Web Services for Presence Based Application Development

Rajesh Karunamurthy

A Thesis

in

The Department

of

Electrical and Computer Engineering

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

September 2004

Your file Votre référence
ISBN: 0-612-94703-3
Our file Notre référence
ISBN: 0-612-94703-3

Abstract

Web Services for Presence Based Application Development

Rajesh Karunamurthy

Presence is a promising technology for developing innovative value added services that differentiates the success of the service providers, seeing that users gaze for novel services. Standardization efforts are ongoing in the presence technology while some organizations like IETF and 3GPP are working in the protocol area. Consortiums like Parlay, JAIN and OMA are working on exposing this potential technology for the application developers via APIs. Interoperability and the flexibility to develop applications with ease would be the foundation for the success of the presence technology. Web services is the next generation middleware that provides flexibility to develop loosely coupled, coarse-grained interfaces with seamless interoperability that makes application development simpler and efficient. These advantages make Web services an attractive middleware for developing ground-breaking value added services using the presence technology. This thesis proposes a novel Web service based API for presence based application development that is protocol, network, language, platform and architecture independent. The proposed API is highly abstracted with the intention that even non-telephony experts like IT developers should be able to incorporate the presence functionalities for creating services. Two case studies are also provided to show the usefulness, completeness and flexibility of the proposed API. A subset of the API is implemented as a Web service gateway and mapped to a SIMPLE based presence server besides that Web service based clients are also prototyped. One of the main aims of this

research work is to see the feasibility of using the Web services middleware for telecommunication application development; therefore performance measurements are also done. The analysis of the results show that although Web services introduces additional overhead because of XML based SOAP messages it is insignificant compared to the interoperability and easier application development features offered by this middleware.

Acknowledgements

Words are not enough for thanking all the people who helped me complete this research work. First of all I would like to take this opportunity to thank my supervisors Dr.Ferhat Khendek and Dr.Roch Glitho for assisting in every move. Although they had busy schedules, they always helped me to solve my problems. I learnt a lot under their tutelage and would always remember them for their guidance, motivation and immense support.

I am thankful to NSERC (the Natural Sciences and Engineering Research Council of Canada) and Ericsson Research and Development, Montreal for assisting me financially. Furthermore I am grateful to Ericsson for providing me an opportunity to work in one of the finest research centers in the world and to Concordia University for giving me all the facilities I needed. I also thank my colleagues for their suggestions and discussions; it was really great working with all of you. I would also like to show my appreciation to Andre Poulin, Jean-Charles Beaudin, Sochea Meas, Vincent Carpentier and last but not least Suresh Krishnan for the help provided by them at Ericsson.

I would like to take this opportunity to thank everyone from my family in India, without them it would have been impossible to complete this work. My uncle, Dr.Radhakrishnan's family in Montreal was of great help and they took care of me very well. I would like to thank them from my heart. For last two years, my friends and roommates were part of my life in Montreal and were always present during the times I needed them the most. I would like to thank everyone for their patience and help.

Rajesh Karunamurthy, September 2004

Table of Contents

List of Figures

Page

List of Tables

Page

List of Acronyms and Abbreviations

3GPP: 3rd Generation Partnership Project

3GPP2: 3rd Generation Partnership Project 2

APEX : Application Extension

BOW: deBate nOW

CLP: Command Line Protocol

CONFAPP: CONFerencing APPlication

CORBA: Common Object Request Broker Architecture

CPIM: Common Presence and Instance Messaging

CS: Circuit Switched

CSP: Client to Server Protocol

DAML-S: DARPA Agent Mark up Language – Services

GIS: Geographic Information System

IANA: Internet Assigned Numbers Authority

IIOP: Internet Inter ORB Protocol

IETF: Internet Engineering Task Force

IMPP: Instant Messaging and Presence Protocol

IMPS: Instant Messaging and Presence Services

IMS: IP Multimedia subsystem

IRC: Internet Relay Chat

J2EE: Java 2 Platform, Enterprise Edition

JCP: Java Community Process

JSPA: Java Specification Participation Agreement

JSR: Java Specification Request

MMS: Multimedia Message Service

MQ: Message Queuing

OASIS: Organization for the Advancement of Structured Information Standards

OGC: Open GIS Consortium, Inc

OMA: Open Mobile Alliance

ORPC: Object Remote Procedure Call

OSA: Open Service Architecture

P&IM: Presence and Instant Messaging

PA: Presence Agent

PAM: Presence and Availability Management

PIDF: Presence Information Data Format

Presentity: Presence Entity

PRIM: Presence and Instant messaging

PS: Packet Switched

QOS: Quality of Service

RFC: Request for Comments

SGML: Standard Genralised Markup Language

SIMPLE: SIP for Instant Messaging and Presence Leveraging Extensions

SIP: Session Initiation Protocol

SMS: Short Message Service

SOA: Service Oriented Architecture

SOAP: Simple Object Acccess protocol

SSP: Server to Server Protocol

UDDI: Universal Description, Discovery and Integration

VHE: Virtual Home Environment

W3C: World Wide Web Consortium

WG: Working Groups

WSDL: Web Services Description Language

WSP: Wireless Session Protocol

WSI: Web Service Interface

WS-I: Web Services Interoperability Organization

WV: Wireless Village

WWW: World Wide Web

XML : Extensible Markup Language

XMPP: Extensible Messaging and Presence Protocol

Chapter 1

Introduction

This chapter presents the background to and motivation for the research. Initially, an overview of the research domain is provided. The problem statement and the proposed solution are discussed in the next section. The last section gives the organization of the remaining part of the thesis.

1.1 Introduction to the Research Domain

This section gives a brief outline of the research area. Initially, an introduction to value added services is given. Afterwards Presence, one of the promising technology for creating value added services, is discussed and finally a small overview of the Web Services technology that allows easier application development is provided.

1.1.1 Value Added Services

Today, telecommunication networks are extensively deployed and used by enormous amount of people for wide variety of services and diverse purposes. Cellular phone has become a basic necessity in this contemporary world and the users gaze for innovative

and attractive services in the market. Basic services like two party voice call are no longer a key issue for the service providers, but the differentiating factor for the service providers are the value added services. Value added services are services that add extra value to the basic service like call forwarding or they could be standalone services like Short Message Service (SMS). There are many value added services available today like call diversion, call screening, call transfer, call hold and SMS. Many innovative technologies like Multimedia Message Service (MMS), Presence, and Push to talk are emerging for enhanced, revolutionary value added services.

1.1.2 The concept of Presence

Thanks to the numerous communication devices, technologies and services available today, however their full potential is not exploited. For instance, according to a research report more than half of the telephone calls made around the globe go unanswered. The presence technology allows entities to disseminate information about their communication capabilities, preferences, availability and status to other entities and allows the prospective usage of all communication devices and technologies.

The birth of World Wide Web (WWW) created a way for global communication and information sharing. Today the most popular application in Internet is the instant messaging application, which is principally a presence-enabled communication service. In instant messaging, users communicate with others using short messages based on the 'availability' of the user, which is basically presence information. The presence technology is used in various areas like gaming, conferencing, collaborative working and

distance learning apart from instant messaging. Some of the prominent presence-enabled applications are Microsoft Office 2003 and IBM Lotus Notes 6.5 [1]. The presence technology can be used as a value added feature with any application that involves interaction between entities.

The presence technology has a wide area of application and its potential has been realized by many standard organizations from Internet and telecommunication domain. Internet Engineering Task Force (IETF) has developed the basic presence model, defined the protocol requirements and also developed an interoperability framework. There are many standard protocols and API's proposed by IETF, 3GPP (3rd Generation Partnership Project), Open Mobile Alliance (OMA), Parlay, and Java API's for Integrated Networks (JAIN) for accessing the presence service. Standardization effect is still on going and the technology is not matured.

1.1.3 Web Services

The World Wide Web is more and more used for application to application communication. The programmatic interfaces made available are referred to as ***Web services*** [2]. The Web services provide a standard means of communication among different software applications, running on a variety of platforms and/or frameworks. According to the World Wide Web Consortium's (W3C) Web Services architecture working group 'Web Services are an amalgam of Distributed Objects, Business to Business and the World Wide Web itself'. Web services is based on Service Oriented Architecture (SOA) and has been adopted widely by many domains like Information

Technology (IT), Telecommunications, Digital Imaging and Geographical Information System, to name a few.

The Web services are the new middleware that provides numerous promising functionalites like high level abstract interfaces, loose coupling between interacting componenets and interoperability. The Web services technology is based on open standards like HTTP, XML (Extensible Markup Language), SOAP (Simple Object Acccess protocol), WSDL (Web Services Description Language) and UDDI (Universal Description, Discovery and Integration). Open standard consortiums like IETF, World Wide Web Consortium, Organization for the Advancement of Structured Information Standards (OASIS) and Web Services Interoperability Organization (WS-I) develop the Web services specifications. Furthermore the high level interfaces provide easier application development. The above mentioned features make Web services an attractive middleware for building standard based, novel interoperable services and to expose the existing functionalities for application development.

1.2 The Problem Statement and the Proposed Solution

In recent years presence technology has gained significant importance and lot of standardization effort is going on within many standard organizations. IP based SIMPLE (SIP for Instant Messaging and Presence Leveraging Extensions), XML based XMPP (Extensible Messaging and Presence Protocol) and Wireless Village (WV) protocols have been proposed and adopted by many companies. These protocols are not interoperable; however there are some solutions proposed by IETF like Common Presence and Instance

Messaging (CPIM) interoperable framework and protocol gateways. Moreover OMA's Interoperability working group has recently started its work on presence interoperability. Interoperability is one of the key issues for the future of presence.

Presence enabled applications have enormous potential and numerous applications can be developed using this technology. While some of the organizations are working on the protocol and interoperability issues of presence, organizations like Parlay and JAIN are working on exposing the presence service to application developers. They have developed interfaces for presence service access irrespective of the underlying protocol, network and architecture. These API's have several problems, some being programming language dependent or technology dependent (Common Object Request Broker Architecture (CORBA) based) and in addition they are at low level of abstraction. The application developers do not have flexibility with the API and more over need to use many methods to even realize simple presence functionality.

The Web Services middleware provides the flexibility to offer high-level, language and platform independent abstract interfaces and besides that provides seamless interoperability. This makes Web Services an attractive technology to develop Application Programming Interfaces (API) for presence. There is no standard Web services based Presence API available, though the work is on going in Parlay-X, which would be eventually used by OMA for providing interoperability.

This thesis proposes a novel Web Services based API for Presence, which is defined at the highest level of abstraction possible. The API defined is protocol, network, language, platform and architecture independent. The proposed API is also flexible and extensible and offers all the potential functionalities possible with the presence service.

These features of the proposed API would make presence based application development easier and faster and furthermore the developer need not be an expert in the presence technology to use the presence functionality with the applications.

As a part of the research work, a subset of the proposed interfaces was implemented as a Web service gateway and mapped to a SIMPLE based presence server. A voice conference application was also developed to show the usability of the interfaces. Performance measurements were taken and analyzed to see the overhead induced by the Web services layer.

1.3 Organization of the Thesis

The second chapter gives an overview of the technologies-Presence and Web Services and discusses the standards available today. The topic on presence technology will explain the basic presence service architecture, presence protocol requirements, proposed presence protocols such as XMPP, SIMPLE, and Wireless Village protocols. A brief overview of the on-going work in the professional organizations is also given. The topic on Web services will elucidate the Web service principles, basic Web service architecture, and the Web service development lifecycle. The Web service protocol stack is also described. The advantages of using Web services in application development conclude the chapter.

The third chapter details about the available APIs for presence based application development. It explains the PAM/Parlay API's; Parlay-X API's and proprietary Web service based presence API's of Nokia. Java based JAIN presence API's is also briefly

discussed and an overview of other relevant Web service API's such as Geographic Information System (GIS) and location based API's is presented. A critical summary on the available API's concludes the chapter and motivates the need for new Web service based API for presence.

The fourth chapter gives a detailed description of the proposed interface and discusses two case studies based on debating and conferencing. This chapter also gives an in-depth explanation of the API and points out the usefulness of all the methods in the API.

The implementation, prototypes and the performance analysis are explained in the fifth chapter. This chapter shows the implementation architecture and explains the details on the mapping of the proposed API to SIMPLE based Ericsson presence server. Details on the test environment, performance measurements and analysis are also specified.

The last chapter concludes the thesis by pointing out the contributions and discusses the possible future work.

Chapter 2

Presence and Web Services

This chapter is organized into two major sections on two technologies, namely Presence and Web Services giving an overview and discussing the standards available in these technologies today. The basic presence model is explained first, after which the presence protocol requirements and the proposed presence protocols such as SIMPLE, XMPP, and Wireless Village protocol are explained. A brief overview of the on-going work in IETF, 3GPP and OMA is also given and a critical summary on the presence technology concludes the discussion on presence. The part on Web services will explain the Web service principles, their architecture, development and the Web services protocol stack. Finally, the advantages of using Web services in application development are explained briefly.

2.1 Presence Technology

Presence technology allows entities to "publish" and "consume" presence information. The presence information may be about capability or preferential information such as communication means, contact address, device status, device priority, device capability, availability, willingness & mood of the user, location or special notes.

2.1.1 A model for Presence Service

Request for Comments (RFC) 2778 [3] describes the basic presence model as shown in figure 2.1. The two main entities involved are the presentity (presence entity) & the watcher, the former is the producer and the latter is the consumer of the presence information. The presence service accepts the presence information from presentity, stores it and distributes to the watchers. Principals are the people, software or groups of people/software that are distinct and act as a single actor for the presence service. Principals may own zero or more presentities and zero or more watchers. They manipulate presentities and watchers using presence user agent and watcher user agent respectively. The presence information that is provided by the presence service to the watchers is determined by the access rules, which are previously defined by the presentities. The presence user agent for a particular Principal, which owns the presentity, verifies the access rules.

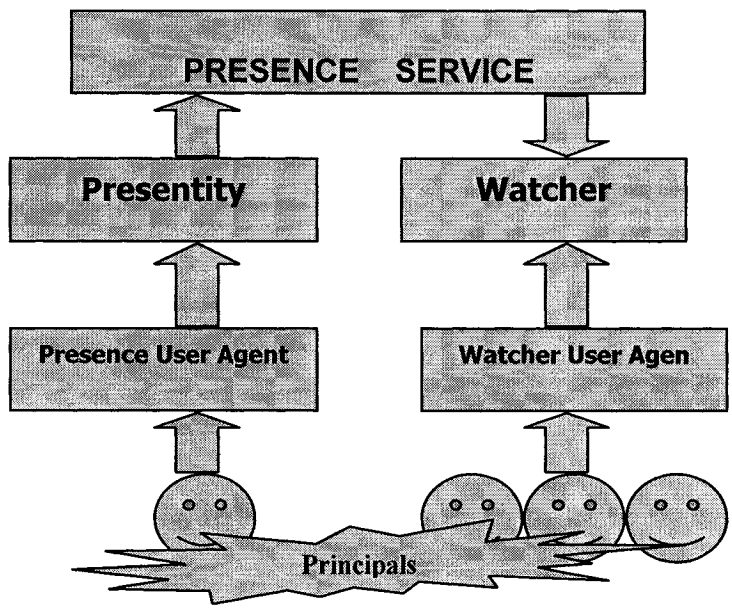


Figure 2.1: Basic Presence Model

Watchers are the entities, which consume the presence information, but the presence service also handles information about watchers called 'watcher information'. The watcher information is also handled similar to presence information. The watcher has visibility rules that are supposed to be manipulated by the watcher user agent before giving out the watcher information. Depending on the consumption of the presence information the watchers are classified into three major types called fetcher, poller and subscriber. Fetcher consumes the presence information as and when required. There is a special kind of fetcher called poller, which fetches presence information at regular intervals. The subscriber is notified about the presence information when there is a change in state of the subscribed presentity.

2.1.2. Presence Protocol Requirements

RFC 2779 [4] describes the protocol requirements for Presence and Instant Messaging (P&IM). This document does not give an exhaustive requirements list, but describes the minimal set of requirements that any P&IM protocol must meet. The document describes a new entity called administrator who controls the local domain and firewall. RFC 2779 describes common security and non-security requirements for both presence and instant messaging services. The shared requirements cover the namespace and administration, scalability, network topology, access control and message encryption and authentication related issues for P&IM. The security requirements further describe security related requirements the protocol must meet. The administrator may have special privileges beyond those allowed by the principal, but it is up to the implementation to impose the

security privileges for administrators. The security requirements covers details on the subscriptions, publications and notifications of the presence information and also requirements related to instant messages. Additional requirements for presence information deals with common presence format, presence lookup and notification, presence caching and replication and performance related issues. Additional instant messaging requirements talks about common message format, reliability, performance and presence format related matters.

2.1.3 Presence Protocols

This section details on the presence protocols available in the public domain. XMPP, SIMPLE and Wireless Village protocol suite will be discussed briefly.

2.1.3.1 Extensible Messaging and Presence Protocol (XMPP)

XMPP [5] is a protocol for streaming XML elements, in order to exchange structured information between any two network-endpoints in close to real-time, however it is mainly used for building presence and instant messaging based applications. The syntax and semantics of the protocol is adopted from the Jabber protocol, which was developed by the Jabber open-source community. The two fundamental concepts of XMPP that helps in the exchange of structured information between presence-aware entities are XML streams and XML stanzas [figure 2.2]. An XML Stream is a container that consists of discrete semantic units of structured information called as XML stanza. <presence/>, <message/> and <iq/> are the three basic XML stanzas defined by the protocol. The core

XMPP protocol is described in the document XMPP: CORE [6] and extensions to the core protocol to provide P&IM service, as specified in RFC 2779, are described in XMPP: Instant Messaging and Presence [7].

XMPP is not devoted to any network architecture although all the deployed systems are based on client-server architecture, where a client communicates with the server using the XMPP protocol over TCP sockets. The server acts as an intelligent abstraction layer for XMPP communication. Gateways are used for communication with non-XMPP messaging systems. Examples are gateways for Internet Relay Chat (IRC), Short Message Service (SMS), SIMPLE, SMTP, and legacy instant messaging networks such as AIM, ICQ, MSN Messenger, and Yahoo! Instant Messenger. The server-to-server communication is an extension of the client-to-server communication and in practice the XMPP system consists of an intercommunicating network of servers. The recommended port for client-server communication is 5222 and for server-server communication is 5269, as registered with the IANA (Internet Assigned Numbers Authority).

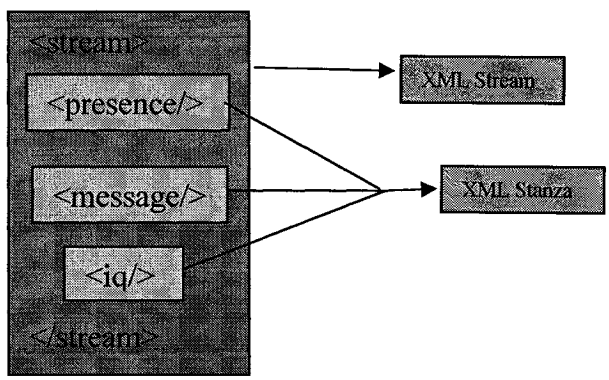


Figure 2.2: XML Streams and XML Stanzas of XMPP Protocol

2.1.3.2 SIP for Instant Messaging and Presence Leveraging Extensions (SIMPLE)

SIMPLE [8] is an application layer protocol that extends SIP for presence and instant messaging. Session Initiation Protocol (SIP) [9] is a text based protocol for creating, modifying and terminating multimedia sessions such as conferences and Internet telephony calls. SIP should be used along with other protocols such as Session Description Protocol (SDP), Media Gateway Control protocol, Real-Time Transport Protocol (RTP) and Real-Time Streaming Protocol (RTSP) to provide a complete service for the user. SIMPLE is defined as an event package that fits within the SIP event framework [10] that is presented briefly.

The SIP event framework is a SIP extension that aims at providing a SIP-specific framework for event notification, which is simple but flexible enough to provide powerful services. The purpose of the event framework is to provide an extensible framework by which SIP nodes can be notified from other nodes when some events have occurred.

The SIP event framework extends the SIP Protocol with two new methods: SUBSCRIBE and NOTIFY and three new headers: Event, Allow-Events and Subscription-State. The resources implementing the event package that suits the framework will publish the event packages it supports using the Allow-Events header. This may be done when it sends an invite request to create a SIP session or when it responds to any of the SIP request messages. The Event header is used by the SIP entities to specify about the interested events of a resource and issues a SUBSCRIBE request to that resource. The Subscription-State header is used to specify the status of the

subscription and is added in the NOTIFY message.

A SUBSCRIBE request establishes a subscription session between the resource and the subscriber. The session established with the INVITE request is orthogonal to the subscription session. The subscription session is valid for a given time called expiration time, which is requested in a SUBSCRIBE message and authorized through one of the 200 response message. The subscriber will be notified about the occurrence of the subscribed events if the subscription session is valid. A subscriber can refresh its subscription before it expires by sending another SUBSCRIBE message within the same subscription session. SUBSCRIBE and NOTIFY requests are handled in the same way as the other SIP messages.

SIMPLE is a specialization of SIP event framework for Presence related events [11]. It is an event package specifying the behavior of the SIP nodes consistent with the event framework, in the generation and processing of SUBSCRIBE and NOTIFY requests for presence related events. One subscribe request is sufficient for a watcher to create subscription sessions with many presentities. This event package introduces a new logical entity called Presence Agent (PA) that is generally co-resident with another entity. PA is capable of accepting subscriptions, storing subscription state and generating notifications when there is a change in the presence state.

SIMPLE works well with both client-server and peer-to-peer based architectures. In a peer-to-peer based architecture the Presence Agent is located in the terminal device, which is responsible for handling the presence related requests for the entities. The presence server in the domain is responsible for handling the presence related requests in the client-server based architecture. Migrating the responsibility of handling the presence

requests is possible to take advantage of both architectures. For example, if a user switches on his/her personal device, which is capable of handling the presence requests, the responsibilities are switched from the presence server to the user's terminal device. Several PA's can handle presence related requests in SIMPLE but all the PA's should have complete knowledge of the presence state of the presentity [12].

SIMPLE inherits the basic mechanisms and capabilities of SIP like addressing, routing, authorization and authentication. SIMPLE uses SIP addresses to identify the presence entities and rely on SIP routing mechanism. SIP proxies and servers handle all the SIMPLE messages like other SIP messages. SIMPLE supports user mobility and multiple devices can manipulate entities presence information. Presence information is very sensitive user related information, so the level of security supported by the basic SIP authentication is inadequate for SIMPLE. SIMPLE does not mandate any authorization mechanism, but it encourages authorization for each presence information access. The implementation can use any existing access control mechanism for the security issues when using SIMPLE. SIMPLE does not support presence information filtering and list management. All the subscriptions have to be maintained by the application using the presence feature.

2.1.3.3 Wireless Village (WV) Protocol Suite

Wireless Village [13] is an organization formed by Ericsson, Motorola and Nokia to promote universal specification for mobile Presence and Instant messaging services. The presence service (defined along with instant messaging, group and shared content

service) by WV is based on client-server model and is accessed by the client by establishing a session with the WV server. The client and server negotiate capabilities with each other when a session (client session) is created; subsequently Client to Server Protocol (CSP) or Command Line Protocol (CLP) governs the client and server interaction. CLP has limited functionalities compared to CSP to support "legacy" clients. The Server to Server Protocol (SSP) is used for the interaction between two servers in which the interactions occurs in a session, different from client session. The CSP, CLP and SSP protocols are binary-based Protocols. The protocol stack is shown in figure 2.3.

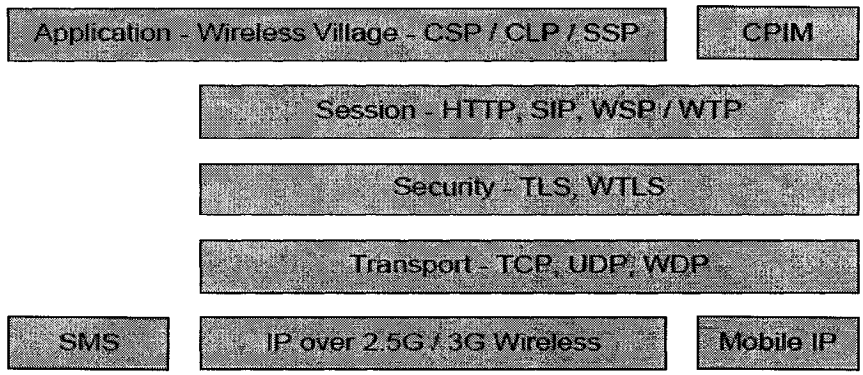


Figure 2.3 : Wireless Village Protocol Stack [13]

The client session once created is destroyed only when the client logs out; otherwise it is independent of the underlying protocols. The destruction and reestablishment of the transport layer also keeps the client session active. All the interactions are through the CSP or CLP requests and responses from client to server or from server to client. WV specifies transport bindings for CSP over WSP [WAP WSP], Hyper Text Transfer Protocol (HTTP) [RFC 2616], HTTPS, or SMS [TS 23.040]. The transport binding for the CSP is logically divided into two separate channels called the data channel and the control channel. The data channel is used for exchanging all CSP

messages and can be temporary unlike the optional control channel. The control channel is used for controlling the data channel, when used should be permanent [14].

The SSP protocol is used to distribute the functionalities between several WV servers and it specifies interaction between inter and intra domain WV servers. In the case of inter domain servers, SSP protocol is used for the interoperation between different domain servers and furthermore SSP protocol is used for bridging the inter domain services with different protocols. CSP and SSP sessions are two different session models, however a single SSP session can support several CSP sessions.

The server session is established when two WV servers authenticate each other and negotiate capabilities, but the session should be established in two-way i.e. from server A to B and also from B to A, in order to initiate SSP transactions between A and B. WV supports HTTP and HTTPS transport bindings for SSP although the SSP transactions are independent of the underlying transport transactions. Symmetric communication is not supported by HTTP/HTTPS hence a SSP session is bound to two HTTP/HTTPS connections. The HTTP/HTTPS connections used for SSP are persistent TCP connections and the HTTP/HTTPS requests and responses are pipelined.

WV supports list management, presence information filtering and partial presence information update. List of presentities can be subscribed for watching which reduces the number of messages transacted and WV has a primitive support for watcher information. WV supports proactive and reactive authorizations of watchers; wherein attribute lists are used for in the former mechanism and "On Request" authorization is done in the latter case. WV does not support soft subscriptions (subscriptions does not time out) and polite blocking i.e.; watchers are always aware of the rejection of presence information access.

2.1.4 Professional Organizations and their Work

This section describes the work done in the various standard organizations in the presence domain. An overview of the ongoing work in IETF, 3GPP and OMA is given.

2.1.4.1 Internet Engineering Task Force (IETF)

Internet Engineering Task Force [15] proposed the presence technology and five Working Groups (WG) were formed to standardize the protocol and data format for the presence and instant messaging. Three of the five WG are actively working on the presence protocol. The Presence and Instant messaging (PRIM) and Application Extension (APEX) working groups have been dissolved. Instant Messaging and Presence Protocol (IMPP) [16] is the lead working group formed in 1998 to develop architecture and define the protocol and data formats for the presence and instant messaging. IMPP WG has published RFC 2778, RFC 2779 and an interoperability framework specifications called Common Presence and Instance Messaging (CPIM) but have not been successful in standardizing a protocol for P&IM.

The Extensible Messaging and Presence Protocol working group is proposing to adopt XMPP, an open XML based protocol; a core in jabber P&IM service, as the presence protocol. The SIMPLE working group is the most active working group and is proposing extensions to SIP for the presence and instant messaging protocol. The XMPP and the SIMPLE protocol is compliant with the presence protocol requirements as proposed in RFC 2779 and also with the CPIM interoperability framework

2.1.4.2 Third Generation Partnership Project (3GPP)

3GPP [17] is working on the standardization of a presence protocol and an API to allow the applications to access the presence service in the wireless domain, while assuring interoperability with the wired domain [18,19]. Initially, 3GPP used the IETF protocol requirements and extended these requirements for the wireless arena. Session Initiation Protocol is already adopted by 3GPP for Internet telephony services and applications. The SIMPLE protocol will also be ultimately extended to encompass aspects such as integrating the presence service in the 3GPP service architecture, charging and administration, maintaining the interoperability with the Internet.

Computing power and network bandwidth are scarce resources in the terminal devices, which access the 3GPP networks through wireless air interfaces, so presence information filtering and partial presence updates are supported by 3GPP. *Presence information filtering* is a mechanism by which watchers can fetch or subscribe required presence information avoiding more traffic in the network.. Publisher also has the facility to update partial presence information and only the values of the updated attributes are notified to the subscribed watchers. 3GPP is also defining attributes to add value to the wireless presence information i.e.; to presentities whose principal is a 3GPP subscriber.

3GPP provides flexibility for the users by supporting different presence information for different watchers, which affects the performance and scalability. Attribute lists are used for authorization and access controls in 3GPP. Additional charging and administration requirements are incorporated in 3GPP, as they are autonomous and independently managed networks.

3GPP networks are very heterogeneous, they contain packet switched (PS), circuit switched (CS), and IP Multimedia subsystem (IMS) domains, moreover they have to interact with PSTN networks and should deal with QoS (Quality of Service) issues. Presence in 3GPP is a inter domain service and so it has to support non-SIP protocols for CS domains.

3GPP defines an open and consistent service architecture based on Virtual Home Environment (VHE) and Open Service Architecture (OSA) concepts. 3GPP adopts Parlay's Presence and Availability Management (PAM) API's for accessing the Presence service by applications. 3GPP, which has adopted client-server based architecture, uses SIP/SIMPLE protocol for non-CS domains and has to support a new protocol for CS domain.

2.1.4.3 Open Mobile Alliance (OMA)

OMA [20] was formed in June 2002, (as on June 2004 is supported by 300 companies from all domains) with a mission to grow the market for the entire mobile industry by removing the barriers and by ensuring seamless application interoperability. OMA works very closely with many international organizations in both wired and wireless domain like IETF, W3C, Java Community Process (JCP), 3GPP and 3rd Generation Partnership Project 2 (3GPP2). There are fourteen working groups functioning in wider areas like architecture, security, device management and mobile protocols. Wireless village, an organization formed by Ericsson, Motorola and Nokia to promote universal specification for mobile Presence and Instant messaging services, consolidated into the OMA and is

now operating as Mobile Instant Messaging and Presence Services (IMPS) WG. The WV specifications are directly adopted and are available as a candidate release version (OMA IMPS V1.1), which are a set of open technical specifications forming an enabler that can be implemented in products and solutions. The WG is concentrating on three main areas of work one being the WV technology, then the SIP/SIMPLE technology, the interoperability between the WV and SIMPLE and also in developing IMPS service enabling interfaces technology.

The IMPS WG is working closely with OMA IOP (Inter Operability) WG for testing the interoperability of SIMPLE and WV technologies; furthermore it is working also with OMA requirements, architecture, security WG for general activities. The OMA MWS (Mobile Web Services) WG is working on the specification that defines the application of Web Services within OMA architecture. The IMPS WG will work with MWS WG to provide mobile web service based specifications and interfaces for presence and instant messaging. The Presence and Availability WG (PAG) is formed to define architecture for OMA presence service that fits the general architecture specified by the architecture WG. This WG is also responsible for specifying enablers for presence and availability (or more generally presence).

The work has just started in the OMA organization and because of the wide acceptance by many big players in the wireless and Internet arena; OMA might bring interoperability in almost across the entire wired and wireless domain.

2.1.5 Discussion

The Presence technology is not matured, but a lot of work is going on in many standard organizations and its working groups. Microsoft, IBM, Sun Microsystems and Novell back up the SIMPLE protocol, while XMPP is supported by Hewlett-Packard, Intel's Wireless Computing Group, Sony and Hitachi, and on the other hand mobile moguls like Ericsson, Nokia and Motorola prefer Wireless Village. The SIMPLE supporters argue that XMPP needs considerable bandwidth, whereas SIMPLE is based on SIP which is not designed for carrying data and the wireless world points out that both the IETF-protocols are not suited for mobile arena. There is a protocol race between XMPP, SIMPLE and Wireless Village, though organizations such as OMA is working on the interoperability between different systems; the ultimate result would be the deployment of systems using all these protocols. Interoperability would be a key issue for the future of presence technology. There is some work done on the XMPP-SIMPLE interoperability in IETF and OMA IOP WG is focusing on presence interoperability using the Web services technology, however they are in the initial stages of work and would take a considerable time to achieve the interoperability.

2.2 Web Services

Web services are programmatic interfaces made available for application-to-application communication. Web services have their roots in component based architecture and borrow concepts like encapsulation and loose coupling from object-oriented programming. They are in the form of software code, which can be invoked and used by the other services/users in a network.

2.2.1 Web Services Principles

The three basic principles of Web Services according to Adam Bosworth [21], one of the pioneers in Web services area, are

1. *Coarse-Grain approach:* This principle deals with the communication efficiency. The interfaces defined for accessing the services that are exposed as Web services should be at a very high level of abstraction, minimizing the interaction between the requesting and the responding components.

2. *Loose Coupling:* Dependency of the interacting components on each other is an important issue in communication. The second principle suggests that the interdependency between two communicating modules must be minimized. Many applications interact with each other for achieving a certain task. If one of the component/application changes, it should not disturb the working of another component.

3. *Asynchronous Communication:* The last principle deals with the mode of communication. The basic method of communication should be asynchronous, though synchronous mechanisms could be used. In Web services world applications interact with many different components at the same time so asynchronous communication, which minimizes the interdependency of the components, is preferred to the synchronous. In practice, the RPC based synchronous mode of communication is more commonly used than the message based asynchronous one.

2.2.2 Web Services Architecture

The Web services architecture [figure 2.4] called as Service Oriented Architecture (SOA) is based on the standard architecture developed by the World Wide Web Consortium's Web Services architecture working group [22]. The basic architecture includes Web services technologies capable of exchanging messages, describing Web services and publishing and discovering Web service descriptions.

2.2.2.1 Roles of Entities

The Web services architecture is based on three different roles played by three different entities namely service provider, service requester and service registry. These entities need not be strictly identical. The service provider is the owner of the service and processes a Web service request. This software agent creates a Web service, describes it

and publishes the service in the service registry. The service is described in a standard formal XML notation called its service description.

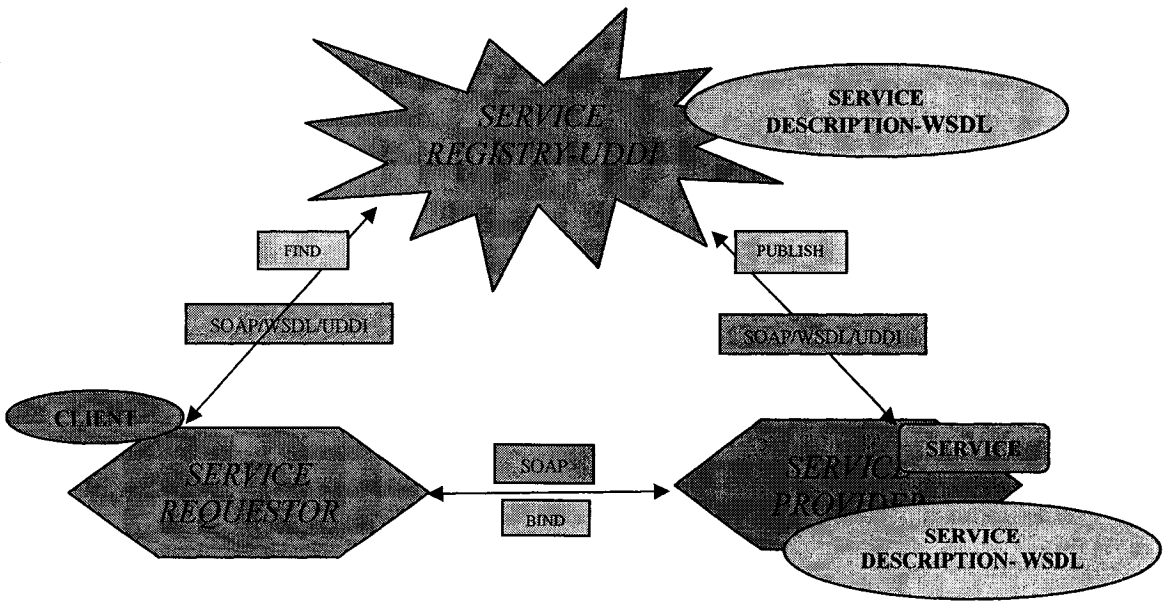


Figure 2.4: Service Oriented Architecture

The service description provides all the information about the service like where to find the service, how to access the service and protocols used. The service registry is an entity, which maintains all service descriptions provided by the service providers. The service requestor is the user of the service. The service requestor software agent can be a service by itself or any user, requiring a service. The service requestor queries about the interested services to the service registry. The service registry provides the lists of available services with its descriptions to the requestor. The service requestor analyses the service description provided by the owner of the service and then interacts with the service provider for using the service. The service requestor can also publish the information to the service requestor directly using any form of communication like email or fax. The service requestor can also be a service provider and vice versa. They are

software agents, which can take dual role. Thus there is a low coupling between the interacting agents in this architecture. The implementation details of the service are never exposed to the service requestor, the service requestor just interacts with the interface provided by the service. The service description may include details about data types, location and binding information.

The communication between the requestor and the provider is mostly by using Simple Object Access Protocol. The services are described using the standard Web Services Description Language. SOAP and WSDL are gaining popularity because they are developed and standardized by the W3C. Other XML based protocol can also be used for communication and similarly other technologies can be used for describing the services. The WSDL and DAML-S are orthogonal. DARPA Agent Mark up Language – Services (DAML-S) can also be used instead of WSDL for Web Services description.

2.2.2.2 Operations Used by Entities

These are the behaviors that must take place for a Web service: publication of the service with its description, finding the service description and binding or invoking the services based on the description of the services. These behaviors are carried out by three different operations namely publish, find and bind as shown in the architecture. The services are of no value unless they are used by some entities.. For a Web service to fetch profit to the provider it needs to be published which can then be discovered and invoked by paying customers. The place to publish the service depends on the application and the

preferences of the service provider. The service requestor uses the find operation to query about the required service locally or from the registry depending on how those types of services are provided. The find operation may be used in the design time or in the runtime. The bind operation is used for invoking or initiating an interaction with the actual service discovered using the find operation. The interactions may be single message one way or multi message conversation or can also be a broadcast from the service requestor to a bunch of services. These interactions can be both synchronous/asynchronous.

2.2.3 Web Services development

The Web Services development lifecycle can be broadly divided into four phases namely the D&D (Development and Description) phase, deployment phase, operational phase and M&M (Maintenance and Management) phase [23].

D&D Phase: The Web Service development and description phase is the phase in which the service is developed from scratch or an already existing application can be modified to a web service or an already available web service and/or application is composed to form a new web service. This is a complex step in which service development itself can have a lifecycle like service design, service implementation and testing. The service interface definition and description is also done in this phase.

Deployment Phase: The deployment phase deals with the publication of the developed

services interfaces and the service descriptions to the service registry/service and also the deployment of the executables for the Web service in the environment.

Operational Phase: In the operational phase the Web Service is ready to be invoked and used. The service provider waits for the usage of the Web service by the service requestor. The operational phase of a particular Web service depends on the type, quality and popularity of the service. If the type of service is a commonly used service like credit card validation service, it may be found, invoked and used by the service requestors often. The Return on Investment (ROI) of a service fully depends on its usage.

M&M Phase: The final phase is the most important and mostly forgotten phase in this life cycle. The ROI for a service is mainly affected by the availability, accessibility, security, performance and quality of service of the underlying services provided. This phase deals with software (service) maintenance and the resulting update and management issues for a web service.

2.2.4 Protocol Stack

IBM developed the conceptual Web Services stack (figure 2.5). The Web services stack is developed based on the Service Oriented Architecture and should support the basic operations: publish, find and bind. The capabilities of the lower layer form the basis for the upper layers. The main issues in the Web services are the QoS, management and security, which should be addressed in each layers of the stack.

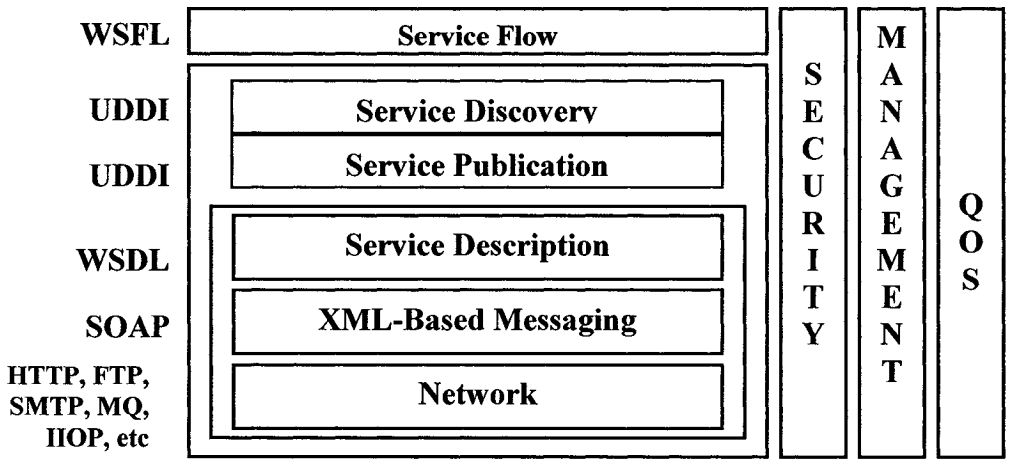


Figure 2.5: Web Services Stack

The technologies that are appropriate for each of the layers of the stack, are HTTP, SOAP [24], WSDL [25] and UDDI [26]. SOAP forms the basic protocol for communication between the service requestor, service provider and service registry. WSDL is the standard service description mechanism, which is based on XML. UDDI is an open initiative, where companies all around the globe can publish and make their service accessible to the requestor. The basis for the Web services stack is the network. It is apparent that Web services are accessed through some network. The Internet accessible Web services should support any of the common network protocols such as HTTP or FTP or SMTP or any transport protocols like TCP. The intranet Web services can be accessed through protocols such as Internet Inter ORB Protocol (IIOP) or Object Remote Procedure Call (ORPC) or Message Queuing (MQ). HTTP is the *de facto* standard because of its wide usage and it passes firewall easily compared to other protocols, thus increasing security. XML is the basis on which other Web service standards such as SOAP and WSDL are built. XML is a standard developed by W3C XML working group. XML is a markup language like HTML and Standard Genralised Markup Language

(SGML) describes a class of data objects called XML documents. XML documents are made up of storage units called entities, which contain either parsed or unparsed data. Service composition and flow is described using Web Services Flow Language (WSFL).

2.2.5 Advantages of using Web Services for Application development

This section discusses the main advantages of Web Services and points out the usefulness of the technology.

1. *Interoperability:* Web Services make disparate systems to communicate effortlessly. They allow developers to build platform and programming language independent applications/services.

2. *Open Standards:* The basic Web services technologies such as SOAP, WSDL and UDDI are open standards developed by World Wide Web Consortium. XML, an open W3C standard is the base for Web services.

3. *Easy Application Development Environment:* Business logic can be exposed as a service using the Web services, so that clients, applications and services can be easily developed by adding minimal application-specific business logic in the client/application side. The high level interfaces hides basic functional details and provides ease in innovative application development.

4. *Reusability:* Most of the applications developed can be exposed as Web services with least amount of coding, providing better Return on Investment. Even legacy based systems and applications can be exposed as Web services.

Chapter 3

Presence Based Application Development: State-of-the-art

This chapter discusses the Application Programming Interfaces available for presence based application development in the public domain. API is a technology that facilitates exchanging messages or data between two or more different software applications. The first section gives an overview on PAM/Parlay interface followed by a brief discussion on Parlay-X API's. The third section describes the proprietary presence Web services interfaces of Nokia, after that Java based JAIN interfaces for presence is discussed. A brief overview of relevant API's such as location based and geographical information system based interfaces is also provided. A critical summary on the different API's available for presence based application development concludes the chapter.

3.1 Presence and Availability Management (PAM)/Parlay

Parlay [27] is developing network independent API's to open the telecom networks for application developers in the IT domain, who outnumber the telecom counterparts. Applications could be developed with ease using the standardized API's, and widely available IT technologies and tools. The IT developers need not be a telecom expert to develop inventive applications. Principally, all the network functionalities should be opened up for application developers to create innovative applications, which are not

only simple to develop and maintain but also must have potential market. The API's are abstracted to be protocol and network independent and are standardized by 3GPP. Parlay API's are based on Common Object Request Broker Architecture.

Initially, PAM forum developed the presence and availability API in the wireless industry. The PAM forum joined Parlay and is now working as Presence and Availability Management working group [28]. This WG is responsible for defining network and protocol independent interfaces for the presence service in Parlay. The PAM API is a part of the Parlay 4.1 specification. The PAM API defines a set of abstract interfaces that can be mapped to any presence protocol in any network. The applications using the PAM API to provide services to the users will be able to handle and manage both preferential and presence information of the agents and identities, independent of the networks and protocols. An identity is the electronic representation or more generally identifier of any entity, while the agent may be any terminal device or software that interacts with the communication networks and services on behalf of the identity. Identities and agents can have aliases and are identified by their names in some naming space. Agents have specific capabilities associated with them, which determine their interactions and participations in some sort of communication. Agents can also have the ability to monitor and proactively report certain information. Identities are associated with the agents and the capabilities of the identities are inherited from the agents.

Dynamic attributes of the identities/agents determine their presence information. The presence information is reported by the identities/agents or by the network. Capabilities of the identities depend on the agents they are associated with. The willingness of the identity to communicate depends on the availability information that

can be calculated by the identity's presence and the preference information. Preferences control the disclosure of presence information of the identity/agent. These PAM interfaces are CORBA based interfaces.

The functionalities covered by the PAM/PARLAY interface are

- Management of identities, their association with agents, agents and their capabilities;
- Management of identity and agent presence;
- Management of availability of the identities;
- Management of events that are related with the above.

3.2 PARLAY-X

Parlay-X [29] working group is focused on providing simple yet powerful, highly abstracted, creative building blocks of telecommunication capabilities. The interfaces provided by this WG will be much simpler than the complex CORBA based interfaces, since these interfaces are abstracted from the CORBA interfaces. The building blocks provided by the Parlay-X group should be easily understandable and utilizable by even non telecom expert to provide highly ground-breaking applications. Homogeneous (E.g.: MMS) or heterogeneous (E.g.: PAM and Mobility) capabilities could be offered as building blocks, but they should always be generic. Figure 3.1 shows the relationship between Parlay and Parlay-X Application Programming Interfaces.

The main goals of Parlay-X are

1. Unless there any convincing services, only functionalities provided by Parlay would be used for making the building blocks.

2. The Interfaces defined should follow KISS (Keep Interface Spec Simple) rule.

3. Only one message exchange between the application and the building block.

Parlay-X Web service specification Version 1.0 [30] exposes SMS, MMS, charging, account management, third party call, user status and terminal location as Web services. As such now, there is no Web Service interface for presence service available in Parlay-X, and it will most likely be available in the next version of the Parlay-X Web service specification.

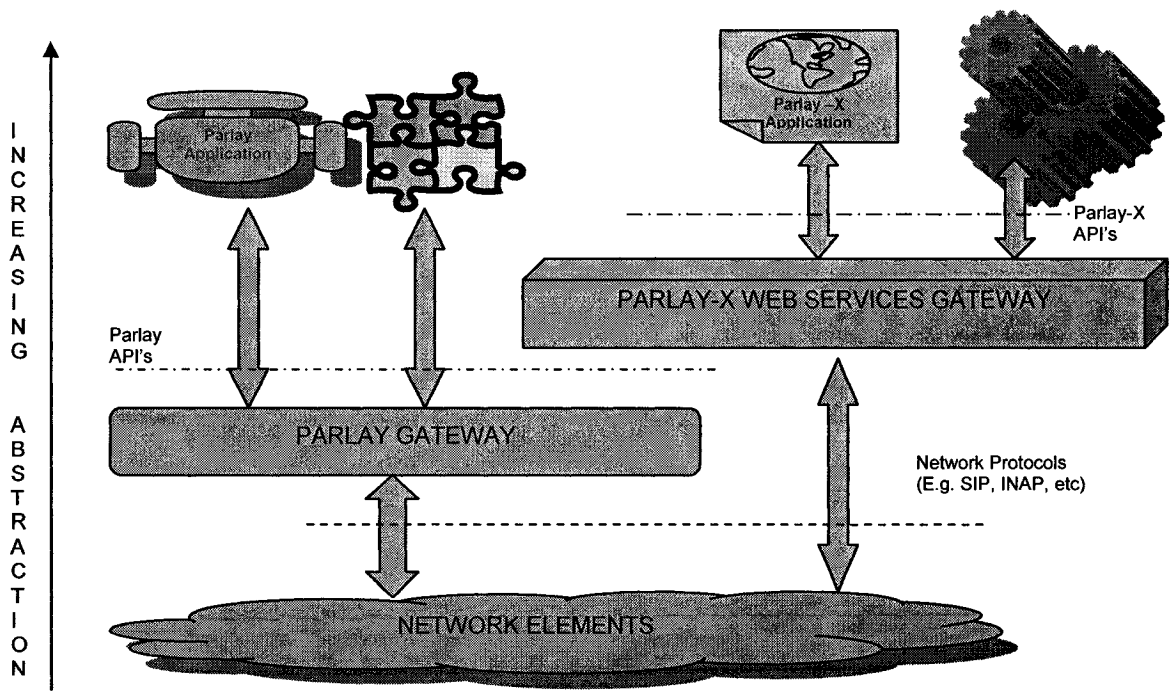


Figure 3.1: Relationship between Parlay and Parlay-X API's

3.3 Nokia's Proprietary Presence API's

Nokia is adopting Web services to expose the functionalities of their server products and have defined Web service interfaces for some of their servers. More information on the Web services usage in Nokia can be found from [31]. Nokia has developed a Presence Server (PS) based on Wireless Village standard. Nokia is exposing the WV based PS using Web services, for applications to employ the presence service using the Web services Interface. The presence Web services interface enables integration of new systems and services with the Nokia presence server and further applications can use these interfaces for using the presence service with ease. The Nokia clients use WV Client Server Protocol for communication with the Presence server. A cluster of servers may also be used for providing high availability, in which case WV Server to Server Protocol will be used for communication between WV servers. The presence Web service interface can be used by clients to send requests to the PS and also by the PS to send notifications to the clients.

The Nokia presence server uses OMA IMPS attributes [32], which can be broadly classified based on the nature of information as client status attributes and user status attributes. The client status attributes gives information about the status, capabilities of the hardware and the software. The status of the clients could be easily reported by the network elements once a client is registered to its network. The user status attributes gives detailed information like availability, preferences, free format status and mood of the user. As Nokia is Wireless Village based it supports contact list management. Users can use many contact lists for distribution of information, proactive authorization and

much more.

The Wireless Village based Presence Web Service Interface 1.1 (WSI) [33] provides access to applications to use the presence functionalities in the Nokia presence server. A wireless village user ID, HTTP authentication username, password and URL are required to access the presence WSI. Authentication and authorization plays a significant security role in the presence WSI. Application acts on behalf of the OMA IMPS user when accessing presence WSI. The transactions can be classified into five categories and there are totally 16 operations in presence WSI. Presence subscription management category has operations to subscribe, unsubscribe and get notified about the presence information, and retrieve watcher and presentity lists. There are two methods in presence fetch and update category to fetch and update the presence information. There are two categories to manage the contact and attribute lists. Creation, updation and deletion of the list members is possible with these category operations. The search category has two operations, one to search for users based on user ID, status, and user name, and the other method is to stop the search.

3.4 Java API's for Integrated Networks (JAIN)

JAIN [34] is an initiative by the Java community for providing Java based open interfaces for accessing communication networks and services. The Interface development is being carried out through the Java Community Process and under the terms of Sun's Java Specification Participation Agreement (JSPA). The main objectives of JAIN initiative are service portability, network independence and open application

development. The goal of the JAIN is to progress the communication industry from many proprietary closed systems to open environment to provide rapid application development and deployment.

JAIN develops application interfaces (E.g.: JAIN SIMPLE Presence) and container interfaces (E.g.: JAIN SLEE-Service Logic Execution Environment) for communication. JAIN has developed three API's for presence based application development. The available presence based API's in JAIN are

A. JAIN Presence (Java Specification Request (JSR) 186) [35]

B. JAIN SIMPLE Presence (JSR 164) [36]

C. JAIN Presence and Availability Management (PAM) API (JSR 123)[37]

JAIN Presence is a generic, protocol agnostic API for presence. This API provides Java based, portable and secured interface for manipulating, managing and controlling presence information between presence servers and clients. The API is independent of the underlying protocol, which would enable innovative application development. This API's status is still "In Progress".

JAIN SIMPLE Presence provides an API to support the use of SIP for Presence and also provides a standard way to interact with the SIP/SIMPLE protocol stack. This API allows manipulation of presence information between clients and servers using SIMPLE protocol. This API development is still in progress.

JAIN SPA Presence and availability Management API is based on PAM Forum 1.0 specification and its adaptation in Parlay 3.0. PAM provides API for management and dissemination of presence and availability information in a secured manner across heterogeneous networks. These interfaces are very abstract and protocol neutral. Users

preferences and enterprise policies determine the manipulation of presence information, which provides both privacy and security. This API specification is withdrawn from the JAIN initiative. JAIN PAM API is at application level, while JAIN SIMPLE API is at protocol level, hence JAIN PAM API can work on top of JAIN SIMPLE Presence API. The JAIN Presence API is the most suitable interfaces for application development compared to the other two JCP specifications.

3.5 Relevant Web Service API's

There are many different kinds of Web service based interfaces defined for many technologies in the recent years. This section briefly discusses some of the API's which are relevant to presence. Presence technology provides user's context related information such as location, status and capabilities. Hence, geographical information system and location based API's would be the most relevant interfaces to be discussed in this section.

3.5.1 Geographical Information System Based API's

Open GIS Consortium, Inc (OGC) [38] is an International, non-profit organization that is leading the development of standards for geospatial and location based services. The main mission of the organization is to deliver open spatial interface specifications. OGC works with government, private and academic sector to create open and extensible API's for GIS. The GIS services can be grouped into three categories: data services, processing services and registry or catalog services. OGC is developing interoperable interfaces for

accessing the basic GIS based services mentioned. OGC has recently published a document that specifies aspects that should be common to multiple or all OWS interface implementation specifications, which includes operation requests and responses, parameters and encoding of the operations. The document specifies a mandatory "Get Capabilities" operation, which allows any client to retrieve metadata about the services available from any server. The server could implement Web Map Service (WMS) or Web Feature Service (WFS) or Web Coverage Service (WCS) as supported by the specification.

3.5.2. Location Based API's

The Open GIS Consortium provides location-based services through 'OpenLS: Core Services' specification. OMA's Location Working Group (LOC), Parlay, JAIN and 3GPP are also participating in the standardization of the location-based services. There are also many proprietary location based Web services like Microsoft MapPoint Web service. Parlay-X Working group has defined open, network and protocol independent location based Web service in the Parlay-X Web Services Specification Version 1.0 [29]. This Web service will be used for retrieving location information, and would enable easy and fast way of developing location-based applications. The location API consist of only one method "getLocation," which takes the users address, whose location information is requested, the desired accuracy and requestor address (optional) as parameters and gives the location related information as output. This Web service API opens the location aware services to IT application developers.

3.6 Summary

There are many API's available for presence based application development but none of them precisely addresses the usage of Web services for presence. The PAM interfaces are designed for server-side aspect of presence. Moreover the PAM Interfaces are CORBA based and not abstract enough to be used directly as Web service interfaces. On the other hand Parlay-X interface for presence service is not defined yet, and their approach will lead to one of the best solutions for Web service based application development. Nokia proposes Web service based Interface but there are number of problems with their interface: - to start with the interface is proprietary, secondly it is Wireless Village dependent and moreover the level of abstraction is very low (protocol level interfaces). JAIN Presence API is protocol independent but it is programming language (Java) dependent, which is not desirable when designing the Web service interfaces. The Web service interface defined for any application development should be protocol, language, platform, network and framework independent and they should be highly abstract and asynchronous. Currently available API standards although satisfy some of the requirements, none of them addresses all the issues. So there is a need for a novel API which would be suitable for presence based application development using Web services.

Chapter 4

The Proposed Web Services Based Application Programming Interface

In the last chapter presence API's available today for application development are thoroughly investigated and found that there is a need for a new API for presence based application development using Web services. This chapter describes in detail the proposed novel Web service API for presence based application development. The first section gives an introduction, which is followed by a description of the methods used in the API. Data types used in the specification is described in the next section. An in-depth discussion of the methods along with the attributes and the status messages is given in the fourth section. Two case studies on the proposed API are presented and a discussion about the proposed API with respect to the case studies is also given. A brief summary of the API concludes the chapter.

4.1 Introduction

The API defined is network, protocol and language independent, i.e., these set of methods constituting the Web service API for presence can be applied to any network or any protocol and suitable for any programming language like Java and C++. The API is

designed at the highest level of abstraction possible and the presence information is notified in asynchronous manner. There are seven Web service based methods defined for presence and watcher information access. There are three methods for presence information access; watcher information access can be done using three methods. The presence information can be consumed in three different ways, and a single subscribe method can handle all these watchers. Contact list management is also supported by the interface and a single method is designed for doing all the operations concerning the contact list.

4.2 Methods used in the API

1. setPresence (UID,UID[ ], CL[ ], TL[ ], AL[ ], Exp_Time, DAL[ ], DAL_only, PD_on, Int_Gra_on, Del_TL[ ]) – This single method supports all the publisher related functionalities of the presence system. This method is used to set and update the partial/full presence information of the presentity (presence entity), to provide preauthorization to the watchers, to enable and set post authorization, to politely deny the watchers and to delete the presence information from the presence server. The rationale for this method is that using this single method all the operations with respect to the presentity can be done.

2. setWatcherInfoAuthorization (UID, UID [], CL [], WAL [], Exp_Time, DWAL [], DWAL_only) – The watcher information is any information about the status of the subscriptions of all watchers. The watcher information typically consists of the status of

the watcher, the event that triggered to the current status, identity of the watchers, duration subscribed for watching and expiration time for the watcher. This method is used to set the authorization rules for "watcher info subscribers" (Users that subscribes to watcher information). The main reason for this method is authorizing the users for getting the watcher information.

3. setCList (UID, AddUsers[ ], DelUsers[ ], MoveUsers[ ], CopyUsers[ ], DelCL[ ], RenameCL[ ], ClearCL[ ]) – Contact list management is one of the most important functions supported by the presence system. This single method is used for doing all the operations for managing the contact list. This method can be extensively used to do all the operations with respect to a contact list (buddy list). This method is used to add, delete, move and copy users in any number of lists and also to rename, delete and clear the users from the lists.

4. subscribe_Presence (UID, UID[ ], CL[ ], AL[ ], DAL_only, Filter_Doc [ ], Exp_Time, R_of_N, Accept_Docs) – The rationale for using this method is all the watcher related functions could be done with this method. This method is used by watchers to subscribe or fetch or poll the presence information of a list of users/contact lists. The user can specify the filtering information using the XML filter document (application/simple-filter +XML [39, 40]), preferred expiration time for the subscription, preferred rate of notification and also the acceptable notification document types. The subscriptions are soft subscriptions, so they have to be refreshed periodically. The allocated expiration time along with the status of the method is returned as a result by this

method. If the application cannot support the filter document and still want to have filtering characteristics it can just specify the set of attributes or the default attribute list to be filtered out. If the subscription duration is set to the current time the presence document is fetched once, but if the duration is greater than zero subscriptions are accepted/refreshed and it is unsubscribed if the expiration time is set to zero. Fetching the presence information with this method at regular intervals of time would take care of the polling mechanism. If the user does not specify the expiration time and rate of notification package specific values are used, in the presence case they are 3600 seconds and 5 seconds respectively [12].

5. notify_Presence (Presence Document) – This method is used to send the presence information document of a list of users/lists in the acceptable format to the watchers. The default document format supported is Presence Information Data Format (PIDF) [41]. If the users request for presence information of a list of users/lists but the particular document format does not support combining all the information in a single document, then the module implementing this method should combine all the documents over time and send the notification as a single file. The main use of this method is to send the requested presence information to the watchers.

6. subscribe_WINFO (UID, UID [ ], CL [ ], WAL [ ], DWAL[], Filter_Doc [ ], Exp_Time, R_of_N, Accept_Docs) – The rationale for using this method is for retrieving the watcher information. This method is used by the watcher info subscribers for subscribing to or fetching or polling the watcher information. This method is used in

the similar way as the Subscribe_ Presence but here watcher information is used instead of presence information. This method also supports filtering using the filter document or by specifying required attributes. Preferred expiration time, rate of notification and the acceptable documents can also be specified. Package specific default values for expiration time and rate of notification are one hour and five seconds respectively [42]. The reason behind not using a generic 'subscribe' method for both the presence and watcher information retrieval is that both the information are fundamentally different and both have different requirement so the same set of attributes could not used for accessing both the information.

7. notify_WINFO (Watcher Info Document) – Acceptable watcher info document is sent, based on the subscription of the user. If the watcher info subscriber doesn't specify the document type then the default document type Watcher Info document [43] is sent. The foremost use of this method is to send the requested watcher information to the entities.

4.3 Detailed Data types used in the proposal

The table 4.1 explains the basic conventions and simple data types used in the proposal. The first column specifies the name of the attribute used in the specification, the next column gives a brief explanation of the attribute and the last column gives the data type.

Polite Denial: Polite denial is a mechanism by which unauthorized watchers are sent false authorization information by the system in order to make them feel they are authorized.

Interactive Granting: Watchers request for presence information access, the presence system checks the publisher's access rules and either grants or rejects authorization for the users. This technique is called as Interactive Granting.

Name	Explanation	Type
UID (PID) - User Identity (Presence Identity)	It can be any ID like SIP ID or Wireless Village ID or any URI or any system specific ID that is unique.	String of alphanumeric characters not more than 30 digits
CL - Contact List	Identity to determine the contact list uniquely.	String of alphanumeric characters not more than 30 digits
AL- Attribute List	List of attributes supported by the presence service. The interfaces neither restrict nor propose any attributes to be supported by the presence service.	Enumerated data type
WAL - Watcher Attribute list	Watcher attributes are very limited set of attributes that specify the state of the watchers, but the information is as	Enumerated data type

	sensitive as presence information itself. The WAL is used to authorize the set of users for accessing the W.I and for filtering the watcher information.	
TL - Tuple List	The list is a structure that contains a tuple ID and a list of tuple values.	Complex data type [Refer Table 4.2]
Exp_Time – Preferred Expiration Time	The time after which the data should not have value. This absolute time is expressed in seconds.	Integer
DAL - Default Attribute List	The default attribute list is set by the presentity	Enumerated data type
DWAL - Default Watcher Attribute List	This default watcher attribute list is set by the presentity	Enumerated data type
DAL_only - Default attribute list enabler	If true, only the default attribute list is requested or disclosed	Boolean
DWAL_only - Default Watcher attribute list enabler	If true, only the default watcher attribute list is requested or disclosed	Boolean
PD_on - Polite Denial enabler	Polite denial is enabled when this value is set to true	Boolean
Int_Gra_on -	Post authorization is enabled when	Boolean

Interactive Granting Enabler	the value is true.	
Del_TL - Delete Tuple List.	List of tuples to be deleted for the presentity	Tuple List. [Refer Table 4.2]
R_of_N - Preferred Rate of Notification.	The rate at which notifications are supposed to be sent. This time is expressed in seconds and is not a relative value.	Integer
Accept_Docs - Acceptable Documents	The list of documents that can be accepted by the subscriber or fetcher. Depending on the list the notifications are sent.	Enumerated data type
Filter_Doc - Filter Document	The document helps in getting filtered presence/watcher information. The filter document could be any document acceptable by both the client and the server. Application/ simple-filer+xml document is recommended as the filter document.	Enumerated data type
AddUsers	This attribute is used to add new users in the contact list.	CL_Type [Refer Table 4.2]
DelUsers	This attribute is used to delete users from the contact list.	CL_Type [Refer Table 4.2]

MoveUsers	Used for moving the users from one contact list to another contact list.	CL_Op [Refer Table 4.2]
CopyUsers	Used for copying a list of users from one contact list to another. The original copy in retained in the old contact list.	CL_Op Complex Data Type. [Refer Table 4.2]
DelCL- Delete Contact List	It is used to delete the contact list. The user list in the contact list is also deleted along with the contact list.	Contact List
RenameCL - Rename Contact List	Used to rename the contact list. The new contact list must be a valid.	CL_Target [Refer table 2]
ClearCL- Clear Contact List	It is used to clear the contents of the contact list, but the list remains empty once it is cleaned.	Complex Data type [Refer table 2]

Table 4.1: Conventions and simple Data types used in the API

Table 4.2 represents the complex data types used in the specification of the web service interface for presence. The first column gives the name of the data type, and the next column briefly describes the data type. The last column explains composition of the complex data type. The "consist of" column (M) should be interpreted as mandatory and (O) as optional.

Name	Description	Consist of
CL_User	Contact List User data	UID (M) –User Identity Nick (O) – Nickname for the user.
CL_Type	Contact List Type	CL_User (M) – Contact List User CL (M)-Contact List
CL_Target	Contact List along with Target CL- When users are copied and moved from one CL to another it require originating and target contact lists.	CL (M)- Originating CL TCL (M)-Target CL
CL_Op	Contact List Operation- Operation on contact list using CL_User and CL_Target	CL_User (M) – Contact List User CL_Target – CL with Target
TL	Tuple List. The list is a structure which contains a tuple ID and a list of tuple values	TID (O) – Tuple ID. The tuple ID can be any unique string of alphanumeric characters. TVAL (O) –Tuple Values. The values to be filled in the tuple for the presence attribute.
Status_Op	Status of the Operation. Gives a Boolean status and status message.	Status (M) (Boolean) –Status of the operation Status_ Mes (O) (String) –Status message
Sub_Status	Status of the operation with status	Status_of_Op (M) (Status_Op) –

	value, message and Expiration time set by the Presence server.	Status of operation Exp_Time (M) – Expiration time set by the server in seconds

Table 4.2: Complex Data Types used in the API

4.4 Methods in detail

The details of the methods with the parameters and error codes are discussed in the following tables. The attribute returned as a result of the method is given in the bracket before every method (For Example: (Status_Op) setPresence means the setPresence methods returns the structure Status_Op). In the following tables (M) should be interpreted as mandatory attribute and (O) should be assumed as optional attribute.

Attributes Taken	***UID (M)*** - User Identity. Identifies the presentity. ***UID [ ] (O)*** - List of user identities for authorization. ***CL [ ] (O)*** - List of contact lists for authorizing ***TL [ ] (O)*** – List of tuples to be set and also used for authorization. ***AL [ ] (O)*** – List of attributes to be authorized for the particular list of users and/or contact lists. ***Exp_Time (O)*** – After this particular expiration time the presence information and authorization information has no value. ***DAL [ ] (O) (AL[ ])*** - Default Attribute list used for authorizing and filtering the presence information with respect to the presentity

	DAL_only (O) (Boolean) – To set Default attribute list for authorization and also for filtering ***PD_On (O) (Boolean)*** - Enable Polite denial ***Int_Gra (O) (Boolean)*** – To enable Post authorization ***Del_TL [ ] (O) (TL[ ])*** – Tuple list to be deleted
Status Messages	Method Succeeded User ID does not exist Contact List does not exist Bad Presence Attribute Tuple ID already exist Bad Tuple Value (Bad Presence Value) Expiration Time too Small Default Attribute List does not exist Polite Denial not supported Interactive Granting not supported Tuple ID does not exist Bad Parameter Permission Denied Server Error Server too busy

Table 4.3: (Status_Op) setPresence Method

Attributes Taken	***UID (M)*** - User Identity. Identifies the user who authorizes the watcher info subscribers. ***UID [ ] (O)*** - List of user identities for authorization. ***CL [ ] (O)*** - List of contact lists for authorizing ***WAL [ ] (O)*** – List of watcher attributes to be authorized for the particular list of users and/or contact lists. ***Exp_Time (O)*** – After this particular expiration times the watcher information and authorization information has no value. ***DWAL [ ] (O) (AL[ ])*** - Default watcher attribute list used for authorizing ***DWAL_only (O) (Boolean)*** – To set default attribute list as authorized list of attribute.
Status Messages	Method Succeeded User ID does not exist Contact List does not exist Bad Watcher Attribute Expiration Time too Small Default Watcher Attribute List does not exist Bad Parameter Permission Denied Server Error/Server too busy

Table 4.4: (Status_Op) setWatcherAuthorization Method

Attributes Taken	***UID (M)*** - User Identity. Identifies the user whose contact list is managed. ***AddUsers[] (O) (CL_Type[ ])*** – To add a list of users to a contact list. The attribute allows adding different users to different contact list at the same time. ***DelUsers[] (O) (CL_Type)*** - To delete a list of users from a contact list. The abstraction level is much higher than the normal delete user methods provided by other interfaces. ***MoveUsers[] (O) (CL_Op)*** - To move users from one contact list to another. ***Copy_Users[] (O) (CL_Op)*** – To copy users from one group to another. ***Del_CL (O) (CL [])*** – To delete the Contact list ***Rename_CL (O) (CL_Target[])*** – To Rename the contact list ***Clear_CL (O) (CL [])*** – To clear the users from the contact list. The list become empty once this is done to any contact list.
Status Messages	Method Succeeded User ID does not exist Contact List Unacceptable Too many users in the contact list Service Not Supported Bad Parameter Permission Denied

	Server Error Server too busy

Table 4.5: (Status_Op) setCList Method

Attributes **Taken**	***UID (M)*** - Identifies the watcher (Subscriber/Fetcher/Poller) ***UID [ ] (O)*** - List of user identities whose P.I. is requested ***CL [ ] (O)*** - To get the presence information of the users in any number of contact lists ***Exp_Time (O)*** – Requested expiration time to get notification documents but could be modified to a time smaller than the specified time, depending on the polices of the server. In order to keep the subscriptions active, it should be refreshed regularly depending on the expiration time. By changing the expiration time the watcher can unsubscribe, resubscribe or fetch the presence document. ***AL [] (O)*** - List of attributes requested by the watcher. This is a preliminary filtering mechanism that can be used by users incapable of using the filtering document method. ***DAL_on (O) (Boolean)*** – To filter out all the attributes except the presentity specified default attribute list. ***FilterDoc[] (O) (String[])*** - List of documents to specify the filters to be used with the presence information retrieval. ***R_of_N[] (O) (integer)*** – Requested Rate of Notification. This can be

	modified because of the server policies and capabilities, but the rate of notification should never be faster than what is requested. ***Accepted_Docs (O) (String [])*** - List of documents that can be accepted by user/application. The document is sent in the notification message after the subscription is accepted.
Status Messages	Method Succeeded User ID does not exist Contact List does not exist Bad Presence Attribute Filter Document not supported No Documents acceptable Rate of notification too fast Expiration Time too Small Default Attribute List does not exist Service Not Supported Bad Parameter Permission Denied Server Error/Server too busy

Table 4.6: (Sub_Status) subscribe_Presence Method

Attributes Taken	***Notify_doc[] (M) (String[])*** - Notification Document is sent which is supported by both client and server. The process and rate of

	notification depends on the module implementing these interfaces.
Status Messages	Method Succeeded Document Not Supported

Table 4.7: (Status_op) notify_Presence Method

Attributes **Taken**	***UID (M)*** - Identifies the watcher info subscribers ***UID [ ] (O)*** - List of user identities whose watcher information is requested ***CL [ ] (O)*** - To get the watcher information of the users in any contact list ***Exp_Time (O)*** – Requested expiration time to get notification documents but this expiration time could be shortened depending on the policies of the server. The subscriptions should be regularly refreshed to keep them active. By changing the expiration time the watcher info subscribers can unsubscribe, re subscribe or fetch the watcher document. ***AL[] (O)*** - List of attributes requested for filtering the unwanted watcher information fetched or sent in the notification. ***DWAL_on (O) (Boolean)*** – To request only the default watcher attribute list, all the other attributes are filtered out. ***FilterDoc[] (O) (String[])*** - List of documents to specify the filters to be used with the watcher information retrieval.

	R_of_N[] (O) (integer) – Requested Rate of Notification. Depending on the local policies server can further increase the rate of notification timing to send notification in longer intervals. ***Accepted_Docs (O) (String[])*** - List of documents that can be accepted by user/application and is fetched or subscribed by the watcher info subscriber
Status **Messages**	Method Succeeded User ID does not exist Contact List does not exist Bad watcher Attribute Filter Document not supported No Documents acceptable Rate of notification too fast Expiration Time too Small Default Watcher Attribute List does not exist Service Not Supported Bad Parameter Permission Denied Server Error Server too busy

Table 4.8: (Sub_Status) subscribe_WINFO Method

Attributes Taken	***Notify_doc[] (M) (String[])*** - Notification document sent to the client by the server. The server based on the local policies can change the subscription duration and the rate of notification.
Status Messages	Method Succeeded Document Not Supported

Table 4.9: (Status_op) notify_WINFO Method

4.5 Two case studies based on the proposed API

This section describes two case studies that are used to describe the usefulness of the Web service based presence API. The two applications described help in realizing the importance of the methods defined in the API. The high level interface definition helps in reducing the number of transactions between the presence server and the clients (users and the application). The dial out conferencing application was developed as a prototype at Ericsson. A small discussion about the usefulness, completeness and flexibility is done at the end of this section.

4.5.1. Case Study One - Application BOW (deBate nOW)

Motivation: Human beings enjoy discussions and debates. Debating application is a service that helps people to debate about a particular topic with voice or video conferencing and with or without text messaging. These applications can aid in having

public or private, pre-arranged or ad hoc debates (debates without a pre fixed time and date). Today there are very few debate services in web that allow for pre-arranged debates. There are almost no Ad hoc debate applications.

Working: BOW [figure 4.1] is a free presence-based Ad hoc debate application. It allows users to debate with the help of multiparty conferencing and text chat. The application chooses a "current" controversial topic and publishes the topic in the presence server WITHOUT specifying the date and time. Interested users watch the topics posted by the application and if interested register for the debate by some means like sending a mail or filling a small form or sending just their contact addresses by instant messaging. The users also authorize the BOW application for watching them. Registered users whenever feel like debating change the status to "For Debate". The application subscribes to the registered users presence (status) information and starts a multiparty conference and chat automatically when at least 5 users are ready "For Debate". New users can also join the on going debates; therefore the application also subscribes to the watcher information of all the registered users. The presence server notifies the watcher information when new users join, or the old users leave the conference. The application then reactively authorizes the new users for joining the on going debates or starts a new debate. Contact list management is not an important issue because the service is a free one.

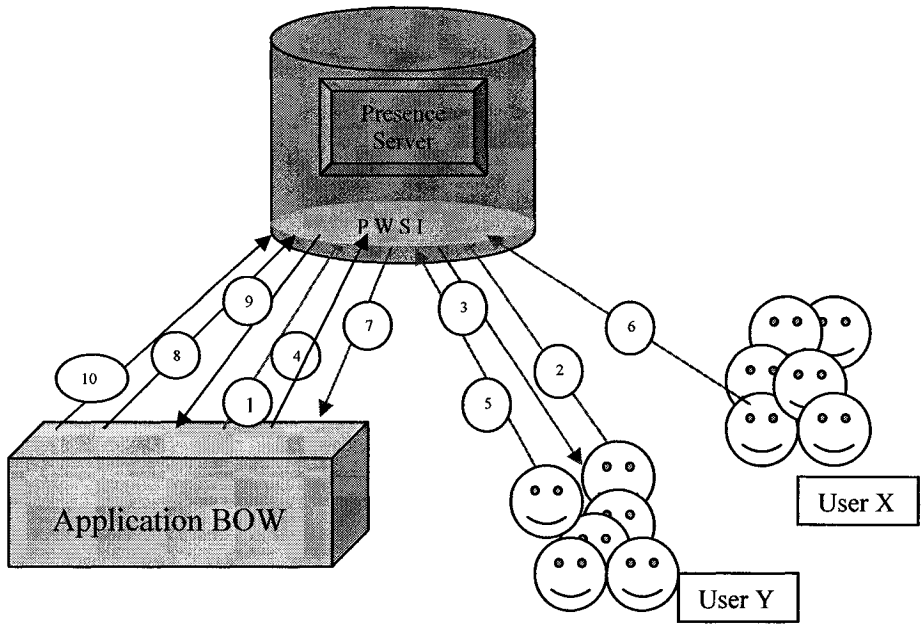


Figure 4.1: deBate nOW application

Explanation on Transacted Messages between Application and Users:

1. *setPresence (BOW, anyone, "IRAQ WAR", 7200, Int_Gra_on, PD_on)* - The BOW application sets the debate topic as "IRAQ WAR" for 2 hours. BOW also preauthorizes any users and enables post authorization.

2. *subscribe_Presence (userX, BOW, Topic, filter_all, 0, any_Document_Type)* - User X fetches only the debate topic and filters out all the other information from application BOW. The user also specifies that he can accept any type of document

3. *notify_Presence ("IRAQ WAR")* - The application sends the document with just topic value as "IRAQ WAR" to user X.

4. *subscribe_Presence (BOW, all debate users, status, filter_all, 3600)* - The users register for the particular debate topic (Iraq war) by some means. The application then susbcribes for the status information of all the registered users for 1 hour.

5. *setPresence (user Y, BOW, "For Debate", 1800)* - User Y authorizes BOW for the subscription of the presence information and sets the status to "For Debate" for 30 minutes.

6. *setWatcherAuthorization (user X,BOW, 4800)* - BOW is authorized by user X for getting its watcher information for 90 minutes.

7. *notify_Presence (PIDF document of User Y with status "For Debate")* - Application BOW receives the PIDF document of user Y with status as "For Debate"
After there are at least 4 users who are interested in the debate are available a audio/video conferencing with text chat adoptability is created by BOW using some technology for example by using the Web service interface for call control and instant messaging.

8. *subscribe_WINFO (BOW, all Debate users, 3600)* - The application BOW subscribes to the watcher information of all the debate users for one hour.

9. *notify_WINFO (winfo document)* - BOW is notified if there is a change in watcher information of some registered user.

10. *setPresence (BOW, all pending users, 3600)* - The debate application authorizes all the pending users for one hour.

4.5.2. Case Study Two - CONFAPP (CONFerencing APPlication)

Motivation: Conferencing is one the most important applications in the communication industry and Internet telephony. The conferencing application can exploit the presence functionalities to check the availability, preference and capabilities of the user and in addition can manage the contact list with the proposed API. There are many conferencing applications available in the market, but there are very few presence-enabled applications that use very primitive and proprietary technologies. There is a need for presence based conferencing application, which is developed using open standards like the proposed API.

Working: CONFAPP [figure 4.2] is a service that helps users to establish prearranged dial in or dial out conferences. Companies can pay and use the application for conferencing between their employees. In the dial-in model users are given an address by the application and they dial in to participate in the conference. In the dial out model the users are watched by the application and when a minimum number of participants are available the conference is automatically started by using some other conferencing API's or by using any other mechanism. The application maintains a contact list for keeping track of the users and the conferences participated by them, which might be then used for billing purposes. As an example if the application has many companies as regular customers, they could maintain a permanent user list for all these companies in their contact list.

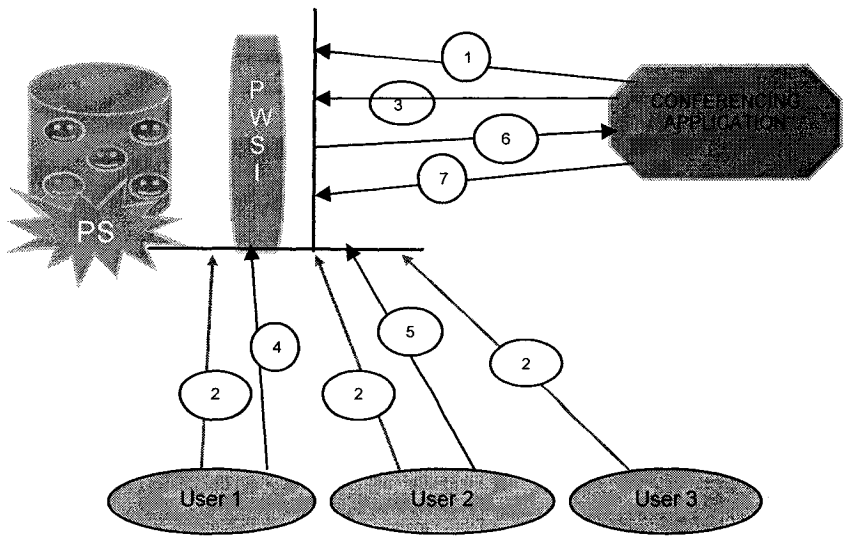


Figure 4.2: Dial-out model conference usage (Presence side)

Explanation on Transacted Messages between Application and Users:

1. *setCList (CONFAPP, AddUsers [{user1, user2, user3}, current_attendees])* - The conferencing application adds the users(user1, user2, user3) to the contact list "current_attendees".

2. *setPresence (user x, CONFAPP, DAL [status],3600, DAL_on)* - The users preauthorize the application and sets the default attribute to be the "status" and enables only that information for the application .

3. *subscribe_Presence (CONFAPP,current_attendees, status , filter_all, 6000, 15, any_doc_type)* - The application subscribes only to the status attribute with rate of notification as 15 seconds for 10 hours and also accepts any type of presence document.

4. *setPresence (user 1, [status=online], 3600)* - User 1 changes the status to online for 1 hour.

5. *setPresence (user 2, [status=online], 3600)* - User 2 also comes online for one hour.

6. *notify_Presence(PIDF documents of user 1 and 2 with status "online")* - The application is notified about the online status of both users. The conference is automatically started when at least two users are online by some mechanism.

7. *setCList(CONFAPP, AddUsers [{user A, user B, user C}, current_attendees], MoveUsers[{user2},current_attendess,ZTEL_employees],CopyUsers[{user3}, current_attendess, 3rdParty_HR] DeleteUsers[{user1}], ClearCL[hackers])* - New users (userA, user B, user C) are added to the current_attendees contact list. User2 is moved from current_attendees to ZTEL_Employees contact list, user3 is copied to a new contact list called 3rdParty_HR and user1 is deleted from the old contact list. Contact list 'hackers' is cleared.

4.5.3 Discussion about the API with respect to the Case Studies

These two case studies show the usefulness of the API by using a wide range of presence functionalities with the help of the proposed API. The watcher information functionalities like watcher information authorization, subscribing to and retrieving the watcher information, and presence information functionalities like preauthorization, presence information setting and retrieval are used in the first case study with the proposed API. The conferencing case study uses the methods of the API to perform the contact list management, and presence information functionalities like post authorization, setting and retrieval of presence information, partial presence information update and presence information filtering. It should be noted that all the presence related functionalities are

used in the two case studies with the help of the API. All the methods designed in the API are used with these two case studies. This proves the completeness of the API for presence functionalities.

The usage of presence functionality in the complex debating and conferencing application becomes simple because of the proposed API. The number of methods used for performing a complex Presence related functionality is reduced by a significant amount. For Example, Presence information setting, authorization and polite denial are done using the single '*setPresence*' method (transacted message 1) in the first case study. In the second case study a single '*setCList*' method (transacted message 7) is used for adding, moving, copying and deleting users from a contact list and also to clear another contact list. This shows that many diverse operations can be performed using the methods of the API. In the first case study the '*subscribe_Presence*' method is used for fetching the presence information (transacted message 2) and subscribing to the presence information (transacted message 4). This shows the flexibility of the methods for performing completely different watcher functions.

4.6 Summary of the API

All the presence functionalities such as presence information creation, deletion, updation and retrieval, watcher information functionalities, contact list management, preauthorization, post authorization, partial presence information update and presence information filtering can be done with the proposed API. The API is designed at a very high level of abstraction; consequently most of the presence functionalities can be

accessed with minimum number of methods using the proposed API. The completeness of the API is also shown with the help of two case studies.

Chapter 5

Prototypes and Performance Study

The implementation, prototypes and the performance analysis are discussed in this chapter. This chapter shows the implementation architecture of the Web service based presence system, gives a concise overview of the different components in the system and explains the mapping of the proposed API to SIMPLE based Ericsson presence server. Details on the test environment, performance measurements and analysis are also provided.

5.1 Architecture of the Web Service based Presence System

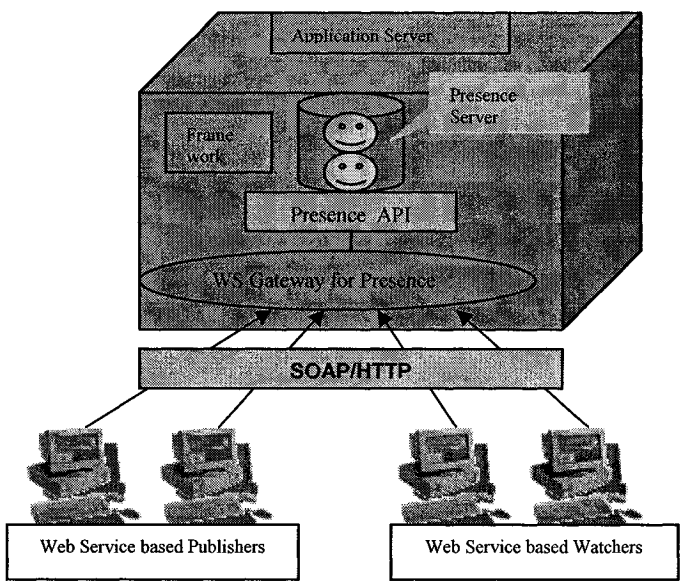


Figure 5.1: Architecture of the Web Service based Presence System

This Web Service based Presence system (figure 5.1) uses the client-server architecture. The presence server runs on an application server that supports Web services. The framework is generally used for configuring presence services and the users for using the presence services. The protocol independent API developed in this work is implemented as a Web service gateway and mapped to the presence server API, which is at a low level of abstraction. The Web service based clients (publisher and watcher) could use RPC (Remote Procedure Call) mode or messaging mode to communicate with the gateway but uses SOAP as the basic protocol for communication. The publisher stores the presence information which is retrieved by the watchers. The Web service gateway for presence, the Web service based watcher client and the Web service based publisher client is developed as a part of the research work. The application server, presence server and the framework were developed by different research teams. Detail on each component in the presence system is discussed now.

5.1.1 Application Server

The application server used is a Java 2 Platform, Enterprise Edition (J2EE) based Apache Axis enabled server built at Ericsson. Apache Axis is an implementation of the SOAP protocol. The application server is platform independent that runs all the other server side components of the presence system.

5.1.2 Framework

The framework is used for registering the presence service to the application server. The framework is also used for creating and deleting the users that can act as both publishers and watchers.

5.1.3 Presence Server

The Presence server is developed as a J2EE application that runs on the Ericsson's application server. This SIMPLE based presence server exposes its capabilities to third party applications using an API.

5.1.4 Presence Web Service Gateway – Prototype

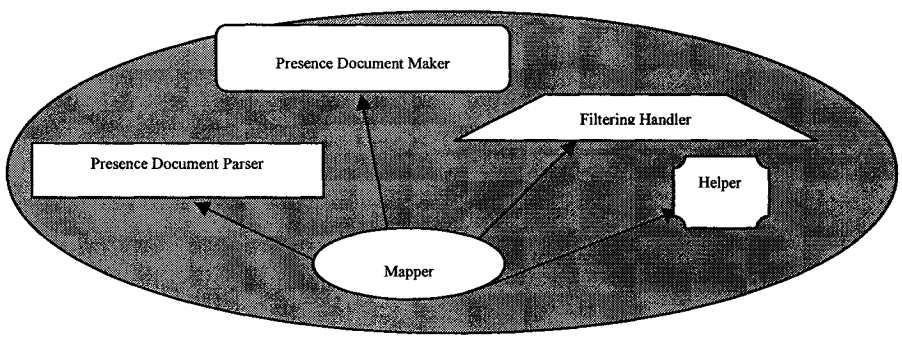


Figure 5.2: Basic Architecture of the Web Service Gateway for Presence

The Web service gateway for presence is to be used along with the SIMPLE based Ericsson presence server. The basic architecture of the gateway is shown in figure 5.2.

The mapper is the coordinating module of the gateway that interacts with the other modules like presence document parser, presence document maker, filtering handler and the helper to process the presence requests. The presence document parser and the presence document maker can parse and create PIDF documents that must be supported by all presence servers. The filtering handler supports the parsing of standard filter document proposed by SIMPLE. The helper is used for coordination between different modules. The architecture of the gateway is flexible enough to be adapted to any presence server, based on any technology. For example, if the underlying presence server is changed to a 3GPP based presence server the mapper is the only building block that has to be changed, the other modules could be used as it is.

The gateway implements only limited presence information functionality (setPresence, subscribe_Presence and notify_Presence methods) that could be mapped to the presence server's Java API. The other functionalities such as the contact list management and watcher Information functionalities are not implemented because of the constraint of the Java API for the third party developers. The detailed class diagram for the gateway is given in the Figure 5.3. The PIAccess is the central mapper class which coordinates with other classes like PIAccessPublishHelper, AccessWatcherHelper, PresenceHandler, PresenceDocumentMaker and FilteringHandler. The classes used by PresenceHandler and FilteringHandler such as Presence, Tuple, Filter, What, Trigger, ThreegppAttributes are generated using the castor Framework [44]. The castor framework generates Java code from the XML schema for accessing the XML elements.

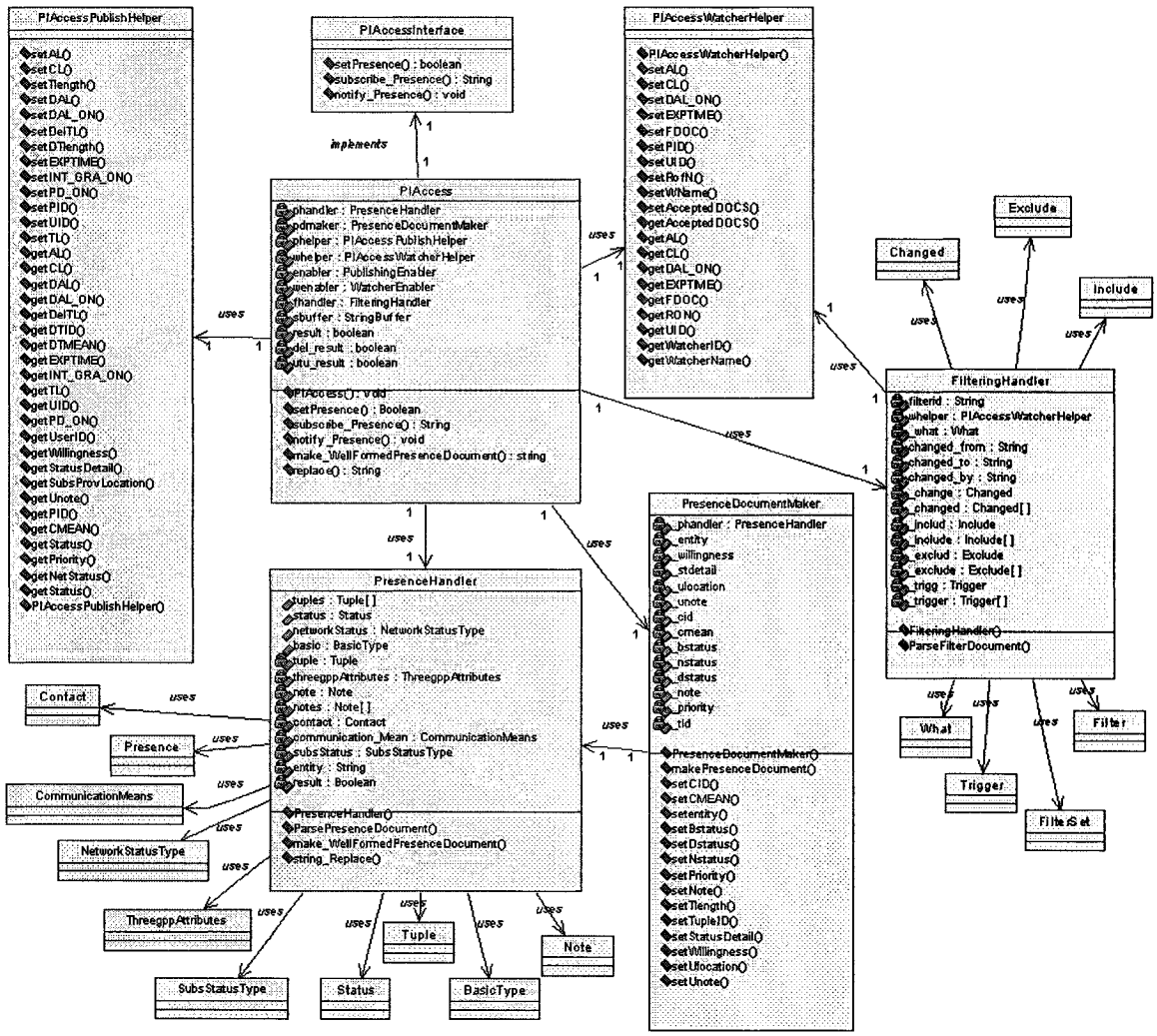


Figure 5.3: Class Diagram of the Web Service Gateway for Presence

5.1.5 Web Service based Presence Clients – Prototypes

The Web service based clients; the publisher and the watcher were also developed as a part of the research work along with the Web service gateway for presence. The publisher and watcher work only in RPC mode of communication and are also Apache Axis based. The clients takes the input from the user using the graphical user interfaces (Figure 5.4

and 5.5), aggregates the information and sends a SOAP based RPC call to the server at a particular endpoint (IP Address and Port number).

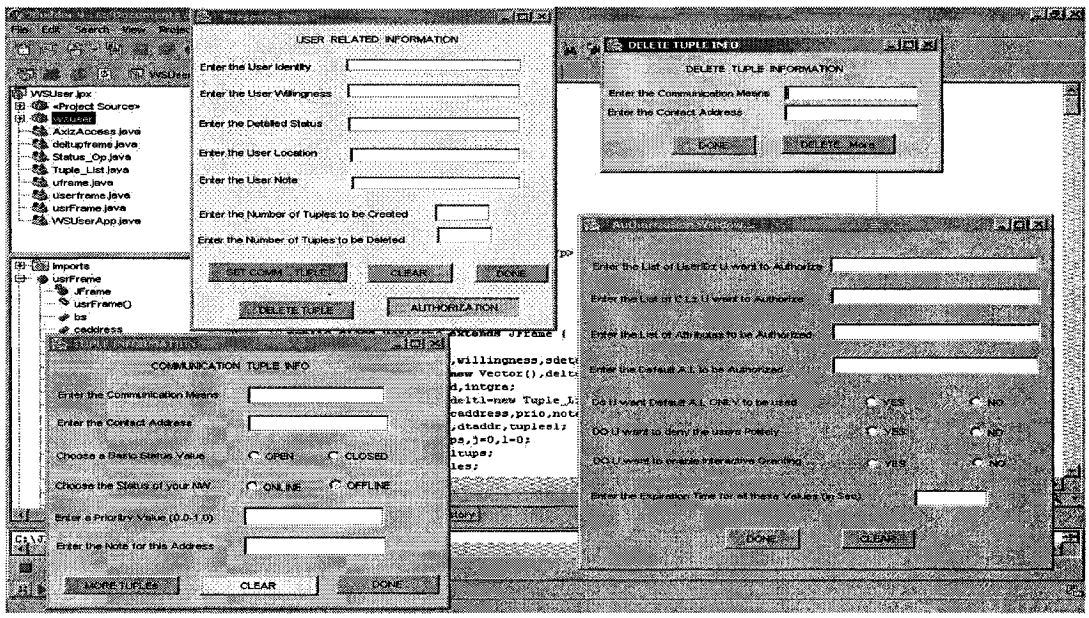


Figure 5.4: User Interface for Publisher

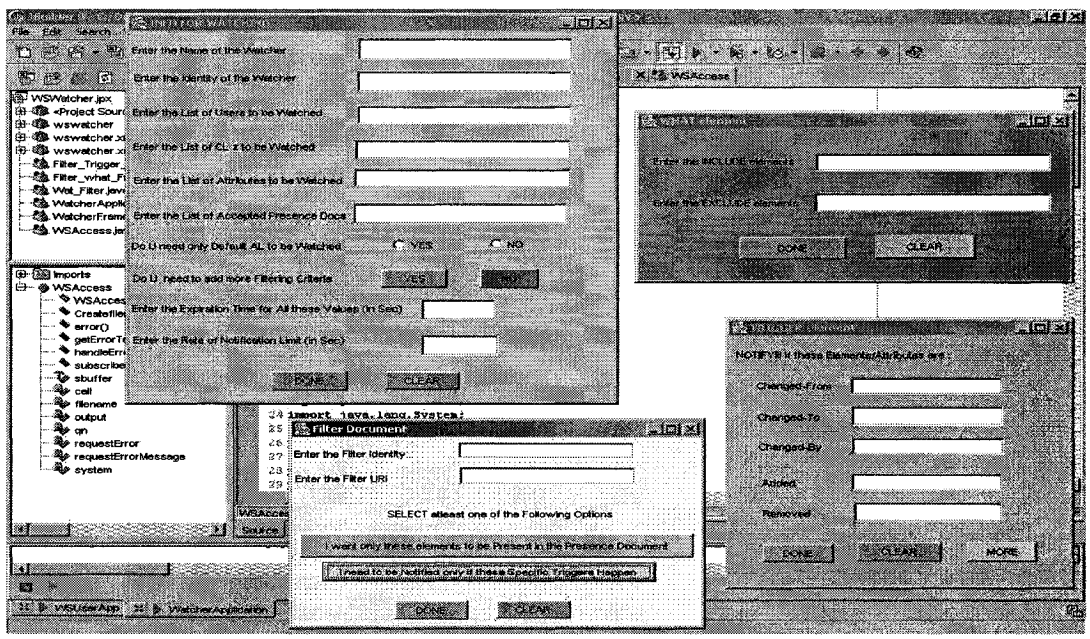


Figure 5.5: User Interface for watcher

The Web Service description of the Presence service is assumed to be published locally to the clients and not in the service registry by the service provider. The publisher uses user GUI, communication tuple GUI, delete tuple GUI and authorization GUI to obtain the presence information from the user. The watcher uses one GUI for getting the information for watching and uses three GUI's for getting the filter related information from the user. The castor framework is used for generating source code for creating the filter document and processing the PIDF document.

5.2 Mapping of the Web Service based Presence Gateway to the SIMPLE based Presence Server

The API developed during the research work is at a very high level of abstraction that helps the application developers to use all the functionalities for publishing the presence information using one method (setPresence). The mapping of the publisher side of the protocol independent API to the Java API of the presence server is explained using the sequence diagram in figure 5.6. The publisher calls the Web service gateway for creating, updating and deleting the presence information of any number of users. The gateway does some processing, then calls the presence server API using low level function calls for creating or updating or deleting one user at a time and sends the result to the publisher after aggregating all the results from the Java API. The setPresence method could also be used for authorization, but the presence server does not expose the functionality for the third party developers using the Java API.

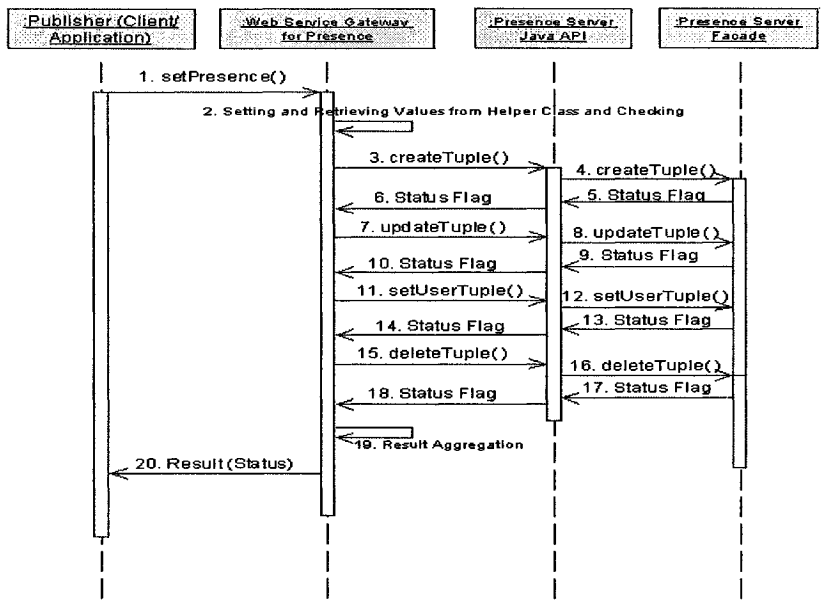


Figure 5.6: Mapping of the Presence Gateway to Presence Server (Publisher Functionalities)

The subscribe_Presence method is used for fetching, polling and subscribing the filtered presence information. The Java based presence server API does not support subscribing for the presence information, so only the fetching and polling mechanisms are supported by the Web service gateway. The sequence diagram for the watcher side operation is shown in figure 5.7. The watcher sends a single method for fetching the filtered presence information of any number of users to the gateway. The presence gateway uses the helper classes to fetch unfiltered presence information of all the specified users, subsequently it parses the presence document, then the gateway filters and makes the new well formed filtered presence document. The aggregated presence document of all the specified users is sent to the watcher. The polling mechanism is implemented in the gateway by fetching the presence information at regular interval of time.

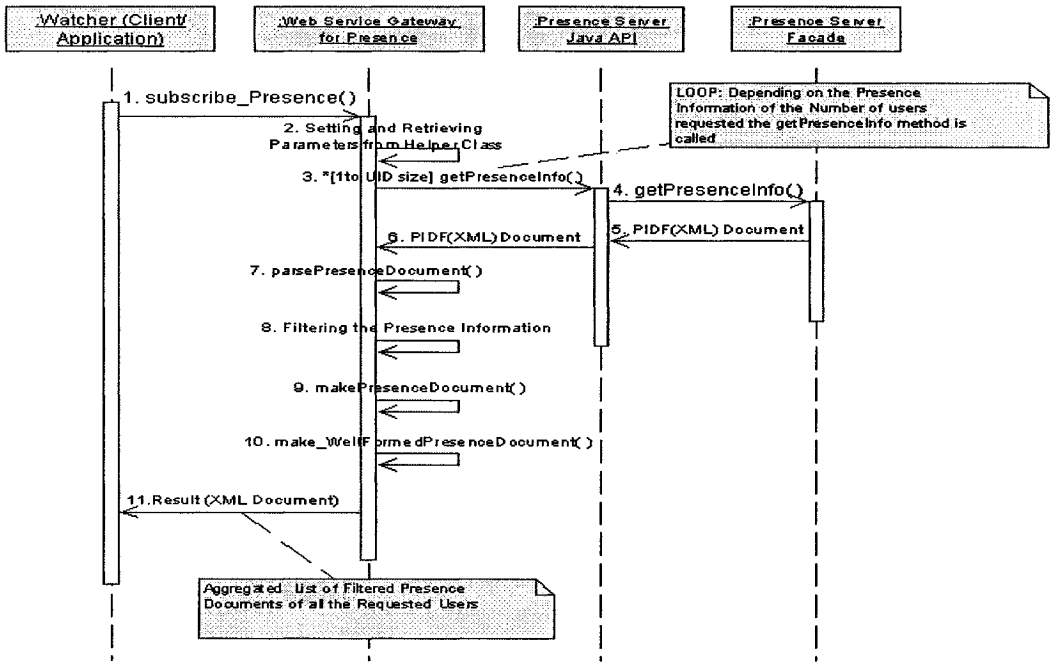


Figure 5.7: Mapping of the Presence Gateway to Presence Server (Watcher Functionalities)

5.3 Performance Evaluation

This section details about the performance evaluation. Firstly, details about the test bed and performance metrics are given, then the performance measurements are shown and the analysis of the results are presented

5.3.1 Test Bed and Performance metrics

The performance measurements were taken to investigate the applicability of Web services paradigm in the telecommunication domain, specifically with the presence technology. The two performance metrics that were considered are network load and

response time. The basic architecture used for taking the performance measurements is given in figure 5.8. The Presence server was deployed on the application server that was running on an Intel Pentium IV 2.4 GHz machine (512 MB RAM) with Red Hat Linux 9.0 platform. The Web service gateway for presence and framework were also deployed on the same application server. The methods supported by the proposed API are atomic methods, so the testing is done with one client, as that would justify the overhead the WS gateway imposes.

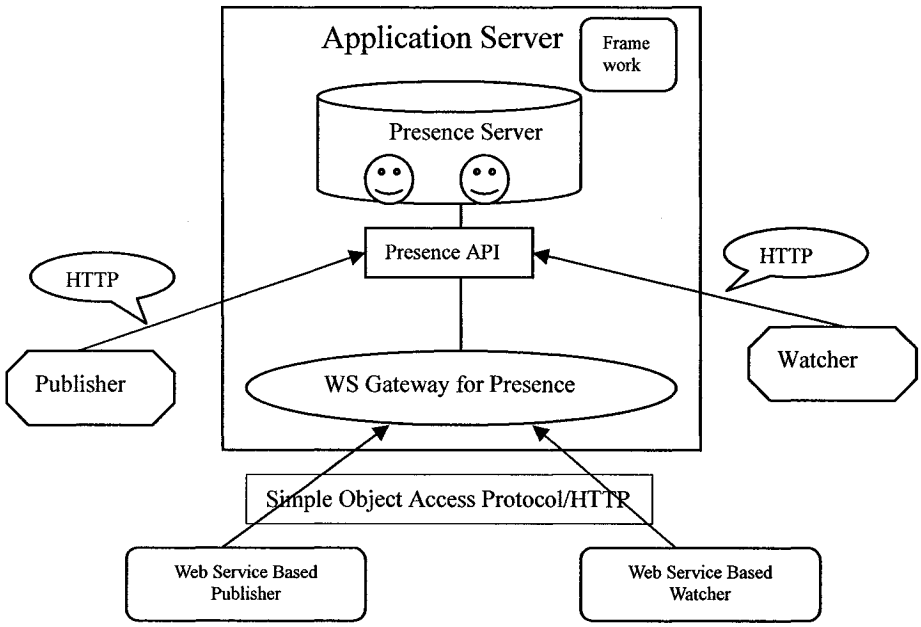


Figure 5.8: Basic architecture of the test environment

The Web service based publisher and watcher were running on Intel Pentium IV 2.4 GHz (512 MB RAM) with Windows XP home edition operating system and were using SOAP/HTTP for communication with the server. The client and server were connected to the same segment of the 100 Mbps Ethernet in our local area network. To compare the result without Web service gateway, we used the same machine to run a

custom built publisher and watcher, which used HTTP for communication with the server. The non Web Service based clients were developed by another team.

We took the response time and the network load during the late hours to minimize the network load due to other activities. Ethereal protocol analyzer was used to catch the packets between the client and server ports. Each packet length was calculated to get a faultless network load. The response time is calculated with an embedded software module in the clients. The response time with and without the gateway is calculated as the time duration between invoking the method in the server and getting a response from it. The performance metrics were taken for 15 trials for all the measurements and the average of all the readings is presented in the next section.

5.3.2 Performance Measurements and Analysis

The performance measurements obtained as a result of the experiments are shown in the graphs (figure 5.9, 5.10, 5.11). In the case of publisher, the response time and network load is calculated with and without the presence gateway for creating and deleting a single tuple. As the gateway supports higher level abstracted methods, the same performance metrics were taken while creating one tuple and deleting one tuple with the setPresence operation. The performance measurements while creating and deleting two tuples using one setPresence method with the Web service gateway is also done. The figure 5.9 shows the final results.

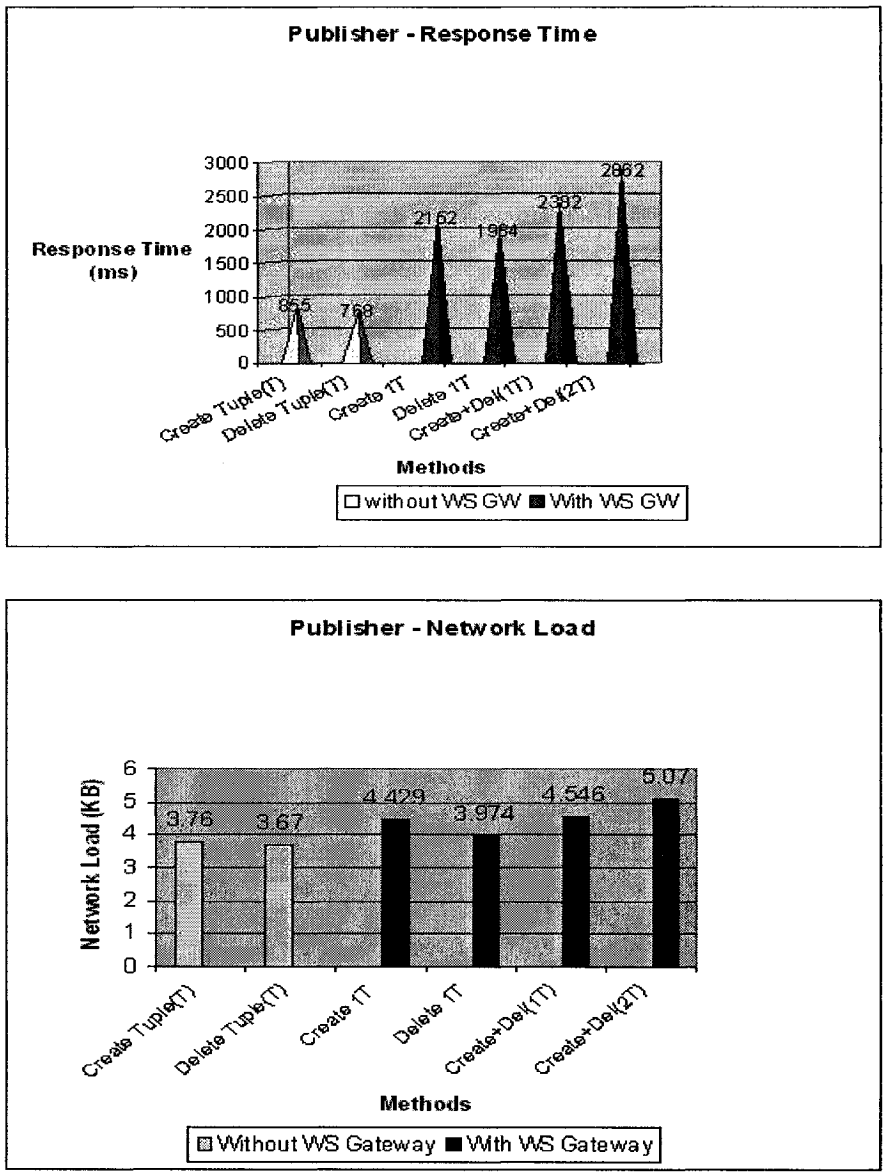
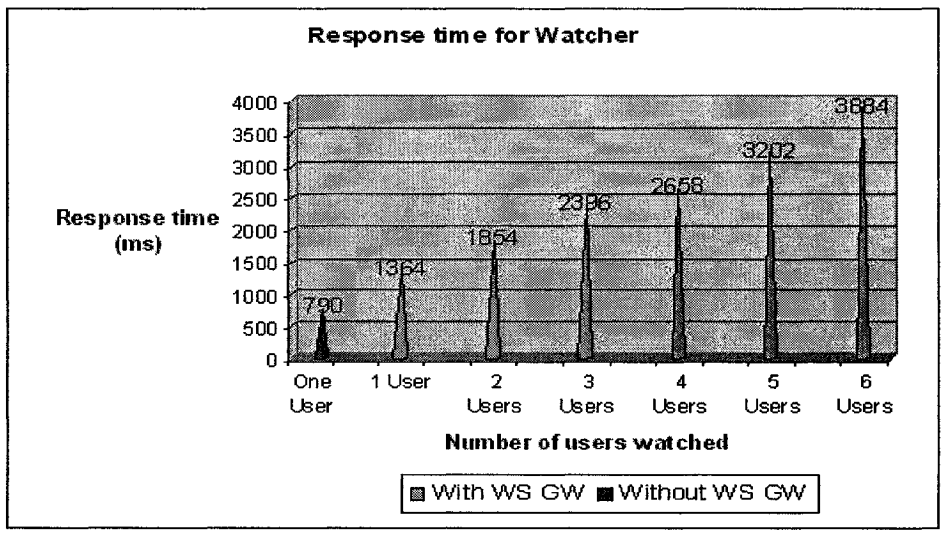


Figure 5.9: Response time and Network Load of the Publisher with and without the Web Service Gateway for Presence

The response time and network load increases with the usage of the gateway. The significant increase in the response time is mainly because of the SOAP decoding and encoding and secondly it is because of the high level abstracted interface. SOAP is a text based protocol that is based on XML, which also introduces additional network load.

Reference [45] discusses the factors that affect the SOAP performance. The results also show that the delete method takes lesser load and response time than the create method for the reason that only two parameters, communication mean and address, are required for performing this method. The main advantage of the high level abstracted interfaces is that, for instance one setPresence method is used to create 2 tuples and delete 2 tuples that obviously increases the response time and network load, which otherwise would take 4 method invocations to perform if the level of abstraction is lower. The end result is that the high level API ultimately helps in reducing the response time and load significantly.



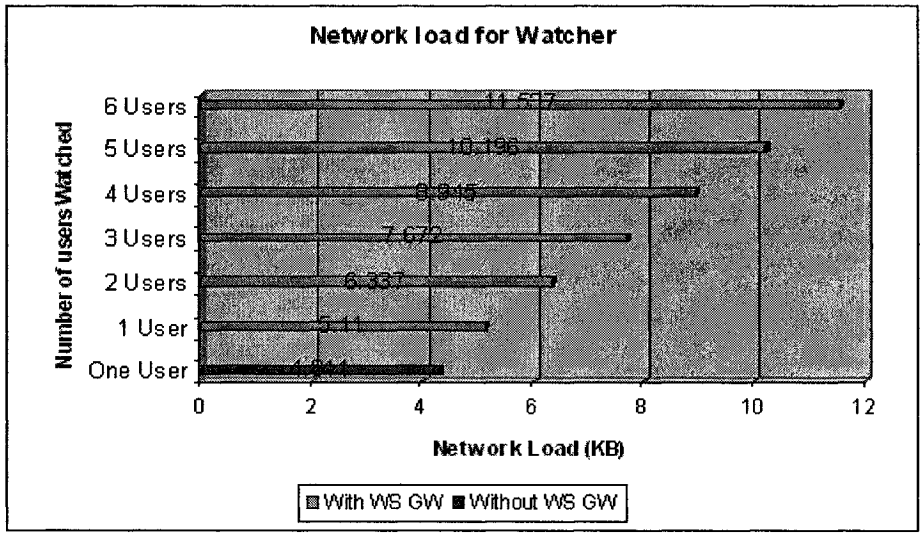
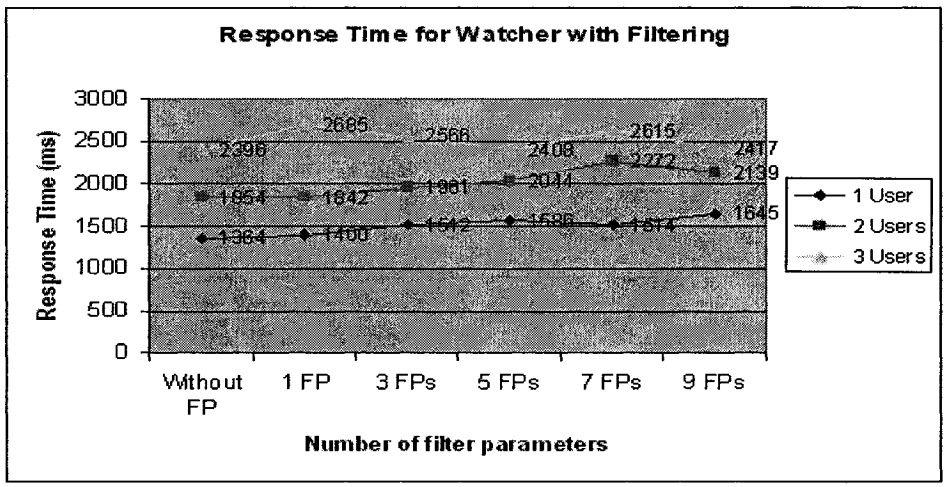


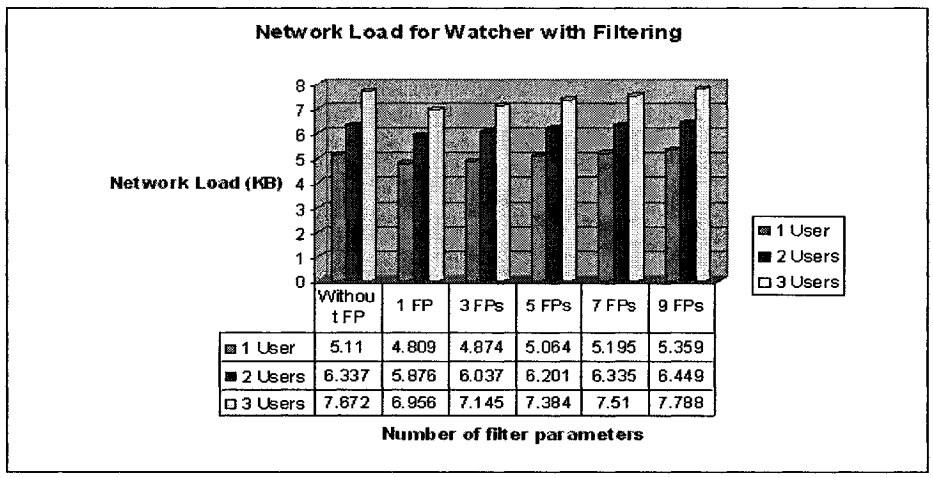
Figure 5.10: Response time and Network Load of the Watcher with and without the Web Service gateway for Presence

The presence server supports only the fetching operation for third party developers, so the performance measurements for watchers are taken only for fetching operation. The measurements with the Web service gateway is taken for getting presence information of one to six users, but without the gateway presence information of only one user can be fetched at a time. The response time and network load measurements of the watcher with and without the gateway is presented in the figure 5.10. As analyzed previously with the publisher, the response time and load increases with the usage of gateway because of SOAP processing and the abstracted API. The network load and response time increases steadily with the number of users being watched.

The presence server does not support the filtering of presence information; on the other hand the Web service gateway for presence supports presence information filtering. The Ericsson SIMPLE based presence server supports the ten presence attributes adopted

by 3GPP. The Web service based watcher specifies the list of attributes that it is interested in fetching, and then the gateway filters the attributes and sends back the filtered presence document to the watcher.





	Without FP	1 FP	3 FPs	5 FPs	7 FPs	9 FPs
1 User	5.11	4.809	4.874	5.064	5.195	5.359
2 Users	6.337	5.876	6.037	6.201	6.335	6.449
3 Users	7.672	6.956	7.145	7.384	7.51	7.788

Figure 5.11: Response time and Network Load of the Watcher with Filtering Parameters

The network load and response time is calculated for three users with filtering criteria. The filtering parameters (FP) specify the number of attributes requested by the user. For example, if the number of FP is three that means the number of attributes requested by the watcher is three. The graphs in figure 5.11 show that even though there

is a small increase in the response time there is a decrease in the network load when filtering is used. It is also observed that it is optimal to filter less than half of the attributes using the gateway, if more than 50% of the attributes are filtered the network load increases considerably. This is because of the fact that when more attributes are requested by the users the content of the filtered presence document is comparable to the unfiltered document which increases the load in the network. The response time is increased because of the additional processing required for filtering the attributes. It is also analyzed that as the number of users increases the response time with filtering parameters remains constant but the network load reduces notably for the optimal filter criteria (up to 50% parameter filtering).

Chapter 6

Conclusion

This chapter gives a summary of the research contributions reported in this thesis. The possible future work is also pointed out.

6.1 Summary of Contributions

In the recent years Web services have emerged as a new standard for easier application development besides that Web services allows loose coupling among applications, and provides effortless interoperability. The main aim of this research work is the investigation of the Web services paradigm for presence based application development.

The core contribution from the research work is the development of the Web service based API for presence based application development. The existing API's for presence is thoroughly analyzed and found that the level of abstraction available from these API's is not suitable to be used directly as high level Web service interfaces. The proposed API is designed to be at a high level of abstraction so that applications could be developed with ease; moreover the API is protocol, network and architecture independent. The application developer need not be an expert in the presence domain to incorporate the presence technology in the applications, which opens the technology to a

wide range of developers. Innovative applications can be effortlessly developed and the number of lines of code required for using different presence functionalities reduces significantly.

Interoperability is one of the key issues for the future of the presence technology as there are different protocols like SIMPLE, 3GPP and XMPP that are widely deployed. The usage of Web services with the presence technology solves this problem for the reason that one of the main advantages of Web Services is that it allows disparate systems to communicate with ease.

A subset of the functionalities of the API is implemented as a Web service gateway for presence. The presence information creation, updation, deletion, retrieval partial presence information update and presence information filtering is supported by the gateway. The rationale for implementing only these functionalities is that the presence server supports only these functionalities as Java API for third party developers. The Web service based publisher and watcher were also developed as prototypes during the course of this research.

Performance measurements were also taken with and without the Web service gateway to explore the usability of Web services with presence. The network load and delay measurements were taken and the results of the experiments were analyzed and presented. The results show that there is an increase in the network load and response time because of the additional SOAP processing and high level abstracted interface. The proposed API's presence information filtering mechanism for watchers reduces the network load by significant amount though the response time increases a little.

Web services though considered being a challenging middleware, the overhead created by Web services is negligible compared to the interoperability and easy application to application interaction features offered by the Web services.

6.2 Future Work

The future work would be to map the native interfaces of the presence server (without using the Java API of the presence server), that would give more flexibility for mapping additional functionalities like subscribing to presence Information, authorization and watcher related functionalities, to the gateway. The API developed is protocol and network independent, so it would be interesting to map the Web service based presence gateway to a 3GPP based presence server that would also allow contact list management along with other presence functionalities to be mapped to the presence gateway. Performance measurements with these new gateways could also be done and compared with the results obtained in this research work. Performance measurements in mobile environment could be done with presence clients on mobile devices like PDA's and mobile phones. These experiments might show different kind of results that would be interesting to analyze.

References

1. Steven J. Vaughan-Nichols, "Presence Technology: More than just Instant Messaging," IEEE Computer, October 2003, pp 11-13

2. Web Services Activity - W3C, http://www.w3.org/2002/ws/

3. M. Day, J. Rosenberg and H. Sugano, "A Model for Presence and Instant Messaging," RFC 2778, Internet Engineering Task Force, February 2000

4. M. Day, S. Aggarwal and J. Vincent, "Instant Messaging/Presence protocol Requirements," RFC 2779, Internet Engineering Task Force, February 2000

5. XMPP W.G, http://www.ietf.org/html.charters/xmpp-charter.html

6. P. Saint-Andre, Ed. "Extensible Messaging and Presence Protocol (XMPP): Core," internet draft, IETF, May 2004. Work in progress

7. P. Saint-Andre, Ed. "Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence," internet draft, IETF, April 2004. Work in progress

8. SIMPLE WG, http://www.ietf.org/html.charters/simple-charter.html

9. J. Rosenberg et al, "SIP: Session Initiation Protocol," RFC 3261, IETF, June 2002

10. A. B. Roach, "Session initiation protocol (SIP)-specific event notification," RFC 3265, IETF, June 2002.

11. J. Rosenberg, "A presence event package for the Session Initiation Protocol (SIP)," internet draft, IETF, January 2004. Work in progress

12. C. Muhugusa, R.H. Glitho, Theo Kanter, "A Protocol Perspective on Presence," Ericsson Internal Document, 2002

13. Wireless Village, http://www.openmobilealliance.org/wirelessvillage/

14. Wireless Village, Client-Server Protocol, Session and Transaction, Version 1.1

15. Internet Engineering Task Force, http://www.ietf.org

16. IMPP W.G, http://www.ietf.org/html.charters/impp-charter.html

17.3rd Generation Partnership Project, www.3gpp.org

18. 3GPP. Technical Specification Group Services and System Aspects, "Presence Service, Stage 1," 3GPP TS 22.141 v6.1.0, September 2002

19.3GPP. Technical Specification Group Services and System Aspects, "Presence Service, Architecture and Functional Description," 3GPP TS 23.141 v6.1.0, Dec 2002

20. Open Mobile Alliance, http://www.openmobilealliance.org/

21. "A Conversation with Adam Bosworth," ACM Queue, March 2003, pp 12-21

22. Web Services Architecture: http://www.w3.org/TR/2002/WD-ws-arch-20021114/

23. Kreger.H; Web Services Conceptual Architecture (WSCA 1.0) IBM white paper

On Web: http://www-3.ibm.com/software/solutions/webservices/pdf/WSCA.pdf

24. Simple Object Access Protocol (SOAP): http://www.w3.org/TR/SOAP/

25. Web Services Description Language (WSDL): http://www.w3.org/TR/wsdl

26. Universal Description, Discovery and Integration (UDDI): http://www.uddi.org

27. Parlay, http://www.parlay.org/

28. PAM WG, http://www.parlay.org/about/pam/index.asp

29. Parlay-X WG, http://www.parlay.org/about/parlay_x/index.asp

30. Parlay-X Web Services Specification Version 1.0,

On Web: http://www.parlay.org/specs/index.asp

31. Forum Nokia, Web Services Interfaces – an engine of innovation, White Paper

32. Forum Nokia, Presence Application Development Guide, version 1.0

33. Forum Nokia, Nokia Presence Server Service Developer's Guide, version 1.0

34. JAIN, http://java.sun.com/products/jain/

35. JAIN Presence, http://jcp.org/en/jsr/detail?id=186

36. JAIN SIMPLE Presence, http://jcp.org/en/jsr/detail?id=164

37. JAIN PAM API, http://jcp.org/en/jsr/detail?id=123

38. Open GIS Consortium, Inc. http://www.opengis.org/

39. H. Khartabil, "An Extensible Markup Language (XML) Based Format for Event Notification Filtering," internet draft, IETF, August 2004, Work in progress

40. H. Khartabil, "Functional Description of Event Notification Filtering," internet draft, IETF, August 2004, Work in progress

41. H. Sugano, "Presence Information Data Format (PIDF)," Internet Draft, IETF, May 2003. Work in progress

42. J. Rosenberg., "A Watcher Information Event Template-Package for the Session Initiation Protocol (SIP)", January 2003. Work in progress

43. J. Rosenberg, "An extensible markup language (XML) based format for watcher information," internet draft, IETF, January 2004. Work in progress

44. The Castor Project, http://www.castor.org/

45. Robert Elfwing et al, Performance of SOAP in Web Service Environment Compared to CORBA, Proceeding of the Ninth Asia-Pacific Software Engineering Conference (APSEC '02), December 04 - 06, 2002 ,Gold Coast, Australia , p.84

Note: All the WebPages were last accessed on August 15, 2004

Appendix A

Deployment and Configuration of the Web Service based Presence System Components

The application server is first started, and then the framework, presence server and the Web service gateway are deployed on the application server. The presence service is then registered to the application server by the framework after which the users for the presence service are created. Figure A.1 shows the deployment and checking of the framework, presence server and gateway.

The registration of the presence service and the creation of the user for presence service access are shown in figure A.2. The publishers can publish the presence information by sending SOAP based RPC calls to the gateway that maps the information to the presence server API to be put in the presence server. The watchers has to be authorized by the publishers before retrieving the presence information, once authorized the watchers can retrieve the filtered presence information using the Web service gateway for presence.

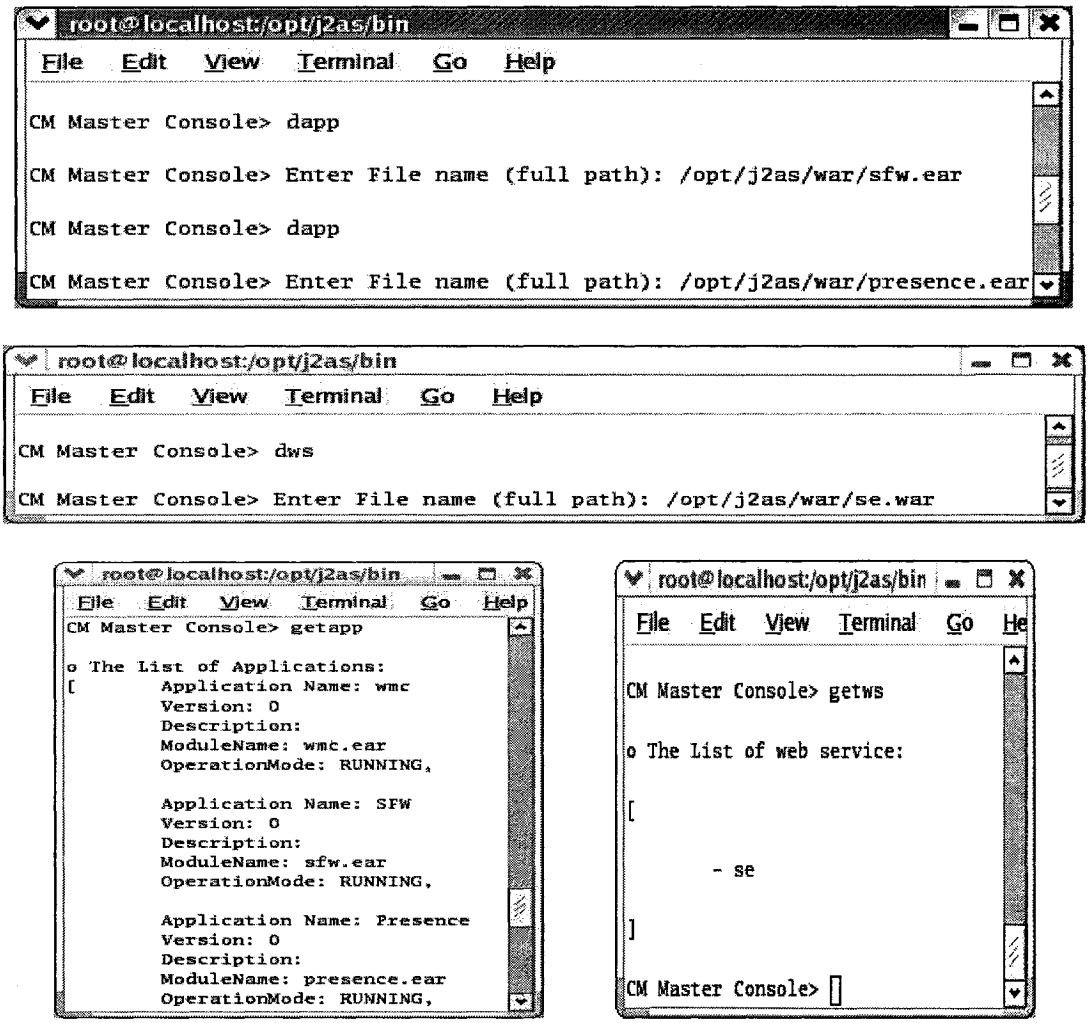


Figure A.1: Deploying and Checking the Framework, Presence Server and Gateway

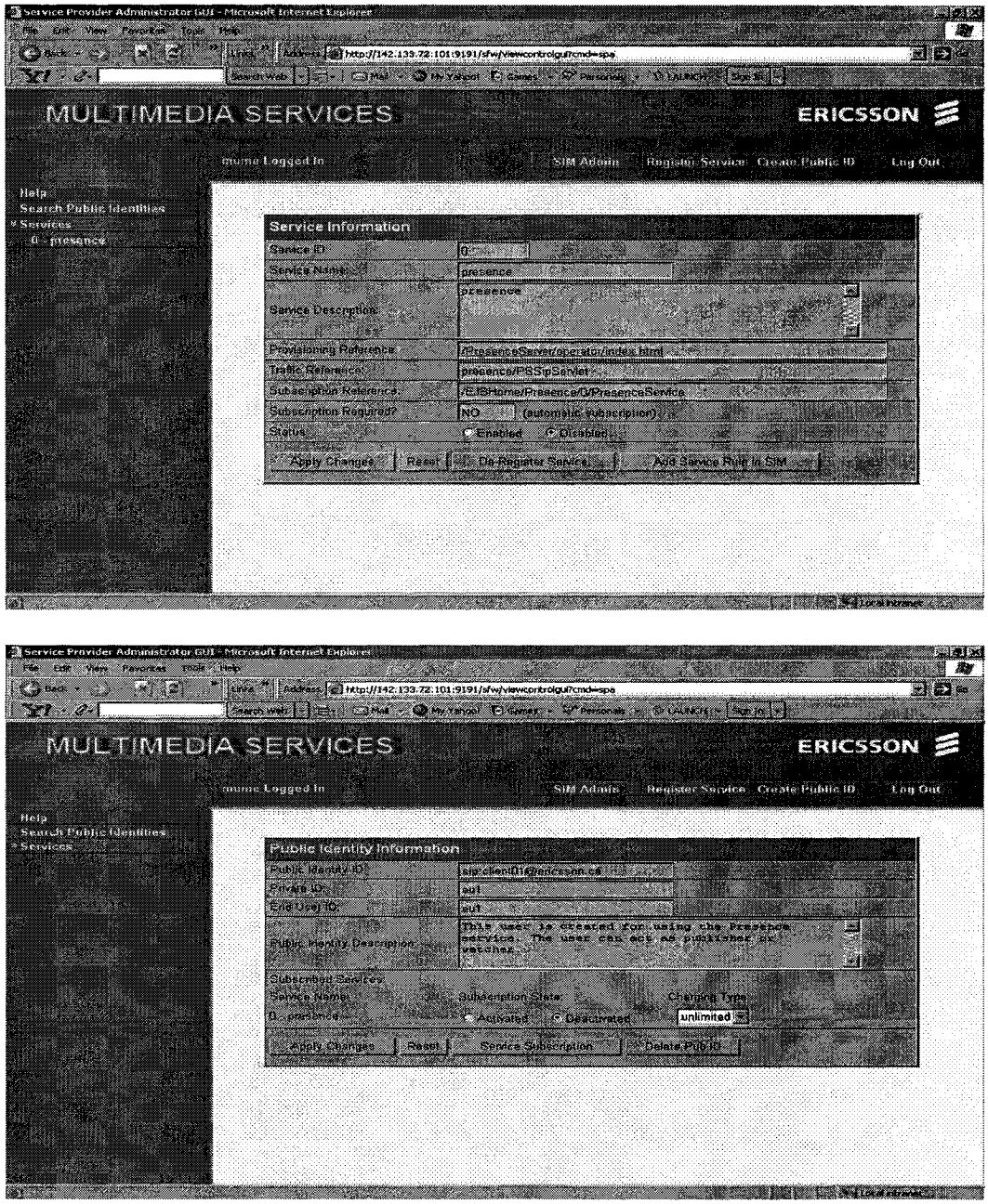


Figure A.2: Registering the Presence service and creating the user for using the Presence service with the Framework