DEVELOPING JXTA APPLICATIONS FOR MOBILE DEVICES AND INVOKING WEB SERVICES DEPLOYED IN JXTA PLATFORM FROM MOBILE DEVICES

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

MESUT BAHADIR

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

DECEMBER 2004

Approval of the Graduate School of Natural and Applied Sciences.

Prof. Dr. Canan Özgen
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. Ayşe Kiper
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Prof. Dr. Asuman Doğaç
Supervisor

Examining Committee Members

Prof. Dr. Asuman Doğaç	(METU,CENG)	
Assoc. Prof. Dr. Ali Doğru	(METU,CENG)	
Assoc. Prof. Dr. Nihan K. Çiçekli	(METU,CENG)	
Assoc. Prof. Dr. İ. Hakkı Toroslu	(METU,CENG)	
Dr. Ayşe Yasemin Seydim	(TCMB)	

ABSTRACT

DEVELOPING JXTA APPLICATIONS FOR MOBILE DEVICES
AND
INVOKING WEB SERVICES DEPLOYED IN JXTA PLATFORM
FROM MOBILE DEVICES

Bahadır, Mesut
M.Sc., Department of Computer Engineering
Supervisor: Prof. Dr. Asuman Doğaç

December 2004, 90 pages

Today, Peer-to-peer (P2P) computing and Web Services play an important role in networking and computing. P2P computing, that aims addressing all the resources in a network and sharing them, is an old paradigm that gains importance nowadays with the advent of popular file sharing and instant messaging applications. On the other hand, a Web service is a software system that has an interface allowing applications to interact with other applications through Internet or intranet. Providing methods for publishing and discovering Web services from which mobile devices can facilitate in a P2P environment enables exploitation of P2P and Web service technologies efficiently by mobile devices. This also extends the range of devices that facilitate P2P and Web services technologies from servers and desktop computers to personal digital assistants (PDAs) and mobile phones.

In this thesis, an architecture that enables publishing and discovering Web services for mobile clients that are inter-connected in a P2P environment is introduced. Key issues in this architecture are allowing mobile devices to join in a

P2P network group, publishing Web services and discovering these services in P2P network. Invoking Web services that are published and discovered is another key issue in this architecture. The architecture introduced exploits P2P and Web services standards using various tools for mobile devices. For the purpose of organizing a P2P environment, JXTA protocols and services are used. WSDL is used for describing Web services. JXTA advertisements help in publication and discovery of Web services; and BPEL enables composition, deployment and execution of Web services. The architecture introduced within the scope of this thesis combines all these standards with tools that enable use of these standards on mobile devices.

The work done in this thesis is realized as a part of Artemis, a project funded by European Commission for providing interoperability of medical information systems.

Keywords: Peer-to-peer (P2P), JXTA, Web Services, BPEL, WSDL, SOAP, J2ME, MIDP, CLDC, Mobile applications

ÖZ

MOBİL CİHAZLARDA JXTA UYGULAMALARI GELİŞTİRİLMESİ
VE
MOBİL CİHAZLARDAN JXTA ORTAMINDAKİ WEB SERVİSLERİNİN
ÇALISTIRILMASI

Bahadır, Mesut
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Asuman Doğaç

Aralık 2004, 90 sayfa

Günümüzde, Eşler-arası (P2P) hesaplama sistemleri ve ağ servisleri iletişim ve hesaplama alanlarında önemli rol oynamaktadır. Amacı, bir ağ üzerindeki bütün kaynakları adreslemek ve paylaşmak olan eşler-arası hesaplama sistemleri, bugünlerde Eşler-arası iletişime dayanan popüler dosya paylaşım ve anında mesajlaşma uygulamalarının çıkışıyla önem kazanmaya başlayan eski bir paradigmadır. Eşler-arası hesaplama sistemleri yanında, bir ağ servisi, internet veya yerel ağlar üzerindeki uygulamalar arasında etkileşim sağlayan bir arayüze sahip bir yazılım sistemidir. Bir Eşler-arası ortamda, ağ servislerinin yayımlanması ve keşfedilmesi için yöntemler sunmak, Eşler-arası ve ağ servisi teknolojilerinin mobil cihazlar tarafından etkili bir şekilde kullanılmasını sağlayacaktır. Bu ayrıca Eşler-arası ve ağ servisi teknolojilerinden faydalanan cihazların kapsamını sunucu ve masaüstü bilgisayarlardan PDA ve mobil telefonlara doğru genişletecektir.

Bu tez çalışmasında, bir Eşler-arası ortama üye mobil cihazlar için ağ servislerinin yayınlanması ve keşfedilmesini sağlayan bilgisayar mimarisi sunulmuştur. Mobil cihazların Eşler-arası ağa dahil edilmesi, ağ servislerinin yayımlanması ve bu ağ servislerinin Eşler-arası ağda keşfedilmesi bu mimari yapıdaki anahtar konulardandır. Yayımlanan ve keşfedilen ağ servislerinin çalıştırılması da bu mimari yapının diğer bir anahtar konusudur. Tanıtılan mimari yapı, Eşler-arası ve ağ servisleri standartlarını, mobil cihazlar için geliştirilen yazılım araçları aracılığıyla kullanmıştır. Eşler-arası ortam oluşturulması amacıyla, JXTA protokolleri ve servisleri kullanmıştır. Ağ servislerinin tanımlanması için WSDL kullanılmıştır. JXTA ilanları, ağ sistemlerinin yayımlanmasına ve keşfedilmesine yardımcı olmaktadır. BPEL, ağ servislerinin oluşturulmasını, yerleştirilmesini ve çalıştırılmasını sağlamıştır. Bu tezin kapsamında sunulan mimari yapı, P2P ve ağ servisi standartlarını, mobil cihazlar için olan aracı yazılımlarla birleştirmiştir.

Bu tez, sağlık bilgi sistemlerinin birlikte çalışırlığını sağlamak için Avrupa Komisyonu tarafından desteklenen Artemis projesinin bir bölümü olarak gerçekleştirilmiştir.

Anahtar Kelimeler: Eşler-arası (P2P), JXTA, Ağ Servisleri, BPEL, WSDL, SOAP, J2ME, MIDP, CLDC, Mobil Uygulamalar

To my family

ACKNOWLEDGMENTS

I would like to thank my supervisor Prof. Dr. Asuman Doğaç for her guidance, support, encouragement and patience during this thesis study.

I would like to thank SRDC team for their support.

I would like to thank my colleagues and classmates, Kırçiçeği Korkmaz, Seda Dağlar, Serpil Memiş, Güzen Erözel, Yasemin Seydim, Mustafa Kemal Kaplan, Sait Korkmaz, Merter Sualp, Serkan Toprak, and especially İlhami Görgün for their encouragements during this study.

Finally, I would like to thank my family for their support and patience.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In recent years, the number and types of devices that connect to the Web for accessing information and services has increased rapidly. These devices differ in types of architectures, capabilities, processing power, as well as connection types. They make use of different architectures and platforms to access information and services on the Web.

Peer-to-peer (P2P) computing and Web services are paradigms that collectively enable various types of devices to connect to Internet or intranet for accessing information and services. They overlap in some properties, such as publication and discovery of services. However, Web services achieve these functionalities in a service-oriented manner, whereas P2P systems achieve in a peer-oriented (node-oriented) manner.

Peer-to-peer (P2P), which is a computing and networking paradigm, has gained great importance in the past decade. It is not a new concept in computer area but with the advent of popular file sharing and instant messaging applications, awareness of this paradigm has increased rapidly. P2P systems consist of self-organizing interconnected nodes, named as peers. P2P computing is the sharing of computer resources and services between interconnected nodes.

Besides P2P computing, Web services is another popular research and application area for enabling various types of devices to use services on the Web. A Web service is a software system for providing services to users or other systems. The technology introduced by Web services builds upon the specifications and the architecture of these specifications that are denoted in Figure 1 [1].

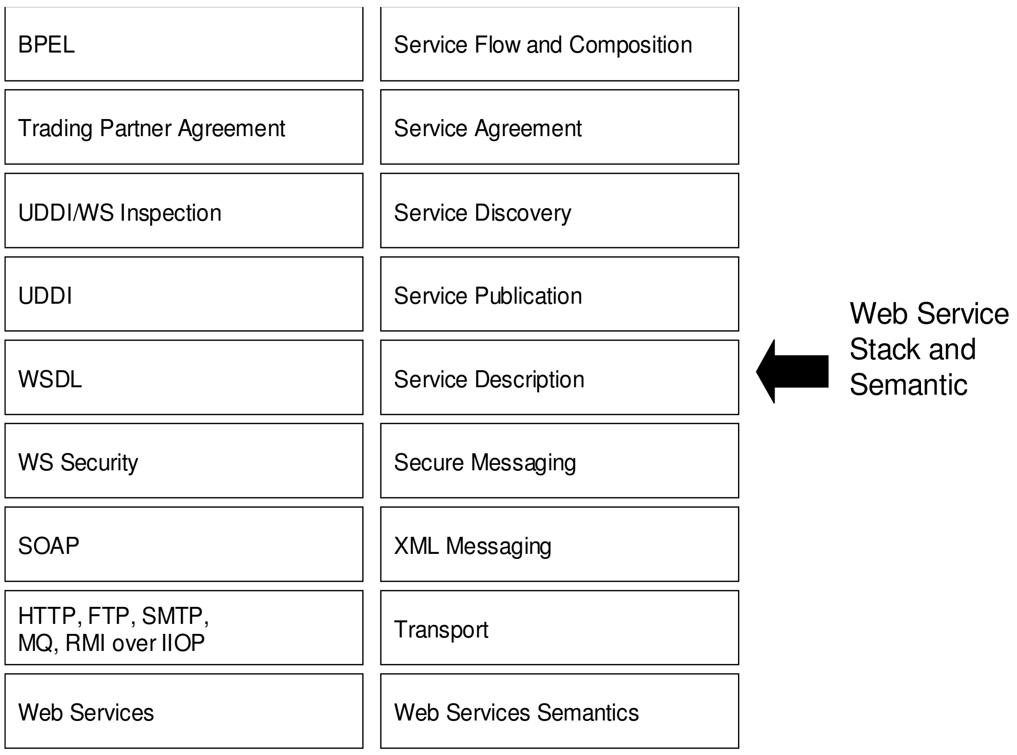


Figure 1: Web Service Stack and Semantic

In this work, the technologies introduced by Web services and P2P computing are brought together in order to realize the application of these technologies on mobile devices. For the purpose of implementing P2P system, the JXTA framework is exploited.

Within the system architecture proposed by this work, the technology stack that is denoted in Figure 2 forms the basis for this work. In this system architecture, service discovery and publication is realized through JXTA.

Considering this stack of specifications that is built upon the mechanisms of Web services, the Web service related specifications, Business Process Execution Language (BPEL), Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL), and P2P computing related specifications provided by JXTA, JXTA Service Advertisement and JXTA Service Discovery, constitute the core building blocks for this work.

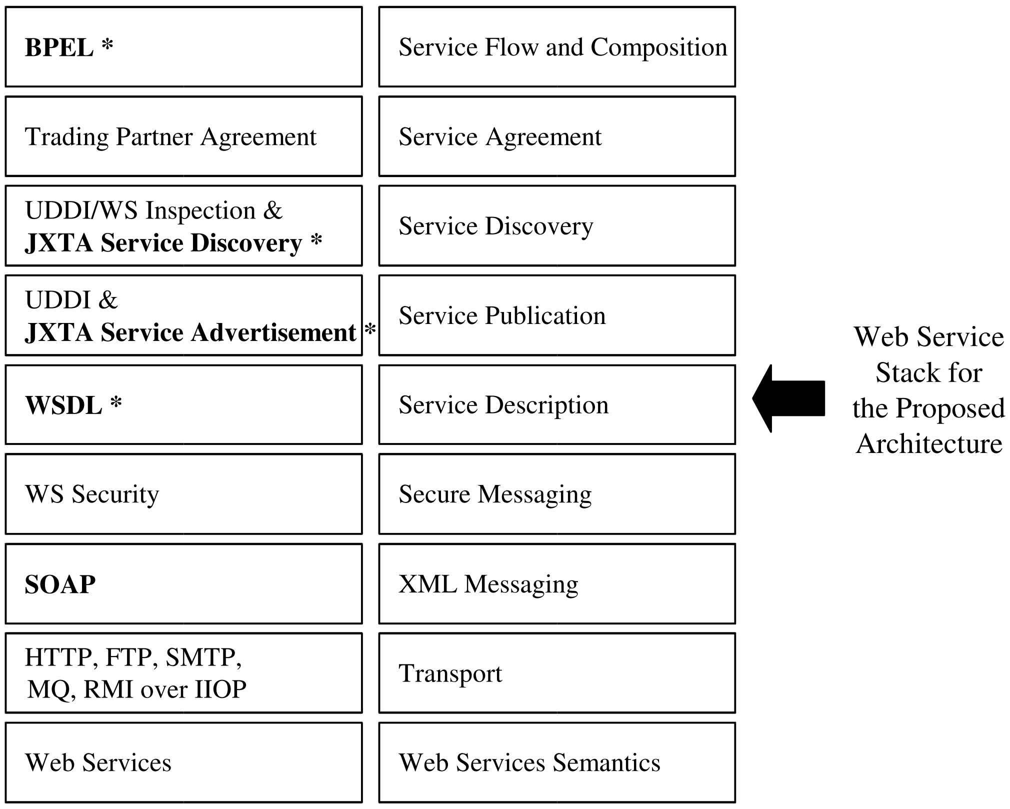

BPEL *
Service Flow and Composition
Trading Partner Agreement
Service Agreement
UDDI/WS Inspection & JXTA Service Discovery *
Service Discovery
UDDI & JXTA Service Advertisement *
Service Publication
WSDL *
Service Description
WS Security
Secure Messaging
SOAP
XML Messaging
HTTP, FTP, SMTP, MQ, RMI over IIOP
Transport
Web Services
Web Services Semantics
Web Service Stack for the Proposed Architecture

Figure 2: Proposed Web Service Stack

The proposed Web service stack is utilized in Artemis [2] Web service architecture for mobile devices. The aim of the Artemis project is to exploit ontologies based on the domain knowledge exposed by the healthcare information standards through HL7 [3], CEN TC251 [4], ISO TC215 [5] and GEHR [6], to provide interoperability of medical information systems.

This thesis is organized as follows:

- Chapter 2 describes the technologies and standards used in this work.
- In Chapter 3, the tools that exploit the standards detailed in Chapter 2 are presented. The components of the applications developed in this work are also presented in this chapter.
- Chapter 4 presents the Artemis system architecture.
- Chapter 5 describes the integration of the technologies and tools, and presents the implementations. Web service invocation through mobile devices, JXTA mobile client development, and Web service discovery

  and invocation through JXTA environment from mobile devices are introduced in subsections of this chapter.
- Finally, the thesis is concluded in Chapter 6.

CHAPTER 2

ENABLING TECHNOLOGIES

In this chapter, the technologies and standards that enable the work done in this thesis are described. In Section 2.1 and Section 2.2, Peer-to-peer (P2P) computing and an introduction to JXTA project is given. Section 2.3 explains Web services and standards on the area of Web services. Finally, in Section 2.4, the Java platform and especially J2ME edition of Java is explained in detail.

2.1 Peer-to-Peer (P2P) Technology

Peer-to-peer (P2P) is a paradigm in network architectures and computation that tries to increase the utilization of computer resources such as (CPU, files, etc..) over the Internet and local intranet. P2P technology focuses on efficient use of computer resources by joining a P2P group as a peer through Internet. P2P computing is characterized by direct connections between interconnected computing nodes. A node in a P2P network may be a client requesting resources such as shared files or processing power, or it may be a server that serves by supplying files and other resources.

2.1.1 Peer-to-Peer Networks

A peer-to-peer network architecture consists of interconnected nodes called peers. The information on one peer can be accessible to other peers. Various network topologies, such as hierarchical, ring, client-server (centralized), decentralized, and hybrid topologies, are used for organizing a P2P environment. The properties of the topologies are given in Table 1 [7].

Table 1: Basic Network Topologies

Topology	Property
Centralized	All function and information about peers and shares is centralized into one server with many peers connecting directly to the server node to send and receive information. Many P2P applications use a centralized component in their architectures.
Ring	A cluster of peers connected together in a ring structure.
Hierarchical	This topology has the form of a tree structure and the head peer sits on the root node of the tree. The node below the root node are responsible from the nodes below it.
Decentralized	All peers can communicate with each other and have equal roles. True P2P networks use this topology or a hybrid topology that has the properties of this topology.

Hybrid topologies are systems that combine at least two basic network topologies.

2.1.2 P2P Applications

There are many popular applications constructed on P2P technology, such as well-known file-sharing programs, Napster, Gnutella and Kazaa, resource sharing programs, SETI@home, and instant messaging programs like AOL Instant Message (AIM) and Yahoo! Messenger. All currently developed P2P solutions rely on platforms, programming language and networking that are peculiar to their own architecture, which implies the lack of interoperability among all P2P applications.

2.2 Introduction to JXTA

JXTA is an open source project that defines generalized peer-to-peer (P2P) protocols for P2P computing. [8] JXTA is a short form of juxtapose, to place side by side, which implies that the aim of Project JXTA is not to replace the client/server computing. JXTA allows any connected device on the network that ranges from PDAs to servers to communicate and interact with each other as peers. Project JXTA started as a research project at Sun Microsystems in 2001 and currently developing as an open source project.

The project devote serious effort to provide P2P applications with an infrastructure where the applications use standards for communication or collaboration.

The main objectives of the Project JXTA are based on the shortcomings in many popular peer-to-peer systems [9]. These aims are:

- Interoperability

  JXTA technology is designed to enable interconnected peers to easily locate each other, communicate and collaborate with each other, and offer services to each other across different P2P systems and different communities.

- Platform independence

  JXTA technology is designed to be independent of programming languages, operating systems, and networking platforms.

- Ubiquity

  JXTA technology is designed to be implementable on every device with a digital heartbeat, including sensors, consumer electronics, PDAs, appliances, network routers, desktop computers, data-center servers, and storage systems.

By defining a standardized set of protocol specification for the peers to operate, the Project JXTA overcomes the lack of interoperability within P2P systems. The JXTA protocols specify the manner in which peers:

- Discover each other
- Self-organize into peer groups
- Publish and discover network services
- Communicate with each other
- Monitor each other

The Project JXTA targets a P2P environment where each peer is independent of software and hardware platform, and can benefit from being connected to other peers.

2.2.1 JXTA Architecture

The Project JXTA architecture consists of three layers; the platform layer, the services layer, and the application layer as shown in Figure 3.

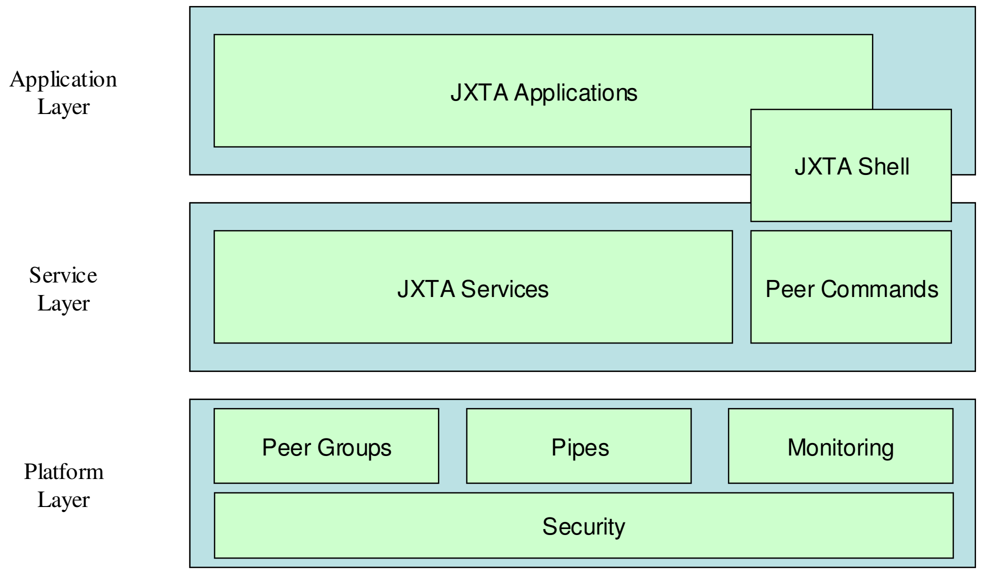


Figure 3: JXTA Architecture

2.2.1.1 Platform Layer

The platform layer provides minimal and essential elements used in P2P networking. This layer is the core of the Project JXTA. Service and Application layers are built on top of the platform layer and this layer provides functionality to the upper layers. The platform layer provides the following core elements, and functions.

- Elements
    - Peers
    - Peer Groups
    - Pipes
    - Endpoints
    - Messages
    - Advertisements
    - Identifiers
- Functions
    - Creation of peers and peergroups
    - Discovery
    - Transport
    - Security

The elements and functions are covered in section 2.2.2 and 2.2.3.

2.2.1.2 Services Layer

The services layer provides network services that may not be necessary for a P2P network to operate, but are common in the P2P environment. Searching and indexing, directory, storage systems, file sharing, authentication are examples of services provided in this layer.

2.2.1.3 Applications Layer

The application layer consists of the integrated applications like instant messaging, document and resource sharing programs. These applications make use of the services provided by service layer.

2.2.2 Concepts

JXTA concepts are the primary components of the JXTA platform that are common for P2P networks. JXTA tries to keep the number of concepts minimal in order to refrain from complexity.

The JXTA network consists of interconnected nodes, namely *peers*. Peers can organize into *peergroups*, which provide *services*. Peers advertise their services to the peergroups in XML documents called *advertisements*. Advertisements enable peers to learn how to interact with a service. Peers use *pipes* for sending *messages* to other peers. Pipes are connected to *endpoints* on peers.

The following subsections will give the definitions and use of the JXTA concepts.

2.2.2.1 Peers

Peers are the interconnected nodes in a JXTA environment that implements the core JXTA protocols. Peers can be any network-aware device ranging from cell phones and PDAs to PCs and supercomputers. They are uniquely identified by a peer ID that are expressed in URNs. Each peer can operate independently and asynchronously from other peers.

Peers can advertise multiple network interfaces as peer endpoints. Endpoints are used by peers for establishing point-yo-point connections with other peers. In order to communicate with another peer, direct point-to-point are not necessarily required; peers can use intermediary peers to communicate. There are five types of peers distinguished according to their responsibilities that are given as follows:

- **Simple Peers** : publish resources to other peers and use resources from other peers in a peer group.
- **Rendezvous Peers** : additionally provides propagation of messages within peer groups. A rendezvous peer knows the peers using itself as rendezvous and also knows other rendezvous for message forwarding between rendezvous.
- **Router Peers** : enables communication through network address translators (NATs) and firewalls.
- **Relay Peer** : spools messages for peers that are not accessible every time. It has also simple peer functionalities.
- **Minimal Peer** : uses a relay peer for simple peer functionalities. It periodically polls relay peers for messaging.

2.2.2.2 Peer Groups

A peer group is a collection of peers that have a common set of interest. Peer groups are formed from self-organization of peers and identified uniquely by a peer group ID. There are three common motivations for creating peer group in P2P network: exchanging services among interested members, securing accessible services offered by a group and monitoring a group of peers. A peer group provides a set of services called peer group services. The set of services includes at least the core JXTA services. The core services are:

- **Discovery Service** **:** for searching peer group resources such as peers, peer groups, pipes, and services.
- **Membership Service** **:** for deciding whether to reject or accept a new group membership application (i.e., to allow a new peer to join a peer group).
  - Peers wanting to join a peer group need to discover at least one member of the peer group and then request to join. The request to join is either rejected or accepted by the collective set of current members.

    - A peer may belong to more than one peer group simultaneously.
- **Access Service** : for validating requests made by one peer to another.
    - The peer receiving the request provides the requesting peer credentials and information about the request being made to the Access Service to determine if the access is permitted. Not all actions within the peer group need to be checked with the Access Service. Only those actions that are restricted to a subset of member peers must be checked.
- **Pipe Service** : for managing and creating pipe connections between the different peer group members.
- **Resolver Service** : for addressing queries to services running on peers in the group and collect responses.
- **Monitoring Service :** for allowing one peer to monitor other members of the same peer group.

2.2.2.3 Services

A service is an abstract resource [10] provided by a peer or peer group. Services can either be deployed or installed on a peer or loaded from the network. In order to run a service on a peer or peer group, a peer may need to locate an implementation suitable for the peer's runtime environment. Invoking or stopping a JXTA service is not defined within the protocol, therefore any existing standard for invoking service can be used.

JXTA protocols define advertisements for publishing and discovery of services in JXTA environment.

2.2.2.4 Advertisements

Advertisements are structures for describing resources such as peer, peer group, pipe, etc. In JXTA, advertisements are represented in XML documents. JXTA defines several advertisement types but any service or peer implementation can create advertisements. The following are the advertisement types defined in JXTA specification:

- Peer Advertisement
- PeerGroup Advertisement

- ModuleClass Advertisement
- ModuleSpec Advertisement
- ModuleImpl Advertisement
- Pipe Advertisement
- Rendezvous Advertisement

2.2.2.5 Modules

Modules are designed to allow a developer to provide functionality within JXTA [11]. They are responsible for providing JXTA's functionality, such as implementations of the peer groups, services and applications provided by peer groups.

The modules are provided by peer groups and they can be initialized, started or stopped by peers. There are also service modules and application modules. Service module is a component of JXTA that is used by peers to run a service and application module is used for running applications.

A module definition consists of three types of advertisements:

– Module Class Advertisement
– Module Specification Advertisement
– Module Implementation Advertisement

Module Class Advertisement

This advertisement only announces the existence of a class of module and the structure of this advertisement is given in Figure 4:

```
<?xml version="1.0" encoding="UTF-8"?>
<jxta:MCA>
        <MCID> … </MCID>
        <Name> … </Name>
        <Desc> … </Desc>
</jxta:MCA>
```

Figure 4: Module Class Advertisement

MCID – Required – Contains a Module Class ID that uniquely identifies a class of modules. This ID is used as the basis for the IDs contained in the

Module Specification and Implementation Advertisements.

Name – Optional – Contains a name for the module class.

Desc – Optional – Contains a description of the module class.

Module Specification Advertisement

This advertisement uniquely identifies a set of protocol-compatible modules and the structure of this advertisement is given in Figure 5:

```
<?xml version="1.0" encoding="UTF-8"?>
<jxta:MSA>
        <MSID> … </MSID>
        <Name> … </Name>
        <Crtr> … </Crtr>
        <SURI> … </SURI>
        <Vers> … </Vers>
        <Desc> … </Desc>
        <Parm> … </Parm>
        <jxta:PipeAdvertisement> … </jxta:PipeAdvertisement>
        <Proxy> … </Proxy>
        <Auth> … </Auth>
</jxta:MSA>
```

Figure 5: Module Specification Advertisement

MSID – Required – Contains a Module Class ID that uniquely identifies a class of modules. This ID includes Module Class ID.

Name – Optional – Contains a name for the module class.

Crtr – Optional – Contains a name for the creator of the module specification.

SURI – Optional – Contains a URI pointing to a specification document.

Vers – Optional – Contains information on the module specification.

Desc – Optional – Contains a description of the module class.

Parm – Optional – Contains parameters for the module class.

jxta:PipeAdvertisement – Optional – Contains a pipe advertisement that describes a pipe for sending data to the module.

Proxy – Optional – Contains MSID of a module that can be used to proxy communication with a module defined by this module specification.

Auth – Optional – Contains MSID of a module that provides authentication services for a module defined by this module specification.

Module Implementation Advertisement

This advertisement provides information on an implementation of a module and the structure of this advertisement is given in Figure 6:

```
<?xml version="1.0" encoding="UTF-8"?>
<jxta:MIA>
        <MSID> … </MSID>
        <Comp> … </Comp>
        <Code> … </Code>
        <PURI> … </PURI>
        <Prov> … </Prov>
        <Desc> … </Desc>
        <Parm> … </Parm>
</jxta:MIA>
```

Figure 6: Module Implementation Advertisement

MSID – Required – Contains a Module Class ID that uniquely identifies a class of modules. This ID includes Module Class ID.

Comp – Optional – Contains compatibility information of the implementation, such as for applications implemented in Java, JVM and binding information is stored in this field.

Code – Optional – Contains any information required to run the code of the module implementation.

PURI – Optional – Contains a URI pointing to the package that contains the module code.

Prov – Optional

Desc – Optional – Contains a description of the module class.

Parm – Optional – Contains parameters for the implementation.

2.2.2.6 Pipes

Pipes are communication channels for sending and receiving messages within JXTA framework. The communicate occurs in asynchronous way. Since pipes are unidirectional channels, there are input and output channels.

2.2.2.7 Endpoints

Endpoints are network interfaces that are used to send and receive data using pipes. That is, endpoints are the initial source or final destination of any message that is transmitted over JXTA P2P network.

2.2.2.8 Messages

Messages are any piece of data structure designed to be usable on top of asynchronous, unreliable and unidirectional transport.

2.2.3 Protocols

The Project JXTA defines several protocols to standardize the P2P networking. These protocols provide functionality for peers on a network. These set of protocols are XML based and describes operations such as peer discovery, endpoint routing, connection binding, message exchange and network propagation. We describe these protocols in the following sections:

2.2.3.1 Peer Discovery Protocol

Peer Discovery Protocol (PDP) is used for discovering published resources. A peer, peer group, pipe, service is a resource in a JXTA environment. The resources are published as advertisements.

A peer can find advertisements (resources) on other peers using PDP. The PDP is the default discovery mechanism for peer groups.

2.2.3.2 Peer Resolver Protocol

The Peer Resolver Protocol (PRP) is one of the core protocols that provides a query/response interface for applications and services. This interface is used for building resolution services.

2.2.3.3 Peer Information Protocol

Peer Information Protocol (PIP) is used for obtaining status information of peers. The status information may be state, uptime or traffic load on a peer. The

PIP protocol provides messages for querying status information of a peer.

2.2.3.4 Pipe Binding Protocol

This protocol allows a peer to bind a pipe advertisement to a pipe endpoint, thus, enables messages to go over the pipe.

2.2.3.5 Endpoint Routing Protocol

Endpoint Routing Protocol allows a peer to ask a router for any available route for sending a message to a destination peer. The response of the router peer can be a list of gateways along the route. By implementing this protocol, any peer can be a router peer.

2.2.3.6 Rendezvous Protocol

The responsibility of rendezvous protocol is to enable peers to propagate messages to each other, and to reduce network traffic by caching most essential information to the operation of the P2P network.

2.3 Web Services

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network [12]. Web services are self-contained, self-describing, modular applications that can be published, located, and invoked across the Web [13]. Web services perform functions, which can be anything from simple requests to complicated business processes. Once a Web service is deployed and published, other applications possibly other Web services can discover and invoke the deployed service. Web Services provides an environment that does not depend on any platform or programming language.

Web services use some standards for enabling use of software elements of Web services. These standards allow Web services to be

- described
- published
- discovered
- invoked

in a standard way and these methods are illustrated in Figure 7.

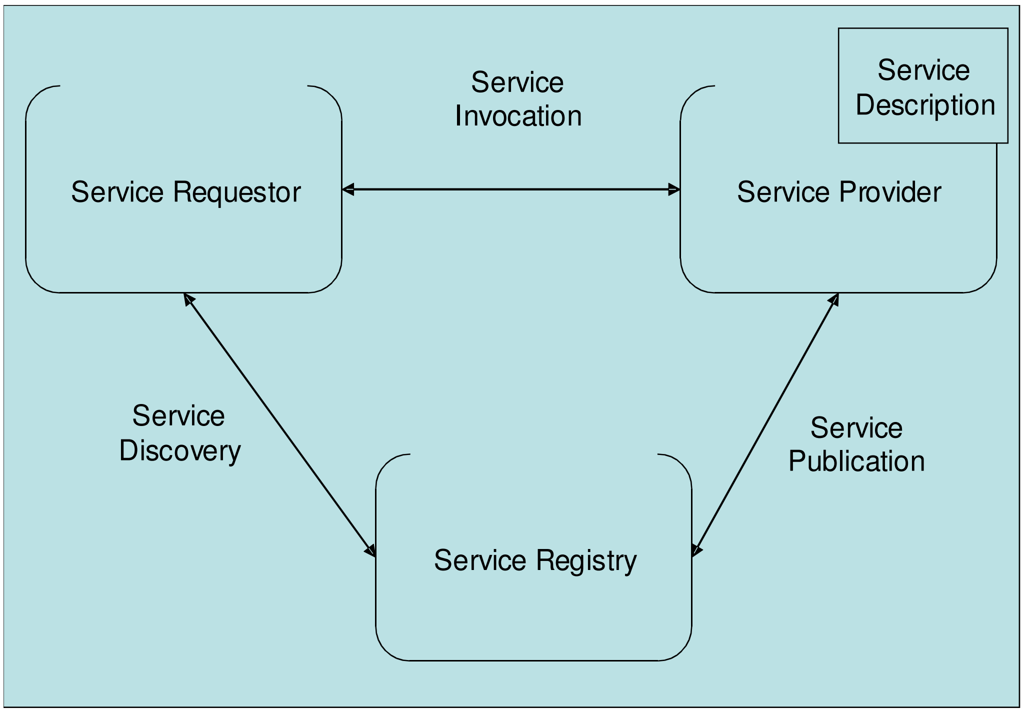


Figure 7: Methods and Roles of Web Service Model

The roles in Figure 7 can defined as follows:

Requestor : Web service client that discovers and invokes the service.

Provider : Web service server that composes, describes and publishes the service.

Registry : Keeps the description of Web services published by providers and allows requestors to search published Web services.

Web services are enabled by use of Extensible Markup Language (XML) for standardization of data formats and exchange of data through Internet or intranet.

For describing Web services, Web Services Description Language (WSDL) is widely used which is a W3C standard. For publishing and discovering Web services, Universal Description and Discovery Integration (UDDI) framework is used. Simple Object Access Protocol (SOAP) is a framework for invoking services via message exchanging.

2.3.1 XML: Extensible Markup Language

Extensible Markup Language (XML) is a simple, flexible, self-describing text format language [14]. XML is a World Wide Web Consortium (W3C) standard and widely adopted by the computer community. It is derived from SGML.

XML takes an important role in the exchange of a wide variety of data on any platform, such as Web services, P2P computing.

2.3.2 WSDL: Web Services Description Language

Web Services Description Language (WSDL) is an XML based language for describing network services as a set of endpoints operating on messages [15]. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. WSDL is extensible to allow description of endpoints and their messages.

A WSDL document defines *services* as collections of network endpoints, or *ports*. In a WSDL document, the abstract definition of endpoints and messages are separated from their concrete network deployment or data format bindings. This allows the reuse of abstract definitions: *messages*, which are abstract descriptions of the data being exchanged, and *port types* which are abstract collections of *operations*. The concrete protocol and data format specifications for a particular port type constitutes a reusable *binding*. A port is defined by associating a network address with a reusable binding, and a collection of ports define a service. Hence, a WSDL document uses the following elements in the definition of network services:

- **Types** – a container for data type definitions
- **Message** – definition of the data being communicated.
- **Operation** – action supported by the service.
- **Port Type** – set of operations supported by one or more endpoints.
- **Binding**– a protocol and data format specification for a particular port type.
- **Port** – a single endpoint defined as a combination of a binding and a network address.
- **Service** – a collection of related endpoints.

Web Services Description Language (WSDL) is the standard for describing a Web service in W3C.

2.3.3 SOAP: Simple Object Access Protocol

Simple Object Access Protocol (SOAP) is a protocol for exchange of information as messages in a decentralized, distributed environment [16]. It is based on XML and consists of three parts: an *envelope* that defines a framework for describing what is in a message and how to process it; a set of *encoding rules* for expressing instances of application-defined datatypes; and a *convention* for representing remote procedure calls and responses.

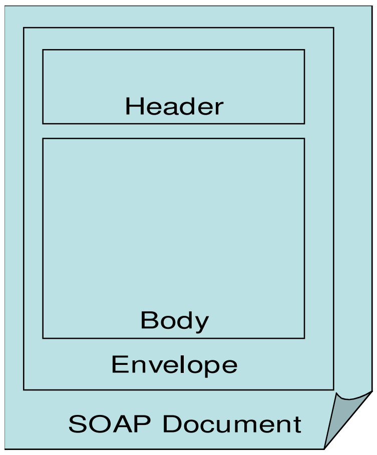


Figure 8: Structure of SOAP Document

2.3.4 BPEL4WS: Business Process Execution Language for Web Services

Business Process Execution Language for Web Services (BPEL4WS) is an XML-based flow language that enables defining business processes as coordinated sets of Web service interactions and defines both abstract and executable processes. Web services interactions may include processes contained within enterprise or between enterprises.

The aim of BPEL4WS is to provide a standard platform for creation of composite Web services and interactions between business partners. The language is based on the IBM's WSFL and Microsoft's XLANG languages and merges the concepts of these two languages. In order to realize its aim, BPEL4WS adds many concepts from structured programming languages such as, sequential execution, iteration, etc.

BPEL4WS allows describing complex business processes and business interactions between or within enterprises. The main activities to provide these complex business process interactions are:

- performing Web service invocations
- manipulating data
- handling fault conditions
- terminating process

The above activities may be nested within structured activities that defines, how these activities run, such as in sequence or in parallel.

2.3.4.1 WSIF: Web Services Invocation Framework

The Web Service Invocation Framework (WSIF) is a set of libraries that provides a simple API for invoking Web services. The features of WSIF is given as follows [17]:

- has an API that provides binding independent access to any Web service.
- has a port type compiler to generate a stub that allows invocation using the abstract service interface.
- allows stubless (completely dynamic) invocation of Web services.
- a new binding can be plugged in at runtime.
- allows the choice of a binding to be deferred until runtime.

WSIF invocation includes the following steps :

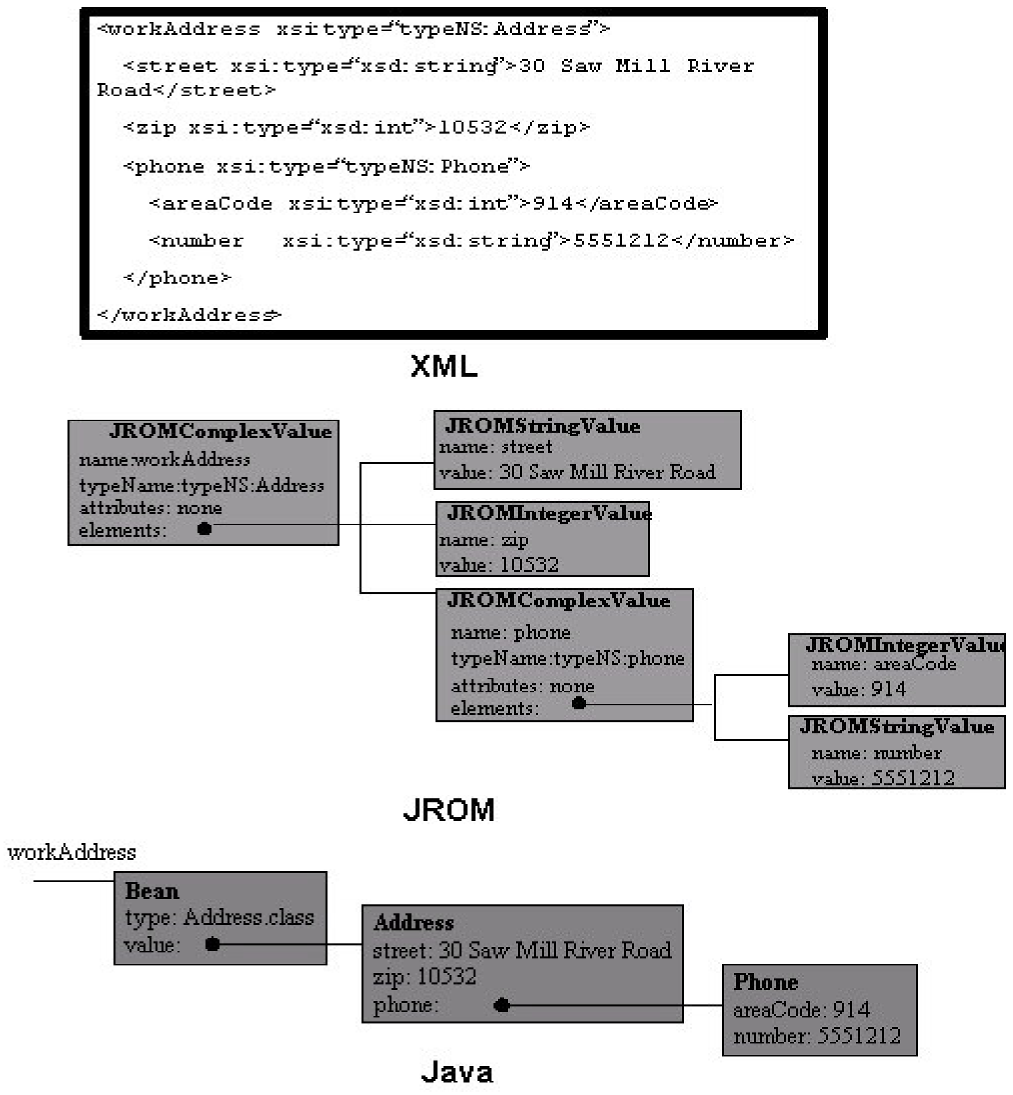
- Loading a WSDL document.
- Creation of a port factory for this service.
- Retrieving the service port using the port factory.
- Creation of messages.
- Invoking by supplying the port with the name of the operation to be invoked, along with an input and/or output message as described by the operation.

2.3.4.2 JROM: Java Record Object Model

Java Record Object Model (JROM) is a tool consisting of a set of interfaces for providing an in-memory tree representation of instances of structured, typed information and that is based on the XML Schema data type system. JROM values can either be simple values, or complex values that can contain an arbitrary number of elements and attributes. [18]

JROM is used in BPEL as an intermediate representation between XML and Java code. JROM enables dynamic invocation without needing Java class generation. It allows swapping in different SOAP engines with no need for the writing of custom de-serializers or serializers for specific Java classes. The simple JROM values contain a Java value that is able to represent the given Schema type. For example, a JROMFloatValue contains a Java float and corresponds to the Schema float simple type. However, all complex types are mapped to the tree structured JROMComplexValue. [19]

In Figure 9, an illustration of JROM and Java representation of an XML document is given.



```
<workAddress xsi:type="typeNS:Address">
  <street xsi:type="xsd:string">30 Saw Mill River
Road</street>
  <zip xsi:type="xsd:int">10532</zip>
  <phone xsi:type="typeNS:Phone">
    <areaCode xsi:type="xsd:int">914</areaCode>
    <number   xsi:type="xsd:string">5551212</number>
  </phone>
</workAddress>
```

Figure 9: Representation of an XML Document in JROM and Java [18]

Dynamic invocation of Web services in BPEL is enabled by the use of JROM and WSIF. Use of JROM with WSIF provides a powerful approach building dynamic Web services systems such as business processes in BPEL. WSIF supports both JROM and Java types.

2.3.4.3 WSGW: Web Services Gateway

Web Services Gateway (WSGW) is a set of run-time libraries for providing a gateway between Internet and intranet application [19]. The gateway enables clients to invoke Web services safely from outside a firewall. WSGW provides the following features:

- **Service mapping:** Maps an existing WSDL-defined Web service to a new service that is served by the gateway to others.
    - **Export mapping:** Maps an internal service to an external service in Internet. The WSDL file in enterprise intranet is regenerated as a new WSDL file. The clients that request the service from outside will use the gateway as the service end-point.
    - **Import Services:** Maps an external service to an internal service. An internal service requestors invoke the service as if it is running on the gateway.
- **Transformation:** The original web service protocol may change to another protocol at the end of service mapping. For example, an internal service available on SOAP over JMS may be invoked using SOAP over HTTP.
- **Security and management:** The web services gateway provides a single point of control, access, and validation of Web service requests.

2.4 Java Technology

Java is an object-oriented programming language and a platform developed by Sun Microsystems. Java technology is based on Java virtual machine (JVM) working as a translator between the language and the operating system and the hardware[20]. JVM has two functions: first, compiles the java programs into an intermediate language, namely java bytecode, then, interprets the java bytecode on each time the program is run. JVM is responsible for the hardware-independence and operating system-independence of the java platform, the small size of compiled code (bytecodes), and platform security. Thus, JVM enables Java programs to run on any system.

Java platform consists of JVM and Java Application Programming Interface (API). API is a collection of libraries used in development of java applications.

There are three versions Java platform that are specialized for different type applications and devices:

- *J2SE (Java 2 Standard Edition)* : Standard edition is the core of the Java technology that includes tools, runtimes, and APIs for developers who write, deploy, and run applets and applications in the Java programming language.

- *J2EE (Java 2 Enterprise Edition)* : This version is used in construction and deployment of multitier enterprise applications by basing them on standardized modular components. J2EE provides a complete set of services to modular components of an enterprise application, and handles many details of application behavior automatically, without complex programming.
- *J2ME (Java 2 Micro Edition)* : Micro edition version of Java is a optimized for Java runtime environment that specifically addresses the small (physically and in memory capability) devices like mobile phones and consumer and embedded devices ranges from smart phones or pagers to the set-top box.

2.4.1 J2SE (Java 2 Standard Edition)

Java 2 Platform, Standard Edition (J2SE) is an edition of Java language and platform that provides a complete programming environment for development of desktop or server applications [21]. J2SE platform consists of two products: Java 2 Runtime Environment, Standard Edition (JRE) and Java 2 Software Development Kit, Standard Edition (SDK). The JRE provides the Java APIs, Java virtual machine, and other components necessary to run applets and applications written in the Java programming language. Java 2 Platform, Enterprise Edition (J2EE) technology is based on JRE. The JRE does not contain tools and utilities such as compilers or debuggers for developing applets and applications.

The Java 2 SDK is a superset of the JRE. It contains everything that is in the JRE as well as tools such as the compilers and debuggers necessary for developing applets and applications.

Figure 10 illustrates all the component technologies in J2SE platform and in which layer these technologies located.

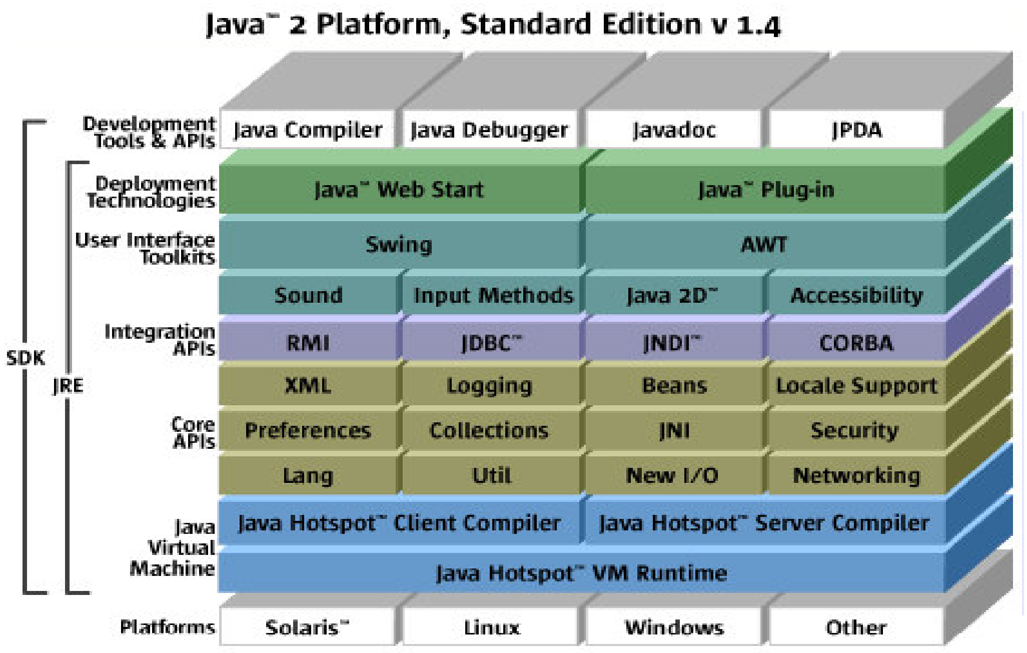


Figure 10: Java 2 Platform, Standard Edition Platform [21]

2.4.2 J2EE (Java 2 Enterprise Edition)

The Java 2 Platform, Enterprise Edition (J2EE) is the Java platform that is based on on JRE. The aim of J2EE is to standardize developing multitier enterprise application and simplify development and deployment of standardized enterprise applications. [22].

The J2EE platform takes advantage of many features of the J2SE environment. The J2EE supports Enterprise JavaBeans components, Java Servlets API, JavaServer Pages and XML technology.

2.4.3 J2ME (Java 2 Micro Edition)

The Micro Edition of the Java 2 Platform (J2ME) is a robust, flexible environment for applications running on embedded or consumer devices[23]. The consumer devices range from mobile devices and personal digital assistants (PDAs) to advanced consumer systems such as TV set-top boxes. J2ME includes Java virtual machines and a set of standard Java APIs defined through the Java Community Process (JCP) [24].

The set of standard Java APIs included in J2ME are flexible user interfaces, a robust security model, a broad range of built-in network protocols, and extensive support for networked and offline applications that can be dynamically and securely downloaded and deployed.

The J2ME platform is deployed on millions of consumer and embedded devices and supported by leading companies.

The J2ME architecture comprises a variety of configurations, profiles, and optional packages. Since there are various types of consumer devices and embedded systems that uses J2ME platform, for each system, a device-specific package is formed from these configurations and profiles to provide a complete Java runtime environment that closely fits the requirements of the system. Each combination is optimized for the memory, processing power, and I/O capabilities of the devices. The components of the J2ME architecture is layered stack and consisting of a virtual machine and core J2ME class libraries besides configurations and profiles. The layered stack of J2ME is illustrated in Figure 11.

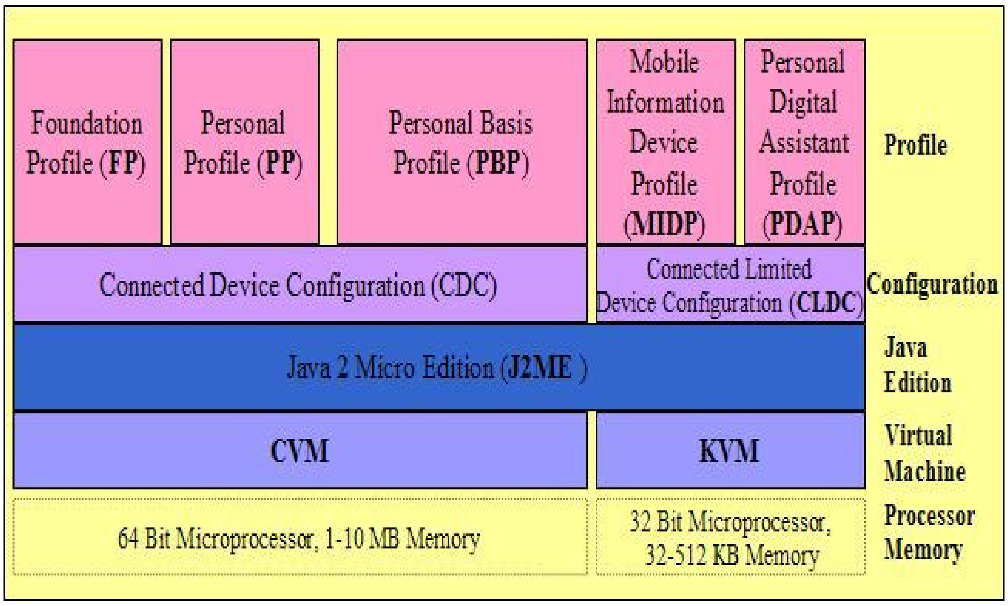

Foundation Profile (FP)
Personal Profile (PP)
Personal Basis Profile (PBP)
Mobile Information Device Profile (MIDP)
Personal Digital Assistant Profile (PDAP)
Profile
Connected Device Configuration (CDC)
Connected Limited Device Configuration (CLDC)
Configuration
Java 2 Micro Edition (J2ME)
Java Edition
CVM
KVM
Virtual Machine
64 Bit Microprocessor, 1-10 MB Memory
32 Bit Microprocessor, 32-512 KB Memory
Processor Memory

Figure 11: J2ME Platform

The components of J2ME architectures can be defined as follows:

- A *configuration* is consisting of a set of libraries and it provides virtual-machine features that must be present in a J2ME environment. The set of libraries with a virtual-machine defines a runtime-environment. Configurations provide the base functionality for a particular range of devices that share similar characteristics, such as network connectivity and memory footprint. Currently, there are two J2ME configurations: Connected Limited Device Configuration (CLDC) and the Connected Device Configuration (CDC). A configuration provides a Java platform for creating applications for consumer and embedded devices when joined with one or more profiles.

- A *profile* is a set of standard Java APIs that support consumer or embedded devices of a chosen configuration. When used with a specific configuration, a profile provides a complete Java application environment for the target device class. Together, they provide a rich run-time environment. Mobile Information Device Profile (MIDP) is a well known profile that is used with CLDC to provide a complete Java application environment for cell phones and other devices with similar capabilities.
- An *optional* package is a set of technology-specific APIs that extends the functionality of a Java application environment. Well known optional packages used in a J2ME environment are Wireless Messaging API (WMA) and the Mobile Media API (MMAPI).

2.4.3.1 CLDC (Connected Limited Device Configuration)

The Connected Limited Device Configuration (CLDC) provides the base set of application programming interfaces and a virtual machine for resource-constrained devices like mobile phones, pagers, and PDAs [25]. CLDC can be used with a profile, such as MIDP, to provide a Java platform for developing applications to run on devices with limited memory, processing power, and graphical capabilities.

The properties of the devices where CLDC can be used are as follows:

- 16-bit or 32-bit processor with a clock speed of 16MHz or higher
- At least 160-192 kilobytes of total memory, including both RAM and flash or ROM, available for the Java platform.
- Limited power, often battery powered operation.

- Connectivity to a network, often with a wireless, non-continuous connection with limited bandwidth.
- User interfaces with varying degrees of sophistication (low-level GUI APIs to high-level APIs) down to and including none.

The goal of the CLDC specification is to standardize a highly portable, minimum-footprint Java application development platform for resource-constrained, network-connected devices. It is developed through the Java Community Process (JCP). We can state the design goals of CLDC as follows:

- Reducing the requirements for deploying application.
- Making use of application portability by abstracting native system operations into standardized APIs
- Extending device functionality by allowing dynamic downloading of applications into the device

The set of base CLDC API packages are:

Table 2: CLDC Package

CLDC Package	Description
java.io	IO classes and packages
java.lang	virtual machine classes and interfaces
java.util	utility classes and interfaces
javax.microedition.io	CLDC generic connection framework classes and interfaces

2.4.3.2 CDC (Connected Device Configuration)

The Connected Device Configuration (CDC) provides Java technology to build and deliver applications that can be shared across a range of network-connected consumer and embedded devices, including high-end PDAs [26].

The properties of the devices where CLDC can be used are as follows:

- At least 32-bit processor with a clock speed of 16MHz or higher
- At least 2-2.5 megabytes of total memory, including both RAM and flash or ROM, available for the Java platform.

CDC is part of the J2ME platform and developed through JCP like other J2ME configurations, profiles and other tools such as optional packages. Target devices of CDC technology consisting of consumer devices and embedded systems those having higher capabilities, displays and processing power. Some properties of CDC configuration are as follows:

- can be integrated with enterprise applications
- provides more secure application, deployment and network environment
- provides rich set of user interface APIs
- supports a wide range of consumer and embedded devices
- supports reuse of code that are developed for J2SE-based applications

As illustrated in Figure 11, CDC can be used with Foundation Profile (FP), Personal Profile (PP) and Personal Basis Profile (PBP). The properties of these profiles are given in Table 3 as follows.

Table 3: J2ME Profiles that are used with CDC

Foundation Profile (FP)	• Java class libraries are based on J2SE edition • There is no GUI support • Provides CLDC compatible library.
Personal Basis Profile (PBP)	In addition to FP features, PBP supports: • lightweight Java Abstract Windowing Tool (AWT) • Xlet class which is an application model based on Java TV API.
Personal Profile (PP)	In addition to PBP APIs, PP supports: • heavyweight and lightweight AWT • applets

2.4.3.3 MIDP (Mobile Information Device Profile)

The Mobile Information Device Profile (MIDP) is a profile that provides Java runtime environment for resource-limited mobile devices [27]. MIDP API allows user interface design, networking and persistent storage support. It is supported by CLDC and when combined with the CLDC, provides a standard Java runtime environment for mobile information devices, such as cell phones and mainstream PDAs. MIDP is the most widely adopted and used profile for mobile devices and

PDAs for enabling for mobile Java applications to run.

CLDC and MIDP provide the core application functionality required by mobile Java applications. The properties of MIDP APIs are as follows:

- **User Interface:** MIDP provides the code graphical user interface classes for Java applications. The GUI is optimized for the small display size, varied input methods of modern mobile devices. MIDP provides intuitive navigation and data entry by taking advantage of phone keypads, extra buttons such as arrow keys, touch screens. MIDP applications are installed and run locally, can operate in both networked and unconnected modes, and can store and manage persistent local data securely.
- **Connectivity:** MIDP enables developers to use the native data network and messaging capabilities of mobile information devices. It supports leading connectivity standards, including HTTP, HTTPS, datagrams, sockets, server sockets, and serial port. MIDP also supports the Short Message Service (SMS) and Cell Broadcast Service of GSM and CDMA networks, through the Wireless Messaging API (WMA) optional package.
- **Multimedia:** MIDP supports development of multimedia applications. Beside high-level UI API, MIDP provides low-level user-interface API for application developers who want to take greater control of graphics and input. Mobile Media API (MMAPI) optional package can also be used for adding video and other rich multimedia content to MIDP applications.
- **Over-the-Air-Provisioning:** MIDP provides dynamic and secure deployment of mobile applications.
- **Security:** The network, application and information security on a mobile device is provided MIDP. MIDP supports HTTPS which enables applications to use existing security standards such as SSL send and receive encrypted data.

The set of base MIDP API packages are:

Table 4: MIDP Package

MIDP Package	Description
javax.microedition.lcdui	UI classes and interfaces
javax.microedition.rms	Record management system classes to support persistent storage
javax.microedition.midlet	MIDP application definition support classes and interfaces
javax.microedition.io	MIDP generic connection framework classes and interfaces
java.io	IO classes and packages
java.lang	virtual machine classes and interfaces
java.util	utility classes and interfaces

A J2ME application build on top of MIDP profile and CLDC configuration is known as MIDlet. MIDlets are similar to applets in that they both have state information. A MIDlet can exist in four different states: loaded, active, paused, and destroyed. The lifecycle of a MIDlet is illustrated in Figure 12.

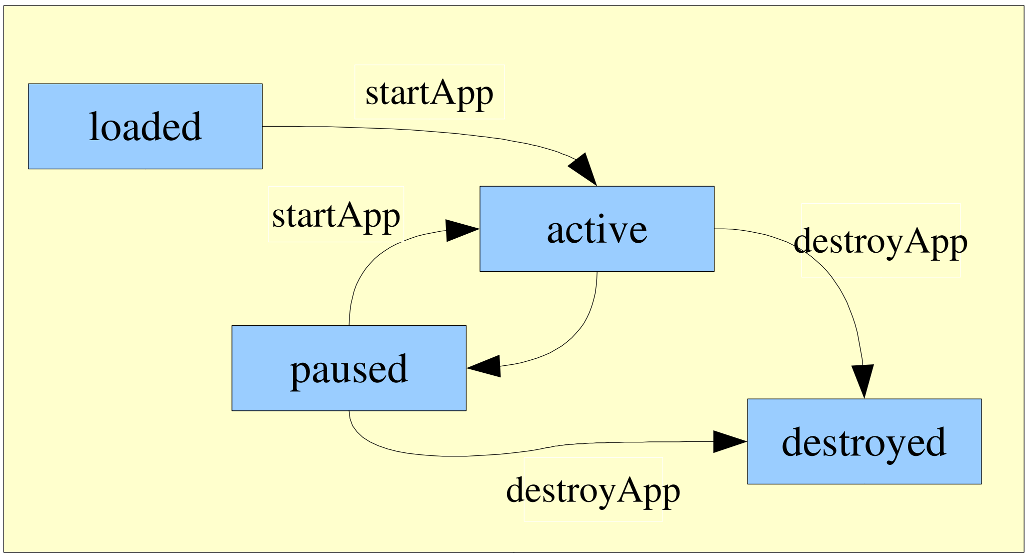


Figure 12: MIDlet lifecycle

First, MIDlet is loaded into the device and the constructor is called, it is in the loaded state. After startApp() is called, the MIDlet is in the active state until the program manager calls pauseApp() or destroyApp(); pauseApp() pauses the MIDlet, and desroyApp() terminates the MIDlet.

CHAPTER 3

SYSTEM ARCHITECTURE

In this chapter, the system architecture is explained together with the tools used. This chapter aims to give a detailed explanation of the components of the system architecture and tools used in the thesis. Firstly, we introduce the hardware and software environment. Then, we explain the web services standards and tools. Finally, we conclude this chapter with the JXTA environment tools.

3.1 Hardware and Development Environment

The client and server of the proposed architecture are implemented on different types of machine architectures. The server is a desktop application, whereas, the client is developed as a mobile application. The mobile device used in this thesis is a personal digital assistant (PDA).

3.1.1 Client

The client device used for invoking Web services and joining to a P2P environment is HP's iPaq 5450 Pocket PC. Microsoft Pocket PC 2002 operating system is running on the device. Other properties of the device is given in the Table 5.

Table 5: Properties of HP iPaq 5450 Pocket PC

Operating System	Microsoft Pocket PC 2002
Processor	400MHz Intel XScale
Memory	64 MB RAM, 48 MB ROM
Display Type	16 bit, 64K color, 240 x 320
Wireless Connectivity	Built in Bluetooth (1.1 compliant) and 802.11b (Wireless Local Area Network (WLAN))

3.1.1.1 Jeode Runtime Environment

Jeode platform is an implementation of Sun Microsystems JVM specification that supplies a java run-time environment for resource-constrained devices that have limited memory, battery power and display screen. Embedded and mobile devices that Jeode supports have the following characteristics:

- diverse architecture types
- less memory
- variety of graphics
- low power processors

Jeode runtime environment provides an environment that compatible with J2ME configurations such as CDC and CLDC.

Jeode platform is implemented by Insignia Solutions Inc. [28] and it is a product of this company.

3.1.1.2 ME4SE

ME4SE is a library that enables J2ME APIs, such as lcdui, persistent storage, networking, and the Generic Connection Framework available for the Java 2 Standard Edition [29]. The motivation of the project is:

- providing limited development support for platforms where no emulator is available.
- allowing demonstration of MIDlets before installation on the device.
- enabling personal java devices to run MIDlets.

3.2 Web Services Tools

In this section, the tools that enables the use of web services standards on server and client machines are introduced.

3.2.1 BPWS4J

The Business Process Execution Language for Web Services (BPWS4J) is a platform on which business processes, written using the Business Process Execution Language for Web Services (BPEL4WS), can be executed.[30] BPWS4J runs in a servlet container.

The BPWS4J engine works as follows:

- The BPWS4J engine takes
    - a BPEL4WS document that describes the process to be executed,
    - a WSDL document (without binding information) that describes the interface that the process will present to clients (partners in BPEL4WS terms),
    - a WSDL documents that describe the services that the process may or will invoke during its execution.
- From this information, the process is made available as a Web service with a SOAP interface. A WSDL file that describes the process's interface may be retrieved from the run-time. The BPWS4J engine supports the invocation, from within the process, of Web services that have a SOAP interface, that are EJBs, or that are normal Java classes.

3.2.2 kXML

kXML is a library that provides an XML pull parser and writer for all Java platforms including the Java 2 Micro Edition (J2ME) (CLDC/MIDP/CDC). It is designed for constrained environments such as MIDP devices. The project is maintained by the Enhydra organization [31]. kXML provides the following key features:

- XML Namespace support
- "Relaxed" mode for parsing HTML or other SGML formats
- Small Memory footprint
- A Pull-based parser for simplified parsing of nested / modularized XML structures
- XML writing support including namespace handling
- Optional kDOM
- Optional WAP support (WBXML/WML)

Despite supporting XML for mobile devices, kXML has the following restrictions:

- kXML does not support user defined (external) entities.
- The doctype declaration can not be directly parsed.

3.2.3 kSOAP

The kSOAP project is a lightweight pull parser based on kXML, designed specifically for use with MIDP devices. The project is maintained by the Enhydra organization [32].

kSOAP is a relatively simple tool for invoking Web services through SOAP messages. It supports capturing and handling faults in SOAP messaging.

kSOAP requires two objects for messaging, namely *SoapObject* and *HttpTransport* and an object for mapping SOAP faults to an exception, namely *SoapFault*.

3.3 JXTA Environment

The Project JXTA envisions creating a P2P environment where interconnected peers easily communicate and collaborate with each other. It provides this flexibility by standardizing only the protocols. This enables JXTA to be architecture-independent, language-independent and ubiquitous platform.

3.3.1 J2SE JXTA Platform

J2SE JXTA platform is a reference implementation of core and standard JXTA protocols in Java programming language using J2SE. It provides sets of libraries that contains Java APIs for programming JXTA applications in Java. The JXTA platform standardizes methods for:

- discovering other peers
- advertising peer resources (Peer, PeerGroup, Service and Pipe Advertisements)
- communicating with each other (Pipes)
- cooperating with each other to form secure peer groups (group membership)

3.3.2 JXME : JXTA for J2ME Devices

JXME is an implementation of a small subset of JXTA platform protocols for resource-limited devices. It provides classes for connecting to a relay peer and polling for message spooled in relay peer. There are three classes mainly used for connecting to a peer, polling for messages, constructing messages that will be send and evaluating the received message. The classes are:

- PeerNetwork
- Message
- Element

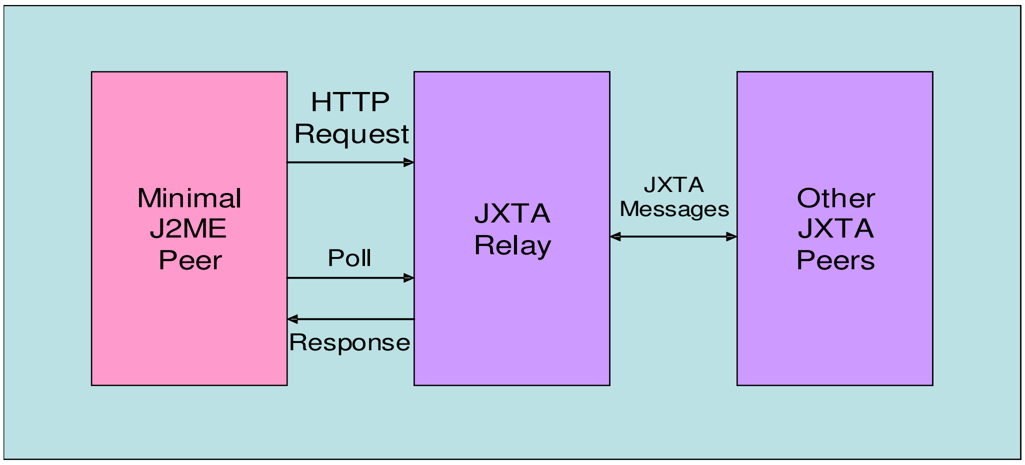


Figure 13: The Flow of Messages in JXME Architecture

CHAPTER 4

ARTEMIS SYSTEM ARCHITECTURE

Most of the health information systems today are proprietary and often only serve one specific department within a healthcare institute. A patient's health information may be spread out over a number of different institutes which do not interoperate. This makes it very difficult for clinicians to capture a complete clinical history of a patient.

On the other hand, the Web services model provides the healthcare industry with an ideal platform to achieve the difficult interoperability problems. Web services are designed to wrap and expose existing resources and provide interoperability among diverse applications.

Introducing Web services to the healthcare domain brings many advantages:

- It becomes possible to provide the interoperability of medical information systems through standardizing the access to data through WSDL and SOAP rather than standardizing documentation of electronic health records.
- Medical information systems suffer from proliferation of standards to represent the same data. Web services allow for seamless integration of disparate applications representing different and, at times, competing standards.
- Web services will extend the healthcare enterprises by making their own services available to others.
- Web services will extend the life of the existing software by exposing previously proprietary functions as Web services.

The Artemis project [2] the interoperability problem in the healthcare domain where organizations have proprietary application systems to access data. To exchange information there are different standards like HL7, GEHR or CEN’s ENV 13606. The aim of the Artemis project is to allow organizations keep their proprietary systems, yet expose the functionality through Web services. Furthermore, an ontology based description of these data exchange standards are proposed. One of the goals of using ontologies is to reduce (or to eliminate) conceptual and terminological differences among the healthcare data exchange standards through semantic mediation.

Mediators are developed to process data from possibly several data sources and to prepare them for the effective use by applications [33]. However with WWW becoming the global communication medium and with the Semantic Web initiative, ontologies are becoming the primary part of the mediation process.

Artemis Web service architecture does not rely on globally agreed ontologies: rather health-care institutes develop their own ontologies. However, it is reasonable to expect healthcare institutes to develop their own ontologies based on the concepts provided by the existing healthcare information standards since considerable semantic information is already captured there.

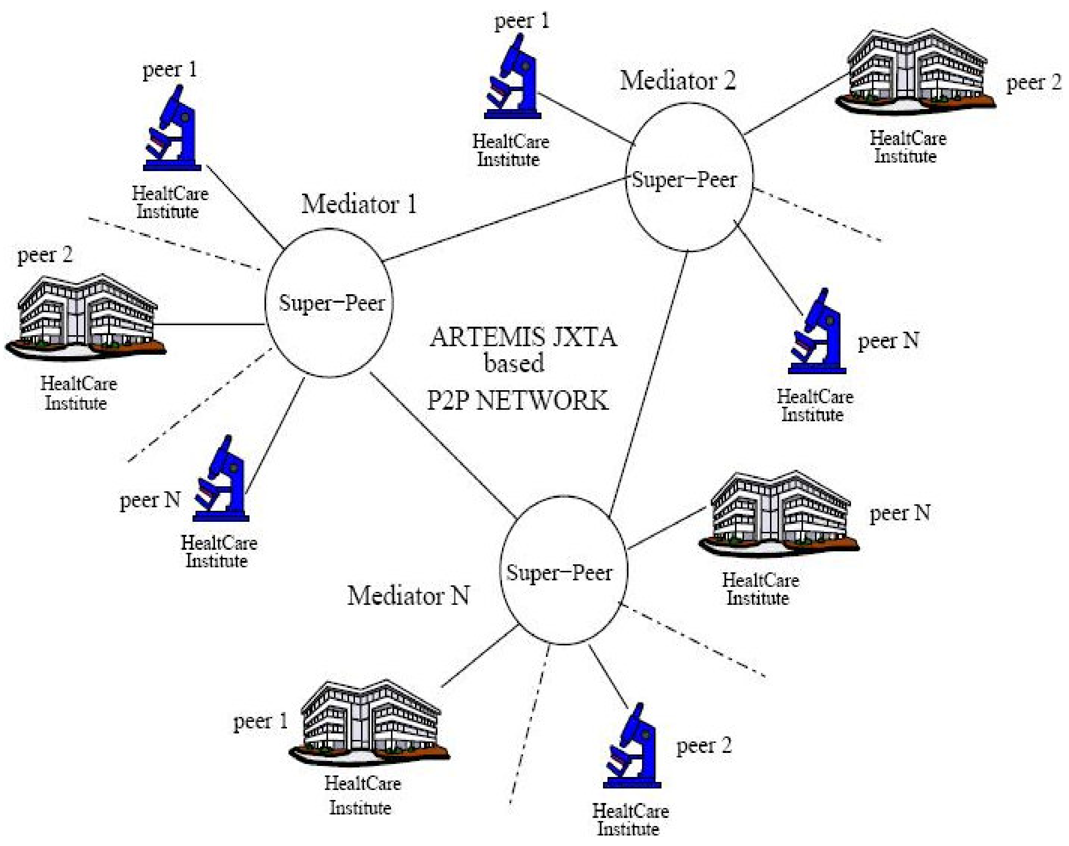


Figure 14: Artemis P2P Architecture

Artemis architecture then helps to reconcile the semantic differences among healthcare institutes through the mediator component. To provide scalability and discovery of other mediators, it has a P2P communication architecture. An overview of Artemis architecture is given in Figure 14.

4.1 Artemis Mediator P2P Architecture

In Artemis, healthcare institutes communicate with each other through mediators which resolve their differences bilaterally. The following observations on organization of mediators are made:

- The mediators must have a distributed architecture to provide for scalability.
- When a healthcare institute, say A, wants to communicate with another healthcare institute, say B, it should be possible to automatically locate the mediator of B.
- There are efficiencies to be gained by logically grouping the healthcare

institutes which communicate often through a single mediator.

With these considerations in mind, Artemis mediators are organized as JXTA super peer groups. In the JXTA super peer based architecture, peers in a peer group communicate with their super peer to advertise their capabilities as well as to search for other peers.

In Artemis, each mediator is a super peer serving the healthcare institutes in its logical peer group. Super-peers employ keyword based routing indices where keywords are used to locate the healthcare institutes. On registration the peer provides this information to its super-peer.

4.2 Artemis Mediator Component

Semantic mapping is the process where two ontologies are semantically related at conceptual level and source ontology instances are transformed into target ontology entities according to those semantic relations. In Artemis, the source and target ontologies belong to the two healthcare institutes willing to exchange information. However, the mapping of these two ontologies are achieved through the reference ontologies stored in the mediator: the generic Service Functionality ontology for classifying Web services in healthcare domain and Service Message ontology for annotating finer granularity services retrieving meaningful electronic healthcare record (EHR) components. The mediator resolves the semantic differences between source and target ontologies by using these ontologies.

Since all the ontologies involved are related with the basic healthcare standards, the mediation process is simpler and hence more efficient. Furthermore, resolved semantic differences are stored as Virtual Web Services (VWS).

The mediator architecture, which is shown in Figure 15, has the following subcomponents:

- Ontology server: The Ontology server contains the following ontologies:

- Service Functionality and Service Message ontologies: Each healthcare institute may develop its own Service Functionality and Service Message ontologies based on existing healthcare information standards. The minimum requirement is annotating their services through such ontologies.
- Virtual Web Services subsystem handles the creation of Virtual Web Services (VWSs) to provide complex aggregations of Web services. The creation of VWSs is realized according to the mappings between the ontologies of Web services' input and output semantics. Newly created VWSs are classified according to the Service Functionality Ontology of the requesting party for its possible future reuse.

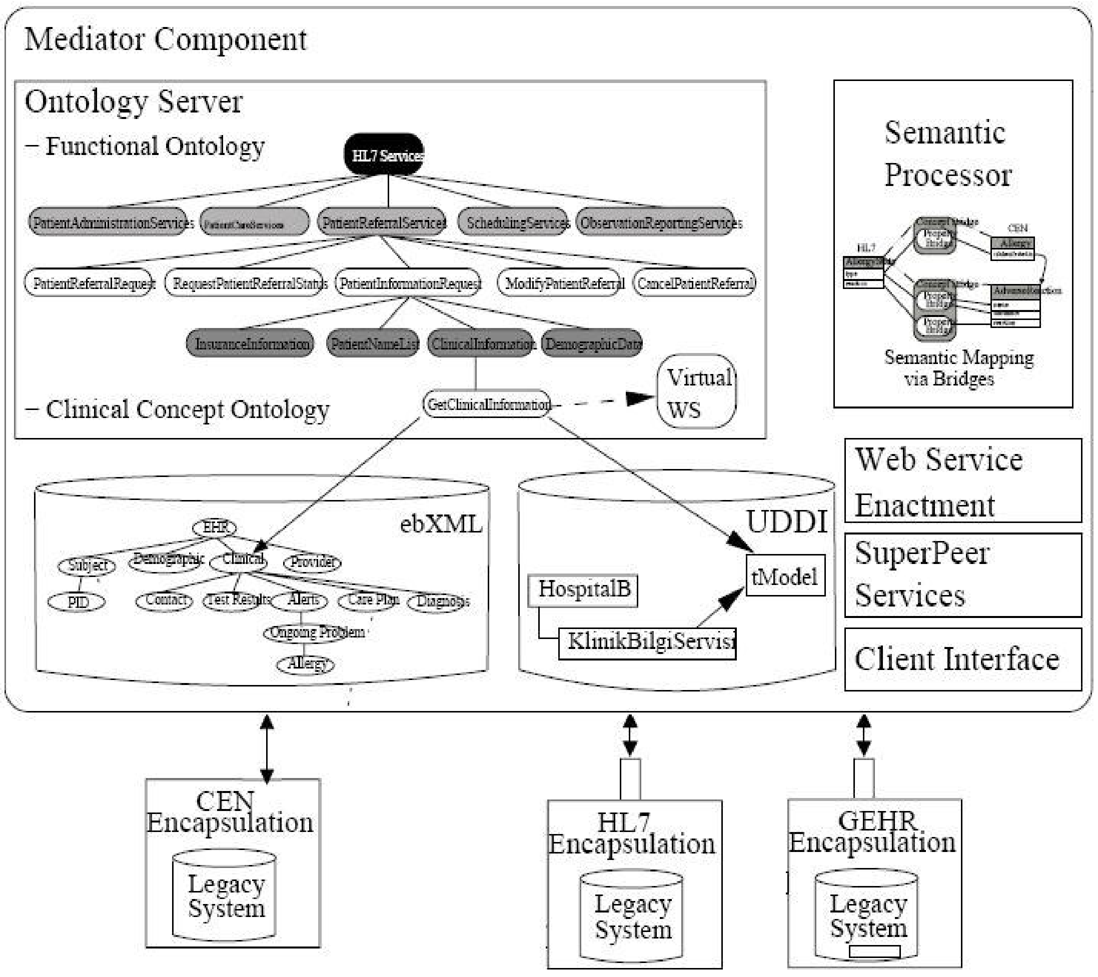


Figure 15: An Overview of the Mediator

- Semantic Processor: There may be more than one Service Functionality and Service Message ontologies in the mediator and the mediator generates the mappings between them using its own reference ontologies based on the healthcare standards. In Artemis, MAFRA [34] is used to represent the mappings and to transform the ontology instances. MAFRA uses the Semantic Bridge Ontology to define the mappings and includes a transformation engine. The mediator stores the previously defined mappings via semantic bridges. For example, the semantic equality relation between the "DiagnosticTestResult" concept in ENV 13606, and the "ObservationResult" concept in HL7 can be represented using MAFRA semantic bridges as follows:

```
<a:ConceptBridge rdf:ID="CB163312">
        <a:relatesTargetEntity                          rdf:resource=
        "http://www.srdc.metu.edu.tr/HL7#ObservationResult"/>
        <a:relatesSourceEntity                          rdf:resource=
        "http://www.srdc.metu.edu.tr/CEN#DiagnosticTestResult"/>
        <a:abstract rdf:resource="&a;True"/>
</a:ConceptBridge>
```

Note that, more complex mappings can be represented using "semantic bridges", such as compositions, alternatives, and transformations aided by external functions.

At runtime the source ontology instances are transformed into target ontology instances by providing the source instance and the rdf representation of mapping to the transformation engine of MAFRA.

- Service registries like UDDI and ebXML: The Web services of the involved healthcare institutes are published in the UDDI or ebXML registries of the mediator.
- Web service Enactment Component handles the invocation of the Web Services and transmits the results of the Web Services. Bridge [35] is used to deploy and invoke Web services in JXTA environment.
- Superpeer Services Component contains the services that provides the communication with other Mediators in a P2P infrastructure. Basically, these services implement the JXTA Protocols. For example, Discovery Service that

implements the JXTA Peer Discovery Protocol is used to find the other Mediators through a keyword based search mechanism.

- Client Interface handles the communication of healthcare institutes with the mediator using client-mediator protocol.

CHAPTER 5

IMPLEMENTATION

In this chapter, first the overall picture of the tools used is illustrated, then the implementation of a JXTA application for deployment and execution of business processes for especially connected-limited devices is explained.

5.1 Components of the Architecture

In Figure 16, the components of the architecture and tools that are covered in Chapter 3 are briefly described.

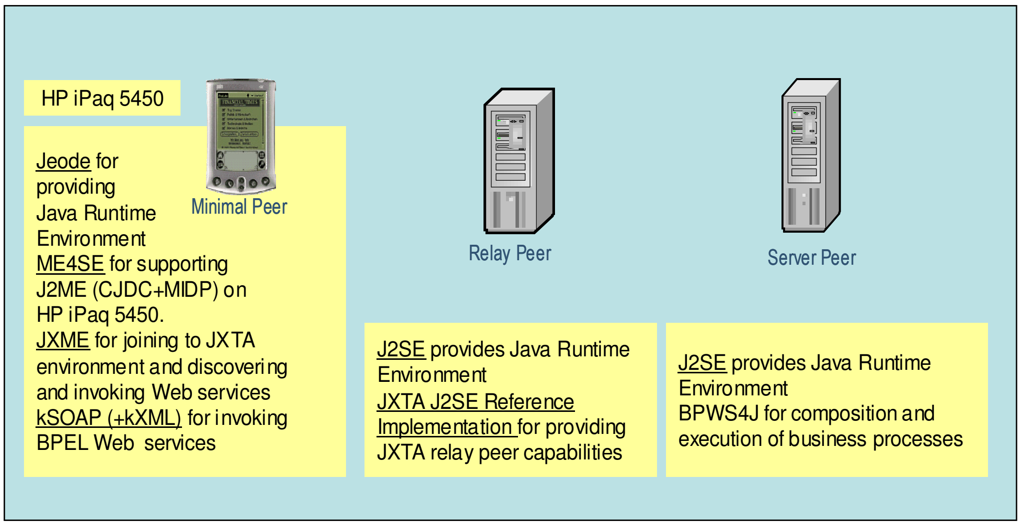

HP iPaq 5450
Minimal Peer
Jeode for providing Java Runtime Environment
ME4SE for supporting J2ME (CJDC+MIDP) on HP iPaq 5450.
JXME for joining to JXTA environment and discovering and invoking Web services
kSOAP (+kXML) for invoking BPEL Web services
Relay Peer
J2SE provides Java Runtime Environment
JXTA J2SE Reference Implementation for providing JXTA relay peer capabilities
Server Peer
J2SE provides Java Runtime Environment
BPWS4J for composition and execution of business processes

Figure 16: Components and Tools of the Architecture

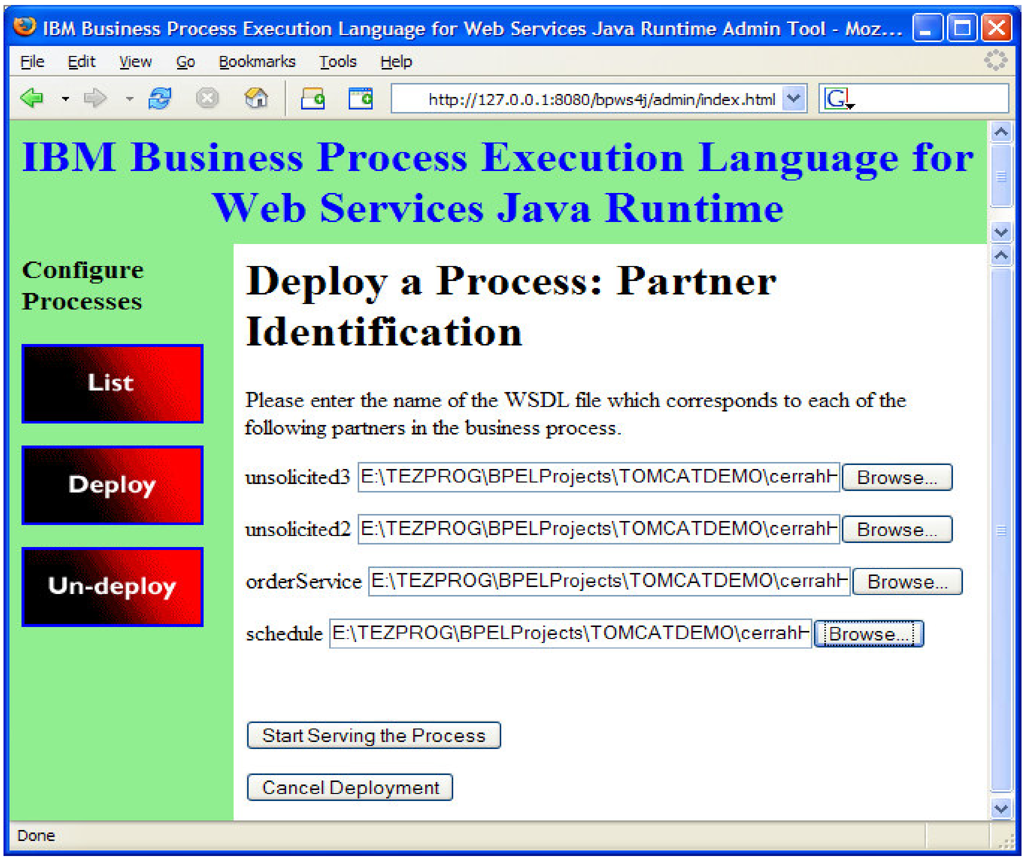
5.2 Deploying Business Processes

In order to use a Web service, first, we have deployed BPEL and WSDL documents in IBM BPEL4WS tool.

IBM BPEL4WS engine provides a user friendly interface for deployment of BPEL documents. The steps in deployment of a business process described in BPEL is as follows:

- The engine takes a BPEL document and related WSDL document.
- The BPEL document and related WSDL document is parsed.
- The engine requests the other WSDL documents that are invoked during the execution of the BPEL document.

The engine gives the Web service invocation point, namely SOAP address, and the methods provided by the service.


IBM Business Process Execution Language for Web Services Java Runtime Admin Tool - Moz...
File Edit View Go Bookmarks Tools Help
http://127.0.0.1:8080/bpws4j/admin/index.html
IBM Business Process Execution Language for Web Services Java Runtime
Configure Processes
List
Deploy
Un-deploy
Browse...
Start Serving the Process
Cancel Deployment
Done

Deploy a Process: Partner Identification

Please enter the name of the WSDL file which corresponds to each of the following partners in the business process.

unsolicited3 E:\TEZPROG\BPELProjects\TOMCATDEMO\cerrahH

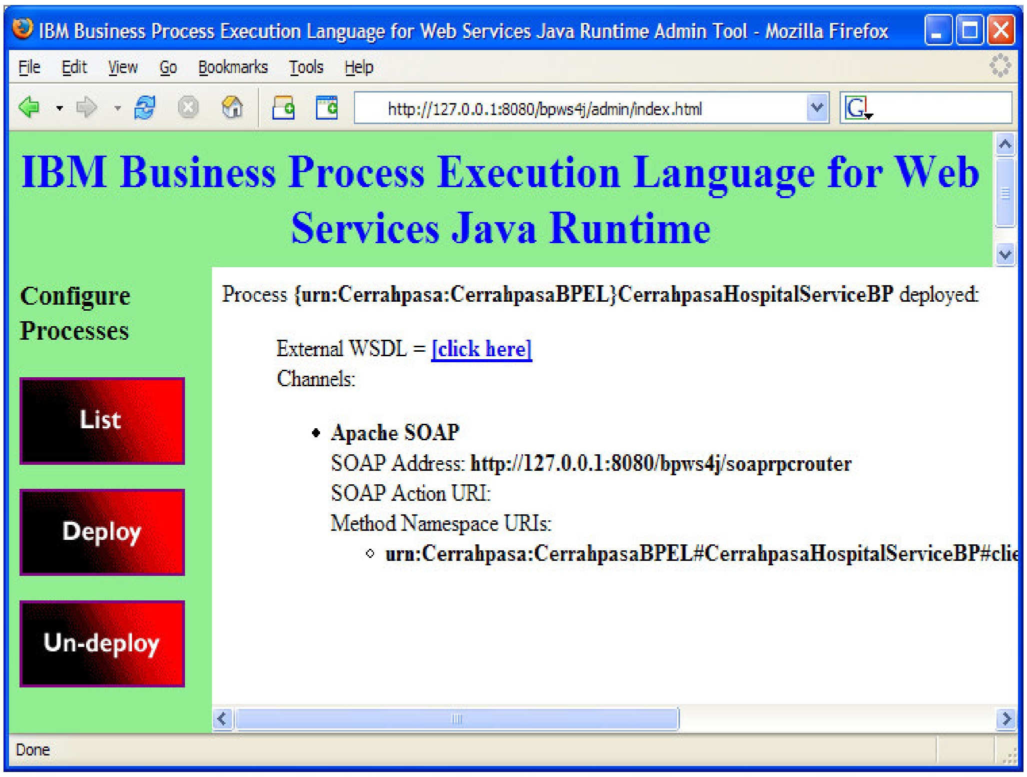
unsolicited2 E:\TEZPROG\BPELProjects\TOMCATDEMO\cerrahH

orderService E:\TEZPROG\BPELProjects\TOMCATDEMO\cerrahH

schedule E:\TEZPROG\BPELProjects\TOMCATDEMO\cerrahH

Figure 17: Deploying WSDL Documents in IBM BPEL4WS

Figure 17 shows the BPEL4WS page that takes the invoked WSDL documents and Figure 18 gives the SOAP addresses and methods of the deployed Web services. The BPEL and WSDL documents used for deployment of a Web service are parts of Artemis project [36].


IBM Business Process Execution Language for Web Services Java Runtime Admin Tool - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

http://127.0.0.1:8080/bpws4j/admin/index.html

IBM Business Process Execution Language for Web Services Java Runtime

Configure Processes

List

Deploy

Un-deploy

Process **{urn:Cerrahpasa:CerrahpasaBPEL}CerrahpasaHospitalServiceBP** deployed:

External WSDL = [click here]

Channels:

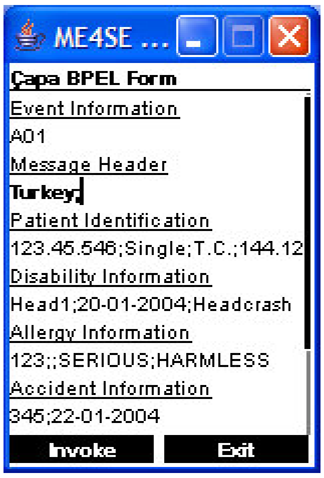
- **Apache SOAP**
  SOAP Address: **http://127.0.0.1:8080/bpws4j/soaprpcrouter**
  SOAP Action URI:
  Method Namespace URIs:
  - **urn:Cerrahpasa:CerrahpasaBPEL#CerrahpasaHospitalServiceBP#clie**

Done

Figure 18: Last Step in Deploying Business Processes in IBM BPEL4WS

5.3 Invoking BPEL Web Services from Mobile Devices

In Figure 19, the invocation of Web services deployed in Section 5.1 is invoked via a mobile device emulator, namely ME4SE.


ME4SE ...

Capa BPEL Form

Event Information

A01

Message Header

Turkey;

Patient Identification

123.45.546;Single;T.C.;144.12

Disability Information

Head1;20-01-2004;Headcrash

Allergy Information

123;;SERIOUS;HARMLESS

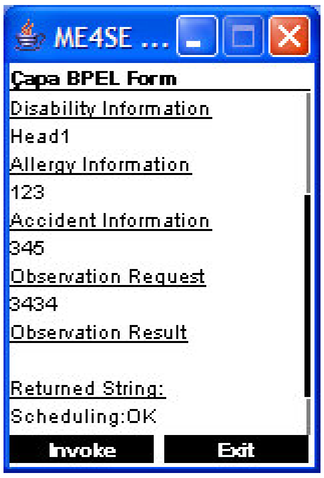
Accident Information

345;22-01-2004

Invoke Exit

Figure 19: Invocation of Deployed Web Service from ME4SE Emulator

In Figure 20, the result of the invoked Web service is given.



Çapa BPEL Form

Disability Information

Head1

Allergy Information

123

Accident Information

345

Observation Request

3434

Observation Result

Returned String:

Scheduling:OK

Figure 20: Result of the Invocation

5.4 Architecture of Publication and Discovery of Web Services in JXTA Environment

The architectural design in Figure 21 and denotes the use of Project JXTA protocols and the tools that enables facilitation of P2P networking on mobile devices with Web service technologies.

Key issues that are considered in this architecture are:

- allowing mobile devices to join in a P2P network group
- publishing Web services in P2P network
- discovering Web services in P2P network
- invoking Web services that are published and discovered

Figure 21 illustrates a P2P network environment where a mobile device can join. The illustration shows that a mobile device can be a minimal peer and join to a P2P environment through JXTA relay peers.

The minimal peers can only connect to a peer group through relay peer, send message to the relay peer and receive message from relay peer by periodically polling the relay peer. Sending and receiving capabilities of minimal peers can enable a mobile device running as a minimal peer to facilitate peer group services and other implemented services.

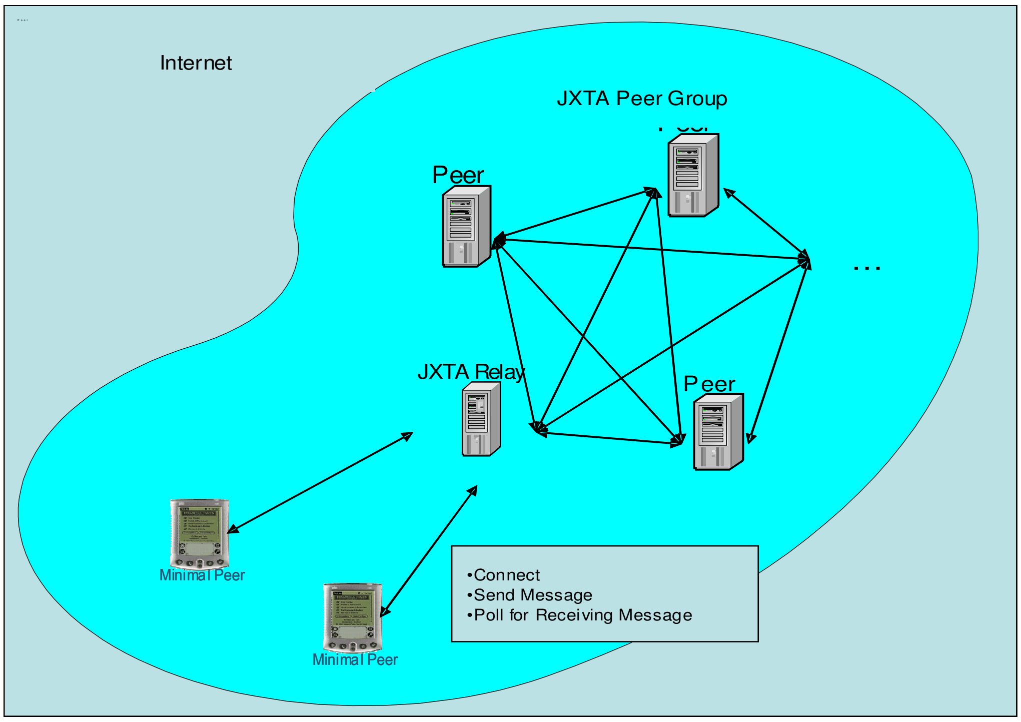


Figure 21: Peer Group with Mobile Devices

denotes the whole picture and describes key issues that are considered, namely, allowing mobile devices to join in a P2P network group, publishing Web services in P2P network, discovering Web services in P2P network and invoking Web services that are published and discovered.

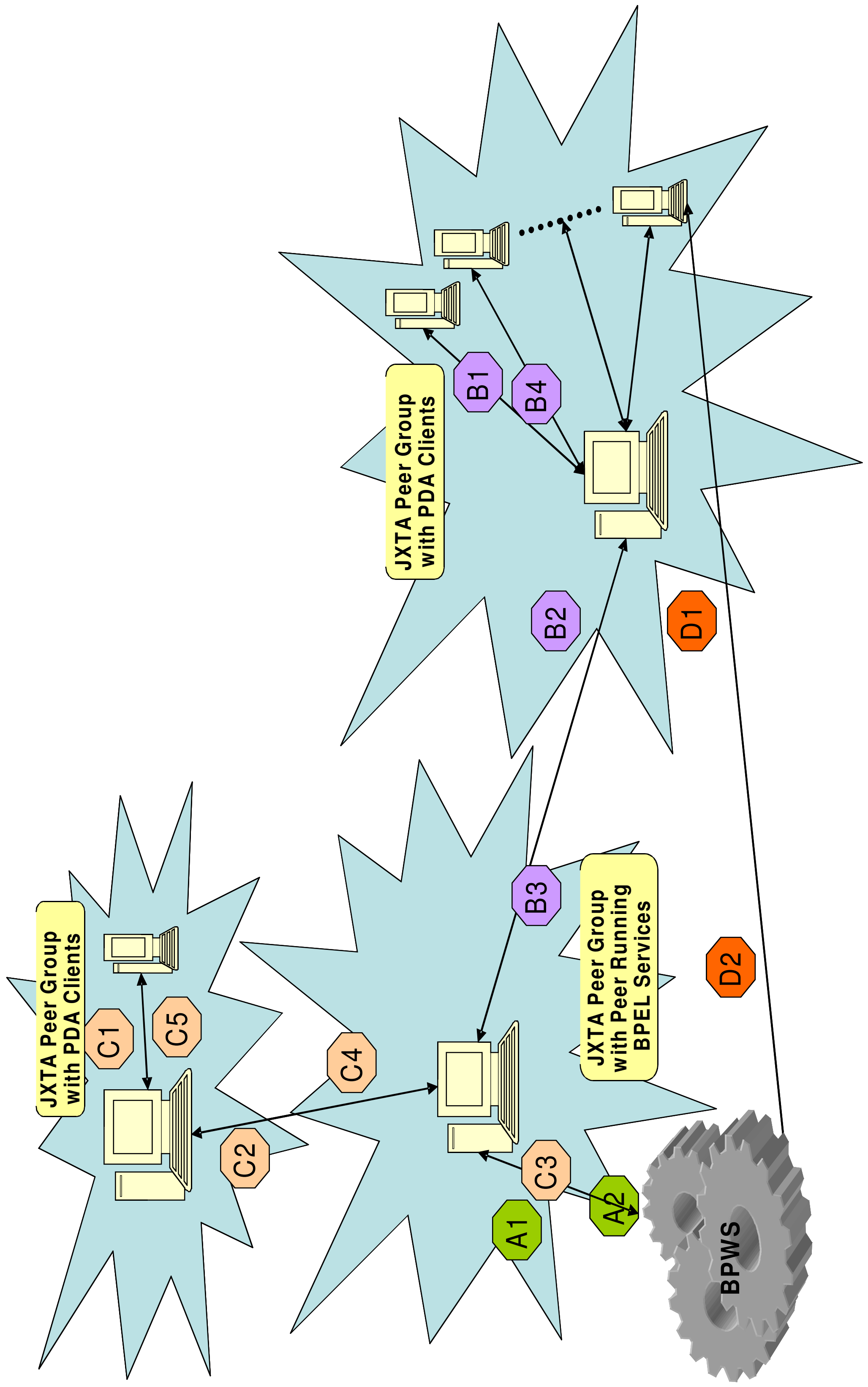
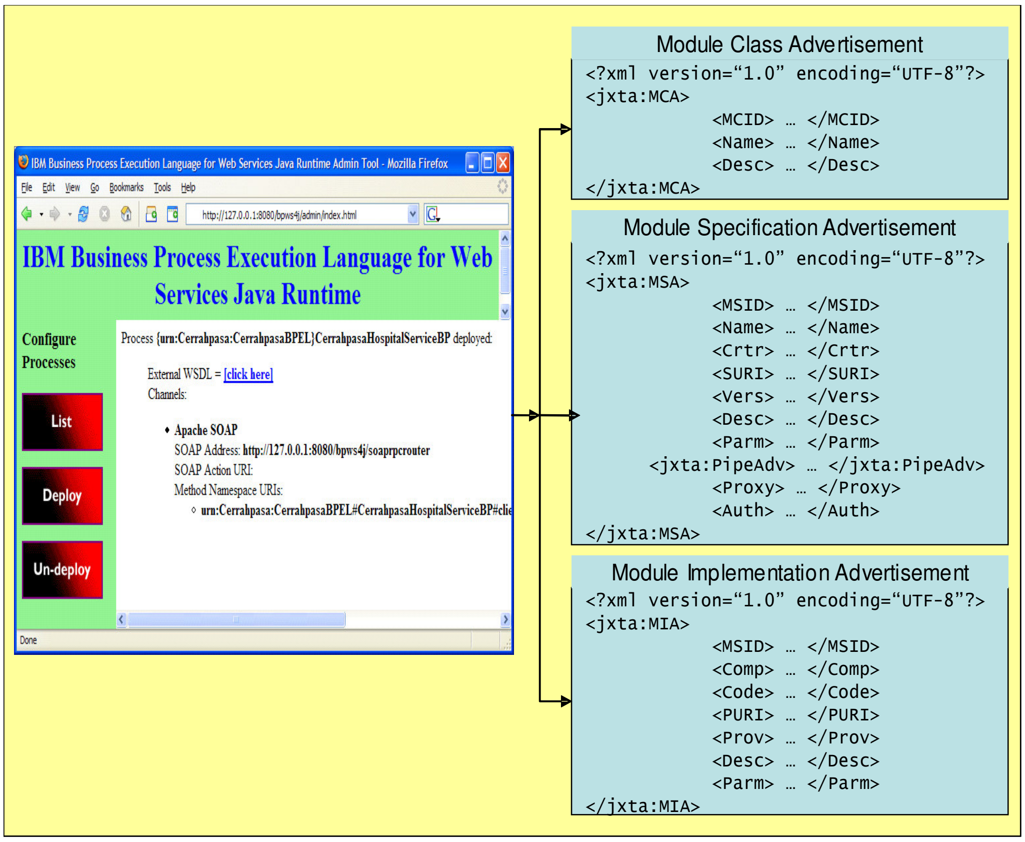

JXTA Peer Group with PDA Clients
B1
B4
B2
D1
JXTA Peer Group with PDA Clients
C1
C5
C2
C4
B3
JXTA Peer Group with Peer Running BPEL Services
D2
C3
A1
A2
BPWS

Figure 22: Architectural Design of the Complete System

Process Type	Step Number	Description
A	1	Convert the BPEL into Module Class Advertisement (MCA), Module Service Advertisement (MSA), Module Implementation Advertisement (MIA)
A	2	Publish the MCA, MSA, MIA through JXTA Peer
B	1	PDA peer sends a message to the relay peer for finding a Web service through JXTA peer group
B	2	JXTA relay peer searches the Web service through JXTA peer group
B	3	JXTA relay peer finds the MCA of the Web services
B	4	JXTA relay peer sends the Web service information to the PDA peer
C	1	PDA peer prepares the input for the Web service and sends it to the JXTA relay peer.
C	2	JXTA relay peer gets the form and sends it to the server peer
C	3	Server peer invokes the Web service
C	4	The result of the invocation of the service is sent back to the relay peer
C	5	JXTA relay peer sends the result to the PDA peer
D	1	PDA peer invokes the Web service outside of the JXTA peer group
D	2	The result of the invocation of the service is sent back to the PDA peer

5.4.1 Publishing BPEL Web Services in JXTA Network

In JXTA, a service is published by an advertisement. A service is defined as a module and modules are published as advertisements, namely, Module Class Advertisement, Module Specification Advertisement and Module Implementation Advertisement. In Figure 23, the BPEL Web service and the JXTA module advertisements to which the Web service is mapped are shown.


IBM Business Process Execution Language for Web Services Java Runtime Admin Tool - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://127.0.0.1:8080/bpws4j/admin/index.html
IBM Business Process Execution Language for Web Services Java Runtime
Configure Processes
List
Deploy
Un-deploy
Done

Process {urn:Cerrahpasa:CerrahpasaBPEL}CerrahpasaHospitalServiceBP deployed:

External WSDL = [click here]
Channels:

- Apache SOAP
  SOAP Address: http://127.0.0.1:8080/bpws4j/soaprpcrouter
  SOAP Action URI:
  Method Namespace URIs:
  - urn:Cerrahpasa:CerrahpasaBPEL#CerrahpasaHospitalServiceBP#clie

Module Class Advertisement

```
<?xml version="1.0" encoding="UTF-8"?>
<jxta:MCA>
            <MCID> … </MCID>
            <Name> … </Name>
            <Desc> … </Desc>
</jxta:MCA>
```

Module Specification Advertisement

```
<?xml version="1.0" encoding="UTF-8"?>
<jxta:MSA>
            <MSID> … </MSID>
            <Name> … </Name>
            <Crtr> … </Crtr>
            <SURI> … </SURI>
            <Vers> … </Vers>
            <Desc> … </Desc>
            <Parm> … </Parm>
      <jxta:PipeAdv> … </jxta:PipeAdv>
            <Proxy> … </Proxy>
            <Auth> … </Auth>
</jxta:MSA>
```

Module Implementation Advertisement

```
<?xml version="1.0" encoding="UTF-8"?>
<jxta:MIA>
            <MSID> … </MSID>
            <Comp> … </Comp>
            <Code> … </Code>
            <PURI> … </PURI>
            <Prov> … </Prov>
            <Desc> … </Desc>
            <Parm> … </Parm>
</jxta:MIA>
```

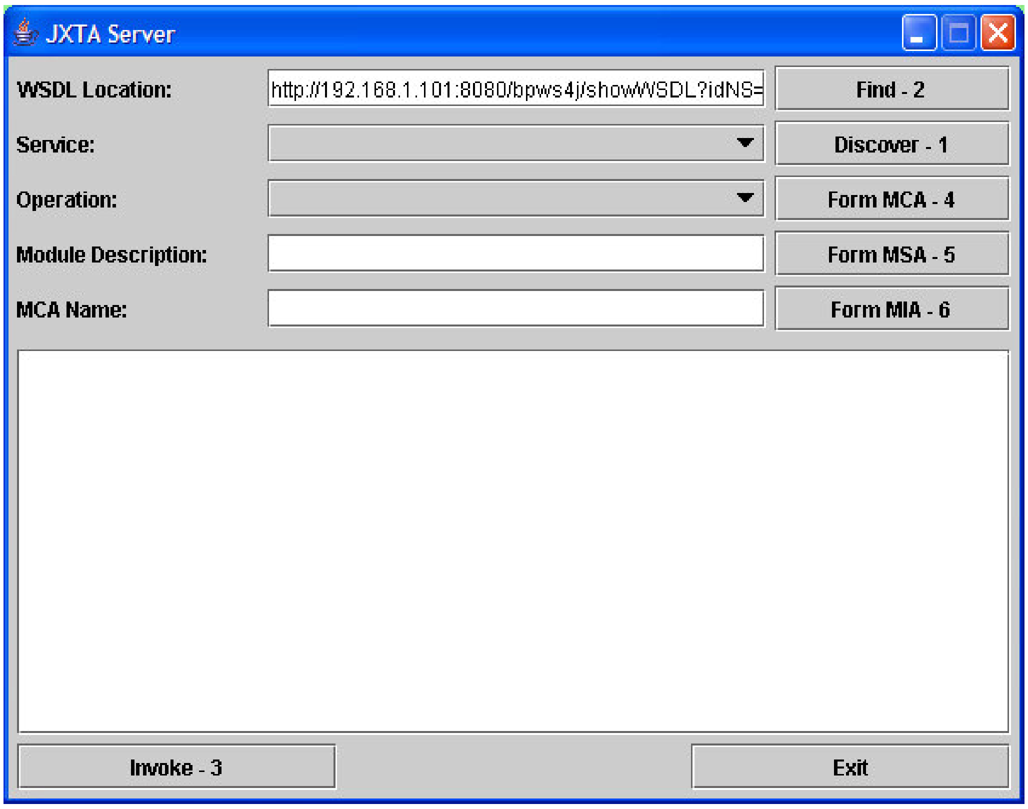
Figure 23: Mapping BPEL Web Services to JXTA Module Advertisements

First, the BPEL Web service is published as Module Class Advertisement (MCA), which defines a specific version of a module. The name of the service and a description of the service is defined with an ID (MCID) that uniquely defines the module.

Module Specification Advertisement (MSA) defines the module, parameters of the service (inputs, outputs, results) and the URI of the specification. An MCA is normally followed by a MSA. An MSA advertisement also contains name, description and ID (MSID) for uniquely identifying the module.

Module Implementation Advertisement defines specific instance of a module on a platform.

In Figure 24, an implementation of a publisher peer that maps BPEL Web services to JXTA module advertisements, is shown.

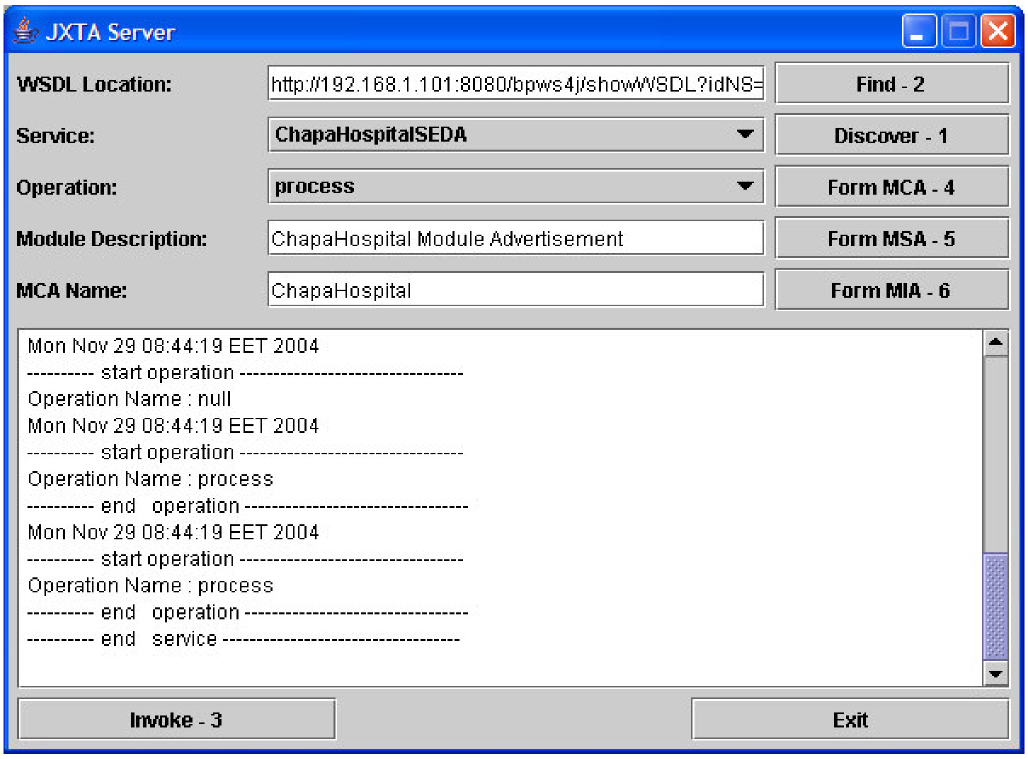


WSDL Location: http://192.168.1.101:8080/bpws4j/showWSDL?idNS=
Service:
Operation:
Module Description:
MCA Name:

Figure 24: GUI of Publisher Peer

The steps in publishing a BPEL Web service into JXTA module is as follows:

1. The discovery service of the advertiser peer is started by pressing the “Discover – 1” button,
2. By filling the “WSDL Location” of the BPEL Web service and pressing the “Find – 2” button, the services and the operations provided by the services is brought to the “Service” and “Operation” lists as in Figure 25.
3. After selecting the service and the operation provided the service, by pressing the “Invoke – 3” button, the input, inout [IN-OUT], output and result parameters of the BPEL service is found.



WSDL Location: http://192.168.1.101:8080/bpws4j/showWSDL?idNS=
Service: ChapaHospitalSEDA
Operation: process
Module Description: ChapaHospital Module Advertisement
MCA Name: ChapaHospital

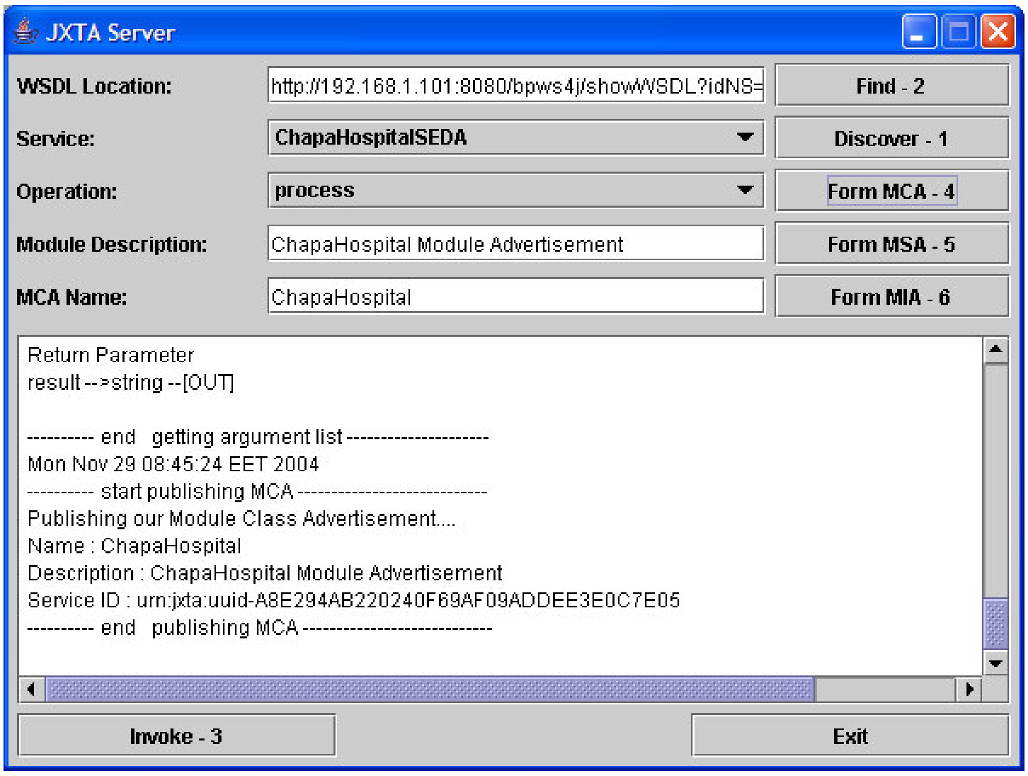
```
Mon Nov 29 08:44:19 EET 2004
---------- start operation --------------------------------
Operation Name : null
Mon Nov 29 08:44:19 EET 2004
---------- start operation --------------------------------
Operation Name : process
---------- end  operation --------------------------------
Mon Nov 29 08:44:19 EET 2004
---------- start operation --------------------------------
Operation Name : process
---------- end  operation --------------------------------
---------- end  service ----------------------------------
```

Figure 25: Finding the WSDL Services and Operations

4. Module Class Advertisement of the BPEL Service is prepared by "Form MCA – 4" button. The name and description of the MCA are written in "MCA Name" and "Module Description" fields. In Table 6 and Figure 26, the MCA of the "Chapa Hospital" BPEL Web service is given.
5. "Form MSA – 5" button is used for publishing the module specification advertisement of the BPEL Web service. The name and description of the MSA are taken from the "MCA Name" and "Module Description" fields. In publishing MSA, inputs, inouts, outputs and results are published in Parm field of the advertisement. Also the value of the "WSDL Location" field is used as the value of SURI (specification URI). In Table 7, the MSA of the "Chapa Hospital" BPEL Web service is given.
6. Finally, the advertisement of module implementation is published by pressing "Form MIA – 6" button.

Table 6: Module Class Advertisement (MCA) of "Chapa Hospital" BPEL Web Service

```
<!DOCTYPE jxta:MCA>
<jxta:MCA xmlns:jxta="http://jxta.org">
      <MCID>
            urn:jxta:uuid-EF12D26C163E44A2AD0BEA9456C3823F05
      </MCID>
      <Name>
            ChapaHospital
      </Name>
      <Desc>
            ChapaHospital Service
      </Desc>
</jxta:MCA>
```



WSDL Location:	http://192.168.1.101:8080/bpws4j/showWSDL?idNS=
Service:	ChapaHospitalSEDA
Operation:	process
Module Description:	ChapaHospital Module Advertisement
MCA Name:	ChapaHospital

```
Return Parameter
result-->string--[OUT]

--------- end   getting argument list --------------------
Mon Nov 29 08:45:24 EET 2004
--------- start publishing MCA ---------------------------
Publishing our Module Class Advertisement....
Name : ChapaHospital
Description : ChapaHospital Module Advertisement
Service ID : urn:jxta:uuid-A8E294AB220240F69AF09ADDEE3E0C7E05
--------- end   publishing MCA --------------------------
```

Figure 26: Publishing Module Class Advertisement

Table 7: Module Specification Advertisement (MSA) of "Chapa Hospital" BPEL Web Service

```
<!DOCTYPE jxta:MSA>
<jxta:MSA xmlns:jxta="http://jxta.org">
      <MSID>
            urn:jxta:uuid-
EF12D26C163E44A2AD0BEA9456C3823F4CFED52F14DE45A2BE8AB465BB69BFB806
      </MSID>
      <Name>
            ChapaHospital
      </Name>
      <Crtr>
            www.srdc.metu.edu.tr
      </Crtr>
      <SURI>

http://192.168.1.101:8080/bpws4j/showWSDL?idNS=urn:Capad:CapadService&amp;idNa
me=ChapaHospitalSEDA
      </SURI>
      <Vers>
            Version 1.0
      </Vers>
      <Desc>
            ChapaHospital Service
      </Desc>
      <Parm>
            <InputSize>
                  8
            </InputSize>
            <string>
                  EVN
            </string>
            <string>
                  MSH
            </string>
            <string>
                  PID
            </string>
            <string>
                  DB1
            </string>
            <string>
                  AL1
            </string>
            <string>
                  ACC
            </string>
            <string>
                  OBR
            </string>
            <string>
                  OBX
            </string>
            <InOutSize>
                  0
            </InOutSize>
            <OutputSize>
                  0
            </OutputSize>
            <ResultSize>
                  1
            </ResultSize>
            <string>
                  result
            </string>
      </Parm>
      <jxta:PipeAdvertisement xmlns:jxta="http://jxta.org">
            <Id>
                  urn:jxta:uuid-
59616261646162614E504720503250333BDEC3ABA324433BA6DFF59B8C9760CB04
            </Id>
            <Type>
                  JxtaUnicast
            </Type>
            <Name>
                  ChapaHospital
            </Name>
      </jxta:PipeAdvertisement>
</jxta:MSA>
```

5.4.2 Discovering BPEL Web Services Published in JXTA Network

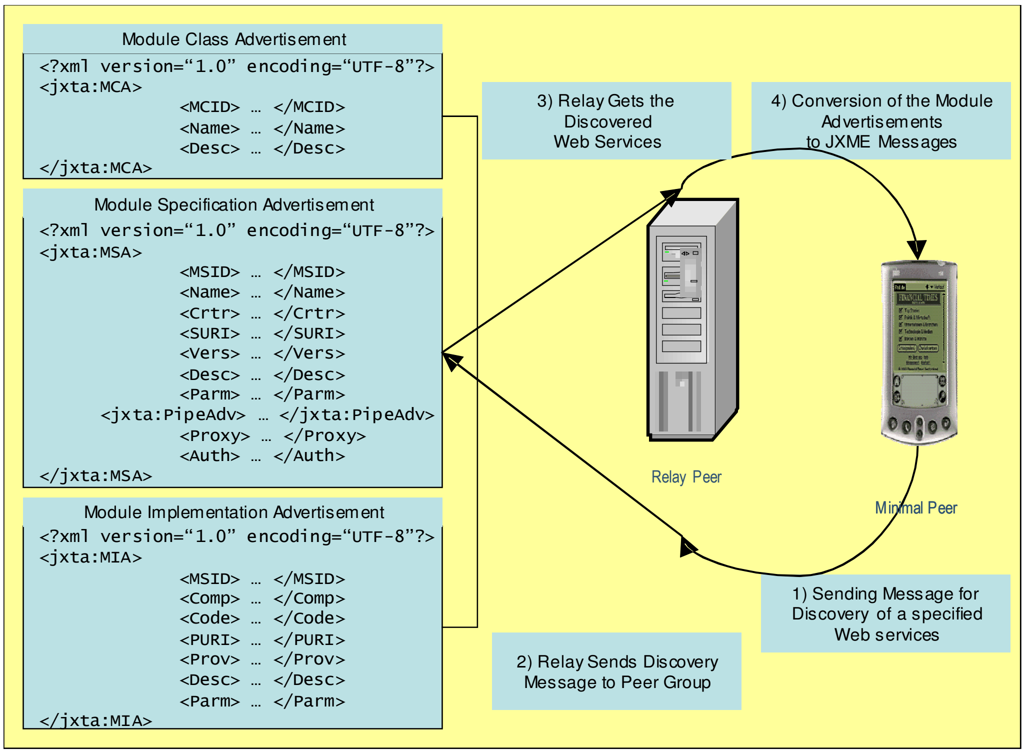
In Figure 23, in order for mobile devices to discover BPEL Web services from JXTA environment, BPEL Web services are mapped into module advertisements. By mapping Web services to advertisements:

- Web services invocation point (WSDL address)
- parameters and type of parameters of the Web services
- return type of Web services
- description of the services
- other module related parameters

are published as modules, so that a peer can discover the publication of Web services through module advertisements. Mobile devices can discover the services as a minimal peer by using JXTA relay peers. By sending and receiving messages, a mobile device can request discovery of a specific service from relay peer and get the module advertisements for the specified services. In Figure 27, discovery of a Web service is illustrated:

The steps for discovering a Web service published in a JXTA environment by a minimal peer (mobile device) is as follows:

1. The request message for discovery of a specific Web services is sent from minimal peer.
2. The relay peer gets the message and converts it to a module discovery message.
3. The module advertisement that is the result of the discovery message is acquired by the relay peer
4. The relay peer converts the module advertisement to a message format for minimal peer.


Module Class Advertisement
Module Specification Advertisement
Module Implementation Advertisement
3) Relay Gets the Discovered Web Services
4) Conversion of the Module Advertisements to JXME Messages
Relay Peer
Minimal Peer
1) Sending Message for Discovery of a specified Web services
2) Relay Sends Discovery Message to Peer Group

```
<?xml version="1.0" encoding="UTF-8"?>
<jxta:MCA>
              <MCID> … </MCID>
              <Name> … </Name>
              <Desc> … </Desc>
</jxta:MCA>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<jxta:MSA>
              <MSID> … </MSID>
              <Name> … </Name>
              <Crtr> … </Crtr>
              <SURI> … </SURI>
              <Vers> … </Vers>
              <Desc> … </Desc>
              <Parm> … </Parm>
      <jxta:PipeAdv> … </jxta:PipeAdv>
              <Proxy> … </Proxy>
              <Auth> … </Auth>
</jxta:MSA>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<jxta:MIA>
              <MSID> … </MSID>
              <Comp> … </Comp>
              <Code> … </Code>
              <PURI> … </PURI>
              <Prov> … </Prov>
              <Desc> … </Desc>
              <Parm> … </Parm>
</jxta:MIA>
```

Figure 27: Discovery of a Web Service through JXTA Relay

The BPEL Web services defined as JXTA modules and published as MCA, MSA, and MIA are discovered by using a protocol between relay peer and minimal peers (PDA). The protocol consists of the messages and response to these messages. The messaging part is consisting of two parts; messages sent by minimal peer and messages sent by relay peer. The following are the messages sent by minimal peers:

Limited Device (Minimal Peer [PDA])

- SEND : Used for messaging between minimal peers and relay peer.

Table 8: Elements of Message "SEND" for Minimal Peer

Element #	Element Name	Element Value
0	"JXTASENDER"	Identity name
1	"JXTAMESSAGETYPE"	"SEND"
2	"JXTAMESSAGE"	Message
3	"JXTAARGS"	"NetPeerGroup"

- FIND : Used for finding the specified keyword in the advertisements published by the relay peer.

Table 9: Elements of Message "FIND" for Minimal Peer

Element #	Element Name	Element Value
0	"JXTASENDER"	Identity name
1	"JXTAMESSAGETYPE"	“FIND”
2	"JXTAMESSAGE"	Keyword
3	"JXTAARGS"	"NetPeerGroup"

- GETW : Used for selecting the appropriate Web service from the list of Web services sent by relay peer.

Table 10: Elements of Message "GETW" for Minimal Peer

Element #	Element Name	Element Value
0	"JXTASENDER"	Identity name
1	"JXTAMESSAGETYPE"	“GETW”
2	"JXTAMESSAGE"	Name of the MSA of Web Service
3	"JXTAMESSAGE"	WSDL Location of the MSA Service
4	"JXTAARGS"	"NetPeerGroup"

- CALL* : Used for calling the Web service with the given parameters. (input, inout, output, result)

  * : Used when invoking the Web service through relay peer.

Table 11: Elements of Message "CALL" for Minimal Peer

Element #	Element Name	Element Value
0	"JXTASENDER"	Identity name
1	"JXTAMESSAGETYPE"	“CALL”
2	"JXTAARGS"	"NetPeerGroup"
3	"JXTAARGS"	Input size (# of Input) (INS)
4	"JXTAARGS"	Inout size (# of Inout) (IOS)
5	"JXTAARGS"	Output size (# of Output) (OUS)

Table 11 (Continued)

Element #	Element Name	Element Value
6	"JXTAARGS"	Result size (# of Result) (RES)
7 -> 7+INS-1	"JXTAARGS"	Input fields
7+INS -> 7+INS+IOS-1	"JXTAARGS"	Inout fields
7+INS+IOS -> 7+INS + IOS+OUS-1	"JXTAARGS"	Output fields
7+INS+IOS+ OUS -> 7+INS + IOS+OUS + RES-1	"JXTAARGS"	Result Fields
7+INS + IOS+OUS + RES	"JXTAARGS"	Name of the BPEL Service
7+INS + IOS+OUS + RES+1	"JXTAARGS"	WSDL Location of the BPEL Service

<u>**Web Service Provider Peer (Relay Peer)**</u>

- SEND : Used for messaging between minimal peers and relay peer.

Table 12: Elements of Message "SEND" for Relay Peer

Element #	Element Name	Element Value
0	"JXTASENDER"	"WSPROVIDER"
1	"JXTAMESSAGETYPE"	“SEND”
2	"JXTAMESSAGE"	Message
3	"JXTAARGS"	"NetPeerGroup"

- FIND : Used for sending the found Web services MSA advertisement to the minimal peer.

Table 13: Elements of Message "FIND" for Relay Peer

Element #	Element Name	Element Value
0	"JXTASENDER"	"WSPROVIDER"
1	"JXTAMESSAGETYPE"	"FIND"
2	"JXTAARGS"	Size of Found List (FOS)
3-> 3+FOS-1	"JXTAARGS"	Found List

- GETW : Used for sending the parameter list (input, inout, output, result list) to the minimal peer.

Table 14: Elements of Message "GETW" for Relay Peer

Element #	Element Name	Element Value
0	"JXTASENDER"	"WSPROVIDER"
1	"JXTAMESSAGETYPE"	"GETW"
2	"JXTAMESSAGE"	Name of the MSA of Web Service
3	"JXTAARGS"	Input size (# of Input) (INS)
4	"JXTAARGS"	Inout size (# of Inout) (IOS)
5	"JXTAARGS"	Output size (# of Output) (OUS)
6	"JXTAARGS"	Result size (# of Result) (RES)
7 -> 7+INS-1	"JXTAARGS"	Input fields
7+INS -> 7+INS+IOS-1	"JXTAARGS"	Inout fields
7+INS+IOS -> 7+INS + IOS+OUS-1	"JXTAARGS"	Output fields
7+INS+IOS+ OUS -> 7+INS + IOS+OUS + RES-1	"JXTAARGS"	Result Fields

Table 14 (Continued)

Element #	Element Name	Element Value
7+INS + IOS+OUS + RES	"JXTAARGS"	Name of the BPEL Service
7+INS + IOS+OUS + RES+1	"JXTAARGS"	WSDL Location of the BPEL Service

- CALL* : Used for sending the result of invocation of the BPEL Web service.
  * : Used when invoking the Web service through relay peer.

Table 15: Elements of Message "CALL" for Relay Peer

Element #	Element Name	Element Value
0	"JXTASENDER"	Identity name
1	"JXTAMESSAGETYPE"	"CALL"
2	"JXTAARGS"	"NetPeerGroup"
3	"JXTAARGS"	Result

Other key issue in this thesis is the invocation of the Web services that are discovered by the minimal peer. There are two ways for invoking the Web service; the first one is invoking the Web service using the parameters of the Web service discovery message sent by relay peer, and the other one is preparing a invocation message and sending it to relay peer for invocation in relay peer or the in the peer which publishes the Web service. Facilitating the peer that publishes the Web service allows using this peer as a relay in invocation of Web services. Figure 28 and Figure 35 illustrate invocation of services through a relay peer and without a relay peer.

5.4.3 Invocation of BPEL Web Services from Mobile Devices through Relay Peer

There are 2 methods for invocation of BPEL Web services published in JXTA network. In both methods, the discovery of Web services, searching from the relay peer and minimal peer, selecting the BPEL Web service and getting the parameter list from the relay peer are same as in Figure 29, Figure 30, Figure 31 and Figure 32. The first method is invocation of the services via relay peer. Figure 28 depicts the first method.

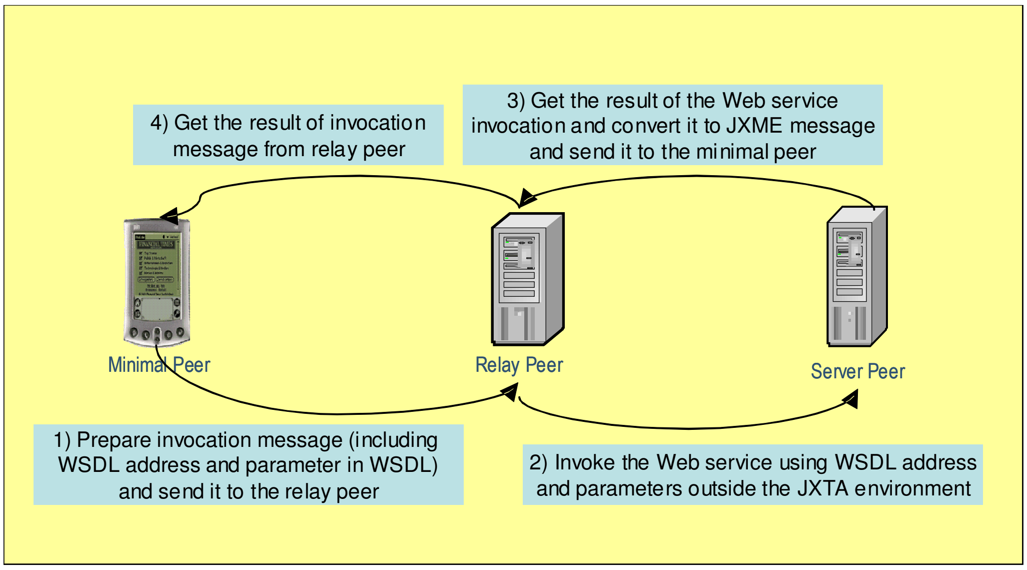
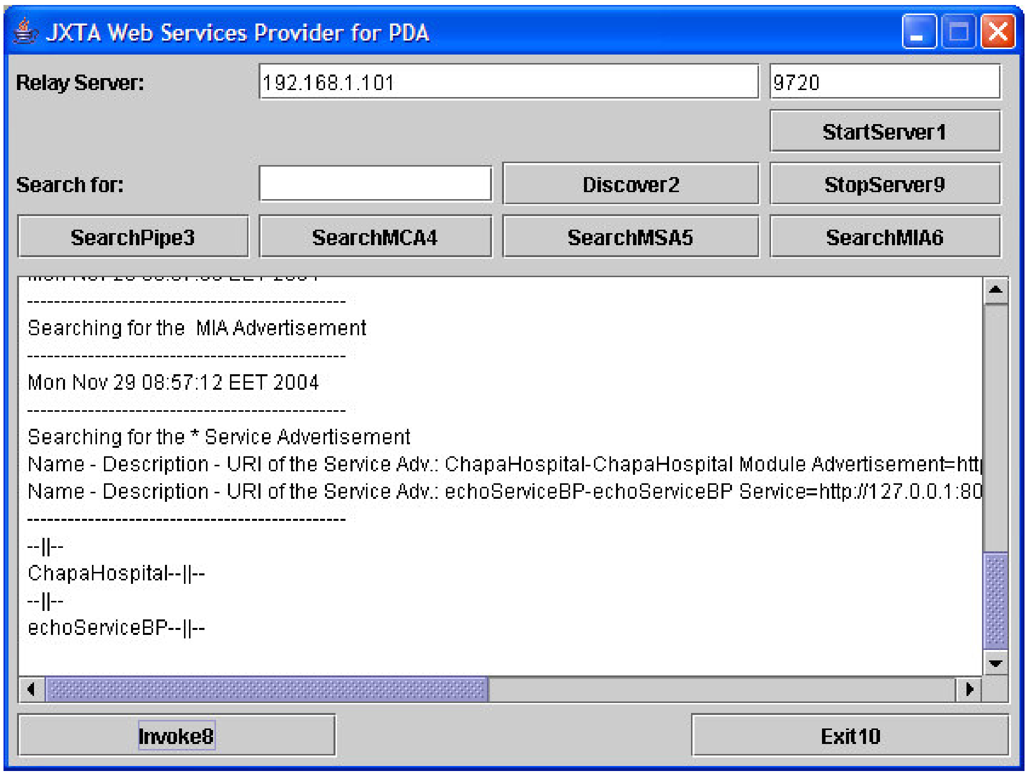

4) Get the result of invocation message from relay peer
3) Get the result of the Web service invocation and convert it to JXME message and send it to the minimal peer
Minimal Peer
Relay Peer
Server Peer
1) Prepare invocation message (including WSDL address and parameter in WSDL) and send it to the relay peer
2) Invoke the Web service using WSDL address and parameters outside the JXTA environment

Figure 28: Invocation of Web Services using Relay Peer

In order to discover a BPEL Web service, first the relay peer discovers the MSA's from the other neighboring peers or rendezvous peers. For this purposes, the following steps should be followed:

1. First, the Web Services Provider for PDA functionality is started by pressing "StartServer1". This server listens the messages sent to the relay peer and processes the messages and take the necessary actions on behalf of the minimal peer.
2. The discovery service of the server is started.
3. Pipe advertisements are searched.

4. Module class advertisements are searched by pressing “SearchMCA4” button.
5. Module specification advertisements are searched by pressing “SearchMSA5” button.
6. Module implementation advertisements are searched by pressing “SearchMIA6” button.
7. By pressing “Invoke8” button, the MSA advertisements are prepared for the minimal peers.



```
--------------------------------------------
Searching for the  MIA Advertisement
--------------------------------------------
Mon Nov 29 08:57:12 EET 2004
--------------------------------------------
Searching for the * Service Advertisement
Name - Description - URI of the Service Adv.: ChapaHospital-ChapaHospital Module Advertisement=htt
Name - Description - URI of the Service Adv.: echoServiceBP-echoServiceBP Service=http://127.0.0.1:80
--------------------------------------------
--||--
ChapaHospital--||--
--||--
echoServiceBP--||--
```

Figure 29: Discovering the MSA from the Relay Peer

In Figure 30, the main option list provided in a minimal peer is depicted. “Send” and “Reply” options are used for messaging between minimal peers and with relay peer. “Find” is used for searching a BPEL Web service. “Configuration” option is used for configuring the address of the relay peer and identity name of the minimal peer.

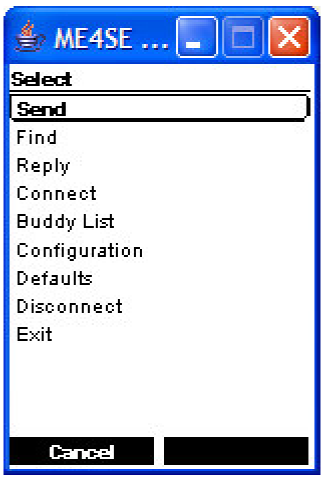


Figure 30: Main Option List of Minimal Peer (PDA).

("Find" is chosen for searching advertisements of BPEL Web services)

In Figure 31, the BPEL Web services those having corresponding MSA advertisements are listed in minimal peer. The listed Web services also meets the criteria that is entered in "Find" option.

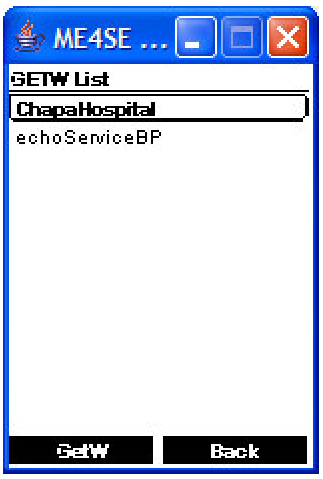
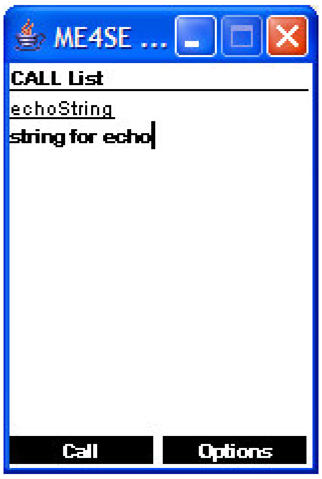


Figure 31: Selecting the BPEL Web services

In Figure 32, the parameter list (input, inout) of the Web service selected in Figure 31, is sent to the minimal peer. And in Figure 33, the first option, "Call" is selected for invoking the selected Web service via relay peer. Finally, Figure 34 denotes the result of the invocation.



string for echo

Figure 32: Parameter List sent by the Relay Peer is filled

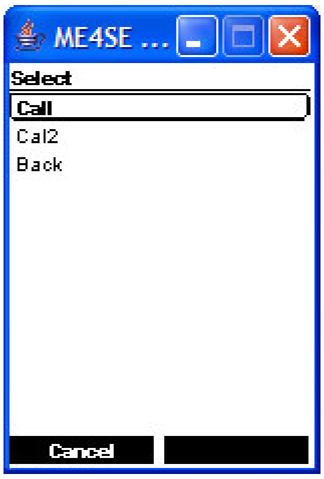
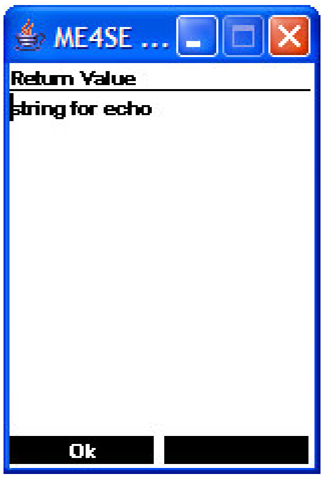


Figure 33: Invocation Option List



string for echo

Figure 34: Result of the Invocation

5.4.4 Direct Invocation of BPEL Web Services from Mobile Devices (without Relay Peer)

The other method for invoking the BPEL Web service, defined in a JXTA module and advertised as MCA and MSA, is invoking the Web service directly from the minimal peer. The difference between this invocation and the other invocation is that this invocation scheme does not use JXTA network in invocation. Figure 35 shows the direct invocation scheme. In order to invoke directly, when calling the Web service, "Cal2" option should be used in Figure 33.

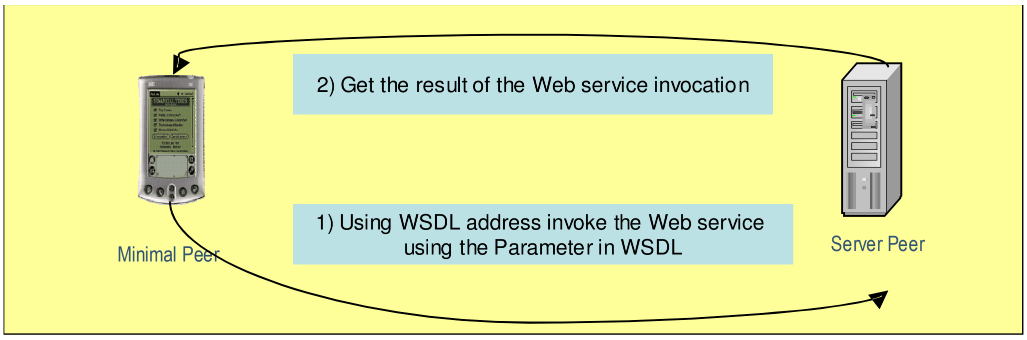

2) Get the result of the Web service invocation
1) Using WSDL address invoke the Web service using the Parameter in WSDL
Minimal Peer
Server Peer

Figure 35: Invocation of Web Services without Relay Peer

CHAPTER 6

CONCLUSIONS

With the increasing popularity of Peer-to-Peer (P2P) computing, P2P technology promises notable progress of spreading towards all devices including the devices only with digital heartbeat.

Mobile devices are expected to become more common in the P2P networks, which is one of the core aims of P2P computing and the Project JXTA. Besides, mobile devices, like personal digital assistants (PDAs), have already started to take an active role in everyday life and to become more widespread.

The Web services technology has gained much popularity, particularly having acquired the great support from industry. Thus, it has been inevitable to accommodate the Web services technology within P2P computing technology, which has also been an effort of the Project JXTA and this thesis work.

This work presents an infrastructure to realize the development of applications that reside on mobile devices and to implement JXTA protocols to make them behave as JXTA peers within the proposed architecture, and the invocation of Web services that reside on machines different from mobile devices but similarly behaving as JXTA peers conforming to the JXTA protocols. Thus, the motivation of this work is to bring the P2P and Web services technologies together for mobile devices, particularly PDAs, by exploiting the JXTA framework for the implementation of P2P environment.

The main question that is investigated for applying the technologies mentioned above and adapting the architecture proposed in this work to mobile devices emerges from the fact that, currently, there exists no standard P2P system implementation for mobile devices. That is, even the JXTA framework does not introduce a solution for this concern for mobile devices. Within this work, in order to overcome this concern, JXTA for J2ME (JXME) has been exploited.

The work done in this thesis exploits three main technologies that gain popularity in recent years, namely P2P computing, Web services and mobile computing, and combine them to provide an environment for mobile devices. In order to realize this, various standards and tools are exploited on these technologies. Since these technologies are relatively new and different platforms are used, many problems appear and these problems are briefly explained as follows:

- Because of the limited display and memory capability of mobile devices, in case of any exception or problem in applications deployed on these devices, the problem can not be easily found and fixed.
- Since BPEL4WS language consists of various set of libraries and tools, for exception and problems that occur on a Web service invocation from mobile device or application, details of the underlying set of libraries and tools have been discovered.

In order to prove the realization of the system structure as a whole, a complete scenario is developed based on the components of the system architecture. This enables us to see the whole system architecture providing mobile devices with the capability to join a P2P environment and facilitate the services published in the environment.

The work designed and developed in this thesis is a part of IST-1-002103-STP Artemis project which is funded by European Commission. The contribution of the work is the architecture designed to provide mobile devices with the services published in a P2P environment.

REFERENCES

1: Bussler, C., Fensel, D., Maedche, A., A Conceptual Architecture for Semantic Web Enabled Web Services, http://lsdis.cs.uga.edu/SemNSF/SIGMOD-Record-Dec02/Bussler.pdf

2: A. Dogac, G. Laleci, S. Kirbas, Y. Kabak, S. Sinir, A. Yildiz, Artemis: Deploying Semanticaly Enriched Web Services in the Healthcare Domain, http://www.srdc.metu.edu.tr/webpage/projects/artemis/publications/_Dogac_InfSys04.pdf

3: Health Level 7 (HL7), http://www.hl7.org

4: CEN TC/251 (European Standardization of Health Informatics) ENV 13606, Electronic Health Record Communication, http://www.centc251.org

5: ISO TC/215, International Organization for Standardization, Health Informatics Technical Committee, http://www.iso.ch/iso/en/stdsdevelopmenttc/tclist/TechnicalCommitteeDetailPage.TechnicalCommitteeDetail?COMMID=4720

6: The Good Electronic Health Record, http://www.gehr.org

7: Nelson Minar, Distributed Systems Topologies: Part 1, http://www.openp2p.com/pub/a/p2p/2001/12/14/topologies_one.html

8: Project JXTA: Java[TM] Programmers Guide, http://www.jxta.org/docs/JxtaProgGuide_v2.pdf

9: Gong, Li, A Network Programming Environment, http://www.jxta.org/project/www/docs/JXTAnetworkProgEnv.pdf

10: Web Services Glossary, http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/

11: Brendon J. Wilson, Projects::JXTA Book, http://www.brendonwilson.com/projects/jxta/

12: Web Services Glossary, http://www.w3.org/TR/ws-gloss/

13: Web Services - The Web's next revolution, http://www6.software.ibm.com/developerworks/education/wsbasics/wsbasics-ltr.pdf

14: World Wide Web Consortium (W3C) XML Standard, http://www.w3.org/XML/

15: Web Services Description Language (WSDL) 1.1, http://www.w3.org/TR/wsdl

16: Simple Objet Access Protocol (SOAP), http://www.w3c.org/TR/SOAP/

17: The architecture of Web Service Invocation Framework (WSIF), http://www-106.ibm.com/developerworks/webservices/library/ws-wsif.html

18: JROM (Java Record Object Model), http://www.alphaworks.ibm.com/tech/jrom

19: Mukhi, N., Khalaf, R., Fremantle, P., Multi-protocol Web Services for enterprises and the Grid, http://ewic.bcs.org/conferences/2002/euroweb/session5/paper2.pdf

20: Java Technology, http://java.sun.com/

21: Java 2 Platform Standard Edition (J2SE), http://java.sun.com/j2se/

22: Java 2 Platform Enterprise Edition (J2EE), http://java.sun.com/j2ee/

23: Java 2 Platform Micro Edition (J2ME), http://java.sun.com/j2me/

24: Java Community Process (JCP), http://jcp.org/

25: Connected Limited Device Configuration (CLDC), http://java.sun.com/products/cldc

26: Connected Device Configuration (CDC), http://java.sun.com/products/cdc/

27: Mobile Information Device Profile (MIDP), http://java.sun.com/products/midp/

28: Insignia Solutions, http://www.insignia.com/

29: ME4SE, http://me4se.org/me4se.html

30: IBM Business Process Execution Language for Web Services Java Run Time (BPWS4J), http://www.alphaworks.ibm.com/tech/bpws4j

31: kXML, http://kxml.objectweb.org/

32: kSOAP, http://ksoap.objectweb.org/

33: Wiederhold, G., Mediators in the Architecture of Future Information Systems, IEEEComputer, Vol.25 No.3, March 1992, IEEEComputer, Vol.25 No.3, March 1992.

34: Maedche, A., Motik, D., Silva, N., Volz, R., MAFRA-A MApping FRAmework for Distributed Ontologies, In Proc. of the 13th European Conf. on Knowledge Engineering and Knowledge Management EKAW-2002, Madrid, Spain, 2002, http://www.sklse.org/group/uml/ebXML/2004.09.17/MAFRA%20-%20A%20MApping%20FRAmework%20for%20Distributed%20Ontologies%20in%20the%20Semantic%20Web(ECAI-WS'2002).pdf

35: Burton, Kevin A., Bridge, http://soap.jxta.org/servlets/ProjectHome

36: Artemis Project, http://www.srdc.metu.edu.tr/webpage/projects/artemis

37: SRDC Team, The StoryBoard of Web Services Orchestration Demo with Collaxa BPEL Server, http://www.srdc.metu.edu.tr/webpage/projects/artemis

APPENDIX A

A Simple Client for Mobile Device That Invokes Web Services

ArtMobileMIDlet.java

```
//*****************************************************************


package tr.edu.srdc;

import java.lang.*;
import java.util.Hashtable;
import javax.microedition.lcdui.*;
import javax.microedition.midlet.MIDlet;

//*****************************************************************

public class ArtMobileMIDlet extends MIDlet implements CommandListener {

  public static void main (String[] args) {
    ArtMobileMIDlet tArtMobileMIDlet = new ArtMobileMIDlet();
  }

  public ArtMobileMIDlet () {
    display       = Display.getDisplay (this);

        mainScreen    = new Form ("Çapa BPEL Form");
    invokeCommand = new Command ("Invoke", Command.SCREEN, 1);
    exitCommand   = new Command ("Exit", Command.EXIT, 2);

    tfEVN = new TextField ("Event Information","",6,0);
    tfMSH = new TextField ("Message Header","",6,0);
    tfPID = new TextField ("Patient Identification","",6,0);
    tfDB1 = new TextField ("Disability Information","",6,0);
    tfAL1 = new TextField ("Allergy Information","",6,0);
    tfACC = new TextField ("Accident Information","",6,0);
    tfOBR = new TextField ("Observation Request","",6,0);
    tfOBX = new TextField ("Observation Result","",6,0);
    stRET = new StringItem ("Returned String:","");

    mainScreen.append (tfEVN);
    mainScreen.append (tfMSH);
    mainScreen.append (tfPID);
    mainScreen.append (tfDB1);
    mainScreen.append (tfAL1);
    mainScreen.append (tfACC);
    mainScreen.append (tfOBR);
    mainScreen.append (tfOBX);

    mainScreen.addCommand (invokeCommand);
    mainScreen.addCommand (exitCommand);

    options = new List ("Options", 3, mainOptions, null);
    options.addCommand (invokeCommand);
```

```
        startApp ();
    }

    public void commandAction(Command command, Displayable displayable) {
        if (command.getCommandType() == Command.EXIT) {
            destroyApp (true);
            notifyDestroyed ();
            return;
        }
        try {
                                hshInput = new Hashtable ();
                                hshInput.put ("EVN", tfEVN.getString());
                                hshInput.put ("MSH", tfMSH.getString());
                                hshInput.put ("PID", tfPID.getString());
                                hshInput.put ("DB1", tfDB1.getString());
                                hshInput.put ("AL1", tfAL1.getString());
                                hshInput.put ("ACC", tfACC.getString());
                                hshInput.put ("OBR", tfOBR.getString());
                                hshInput.put ("OBX", tfOBX.getString());

                                InvokeArtMob tInvokeArtMob = new InvokeArtMob();

                        String tRetVal = tInvokeArtMob.process("http://ugimbtp3:8080/bpws4j/soaprpcrouter",
hshInput);

                        stRET = new StringItem("Returned String:",tRetVal);
                        mainScreen.append(stRET);
        } catch (Exception e) {
                        e.printStackTrace();
        }
    }

    public void destroyApp (boolean flag) {
    }

    public void pauseApp () {
    }

    public void startApp () {
        mainScreen.setCommandListener(this);
        display.setCurrent(mainScreen);
    }

    private Display display;
    private Form mainScreen;
    private List options;
    private Command invokeCommand;
    private Command exitCommand;
    private String mainOptions[] = { };
    private TextField tfEVN;
    private TextField tfMSH;
    private TextField tfPID;
    private TextField tfDB1;
    private TextField tfAL1;
    private TextField tfACC;
    private TextField tfOBR;
    private TextField tfOBX;
    private StringItem stRET;
    private Hashtable hshInput;
}

//*****************************************************************
```

APPENDIX B

WSDL Documents of Artemis Web Service

The WSDL documents given in this appendix are described for the Web Service Orchestration Demo of Artemis Project [2][37][36] and deployed for Web service invocation from mobile devices.

ChapaHospital.wsdl

```
<definitions targetNamespace="urn:Capad:CapadService"
        xmlns:tns="urn:Capad:CapadService"
        xmlns:slt="http://schemas.xmlsoap.org/ws/2002/07/service-link/"
        xmlns="http://schemas.xmlsoap.org/wsdl/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema">

        <message name="ChapaHospitalRequestMessage">
                <part name="EVN" type="xsd:string"/>
                <part name="MSH" type="xsd:string"/>
                <part name="PID" type="xsd:string"/>
                <part name="DB1" type="xsd:string"/>
                <part name="AL1" type="xsd:string"/>
                <part name="ACC" type="xsd:string"/>
                <part name="OBR" type="xsd:string"/>
                <part name="OBX" type="xsd:string"/>
        </message>
        <message name="ChapaHospitalResponseMessage">
                <part name="result" type="xsd:string"/>
        </message>

        <portType name="ChapaPT">
                <operation name="process" parameterOrder="EVN MSH PID DB1 AL1 ACC OBR
OBX">
                        <input  message="tns:ChapaHospitalRequestMessage"/>
                        <output message="tns:ChapaHospitalResponseMessage"/>
                </operation>
        </portType>

        <slt:serviceLinkType name="ChapaHospitalSLT">
                <slt:role name="service">
                        <slt:portType name="tns:ChapaPT"/>
                </slt:role>
        </slt:serviceLinkType>
        <service name="ChapaHospital">
        </service>
</definitions>
```

CerrahpasaHospital.wsdl

```
<?xml version="1.0" encoding="UTF-8" ?>
<definitions targetNamespace="urn:Cerrahpasa:CerrahpasaBPEL"
        xmlns:tns="urn:Cerrahpasa:CerrahpasaBPEL"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:slt="http://schemas.xmlsoap.org/ws/2002/07/service-link/"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
        xmlns="http://schemas.xmlsoap.org/wsdl/">

        <message name="CerrahpasaResultMessage">
                <part name="result" type="xsd:string" />
        </message>
        <message name="CerrahpasaRequestMessage">
                <part name="msh" type="xsd:string" />
                <part name="pid" type="xsd:string" />
                <part name="al1" type="xsd:string" />
                <part name="obr" type="xsd:string" />
                <part name="obx" type="xsd:string" />
        </message>
        <portType name="Cerrahpasa">
                <operation name="initiate">
                        <input message="tns:CerrahpasaRequestMessage" />
                        <output message="tns:CerrahpasaResultMessage" />
                </operation>
        </portType>
        <binding name="clientCerrahpasaApacheSOAPSOAPBinding" type="tns:Cerrahpasa">
                <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
                <operation name="initiate">
                        <soap:operation soapAction="" style="rpc" />
                        <input>
                                <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="urn:Cerrahpasa:CerrahpasaBPEL#CerrahpasaHospitalServiceBP#client#urn:Cerrahpasa:Cerrah
pasaBPEL#Cerrahpasa" />
                        </input>
                        <output>
                                <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="urn:Cerrahpasa:CerrahpasaBPEL#CerrahpasaHospitalServiceBP#client#urn:Cerrahpasa:Cerrah
pasaBPEL#Cerrahpasa" />
                        </output>
                </operation>
        </binding>
        <service name="CerrahpasaHospitalServiceBP">
                <port name="clientCerrahpasaApacheSOAPSOAPBindingPort"
binding="tns:clientCerrahpasaApacheSOAPSOAPBinding">
                        <soap:address location="http://localhost:8080/bpws4j/soaprpcrouter" />
                </port>
        </service>
        <slt:serviceLinkType xmlns:slt="http://schemas.xmlsoap.org/ws/2002/07/service-link/"
name="CerrahpasaSLT">
                <slt:role name="CerrahpasaProvider">
                        <slt:portType name="tns:Cerrahpasa" />
                </slt:role>
        </slt:serviceLinkType>
</definitions>
```

AdmitVisit.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="MyAdmitVisit"
        targetNamespace="urn:Foo"
        xmlns:tns="urn:Foo"
        xmlns="http://schemas.xmlsoap.org/wsdl/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">

        <types/>

        <message name="AdmitVisit_admit">
                <part name="String_1" type="xsd:string"/>
                <part name="String_2" type="xsd:string"/>
                <part name="String_3" type="xsd:string"/>
                <part name="String_4" type="xsd:string"/>
                <part name="String_5" type="xsd:string"/>
                <part name="String_6" type="xsd:string"/>
        </message>
        <message name="AdmitVisit_admitResponse">
                <part name="result" type="xsd:string"/>
        </message>

        <portType name="AdmitVisit">
        <operation name="admit" parameterOrder="String_1 String_2 String_3 String_4 String_5
String_6">
        <input message="tns:AdmitVisit_admit"/>
        <output message="tns:AdmitVisit_admitResponse"/>
        </operation>
        </portType>

        <binding name="AdmitVisitBinding" type="tns:AdmitVisit">
                <operation name="admit">
                        <input>
                                <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
use="encoded" namespace="urn:Foo"/>
                        </input>
                        <output>
                                <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
use="encoded" namespace="urn:Foo"/>
                        </output>
                        <soap:operation soapAction=""/>
                </operation>
                <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
        </binding>

        <service name="MyAdmitVisit">
                <port name="AdmitVisitPort" binding="tns:AdmitVisitBinding">
                        <soap:address location="http://ugimbtp3:1024/admit/admitpatient"/>
                </port>
        </service>
</definitions>
```

Unsolicited.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions
        name="MyUnsolicited"
        targetNamespace="urn:Foo"
        xmlns:tns="urn:Foo"
        xmlns="http://schemas.xmlsoap.org/wsdl/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">

        <types/>

        <message name="Unsolicited_transmission">
                <part name="String_1" type="xsd:string"/>
                <part name="String_2" type="xsd:string"/>
                <part name="String_3" type="xsd:string"/>
                <part name="String_4" type="xsd:string"/>
        </message>
        <message name="Unsolicited_transmissionResponse">
                <part name="result" type="xsd:string"/>
        </message>

        <portType name="Unsolicited">
                <operation name="transmission" parameterOrder="String_1 String_2 String_3 String_4">
                        <input message="tns:Unsolicited_transmission"/>
                        <output message="tns:Unsolicited_transmissionResponse"/>
                </operation>
        </portType>

        <binding name="UnsolicitedBinding" type="tns:Unsolicited">
                <operation name="transmission">
                        <input>
                                <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
use="encoded" namespace="urn:Foo"/>
                        </input>
                        <output>
                                <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
use="encoded" namespace="urn:Foo"/>
                        </output>
                        <soap:operation soapAction=""/>
                </operation>
                <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
        </binding>

        <service name="MyUnsolicited">
                <port name="UnsolicitedPort" binding="tns:UnsolicitedBinding">
                <soap:address location="http://ugimbtp3:1024/unsolicited/unsolinfo"/>
                </port>
        </service>
</definitions>
```

Cerrahpasa.wsdl

```
<definitions targetNamespace="urn:Cerrahpasa:CerrahpasaBPEL"
        xmlns:tns="urn:Cerrahpasa:CerrahpasaBPEL"
        xmlns:slt="http://schemas.xmlsoap.org/ws/2002/07/service-link/"
        xmlns="http://schemas.xmlsoap.org/wsdl/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema">

        <message name="CerrahpasaRequestMessage">
                <part name="msh" type="xsd:string" />
                <part name="pid" type="xsd:string" />
                <part name="al1" type="xsd:string" />
                <part name="obr" type="xsd:string" />
                <part name="obx" type="xsd:string" />
        </message>

        <message name="CerrahpasaResultMessage">
                <part name="result" type="xsd:string"/>
        </message>


        <portType name="Cerrahpasa">
                <operation name="initiate">
                        <input message="tns:CerrahpasaRequestMessage"/>
                        <output message="tns:CerrahpasaResultMessage"/>
                </operation>
        </portType>


        <slt:serviceLinkType name="CerrahpasaSLT">
                <slt:role name="CerrahpasaProvider">
                <slt:portType name="tns:Cerrahpasa"/>
                </slt:role>
        </slt:serviceLinkType>

        <service name="CerrahpasaHospitalServiceBP">
        </service>
</definitions>
```

OrderEntry.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions
        name="MyOrderEntry"
        targetNamespace="urn:Foo"
        xmlns:tns="urn:Foo"
        xmlns="http://schemas.xmlsoap.org/wsdl/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">

        <types/>

        <message name="OrderEntry_order">
                <part name="String_1" type="xsd:string"/>
                <part name="String_2" type="xsd:string"/>
                <part name="String_3" type="xsd:string"/>
                <part name="String_4" type="xsd:string"/>
                <part name="String_5" type="xsd:string"/>
        </message>
        <message name="OrderEntry_orderResponse">
                <part name="result" type="xsd:string"/>
        </message>

        <portType name="OrderEntry">
                <operation name="order" parameterOrder="String_1 String_2 String_3 String_4
String_5">
                        <input message="tns:OrderEntry_order"/>
                        <output message="tns:OrderEntry_orderResponse"/>
                </operation>
        </portType>

        <binding name="OrderEntryBinding" type="tns:OrderEntry">
                <operation name="order">
                        <input>
                                <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
use="encoded" namespace="urn:Foo"/>
                        </input>
                        <output>
                                <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
use="encoded" namespace="urn:Foo"/>
                        </output>
                        <soap:operation soapAction=""/>
                </operation>
                <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
        </binding>

        <service name="MyOrderEntry">
                <port name="OrderEntryPort" binding="tns:OrderEntryBinding">
                        <soap:address location="http://ugimbtp3:1024/lab/order"/>
                </port>
        </service>
</definitions>
```

Unsolicited2.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="MyUnsolicited" targetNamespace="urn:Foo" xmlns:tns="urn:Foo"
xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
	<types/>

	<message name="Unsolicited_transmission">
		<part name="String_1" type="xsd:string"/>
		<part name="String_2" type="xsd:string"/>
		<part name="String_3" type="xsd:string"/>
		<part name="String_4" type="xsd:string"/>
	</message>

	<message name="Unsolicited_transmissionResponse">
		<part name="result" type="xsd:string"/>
	</message>

	<portType name="Unsolicited">
		<operation name="transmission" parameterOrder="String_1 String_2 String_3 String_4">
			<input message="tns:Unsolicited_transmission"/>
			<output message="tns:Unsolicited_transmissionResponse"/>
		</operation>
	</portType>

	<binding name="UnsolicitedBinding" type="tns:Unsolicited">
		<operation name="transmission">
			<input>
				<soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
use="encoded" namespace="urn:Foo"/>
			</input>
			<output>
				<soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
use="encoded" namespace="urn:Foo"/>
			</output>
			<soap:operation soapAction=""/>
		</operation>
		<soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
	</binding>

	<service name="MyUnsolicited">
		<port name="UnsolicitedPort" binding="tns:UnsolicitedBinding">
			<soap:address location="http://ugimbtp3:1024/unsol1/soli1"/>
		</port>
	</service>
</definitions>
```

Unsolicited3.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions
        name="MyUnsolicited"
        targetNamespace="urn:Foo"
        xmlns:tns="urn:Foo"
        xmlns="http://schemas.xmlsoap.org/wsdl/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">

        <types/>

        <message name="Unsolicited_transmission">
                <part name="String_1" type="xsd:string"/>
                <part name="String_2" type="xsd:string"/>
                <part name="String_3" type="xsd:string"/>
                <part name="String_4" type="xsd:string"/>
        </message>
        <message name="Unsolicited_transmissionResponse">
                <part name="result" type="xsd:string"/>
        </message>

        <portType name="Unsolicited">
                <operation name="transmission" parameterOrder="String_1 String_2 String_3 String_4">
                        <input message="tns:Unsolicited_transmission"/>
                        <output message="tns:Unsolicited_transmissionResponse"/>
                </operation>
        </portType>

        <binding name="UnsolicitedBinding" type="tns:Unsolicited">
                <operation name="transmission">
                        <input>
                                <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
use="encoded" namespace="urn:Foo"/>
                        </input>
                        <output>
                                <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
use="encoded" namespace="urn:Foo"/>
                        </output>
                        <soap:operation soapAction=""/>
                </operation>
                <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
        </binding>

        <service name="MyUnsolicited">
                <port name="UnsolicitedPort" binding="tns:UnsolicitedBinding">
                        <soap:address location="http://ugimbtp3:1024/unsol2/soli2"/>
                </port>
        </service>
</definitions>
```

Scheduling.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions
	name="MyScheduling"
	targetNamespace="urn:Foo"
	xmlns:tns="urn:Foo"
	xmlns="http://schemas.xmlsoap.org/wsdl/"
	xmlns:xsd="http://www.w3.org/2001/XMLSchema"
	xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">

	<types/>

	<message name="Scheduling_schedule">
		<part name="String_1" type="xsd:string"/>
		<part name="String_2" type="xsd:string"/>
		<part name="String_3" type="xsd:string"/>
		<part name="String_4" type="xsd:string"/>
		<part name="String_5" type="xsd:string"/>
		<part name="String_6" type="xsd:string"/>
	</message>
	<message name="Scheduling_scheduleResponse">
		<part name="result" type="xsd:string"/>
	</message>

	<portType name="Scheduling">
		<operation name="schedule" parameterOrder="String_1 String_2 String_3 String_4
String_5 String_6">
			<input message="tns:Scheduling_schedule"/>
			<output message="tns:Scheduling_scheduleResponse"/>
		</operation>
	</portType>

	<binding name="SchedulingBinding" type="tns:Scheduling">
		<operation name="schedule">
			<input>
				<soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
use="encoded" namespace="urn:Foo"/>
			</input>
			<output>
				<soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
use="encoded" namespace="urn:Foo"/>
			</output>
			<soap:operation soapAction=""/>
		</operation>
		<soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
	</binding>

	<service name="MyScheduling">
		<port name="SchedulingPort" binding="tns:SchedulingBinding">
			<soap:address location="http://ugimbtp3:1024/oper/room"/>
		</port>
	</service>
</definitions>
```

APPENDIX C

Business Process (BPEL) Documents of Artemis Web Service

The BPEL documents given in this appendix are composed and described for the Web Service Orchestration Demo of Artemis Project [2][37][36].

ChapaHospital.bpel

```
<process name="ChapaHospital"
  targetNamespace="urn:Capad:CapadService"
  xmlns:tns="urn:Capad:CapadService"
  xmlns="http://schemas.xmlsoap.org/ws/2002/07/business-process/"
  xmlns:ns0="urn:Foo"
  xmlns:ns1="urn:Cerrahpasa:CerrahpasaBPEL">

  <containers>
    <container name="input" messageType="tns:ChapaHospitalRequestMessage"/>
    <container name="output" messageType="tns:ChapaHospitalResponseMessage"/>
    <container name="admitin" messageType="ns0:AdmitVisit_admit"/>
    <container name="admitout" messageType="ns0:AdmitVisit_admitResponse"/>
    <container name="unsolin" messageType="ns0:Unsolicited_transmission"/>
    <container name="unsolout" messageType="ns0:Unsolicited_transmissionResponse"/>
    <container name="cerrahin" messageType="ns1:CerrahpasaRequestMessage"/>
    <container name="cerrahout" messageType="ns1:CerrahpasaResultMessage"/>
  </containers>

  <partners>
    <partner name="caller" serviceLinkType="tns:ChapaHospitalSLT"/>
    <partner name="AdmitService" serviceLinkType="ns0:AdmitVisitSLT"/>
    <partner name="UnsolicitedService" serviceLinkType="ns0:UnsolicitedSLT"/>
    <partner name="Cerrahpasa" serviceLinkType="ns1:CerrahpasaSLT"/>
  </partners>

  <sequence name="main">
    <receive name="receiveInput" partner="caller" portType="tns:ChapaPT" operation="process"
container="input" createInstance="yes"/>
    <scope name="AdmitVisit">
      <sequence>
          <assign name="Input_to_AdmitVisit">
            <copy>
              <from container="input" part="EVN">
              </from>
              <to container="admitin" part="String_1"/>
            </copy>
            <copy>
              <from container="input" part="MSH">
              </from>
              <to container="admitin" part="String_2"/>
            </copy>
            <copy>
              <from container="input" part="PID">
```

```
        </from>
        <to container="admitin" part="String_3"/>
       </copy>
       <copy>
        <from container="input" part="DB1">
        </from>
        <to container="admitin" part="String_4"/>
       </copy>
       <copy>
        <from container="input" part="AL1">
        </from>
        <to container="admitin" part="String_5"/>
       </copy>
       <copy>
        <from container="input" part="ACC">
        </from>
        <to container="admitin" part="String_6"/>
       </copy>
      </assign>
      <invoke partner="AdmitService" portType="ns0:AdmitVisit" operation="admit"
inputContainer="admitin" outputContainer="admitout"/>
      <assign name="Output_of_AdmitVisit">
       <copy>
        <from container="admitout" part="result">
        </from>
        <to container="output" part="result"/>
       </copy>
      </assign>
   </sequence>
  </scope>
  <switch name="Check_AdmitVisit">
   <case condition="bpws:getContainerData('output','result') = 'Admit:OK'">
      <sequence>
       <scope name="Unsolicited_Transmission">
        <sequence>
         <assign name="Input_to_Unsolicited">
            <copy>
             <from container="input" part="MSH">
             </from>
             <to container="unsolin" part="String_1"/>
            </copy>
            <copy>
             <from container="input" part="PID">
             </from>
             <to container="unsolin" part="String_2"/>
            </copy>
            <copy>
             <from container="input" part="OBR">
             </from>
             <to container="unsolin" part="String_3"/>
            </copy>
            <copy>
             <from container="input" part="OBX">
             </from>
             <to container="unsolin" part="String_4"/>
            </copy>
         </assign>
         <invoke partner="UnsolicitedService" portType="ns0:Unsolicited" operation="transmission"
inputContainer="unsolin" outputContainer="unsolout"/>
         <assign name="Output_of_Unsolicited">
            <copy>
             <from container="unsolout" part="result">
             </from>
```

```
                <to container="output" part="result"/>
              </copy>
            </assign>
          </sequence>
        </scope>
        <switch name="Check_Unsolicited">
          <case condition="bpws:getContainerData('output','result') = 'Unsolicited:OK'">
            <scope name="CerrahpashaHospital">
                <sequence>
                  <assign name="Input_to_Cerrahpasha">
                    <copy>
                      <from container="input" part="MSH">
                      </from>
                      <to container="cerrahin" part="msh"/>
                    </copy>
                    <copy>
                      <from container="input" part="PID">
                      </from>
                      <to container="cerrahin" part="pid"/>
                    </copy>
                    <copy>
                      <from container="input" part="AL1">
                      </from>
                      <to container="cerrahin" part="al1"/>
                    </copy>
                    <copy>
                      <from container="input" part="OBR">
                      </from>
                      <to container="cerrahin" part="obr"/>
                    </copy>
                    <copy>
                      <from container="input" part="OBX">
                      </from>
                      <to container="cerrahin" part="obx"/>
                    </copy>
                  </assign>
                  <invoke partner="Cerrahpasa" portType="ns1:Cerrahpasa" operation="initiate"
inputContainer="cerrahin" outputContainer="cerrahout"/>
                  <assign name="Last">
                    <copy>
                      <from container="cerrahout" part="result">
                      </from>
                      <to container="output" part="result"/>
                    </copy>
                  </assign>
                </sequence>
            </scope>
          </case>
          <otherwise>
            <assign name="Unsolicited_Failed"><copy>
                  <from expression="'Unsolicited is not OK'">
                  </from>
                  <to container="output" part="result"/>
                </copy>
            </assign>
          </otherwise>
        </switch>
      </sequence>
  </case>
  <otherwise>
      <assign name="Admit_Failed">
        <copy>
          <from expression="'Admit is not OK'">
```

```
            </from>
            <to container="output" part="result"/>
          </copy>
        </assign>
    </otherwise>
   </switch>
   <reply name="replyOutput" partner="caller" portType="tns:ChapaPT" operation="process"
container="output"/>
  </sequence>
</process>
```

Cerrahpasa.bpel

```
<process name="Cerrahpasa"
  targetNamespace="urn:Cerrahpasa:CerrahpasaBPEL"
  xmlns:tns="urn:Cerrahpasa:CerrahpasaBPEL"
  xmlns:ns0="urn:Foo"
  xmlns="http://schemas.xmlsoap.org/ws/2002/07/business-process/" >

  <containers>
    <container name="input" messageType="tns:CerrahpasaRequestMessage"/>
    <container name="output" messageType="tns:CerrahpasaResultMessage"/>
    <container name="orderInput" messageType="ns0:OrderEntry_order"/>
    <container name="orderOutput" messageType="ns0:OrderEntry_orderResponse"/>
    <container name="unsolicited2Input" messageType="ns0:Unsolicited_transmission"/>
    <container name="unsolicited2Output" messageType="ns0:Unsolicited_transmissionResponse"/>
    <container name="unsolicited3Input" messageType="ns0:Unsolicited_transmission"/>
    <container name="unsolicited3Output" messageType="ns0:Unsolicited_transmissionResponse"/>
    <container name="scheduleInput" messageType="ns0:Scheduling_schedule"/>
    <container name="scheduleOutput" messageType="ns0:Scheduling_scheduleResponse"/>
  </containers>

  <partners>
    <partner name="client" serviceLinkType="tns:CerrahpasaSLT"/>
    <partner name="orderService" serviceLinkType="ns0:OrderEntrySLT"/>
    <partner name="unsolicited2" serviceLinkType="ns0:Unsolicited2SLT"/>
    <partner name="unsolicited3" serviceLinkType="ns0:UnsolicitedS3LT"/>
    <partner name="schedule" serviceLinkType="ns0:SchedulingSLT"/>
  </partners>

  <sequence name="main">
    <receive name="receiveInput" partner="client" portType="tns:Cerrahpasa"
      operation="initiate" container="input"
      createInstance="yes"/>
    <scope>
     <sequence>
          <scope>
            <flow>
              <sequence>
               <scope name="UnsolicitedTransmissionOfPatient">
                     <sequence>
                       <assign>
                         <copy>
                           <from container="input" part="msh">
                           </from>
                           <to container="unsolicited2Input" part="String_1"/>
                         </copy>
                         <copy>
                           <from container="input" part="pid" >
                           </from>
                           <to container="unsolicited2Input" part="String_2"/>
                         </copy>
                         <copy>
                           <from container="input" part="obr" >
                           </from>
                           <to container="unsolicited2Input" part="String_3"/>
                         </copy>
                         <copy>
                           <from container="input" part="obx">
                           </from>
                           <to container="unsolicited2Input" part="String_4"/>
                         </copy>
                       </assign><invoke partner="unsolicited2" portType="ns0:Unsolicited"
operation="transmission" inputContainer="unsolicited2Input" outputContainer="unsolicited2Output"/>
```

```
            </sequence>
      </scope>
    </sequence>
    <sequence>
      <scope name="UnsolicitedTransmissionOfRecordSystem">
            <sequence>
              <assign>
                <copy>
                  <from container="input" part="msh" >
                  </from>
                  <to container="unsolicited3Input" part="String_1"/>
                </copy>
                <copy>
                  <from container="input" part="pid">
                  </from>
                  <to container="unsolicited3Input" part="String_2"/>
                </copy>
                <copy>
                  <from container="input" part="obr">
                  </from>
                  <to container="unsolicited3Input" part="String_3"/>
                </copy>
                <copy>
                  <from container="input" part="obx">
                  </from>
                  <to container="unsolicited3Input" part="String_4"/>
                </copy>
              </assign>
              <invoke operation="transmission" inputContainer="unsolicited3Input"
outputContainer="unsolicited3Output" partner="unsolicited3" portType="ns0:Unsolicited"/>
            </sequence>
      </scope>
    </sequence>
    <sequence>
      <scope name="SchedulingOperationRoom">
            <sequence>
              <sequence>
                <assign>
                  <copy>
                        <from container="input" part="msh">
                        </from>
                        <to container="scheduleInput" part="String_1"/>
                  </copy>
                  <copy>
                        <from expression="'booo'">
                        </from>
                        <to container="scheduleInput" part="String_2"/>
                  </copy>
                  <copy>
                        <from container="input" part="pid">
                        </from>
                        <to container="scheduleInput" part="String_3"/>
                  </copy>
                  <copy>
                        <from container="input" part="obx">
                        </from>
                        <to container="scheduleInput" part="String_4"/>
                  </copy>
                  <copy>
                        <from expression="'no dg1 info found'">
                        </from>
                        <to container="scheduleInput" part="String_5"/>
                  </copy>
                  <copy>
                        <from expression="'no ais info found'">
                        </from>
                        <to container="scheduleInput" part="String_6"/>
                  </copy>
                 </assign>
                 <invoke operation="schedule" inputContainer="scheduleInput"
outputContainer="scheduleOutput" partner="schedule" portType="ns0:Scheduling"/>
                 <assign>
                  <copy>
                        <from container="scheduleOutput" part="result">
                        </from>
                        <to container="output" part="result"/>
                  </copy>
                 </assign>
                </sequence>
               </sequence>
          </scope>
         </sequence>
        </flow>
       </scope>
       <sequence name="LabOrderEntry">
        <assign>
         <copy>
          <from container="input" part="msh">
          </from>
          <to container="orderInput" part="String_1"/>
         </copy>
         <copy>
          <from container="input" part="pid">
          </from>
          <to container="orderInput" part="String_2"/>
         </copy>
         <copy>
          <from container="input" part="al1">
          </from>
          <to container="orderInput" part="String_3"/>
         </copy>
         <copy>
          <from expression="'boooo'">
          </from>
          <to container="orderInput" part="String_4"/>
         </copy>
         <copy>
          <from expression="'booooo'">
          </from>
          <to container="orderInput" part="String_5"/>
         </copy>
        </assign>
        <invoke operation="order" inputContainer="orderInput" outputContainer="orderOutput"
partner="orderService" portType="ns0:OrderEntry"/>
       </sequence>
     </sequence>
   </scope>
   <reply name="callbackClient" partner="client" portType="tns:Cerrahpasa" operation="initiate"
container="output"/>
  </sequence>
</process>
```