DESIGN AND IMPLEMENTATION OF SEMANTICALLY ENRICHED WEB SERVICES IN THE HEALTHCARE DOMAIN

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

BY

ÜMİT LÜTFÜ ALTINTAKAN

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

Assoc. Prof. Dr.İsmail Hakkı Toroslu	(METU,CENG)	____________________
Prof. Dr. Asuman Doğaç	(METU,CENG)	____________________
Assoc. Prof. Dr. Nihan Kesim Çiçekli	(METU,CENG)	____________________
Assoc. Prof. Dr. Ali Doğru	(METU,CENG)	____________________
Bülent Kunaç	(TEPE Tech.)	____________________

ABSTRACT

DESIGN AND IMPLEMENTATION OF SEMANTICALLY ENRICHED WEB SERVICES IN THE HEALTHCARE DOMAIN

Altıntakan, Ümit Lütfü
M.S., Department of Computer Engineering
Supervisor: Prof. Dr. Asuman Doğaç

December 2004, 123 pages

Healthcare Informatics suffers from the lack of information exchange among domain partners. Allowing cooperation among distributed and heterogeneous applications is a major need of current healthcare information systems. Beyond the communication and integration problems, medical information itself is by nature complex, combined with data and knowledge. The increasing number of standards and representation of the same data in different structures using these standards constitute another problem in the domain.

Platform and implementation independency makes Web service technology the natural way to solve the interoperability problems in the healthcare domain. Standardizing the access to data through WSDL and SOAP rather than standardizing the electronic health record will help to overcome the integration problems among different standards in medical information systems. However, introducing Web services to the healthcare systems will not suffice to solve the problems in the domain unless the semantics of the services are exploited.

This thesis aims to show that by generating web services and classifying these services through their functionalities, it is possible to achieve the interoperability among healthcare institutes, such as hospitals. The designed system is based on Artemis P2P Framework, and the annotation of the system is realized in the same framework.

Keywords: Medical Informatics, Web Service, Semantic, Healthcare, Interoperability

ÖZ

MANTIKSAL OLARAK ZENGİNLEŞTİRİLMİŞ AĞ SERVİSLERİNİN SAĞLIK ALANINDA TASARLANMASI VE UYGULANMASI

Altıntakan, Ümit Lütfü
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Asuman Doğaç

Aralık 2004, 123 sayfa

Sağlık Bilgi Sistemleri, sağlık alanındaki ortakların bilgi değişim noksanlığından olumsuz etkilenmektedir. Dağınık ve heterojen uygulamalar arasında birlikte işleyebilirliğin sağlanması, mevcut sağlık bilgi sistemlerinin temel ihtiyaçlarından biridir. Sağlık bilişimi, iletişim ve entegrasyon sorunlarının yanısıra, tabiatı gereği kompleks bir yapı taşımaktadır; veri ve bilgiyi beraber içermektedir. Her geçen gün artan standartlar ve bu standartların aynı bilgi parçacıklarını farklı yapılarda sunmaları, alandaki diğer bir problemi oluşturmaktadır.

Ortamdan ve uygulamadan bağımsız olmaları, Ağ servislerinin, sağlık alanındaki birlikte işleyebilirlik probleminin çözümünde doğal bir tercih olacağını göstermektedir. Elektronik sağlık kayıtlarının standartlaştırılması yerine, bu kayıtlara erişimin WSDL ve SOAP gibi kabul edilmiş standartlar kullanılarak sağlanması, sağlık bilgi sistemleri arasındaki entegrasyon probleminin çözümüne katkı sağlayacaktır. Ancak, mantıksal tanımlamaları yapılmadan, sağlık sistemlerinde Ağ servislerinin kullanılması alandaki sorunları çözmeye yeterli olmayacaktır.

Bu tez çalışmasında, sağlık alanında ağ servisleri oluşturup, bu servislerin fonsiyonalitelerine göre sınıflandırılması ile, hastane gibi sağlık kuruluşları arasında, birlikte işleyebilirliğin sağlanabileceğinin gösterilmesi hedeflenmiştir. Geliştirilen sistem, Artemis P2P ortamına dayanmakta ve sistemin örnek uygulaması da aynı ortamda gerçekleştirilmiştir.

Anahtar Kelimeler: Tıp Bilişimi, Ağ Servisleri, Mantık, Birlikte İşleyebilirlik.

To My Family

ACKNOWLEDGMENTS

I would like to thank my supervisor Asuman Doğaç for all her guidance, advice, encouragements and support during this study.

I would like to thank all members of SRDC team for their technical guidance and support during this study.

I would also thank to Yıldıray Kabak, Gökçe Banu Laleci and İlhami Görgün for their technical assistance and support throughout the study.

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

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Healthcare Informatics is one of the few domains that constitute huge domain knowledge and standards. Large volumes of data are generated from hospitals, primary care surgeries, clinics, and laboratories every day. The single doctor-patient relationship is being replaced by one in which the patient is managed by a team of healthcare professionals each specialized in one aspect of care. Clinical information of a patient is spread out over a number of medical centers which makes it difficult to get the exact history of a patient.

In order to improve the quality of care and to reduce cost, the ability to share information easily between care providers must be supplied. Cooperation among different medical information systems within a regional, national or even international scope is also strongly required. However, these systems are developed progressively and in isolation. They do not communicate with each other. Redeveloping all the applications in a new standard way can solve the problem but that is an expensive process and wastes resources. Integrating medical information systems becomes the only flexible solution.

In addition, medical informatics is by nature complex; combined with data and knowledge. For example, the value "sex" of a patient can be coded as a symbol ["M" | "F"] or as an integer [0 | 1]; it can also reference to different vocabularies, nomenclatures or classifications (e.g., ICD9-CM, ICD10, Mesh, SNOMED International, UMLS, etc.) [8]. Medical knowledge such as the semantic relations among medical concepts can be represented in different ways. Existing standards in the healthcare domain, such as HL7 [17], CEN TC251 [5], ISO TC215 [23] and GEHR [58], represent the same information in different structures and vocabularies, which produces another problem for interoperability.

Web services, on the other hand, are mainly designed to solve such interoperability problems. Hence introducing Web service technology to the healthcare domain seems to solve most of the problems in medical informatics.

Well accepted standards like Web Services Description Language (WSDL) [66] and Simple Object Access Protocol (SOAP) [54] make it possible to dynamically invoke Web services. However to be able to exploit the Web services to their full potential, their semantic information should be available.

Introducing Web services to healthcare domain will bring the following advantages:

- By standardizing the access to data through WSDL and SOAP, it will be easier to provide the interoperability of medical systems.
- 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.

Within the scope of this thesis, a real-life application software running on a hospital is examined and a number of sample web services are generated. To make the services useful for other health centers, the semantics of them are described based on ontologies, so that the consumers will be able to search and use the correct services. However, there does not exist global ontologies in healthcare informatics and it is not realistic to expect global ontologies. Although there is a huge effort to develop a metathesaurus, called Unified Medical Language System (UMLS) [62], this is not an ontology. In this study, to semantically markup Web services, simple ontologies are developed based on prominent healthcare standards such as HL7 [17], GEHR [58], and CEN ENV 13606-2 [5]. It should be noted that the aim of this thesis is not to propose ontologies, but to show how they can be made use of once they are defined by standard bodies. The web services are classified through their functionality and a service functionality ontology is created according to the semantics of the services offered. Since there can be quite complex service parameters coming from different information systems, additional message semantics is required, and the semantics of service parameters are defined using message ontologies.

The thesis is organized as follows: In Chapter 2, the background information and the related work done in healthcare information are introduced. Specifically, healthcare domain and health information standards including documentation, coding and classification are described. In addition, the enabling technologies, such as Web services; semantics of services; peer to peer (P2P) computing paradigm are also included in this chapter.

In Chapter 3, the system designed in this study is introduced. First, two major examples are given to show the benefits gained after the integration in medical information systems is achieved. Then the examined hospital information system and the problems of the real life system are introduced. Next, the need for functionality and message ontologies are described and both are developed based on HL7 standards. This chapter concludes with the descriptions of the generated sample services.

Chapter 4 presents the implementation platform, tools and an annotation of the system. In order to clarify the concepts a sample scenario is given at the end of this chapter.

Finally the conclusion of the study and the future work is presented in Chapter 5.

CHAPTER 2

RELATED WORK

2.1 Healthcare Domain

A good understanding of existing healthcare standards is essential to realize the work achieved in this thesis. This chapter is mainly about some of the most wide spread techniques, models and standards in the healthcare domain.

2.1.1 Health Level Seven (HL7)

HL7 is both an American National Standards Institute (ANSI)-accredited standards organization and a standard. As a standards development organization, HL7's mission is to provide standards for the exchange, management and integration of data that support clinical patient care, and the management and delivery of healthcare services by defining the protocol for exchanging clinical data between diverse healthcare information systems [17]. Furthermore HL7 is a standard in healthcare that does not focus on the requirements of a particular department in a healthcare organization but it addresses the entire healthcare organization.

HL7's purpose is to facilitate communication in healthcare settings. The primary goal is to provide standards for the exchange of data among healthcare computer applications that eliminate or substantially reduce the custom interface programming and program maintenance that may otherwise be required [17].

The HL7 organization works through to create flexible, cost-effective approaches, standards, guidelines, methodologies and related services for interoperability between healthcare information systems.

In 1999, two HL7 protocols were published in an effort to move HL7 beyond its traditional message-based functionality. First, the HL7 organization published the Clinical Context Management Specification Version 1.0 (CCM) as an ANSI standard. The CCM standard establishes nationwide support for the visual integration of disparate healthcare applications on the clinical desktop. Vendors supporting this standard enable end users to seamlessly view results from different back-end clinical systems as if they were totally integrated.

As a standard, HL7 is widely accepted and used. Since its creation, this standard has grown from a user-based consensus standard to an international standard with affiliate groups in Australia, Canada, Finland, Germany, India, The Netherlands, New Zealand, South Africa and the United Kingdom. Since 1987 several versions of the HL7 standard have been published. The current version is 2.5; version 3.0 is still in a development phase.

HL7 does not assume nor does it make any assumption of data storage within applications but it is designed to support various healthcare systems, both centralistic and distributed systems, in heterogeneous environments to communicate with each other [17]. In addition the standard's goals are to standardise the content and therefore the interfaces between systems, to increase the efficiency of communication, and to decrease the number of interfaces between healthcare systems.

The term "Level 7" refers to the highest level of the Open System Interconnection (OSI) model of the International Organization for Standardization (ISO). This is not to say that HL7 conforms to ISO defined elements of the OSI's seventh level. Also, HL7 does not specify a set of ISO approved specifications to occupy layers 1 to 6 under HL7's abstract message specifications. HL7 does, however, correspond to the conceptual definition of an application-to-application interface placed in the seventh layer of the OSI model [17].

In the OSI conceptual model, the functions of both communications software and hardware are separated into seven layers, or levels. The HL7 Standard is primarily focused on the issues that occur within the seventh, or application, level. These are the definitions of the data to be exchanged, the timing of the exchanges, and the communication of certain application-specific errors between the applications.

2.1.1.1 Version 2

The HL7 Version 2 Messaging Standard — Application Protocol for Electronic Data Exchange in Healthcare Environments — is considered to be the workhorse of data exchange in healthcare and is the most widely implemented standard for healthcare information in the world. These Version 2 standards are developed with the assumption that an event in the real world of healthcare creates the need for data to flow among systems. The real-world event is called the **trigger** event. For example, the trigger event a "patient is admitted" may cause the need for data about that patient to be sent to a number of other systems. The trigger event, an observation for a patient is available, may cause the need for that observation to be sent to a number of other systems.

The standard groups the events into the following clusters:

- *Patient Administration*: Admit, Discharge, Transfer, and Demographics.
- *Order Entry*: Orders for Clinical Services and Observations, Pharmacy, Dietary, and Supplies.
- *Query:* Rules applying to queries and to their responses
- *Financial Management*: Patient Accounting and Charges
- *Observation Reporting*: Observation Report Messages.
- *Master Files*: Health Care Application Master Files.
- *Medical Records/Information Management*: Document Management Services and Resources
- *Scheduling*: Appointment Scheduling and Resources.
- *Patient Referral*: Primary Care Referral Messages
- *Patient Care*: Problem-Oriented Records.

The HL7 Version 2 Standard consists of a simple exchange of messages between a pair of applications: the unsolicited update and its acknowledgment or the query and its response. The underlying operational model is that of a **client** and a **server**. An application interfaces with another application using an event code that identifies the transaction. The other application responds with a message that includes data or an error indication. The initiating application may receive a reject status from the other application or from lower level software indicating that its message was not received correctly.

A **message** is the atomic unit of data transferred between systems. It is comprised of a group of segments in a defined sequence. Each message has a three-character code **message type** that defines its purpose. For example the ADT Message type is used to transmit portions of a patient's Patient Administration (ADT) data from one system to another.

There is a one-to-many relationship between message types and trigger event codes. The same trigger event code may not be associated with more than one message type; however a message type may be associated with more than one trigger event.

A **segment** is a logical grouping of **data fields**. Segments of a message may be required or optional. They may occur only once in a message or they may be allowed to repeat. Each segment is given a name. For example, the ADT message may contain the following segments: Message Header (MSH), Event Type (EVN), Patient ID (PID), and Patient Visit (PV1).

2.1.1.2 Version 3

The V2.x series of messages are widely implemented and very successful. These messages evolved over several years using a "bottom-up" approach that has addressed individual needs through an evolving ad-hoc methodology. However there was neither a consistent view of that data that HL7 moves nor that data's relationship to other data.

While providing great flexibility, its optionality also makes it impossible to have reliable conformance tests of any vendor's implementation and also forces implementers to spend more time analyzing and planning their interfaces to ensure that both parties are using the same optional features [1]. To remedy and avoid these problems the message based communication will be based on an object-oriented data model, the so called Reference Information Model (RIM) and on Domain Information Models (DIMs) to create messages [1].

The RIM is a static model of health and healthcare information as viewed within the scope of HL7 development activities [18]. In these data model semantic and lexical connections that exist between different fields of HL7 messages will be modelled. It is the ultimate source from which all HL7 version 3.0 protocol specification standards draw their information related content.

Furthermore, the following features will extend and advance the usability and acceptance of the upcoming HL7 version 3.0 of the standard [1]:

- Whereas in previous versions of the standard many fields in messages have been optionally, the number of optional fields is going to be reduced much.
- Future message communication will be based on XML.
- A Message Development Framework (MDF) will be provided by the HL7 group.
- Test criteria to verify application conformance to the standard will be provided.

2.1.2 Electronic Health Record (EHR) and Clinical Document Architectures

There is no commonly agreed nor ISO standardised or certified definition of the term EHR, the architecture of an EHR, and the access-mechanisms for the retrieval of information that is stored in an EHR. The ISO lists the following principles underpinning the EHR [24]:

- The EHR should be timely, reliable, complete, accurate, secure and accessible and designed to support the delivery of healthcare services regardless of the model of healthcare being applied. It should interoperate in a way which is truly global yet respects local customs, language and culture.
- The EHR should not be considered applicable only to patients, that is, individuals with the presence of some pathological condition. Rather, the focus should be on individual’s health, encompassing both well-being and morbidity.
- The EHR recognises that an individual’s health data will be distributed over different systems, and in different locations around the world. To achieve the integration of data, the EHR will require the adoption of a common information model by compliant systems and the adoption of relevant international standards wherever possible.

An open standardised EHR architecture would be the key to interoperability at the information level. Such a standardised architecture would enable the whole or parts of the EHR to be shared and exchanged, independently of any particular EHR system.

2.1.2.1 Comitè Europèen de Normalisation (CEN) TC251

The technical committee TC251 “Health Informatics” is a subdivision of the European Committee for Standardization (CEN). The scope of TC251 is the standardization in the field of Health Information and Communications Technology (ICT) to achieve compatibility and interoperability between independent systems and to enable modularity.

The European pre-standard ENV 13606:2000 was first published in 1999 by the TC251. It proposes a set of interoperability measures to facilitate the communication between heterogeneous systems with respect to the requirement, that the meaning of clinical data, primarily intended by the original author, must be preserved faithfully and presented by the receiving system, even if the underlying system architectures of the sender and receiver vary.

To improve the pre-standard ENV 13606 to a full "EN"-standard a CEN Task Force "EHRcom" was initiated in December 2001. Multiple inputs with emphasis on implementation experiences have been considered and cooperative work with HL7 (CDA) and openEHR has led to a first revised version of the ENV 13606, prEN 13606:2003. Whereas ENV 13606 consists of four parts the revised prEN 13606 will consist of five parts:

- Part 1: Reference Model (refinement of ENV13606 Part 1)
- Part 2: Archetype Interchange Specification (new)
- Part 3: Reference Archetypes and Term Lists (will draw on ENV13606 Part 2)
- Part 4: Security Features (will draw on ENV13606 Part 3)
- Part 5: Exchange Models (fulfilling the same role as ENV13606 Part 4)

2.1.2.2 Good Electronic Healthcare Record (GEHR)

GEHR first started as a European Project between 1992 and 1995. The main objective of the project was to develop a common EHR architecture for Europe.

The GEHR approach uses a formal semantic model, known as the GEHR Object Model (GOM). Rather than try to model a myriad of possible clinical concepts, the GOM provides concepts at a number of levels [59]:

- EHR and Transaction level
- Navigation level
- Content (e.g. observation, subjective, instruction) level
- Data types (e.g. quantity, multimedia) level

Clinical models are expressed outside the GOM in the form of *archetypes*. These clinical archetypes define the valid GOM information structures for particular clinical concepts such as "blood pressure", "family history" and so on.

Archetypes are expressed in XML-Schema, allowing them to be authored, viewed and managed outside the record; from the point of view of an EHR system, they are simply treated as input documents. The meta-model approach fulfills the requirement that GEHR not constrain the way medicine is practiced, but rather provides a way in which information created under any practice model can be stored.

2.1.2.3 HL7 Clinical Document Architecture (CDA)

The Clinical Document Architecture is part of the HL7 version 3 family of standards for the representation and machine processing of clinical documents in a way which makes the documents both human readable and machine processable, and guarantees preservation of the content by using XML [64]. More precisely, the CDA standard does standardize the markup of structure and semantics for the exchange of CDA documents but does not model the clinical content.

The CDA specifies the structure and semantics of clinical documents in health care. A document can be defined as a piece of text or information that would usually be authenticated by a signature eg. a progress note, a pathology request, a radiology report, or an account. A CDA document may contain text, images and multimedia, coded data. The CDA document can be:

- Stored either permanently or temporarily as a document in a computer system; and
- Transmitted as the content of a message using email, HL7 or any other messaging system.

A clinical document has the following features which form the framework for the CDA:

- Persistence
- Stewardship
- Authentication
- Wholeness and context
- Human readability

The CDA document architecture is structured in three levels: Level 1 through Level 3. Level 1 has a structured header and structured body of message with limited coding capacity for content. Levels 2 and 3 impose more structure to allow the representation of "context" or constrained fields and more coded data.

2.1.3 Documentation / Coding / Classification Standards

There are several possibilities how data might be recorded for diagnoses and procedures. However, to allow for analyses of collected data or exchange of patient data between medical institutes it is necessary to store patient records in well defined data structures, and to codify diagnoses and procedures. In this section brief summary of some of the main documentation, coding, and classification standards are given.

2.1.3.1 International Statistical Classification of Diseases (ICD)

The *International Classification of Diseases* (ICD) is designed to promote international comparability in the collection, processing, classification, and presentation of **mortality statistics.**

The International Classification of Diseases and Related Health Problems is the oldest and most widely recognised classification of diseases. Later, The World Health Organization (WHO) extended the ICD to cover diseases and injuries.

The ICD is developed collaboratively between the WHO and several international institutions so that the medical terms reported by physicians, medical examiners, and coroners on death certificates can be grouped together for statistical purposes. The purpose of the ICD and of WHO sponsorship is to promote international comparability in the collection, processing, and presentation of mortality and morbidity statistics.

The ICD is used to translate terms of disease diagnoses and other health problems into alphanumeric codes. In the ICD diseases and injuries are arranged in groups by means of established criteria, such as epidemic diseases, constitutional or general diseases, endemic diseases (ordered by localisation), diseases due to development, and injuries.

Although ICD Tenth Revision has been published in 1992 already, many systems are still based on ICD Ninth Revision. Both revisions contain a set of expansions for other families of medical terms than diagnostic terms. About every decade the ICD is revised by the WHO and their partners to incorporate changes in the medical field by means of advances in medical science.

In ICD-9 all disease terms are separated into 17 chapters of diseases. Each chapter contains up to 24 groups of diseases, all chapters together contain 110 groups. Each group contains up to 20 three-digit categories, all groups together contain 909 categories. Every three-digit category contains up to 10 four-digit subcategories which might contain up to 10 subdivisions at the fifth digit. ICD codes with five digits represent the most detailed level whereas four- and three-digit codes are more general.

The structure of the tenth revision of the ICD is similar to the ninth revision, but whereas characters are used at the first position of a code in the ninth revision, in the tenth revision all codes begin with a character followed by two digits and another digit after a point. Each three-digit category contains up to 10 four-digit subcategories which might contain up to 10 subdivisions at the fifth digit. ICD codes with five digits represent the most detailed level whereas four- and three-digit codes are more general.

2.1.3.2 International Classification of Primary Care (ICPC)

ICPC was primarily designed for facilitating primary care providers to classify patient reasons for encounter at the time of consultation. But it can also be used to codify the practitioner’s assessment of the problem (diagnoses), to the diagnostic interventions, and to the treatment for the problem at the time of the encounter. With this classification healthcare providers can classify three important elements of the health care encounter using a single classifier: reasons for encounter, diagnoses or problems, and process of care. ICPC enables the health care provider to differentiate between episodes of care and permanent diagnoses within the classification system. Furthermore the ICPC is problem-oriented and not disease-oriented.

The first version of the ICPC (ICPC-1) was published in 1987 by the World Organization of National Colleges, Academics, and Academic Associations of General Practitioners/Family

Physicians (WONCA). The second version (ICPC-2) was published in 1998 and the current version (ICPC-2-E) was published in 2000. Conversion tables ICPC-2/ICD-10 are available.

2.1.3.3 International Classification of Procedures in Medicine (ICPM)

ICPM was published in 1976 by the WHO for trial purposes. It contained chapters on diagnostic, laboratory, preventive, surgical, other therapeutic and ancillary procedures. ICPM has been a source of inspiration for a number of other procedural classifications.

The ICPM is a four-digit systematic of procedures in medicine. ICPM covers therapeutic, diagnostic, and prophylactic measures. The ICPM is subdivided into 9 parts:

- Procedures for medical diagnosis
- Laboratory procedures
- Preventive procedures
- Surgical procedures
- Radiology and certain other applications of physics in medicine
- (the same as 7.)
- Drugs, medicaments and biological agents
- Other therapeutic procedures
- Ancillary procedures

2.1.3.4 The Procedure Coding System (PCS)

The US-American Healthcare Financing Administration (HCFA) initiated the development of a totally new procedure coding system (ICD-10-PCS). Many European countries are also interested in this emerging PCS, especially for the definition of case groups and reimbursement.

The PCS is a multi-dimensional or multi-axial classification for medical and surgical procedures with a seven-digit alphanumerical notation. The meaning of the following six characters depends on the first character describing the section:

- Section (0: Medical and Surgical, …)
- Body System (0: Central Nervous System, 1: Peripheral Nervous System, …)
- Root Operation (0: Bypass, 1: Change, 2: Creation, …)

- Body Part (depends completely on the previous digits)
- Approach (0: Open, 1: Open Intraluminal, …)
- Device (depends completely on the previous digits)
- Qualifier (depends completely on the previous digits)

The most important axis is the “root operation” which specifies the underlying objective of the procedure. All possible medical and surgical procedures comprise now 30 different root operations. PCS avoids multiple meanings for the same term by means of a standardised terminology and thus different procedures are labelled with a unique term.

2.1.4 Linking Different Classifications and Nomenclatures

In course of time several different controlled vocabularies have been introduced. This diversity has resulted in confusion about how to compare medical expressions which have been classified of indexed by different controlled vocabularies. These problems and the wish for a better standardisation and harmony led to some approaches. One of these approaches is described in this section briefly.

2.1.4.1 Unified Medical Language System (UMLS)

The UMLS is developed by the USA National Library of Medicine (NLM) and it is a long-term research and development project. The purpose of the UMLS project is to facilitate the retrieval and integration of information from multiple machine-readable biomedical information sources, such as bibliographic biomedical databases, clinical records, knowledge-based systems, and directories of people and organizations. The UMLS project develops and distributes multi-purpose, electronic “Knowledge Sources” and associated lexical programs.

The UMLS Metathesaurus is one of three knowledge sources the NLM provides as part of the UMLS project. It preserves the names, meanings, hierarchical contexts, attributes, and inter-term relationships present in its source vocabularies. It adds certain basic information to each concept and establishes synonymy and new relationships between terms from different source vocabularies. The Metathesaurus contains concepts and concept names from more than 100 vocabularies and classifications, e.g. ICD-10 and ICPC, most of them are included entirely and some in multiple editions. The structure of the Metathesaurus can also incorporate translations of vocabularies other than English.

The purpose of the Methatesaurus is to link alternative names and views of the same concept together and to identify useful relationships between different concepts and hence the Metathesaurus is organized by concept or meaning. Each concept or meaning has a unique identifier (CUI) with no intrinsic meaning. Each unique concept name or string in each language has a unique string identifier (SUI) and for English language entries only each string is linked to all of its lexical variants or minor variations by means of a common term identifier (LUI). All string and term identifiers are linked to at least one concept identifier and different terms with the same meaning are linked to the same concept identifier.

Relationships between different concepts in the Metathesaurus are derived from source vocabularies, e.g. from the hierarchical structure of the source vocabulary, or relationships are created by Metathesaurus editors to link concepts that would not otherwise be linked.

Annual editions of the Metathesaurus have been distributed since 1990 and the current version was released in November 2003.

The Semantic Net of the UMLS offers a special view on the various links established between medical terms within a vocabulary and between vocabularies. Several hierarchies and other generic or semantic relationships can be viewed simultaneously and the granularity of a classification structure can be chosen on each level.

2.1.5 IHE - Integrating the Healthcare Enterprise

Integrating the Healthcare Enterprise (IHE) is an initiative designed to promote the integration of information systems that support modern healthcare institutions. Its fundamental objective is to ensure that in the care of patients all required information for medical decisions is accurate and available to healthcare professionals [22].

IHE is not a standards organization. Instead it promotes coordinated use of existing standards such as DICOM and HL7 to develop workflow solutions for the healthcare enterprise. Systems designed in agreement with IHE profiles communicate with one another better, are easier to implement, and facilitate the efficient access of information. Physicians, nurses, administrators and other healthcare professionals foresee a day when vital information can flow seamlessly from system to system and be readily available at the point of care. IHE is intended to make this vision a reality by improving the condition of systems integration and eliminating barriers to optimal patient care.

The IHE Technical Framework identifies functional components of a distributed healthcare environment (referred to as IHE actors), solely from the point of view of their interactions in the healthcare enterprise [22].

Actors and transactions are defined to provide a basis for defining the interactions among functional components of the healthcare information system environment. The IHE actors and transactions described in the IHE Technical Framework are abstractions of the real-world healthcare information system environment.

2.1.5.1 Integration Profiles

IHE IT Infrastructure Integration Profiles (Figure 2.1), offer a common language that healthcare professionals and vendors can use to discuss integration needs of healthcare enterprises and the integration capabilities of information systems in precise terms. Integration Profiles specify implementations of standards that are designed to meet identified clinical needs. They enable users and vendors to state which IHE capabilities they require or provide, by reference to the detailed specifications of the IHE IT Infrastructure Technical Framework.

Integration profiles are defined in terms of IHE Actors and transactions. Actors are information systems or components of information systems that produce, manage, or act on information associated with clinical and operational activities in the enterprise. Transactions are interactions between actors that communicate the required information through standards-based messages.

Vendor products support an Integration Profile by implementing the appropriate actor(s) and transactions. A given product may implement more than one actor and more than one integration profile.

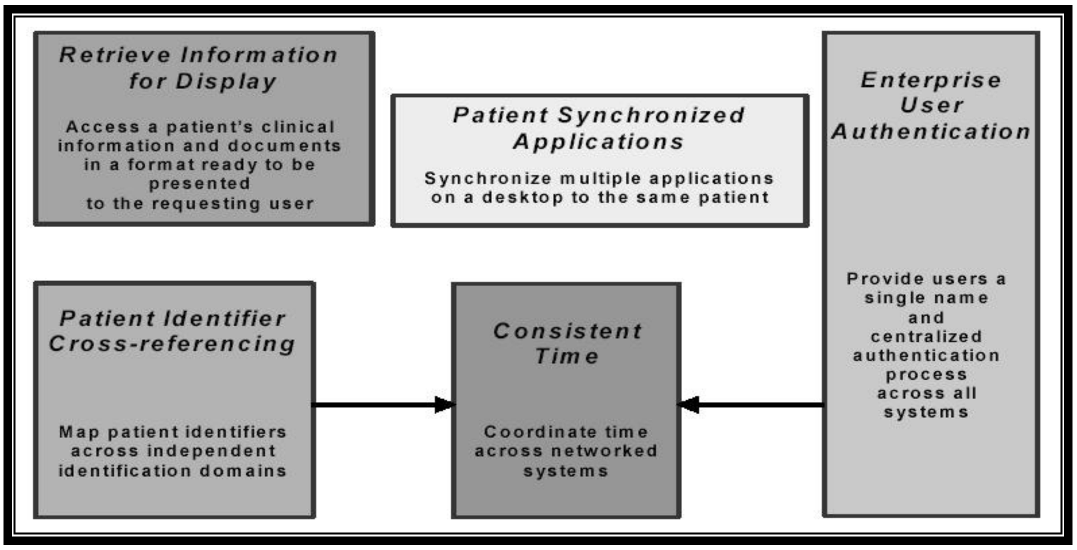
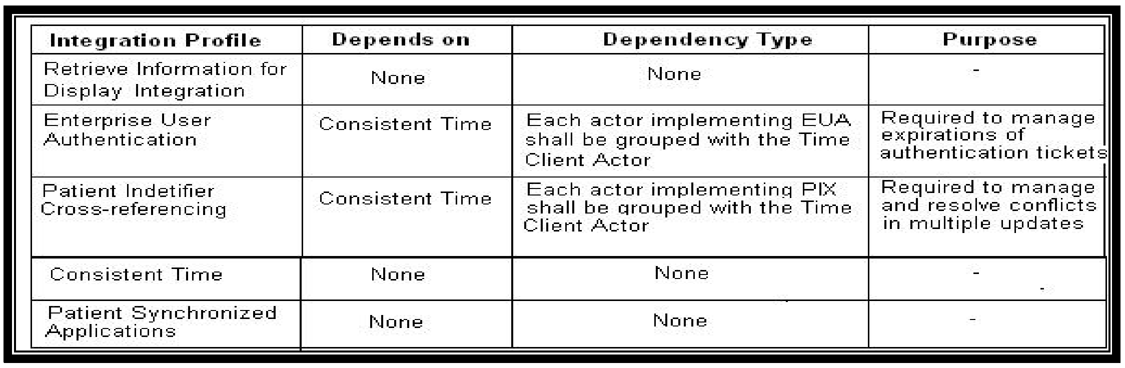

Retrieve Information for Display
Access a patient's clinical information and documents in a format ready to be presented to the requesting user
Patient Synchronized Applications
Synchronize multiple applications on a desktop to the same patient
Enterprise User Authentication
Provide users a single name and centralized authentication process across all systems
Patient Identifier Cross-referencing
Map patient identifiers across independent identification domains
Consistent Time
Coordinate time across networked systems

Figure 2.1: IHE IT Infrastructure Integration Profiles

Dependencies among IHE Integration Profiles exist when implementation of one integration profile is a prerequisite for achieving the functionality defined in another integration profile. Figure 2.1 provides a graphical view of the dependencies among IHE IT Infrastructure Integration Profiles.

The arrows in the figure point from a given integration profile to the integration profile(s) upon which it depends. Figure 2.2 defines these dependencies in tabular form.

Some dependencies require that an actor supporting one profile be grouped with one or more actors supporting other integration profiles. For example, Enterprise User Authentication (EUA) requires that different participating actors be grouped with the Time Client Actor that participates in the Consistent Time (CT) Integration Profile. The dependency exists because EUA actors must refer to consistent time in order to function properly.



Integration Profile	Depends on	Dependency Type	Purpose
Retrieve Information for Display Integration	None	None	-
Enterprise User Authentication	Consistent Time	Each actor implementing EUA shall be grouped with the Time Client Actor	Required to manage expirations of authentication tickets
Patient Indetifier Cross-referencing	Consistent Time	Each actor implementing PIX shall be grouped with the Time Client Actor	Required to manage and resolve conflicts in multiple updates
Consistent Time	None	None	-
Patient Synchronized Applications	None	None	-

Figure2.2: Integration Profiles Dependencies

To support a dependent profile, an actor must implement all required transactions in the prerequisite profiles in addition to those in the dependent profile. In some cases, the prerequisite is that the actor selects any one of a given set of profiles.

2.2 Web Services

The interoperability problem described in Chapter 1 can be solved by introducing the Web service technology to the healthcare domain. In this section Web services and the underlying standards are briefly described.

A Web service is the programmable application logic accessible using standard Internet protocols. Web services combine the best aspects of component-based development and the Web. Unlike current component technologies, Web services are not accessed via object-model-specific protocols, such as the distributed Component Object Model (DCOM), Remote Method Invocation (RMI), or Internet Inter-ORB Protocol (IIOP). Instead, Web services are accessed via ubiquitous Web protocols and data formats, such as Hypertext Transfer Protocol (HTTP) and Extensible Markup Language (XML).

There is not a standard definition of a Web service.

A definition of web service from IBM’s tutorial [21] is as follows:
“Web services are a new breed of Web applications. They are self-contained, self describing, modular applications that can be published, located, and invoked across the Web. Web services perform functions, which can be anything from simple request to complicated business processes... Once a Web service is deployed, other applications (and other Web services) can discover and invoke the deployed service.”

Web service based applications to be loosely coupled, component-oriented, cross-technology implementations. Web services can be used alone or in conjunction with other Web services to carry out a complex aggregation or a business transaction [20].

Microsoft has a couple of definitions for Web service:
“A Web service is a unit of application logic providing data and services to other applications. Applications access Web services via ubiquitous Web protocols and data

formats such as HTTP, XML, and SOAP, with no need to worry about how each Web service is implemented. Web services are a cornerstone of the Microsoft .NET programming model [36].”

Consumers of the Web service can be implemented on any platform in any programming language, as long as they can create and consume the messages defined for the Web service interface [35].

Sun provides the following definition:
“Web services are software components that can be spontaneously discovered, combined, and recombined to provide a solution to the user's problem/request. The Java language and XML are the prominent technologies for Web services [56].”

In general, a Web service is a *platform and implementation independent* software component that can be:

- *Described* using a service description language
- *Published* to a registry of services
- *Discovered* through a standard mechanism (at runtime or design time)
- *Invoked* through a declared API, usually over a network
- *Composed* with other services

Any type of application can be offered as a Web service. One important point is that a Web service need not necessarily exist on the World Wide Web, applicable to any type of Web environment: Internet, intranet, or extranet.

A standardized XML messaging and not being tied to any one operating system or programming language is vital for the web services.

Although they are not required, a web service may also have two additional (and desirable) properties:

- A web service should be self-describing. If you publish a new web service, you should also publish a public interface to the service. At a minimum, your service should include human-readable documentation so that other developers can more easily integrate your service.

- A web service should be discoverable. If you create a web service, there should be a relatively simple mechanism for you to publish this fact. Likewise, there should be some simple mechanism whereby interested parties can find the service and locate its public interface. The exact mechanism could be via a completely decentralized system or a more logically centralized registry system.

Interactions among Web services involve three types of participants: Service Provider, Service Consumer and Service Registry (Figure 2.3).

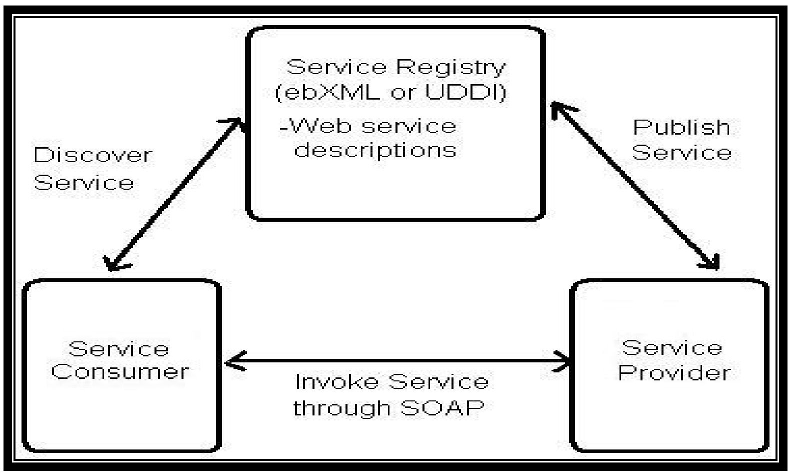


Figure 2.3: Web Services Model

Service providers are the parties that offer services. They publish the descriptions of their services in the Service Registry, which is a searchable repository of service descriptions. Service descriptions provide information on the data being exchanged, the sequence of messages for an operation, the location of the service. Service Consumers find the services by using the APIs provided by the service registries. The registry returns the description of the service which is used in invoking the service.

Considerable progress has been made in the area of Web service description and invocation. There are two almost universally accepted standards for these purposes: SOAP (Simple Object Access Protocol) [54] for invoking services and WSDL (Web Services Description Language) [66] for describing the technical specifications of the services. There are also two well-known service registry specifications, UDDI by Microsoft and IBM, and ebXML by UN/CEFACT. There are a number of implementations of these registries such as [20] and [37].

2.2.1 SOAP

SOAP is an XML-based protocol for exchanging information between computers. Although SOAP can be used in a variety of messaging systems and can be delivered via a variety of transport protocols, the initial focus of SOAP is remote procedure calls transported via HTTP. SOAP therefore enables client applications to easily connect to remote services and invoke remote methods. A client application can immediately add language translation to its feature set by locating the correct SOAP service and invoking the correct method.

SOAP therefore represents a cornerstone of the web service architecture, enabling diverse applications to easily exchange services and data. SOAP has received widespread industry support. Dozens of SOAP implementations now exist, including implementations for Java, COM, Perl, C#, and Python.

2.2.2 WSDL

WSDL is an XML based language that describes how to invoke a service. It provides information on the data being exchanged, the sequence of messages for an operation, the location of the service and the description of bindings (e.g. SOAP). WSDL descriptions are composed of interface and implementation definitions. The interface is an abstract and reusable service definition that can be referenced by multiple implementations. The implementation describes how the interface is implemented by a given service provider.

Using WSDL, a client can locate a web service and invoke any of its publicly available functions. WSDL-aware tools enable applications to easily integrate new services with little or no manual code. WSDL therefore represents a cornerstone of the web service architecture, providing a common language for describing services and a platform for automatically integrating those services.

WSDL has an XML grammar for describing web services. The specification itself is divided into six major elements: definitions, types, message, portType, binding and service.

2.2.3 Web Service Registries

As mentioned before, Interactions among Web services involve three types of participants: service provider, service registry and service consumer. Service registries are searchable repositories of Web Service descriptions. There are two well known service registries: Electronic Business XML (ebXML) Registries and the Universal Description, Discovery, Integration framework (UDDI) Registries.

2.2.3.1 ebXML Registries

Electronic Business XML [12] is an initiative from OASIS and United Nations Centre for Trade Facilitation and Electronic Business UN/CEFACT [63]. ebXML aims to provide the exchange of electronic business data in Business-to-Business and Business-to-Customer environments. The vision of ebXML is to create a single set of internationally agreed upon technical specification that consists of common XML semantics and related document structures to facilitate global trade.

The ebXML architecture provides the following functional components:

- Registry/Repository: A registry is a mechanism where business documents and relevant metadata can be registered, and can be retrieved as a result of a query. A registry can be established by an industry group or standards organization. A repository is a location (or a set of distributed locations) where a document pointed at by the registry resides and can be retrieved by conventional means (e.g., http or ftp).
- Trading Partner Information: The Collaboration Protocol Profile (CPP) provides both DTD and XML Schema definitions of an XML document that specifies the details of how an organization is able to conduct business electronically. It specifies such items as how to locate contact and other information about the organization, the types of network and file transport protocols it uses, network addresses, security implementations, and how it does business (a reference to a Business Process Specification). The Collaboration Protocol Agreement (CPA) specifies the details of how two organizations have agreed to conduct business electronically. It is formed by combining the CPPs of the two organizations.

- Business Process Specification Schema: The Business Process Specification Schema provides the definition of an XML document (in the form of an XML DTD) that describes how an organization conducts its business. While the CPA/CPP deals with the technical aspects of how to conduct business electronically, the Business Process Specification Schema deals with the actual business process.
- Messaging Service: ebXML messaging service provides a standard way to exchange messages between organizations reliably and securely. It does not dictate any particular file transport mechanism, such as SMTP, HTTP, or FTP.
- Core Components: ebXML provides a core component architecture where a core component is a general building block that basically can be used to form business documents.

An ebXML registry is a mechanism where business documents and relevant metadata can be registered and later can be retrieved as a result of a query. Registry information model [12] defines the types of objects stored in the registry and the way they are organized.

The ebXML registry provides a set of services to enable information sharing among interested parties to facilitate business process integration. The shared information is maintained in a repository and managed by the ebXML Registry Services [13].

An important characteristic of ebXML registry is that it allows metadata to be stored in the registry. This is achieved through a “classification" mechanism, called ClassificationScheme which helps to classify the objects in the registry. ClassificationScheme defines a hierarchy of ClassificationNodes. The nodes in this hierarchy are related with registry objects through Classification objects. A Classification instance classifies a RegistryObject instance by referencing a node defined within a particular classification scheme. An internal classification always references the node directly, by its id, while an external classification references the node indirectly by specifying a representation of its value that is unique within the external classification scheme.

2.2.3.2 The Universal Description, Discovery, Integration Framework (UDDI) Registries

UDDI is jointly proposed by IBM, Microsoft and Ariba. It is a service registry architecture that presents a standard way for businesses to build a registry, discover each other, and describe how to interact over the Internet.

Currently IBM and Microsoft are running public registries. UDDI defines a programmatic interface for publishing (publication API) and discovering (inquiry API) Web services. The core component of UDDI is the business registry, an XML repository where businesses advertise services so that other businesses can find them. Conceptually, the information provided in a UDDI registry consists of white pages (contact information), yellow pages (industrial categorization) and green pages (technical information about services).

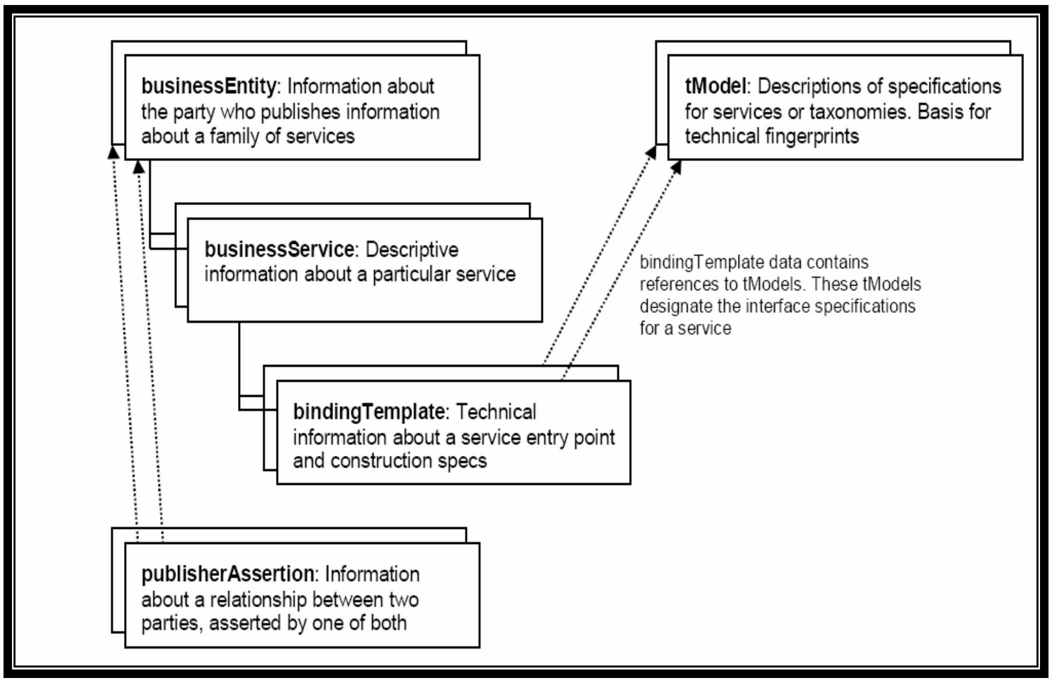

businessEntity: Information about the party who publishes information about a family of services
tModel: Descriptions of specifications for services or taxonomies. Basis for technical fingerprints
businessService: Descriptive information about a particular service
bindingTemplate data contains references to tModels. These tModels designate the interface specifications for a service
bindingTemplate: Technical information about a service entry point and construction specs
publisherAssertion: Information about a relationship between two parties, asserted by one of both

Figure 2.4: UDDI Core Types

The UDDI information model, defined through an XML schema, identifies five core types of information as shown in Figure 2.4. These core types are business, service, binding, service specifications information and relationship information between two parties.

In UDDI, services use category bags for semantic information. An item in a category bag contains a tModel key and an associated OverviewDoc element. tModels provide the ability to describe compliance with a specification, a concept, or a shared design. When a particular specification is registered with the UDDI as a tModel, it is assigned a unique key, which is then used in the description of service instances to indicate compliance with the specification.

The specification is not included in the tModel itself. The "OverviewDoc" and "OverviewURL" elements of tModels are used to point at the actual source of a specification. More precisely, the use of tModels in UDDI is two-fold [61]:

- Defining the technical fingerprint of services: The primary role that a tModel plays is to represent a technical specification on how to invoke a registered service, providing information on the data being exchanged, the sequence of messages for an operation and the location of the service. Examples of such technical specifications include WSDL descriptions and RosettaNet PIPs. Note that the tModel mechanism describes only the signature of the services; it does not provide any information on the functionality of the service.
- Providing abstract namespace references: In UDDI, businesses, services and tModels can specify the categories to which they belong in their category bags. Categorization facilitates to locate businesses and services by relating them to some well-known industry, product or geographic categorization code set. Currently UDDI uses the North American Industrial Classification Scheme (NAICS) taxonomy for describing what a business does; the Universal Standard Products and Services Classification (UN-SPSC) for describing products and services offered; and ISO3166, a geographical taxonomy for determining where a business is located. It should be noted that any number of categories could be referenced in category bags.

It is not possible to query the attributes of tModels since they do not have any formal description. Also expressing complementary services is not possible because UDDI does not provide a mechanism to define relationships between tModels.

To invoke a service in an UDDI registry, it is necessary to know either its key or the business the service belongs. If the service key is not known, the service is located through its business in the UDDI registry using the APIs provided by the registry (e.g. IBM's UDDI4J API), and the corresponding WSDL description is accessed. After gathering all the information about the service to be invoked, the necessary SOAP calls are made as specified in the WSDL description.

2.3 Semantic, Ontology, Web Service Semantics

Considering the volume of information available on the Web, it is not possible to deal with it manually. This lack of means for applications to automatically process and meaningfully share information on the Web, constitutes one of the main barriers to the so called “Semantic Web” [57].

In order to fully exploit the opportunities brought by Web, metadata (semantic representation) of the resources need to be made explicit to make it automatically processable. Metadata can be defined as “structured data about data”. For example, library card catalogs represent a well-established type of metadata that help discovering resources like books and journals in the libraries.

A well-defined metadata description language gives the ability to automatically process the description to obtain the metadata. However to interpret the metadata automatically through a program or a software agent, its meaning (semantics) must also be known. Meaning of data is given through domain specific ontologies. Ontology is a schema for a domain, in other words, it is the explicit formal specification of the terms in the domain and relations among them.

One of the most important initiatives for describing the semantic of Web resources is the Resource Description Framework (RDF) developed by the World Wide Web Consortium [65]. RDF fixes the syntax and structure of describing metadata through RDF Syntax [48] and it allows meaning to be defined and associated with data through RDF Schema [47], which gives facilities to define domain specific ontologies.

By observing that the Semantic Web should enable greater access not only to content but also to services on the Web, a semantic markup for web services, based on DAML+OIL, has also been defined in [8]. Recently, World Wide Web Consortium [65] has proposed OWL, Web Ontology Language, [39] for facilitating greater machine interpretability of Web content. It should be noted that OWL is a revision of the DAML+OIL web ontology language incorporating lessons learned from the design and application of DAML+OIL.

2.3.1 Ontology

An ontology defines a common vocabulary for researchers who need to share information in a domain. It includes machine-interpretable definitions of basic concepts in the domain and relations among them. By developing ontologies it will be possible:

- To share common understanding of the structure of information among people or software agents
- To enable reuse of domain knowledge
- To make domain assumptions explicit
- To separate domain knowledge from the operational knowledge
- To analyze domain knowledge

An ontology is generally composed of the following items:

- a formal explicit description of concepts in a domain of discourse (i.e. classes which are also called concepts),
- properties of each concept describing various features and attributes of the concept (i.e. slots which are also called roles or properties) and,
- restrictions on slots (i.e. facets which are also called role restrictions).

An ontology together with a set of individual instances of classes constitutes a knowledge base. The knowledge included in an ontology can be shared because the use of this knowledge is consensual, that is, it has been accepted by a group, not by a single individual.

Developing an ontology includes:

- defining classes in the ontology,
- arranging the classes in a taxonomic (subclass-superclass) hierarchy,
- defining slots and describing allowed values for these slots,
- filling in the values for slots for instances.

A knowledge base can be created by defining individual instances of these classes filling in specific slot value information and additional slot restrictions.

Classes are the focus of most ontologies. Classes describe concepts in the domain. For example, a class of wines represents all wines. Specific wines are instances of this class.

A class can have **subclasses** that represent concepts that are more specific than the superclass. For example, we can divide the class of all wines into red, white, and rosé wines. Slots describe properties of classes and instances.

There are libraries of reusable ontologies on the Web and in the literature such as the Ontolingua ontology library, or the DAML ontology library [7].

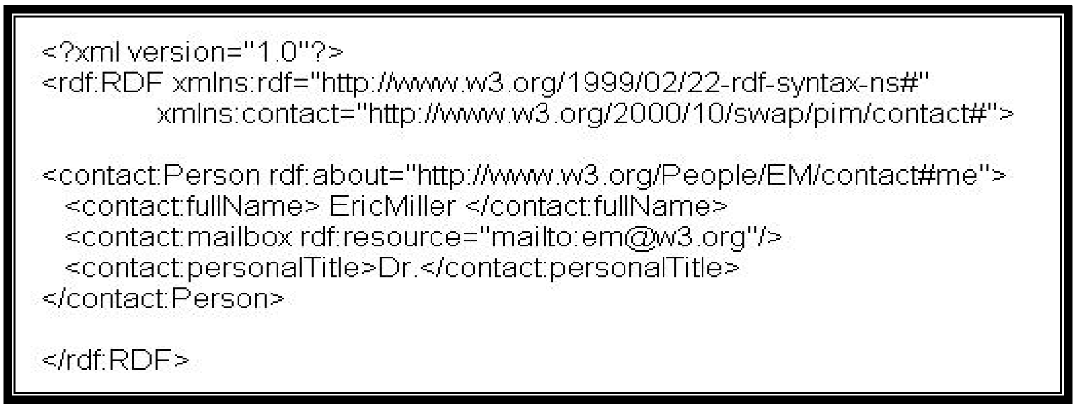
In order for ontologies to have the maximum impact, they need to be widely shared. In order to minimize the intellectual effort involved in developing an ontology they need to be re-used. In the best of all possible worlds they need to be composed and mapped to each other. There is a considerable amount of research in Ontology Mapping.

2.3.2 Resource Description Framework (RDF)

The Resource Description Framework (RDF) is a language for representing information about resources in the World Wide Web [65]. It is particularly intended for representing metadata about Web resources, such as the title, author, and modification date of a Web page. RDF can also be used to represent information about things that can be *identified* on the Web, even when they cannot be directly *retrieved* on the Web.

The basic RDF data model expressing the syntax and structure consists of three object types: *resources* which are the things being described by RDF, *properties* which are specific aspects, attributes or relations describing a resource and *statements* that assign a *value* to a property of a resource.

A resource can be any object that is uniquely identifiable by a Uniform Resource Identifier (URI). For example, "http://www.w3.org/People/EM/contact#me" is a resource in Figure 2.5.



```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:contact="http://www.w3.org/2000/10/swap/pim/contact#">

<contact:Person rdf:about="http://www.w3.org/People/EM/contact#me">
  <contact:fullName> EricMiller </contact:fullName>
  <contact:mailbox rdf:resource="mailto:em@w3.org"/>
  <contact:personalTitle>Dr.</contact:personalTitle>
</contact:Person>

</rdf:RDF>
```

Figure 2.5: An RDF/XML Example

Values may be atomic in nature or can be other resources, which in turn may have their own properties. A *value* can just be a string, for example “Eric Miller” or it can be another resource, for example “mailto:em@w3.org”. A collection of these properties that refers to the same resource is called a description.

In fact, RDF models descriptions and statements in terms of a directed graph consisting of nodes and arcs. The nodes describe resources, and properties represented by directed arcs that connect subject nodes to object nodes. A property arc is interpreted as an attribute, relationship or predicate of the resource, with a value given by the object node. Figure 2.6 shows the graph representation of the sample RDF description given in Figure 2.5.

The namespace URI in a namespace declaration, like *xmlns:rdf=”http://www.w3.org/1999 /rdf-syntax-ns"* in the example of Figure 2.5, is a globally unique identifier for a particular schema, and it indicates that *rdf:Description* tag is defined in this schema.

Once the meanings of the attributes are known, this description becomes machine processable. Any program (having a prior knowledge of the syntax and semantics of RDF) can parse the description, extract meta-data, interpret it and use it for its purposes. Note however that even when an application knows only the RDF Syntax and has no understanding of a particular schema will still be able to parse the description into the property-type and corresponding values and will still be able to deduce that a resource is being described with a certain property and a property value.

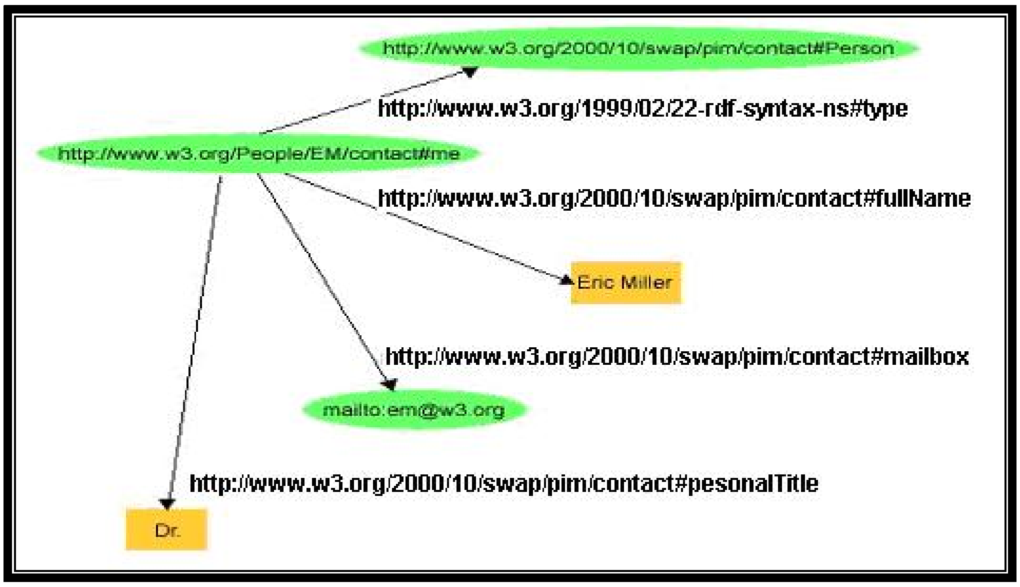

http://www.w3.org/2000/10/swap/pim/contact#Person
http://www.w3.org/1999/02/22-rdf-syntax-ns#type
http://www.w3.org/People/EM/contact#me
http://www.w3.org/2000/10/swap/pim/contact#fullName
Eric Miller
http://www.w3.org/2000/10/swap/pim/contact#mailbox
mailto:em@w3.org
http://www.w3.org/2000/10/swap/pim/contact#pesonalTitle
Dr.

Figure 2.6: An RDF Graph

The declaration of properties and their corresponding semantics are defined in the context of RDF as a schema. [47] defines a Schema Specification Language for this purpose and provides a basic type system for use in RDF Models. It defines resources and properties such as “Class” and “subClassOf” that are used in specifying application-specific schemas.

The RDF Schema mechanism defines the following core *classes*:

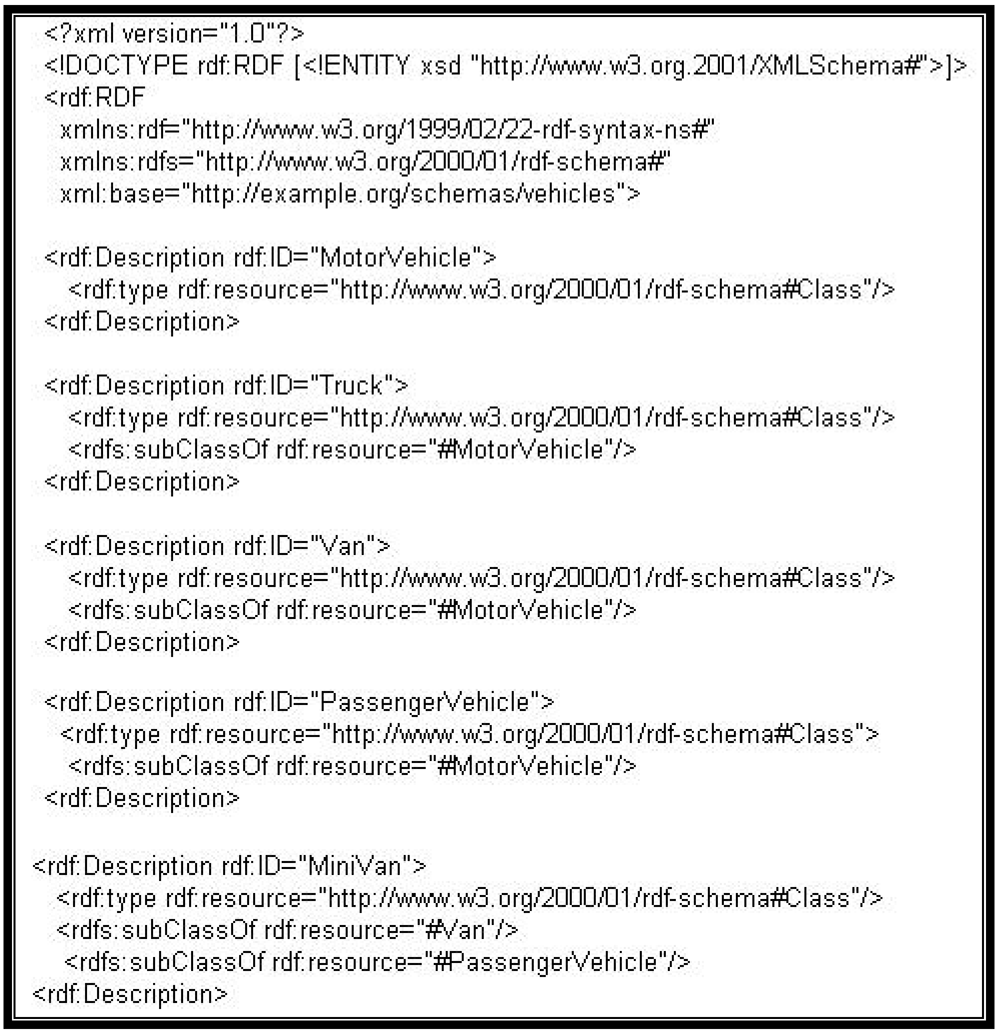
- ***rdfs:Resource***: This class is the top class, i.e., all resources are considered to be instances of this class.
- ***rdf:Property***: This class represents the subset of RDF resources that are properties.
- ***rdfs:Class***: The class *rdfs:Class* represents the subset of RDF resources that are classes. An RDF schema is organized into a collection of classes.

The RDF Schema mechanism also defines the following core *properties*:

- ***rdf:type***: This property indicates which class a resource belongs to. Resources may be instances of one or more classes by multiple inheritance. Note that when a schema defines a new class, the resource representing that class must have an *rdf:type* property whose value is the resource *rdfs:Class*. The resource known as *rdfs:Class* is itself a resource of *rdf:type rdfs:Class*.
- ***rdfs:subClassOf***: Classes are often organized in a hierarchy and *rdfs:subClassOf* property is defined to express subclass relationship among classes.

- ***rdfs:subPropertyOf***: This core property specifies that one property is a specialization of another.

The example provided in Figure 2.7 defines such a class hierarchy where “MotorVehicle” is the subclass of the top level class “resource”, and “Truck” is a subclass of “MotorVehicle”.



```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org.2001/XMLSchema#">]>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://example.org/schemas/vehicles">

<rdf:Description rdf:ID="MotorVehicle">
   <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
<rdf:Description>

<rdf:Description rdf:ID="Truck">
   <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
   <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
<rdf:Description>

<rdf:Description rdf:ID="Van">
   <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
   <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
<rdf:Description>

<rdf:Description rdf:ID="PassengerVehicle">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class">
   <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
<rdf:Description>

<rdf:Description rdf:ID="MiniVan">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:subClassOf rdf:resource="#Van"/>
   <rdfs:subClassOf rdf:resource="#PassengerVehicle"/>
<rdf:Description>
```

Figure 2.7: The Vehicle Class Hierarchy in RDF/XML

2.3.3 Web Ontology Language (OWL)

OWL is intended to be used when the information contained in documents needs to be processed by applications, as opposed to situations where the content only needs to be presented to humans [65]. OWL can be used to explicitly represent the meaning of terms in vocabularies and the relationships between those terms. OWL has more facilities for expressing meaning and semantics than XML, RDF, and RDF-S, and thus OWL goes beyond these languages in its ability to represent machine interpretable content on the Web. OWL is a revision of the DAML+OIL web ontology language incorporating lessons learned from the design and application of DAML+OIL [39].

W3C has expressed the need for a Web Ontology language on top of the existing XML, and RDF. These can be summarized as follows: “Ontologies are critical for applications that want to search across or merge information from diverse communities. Although XML DTDs and XML Schemas are sufficient for exchanging data between parties who have agreed to definitions beforehand, their lack of semantics prevent machines from reliably performing this task given new XML vocabularies. RDF and RDF Schema begin to approach this problem by allowing simple semantics to be associated with identifiers. However, in order to achieve interoperation between numerous, autonomously developed and managed schemas, richer semantics are needed. For example, RDF Schema cannot specify that the Person and Car classes are disjoint, or that a string quartet has exactly four musicians as members.”

- OWL has been designed to meet this need for a Web Ontology Language. OWL is part of the growing stack of W3C recommendations related to the Semantic Web [65].
- XML provides a surface syntax for structured documents, but imposes no semantic constraints on the meaning of these documents.
- XML Schema is a language for restricting the structure of XML documents and also extends XML with data types.
- RDF is a data model for objects ("resources") and relations between them provides a simple semantics for this data model, and these data models can be represented in XML syntax.
- RDF Schema is a vocabulary for describing properties and classes of RDF resources, with a semantics for generalization-hierarchies of such properties and classes.
- OWL adds more vocabulary for describing properties and classes: among others, relations between classes (e.g. disjointness), cardinality (e.g. "exactly one"), equality, richer typing of properties and characteristics of properties (e.g. symmetry), and enumerated classes.

OWL provides three increasingly expressive sublanguages designed for use by specific communities of implementers and users:

- *OWL Lite* supports those users primarily needing a classification hierarchy and simple constraints. For example, while it supports cardinality constraints, it only permits cardinality values of 0 or 1. It should be simpler to provide tool support for OWL Lite than its more expressive relatives, and OWL Lite provides a quick migration path for thesauri and other taxonomies.
- *OWL DL* supports those users who want the maximum expressiveness while retaining computational completeness (all conclusions are guaranteed to be computable) and decidability (all computations will finish in finite time). OWL DL includes all OWL language constructs, but they can be used only under certain restrictions (for example, while a class may be a subclass of many classes, a class cannot be an instance of another class).
- *OWL Full* is meant for users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees. For example, in OWL Full a class can be treated simultaneously as a collection of individuals and as an individual in its own right. OWL Full allows an ontology to augment the meaning of the pre-defined (RDF or OWL) vocabulary.

2.3.4 Web Service Semantics

Well accepted standards like Web Services Description Language [66] and Simple Object Access Protocol [54] make it possible to dynamically invoke Web services. That is, when the semantic of the service to be used is known, its WSDL description can be accessed by a program which uses the information in the WSDL description like the interface, binding and operations to dynamically access the service. However to be able to exploit the Web services to their full potential, their semantic information should be available. In this way, not only humans can query the service registries to discover and compose services, but more importantly, this also opens up the way for software agents to automatically exploit Web services on behalf of users since formal semantic descriptions are machine processable [11].

OWL-S (formerly DAML-S) is an OWL Web service ontology which supplies Web service providers with a core set of markup language constructs for describing the properties and capabilities of their Web services in unambiguous, computer-interpretable form. OWL-S markup of Web services aims to facilitate the automation of Web service tasks including automated Web service discovery, execution, interoperation, composition and execution monitoring [40].

OWL-S classifies the Web Services in to two categories as:

- “primitive” in the sense that they invoke only a single Web-accessible computer program, sensor, or device that does not rely upon another Web service, and there is no ongoing interaction between the user and the service, beyond a simple response.
- “complex” that are composed of multiple primitive services, often requiring an interaction or conversation between the user and the services, so that the user can make choices and provide information conditionally.

OWL-S is meant to support both categories of services, in order to facilitate the following functionalities:

- *Automatic Web service discovery* involves the automatic location of Web services that provide a particular service and that adhere to requested constraints.
- *Automatic Web service invocation* involves the automatic execution of an identified Web service by a computer program or a software agent.
- *Automatic Web service composition and interoperation* involves the automatic selection, composition and interoperation of Web services to perform some tasks, given a high-level description of an objective.
- *Automatic Web service execution monitoring:* Individual services and, even more, compositions of services, will often require some time to execute completely. Users may want to know during this period what the status of their request is, or their plans may have changed requiring alterations in the actions the software agent takes.

OWL-S provides an upper ontology for Service definition (Figure 2.8). The top level class in this ontology is the “Service” class.

Service class has the following three properties:

- *presents:* The range of this property is ServiceProfile class. That is, the class Service presents a ServiceProfile to specify what the service provides for its users as well as what the service requires from its users.

- *describedBy:* The range of this property is ServiceModel class. That is, the class Service is describedBy a ServiceModel to specify how it works.
- *supports:* The range of this property is ServiceGrounding. That is, the class Service supports a ServiceGrounding to specify how it is used.

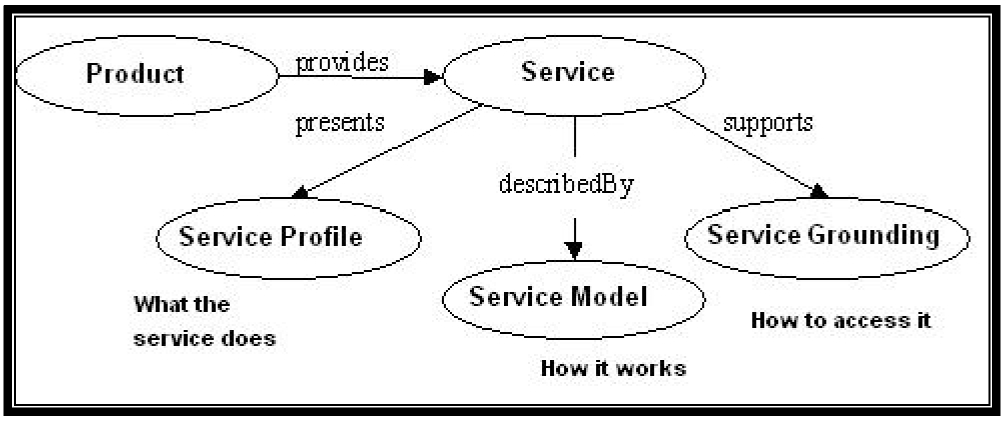


Figure 2.8: OWL-S Top Level Service Ontology

2.4 Peer-To-Peer (P2P) Computing

In this section, a brief summary of P2P computing paradigm is introduced. It is required to have a general knowledge of P2P systems, since the generated sample Web services during this study is annotated in Artemis P2P Network. The concepts described in this section are the general aspects of P2P computing.

Peer-to-Peer (P2P) refers to a class of systems and applications that employ distributed *resources* to perform a *critical function* in decentralized manner [43]. Computing power, data, network bandwidth and presence (computers, human and other resources) are all thought in term *resources.* The critical function can be distributed computing, data/content sharing, communication and collaboration.

P2P gained popularity with Napster’s support for music sharing on the Web. Then it become an important technique in various areas, such as distributed computing. P2P has received the attention of both industry and academia.

P2P is based on the principle that the world will be connected and widely distributed, and it will not be possible to manage things in centralized manner. P2P computing is an alternative to the centralized and client-server models of computing (Figure 2.9). In its purest form, P2P model has no concept of server; all participants are peers. The term "peer", is defined as "like each other".

A P2P system then can be viewed as peers depend on other peers, and the communication among the peers. Peers are not wholly controlled by each other or some authority.

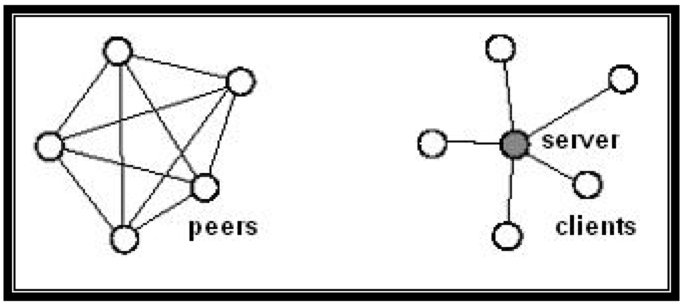


Figure 2.9: High Level View of P2P vs. Client-Server

2.4.1 Goals

Selecting a P2P approach is driven by one or more of the following goals [41]:

- **Cost sharing/reduction:** Opposite to the centralized systems that load the cost of the system to the servers, P2P architecture spread the cost over all the peers.
- **Resource aggregation and interoperability:** Each node (peer) in the P2P system brings with it certain resources such as compute power or storage space. Applications that require huge amounts of these resources naturally choose P2P approach.
- **Increased autonomy:** Most important examples of this property are various file sharing systems, such as Napster, Gnutella. Users of those systems are able to get files that would not be available at any central.
- **Anonymity/privacy:** With a central server it is hard to ensure anonymity since the client is identified by a server, at least from internet addresses. P2P structure helps users to avoid having to provide any information about them. FreeNet is a good example, which uses a forwarding algorithm for messages to ensure that the original requester and sender cannot be tracked.
- **Dynamism:** It is a must for P2P systems to cope with dynamism. Since resources as compute nodes, will be entering and leaving the system continuously. So if an

application is intended to support a highly dynamic environment, the P2P approach is a natural choise.

- **Enabling communication and collaboration in ad-hoc environments:** Ad-hoc environments such that members arrive and leave depending on perhaps their current physical locations or their interests, are example areas that P2P fits.

2.4.2 P2P Systems

P2P System can be classified as shown in Figure 2.10.

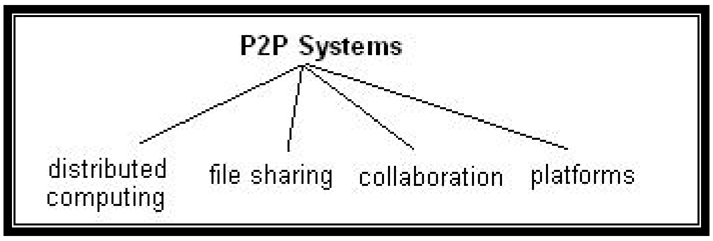


Figure 2.10: A Taxonomy of P2P systems

The well known examples (applications and systems) for each classification are shown in Figure 2.11.

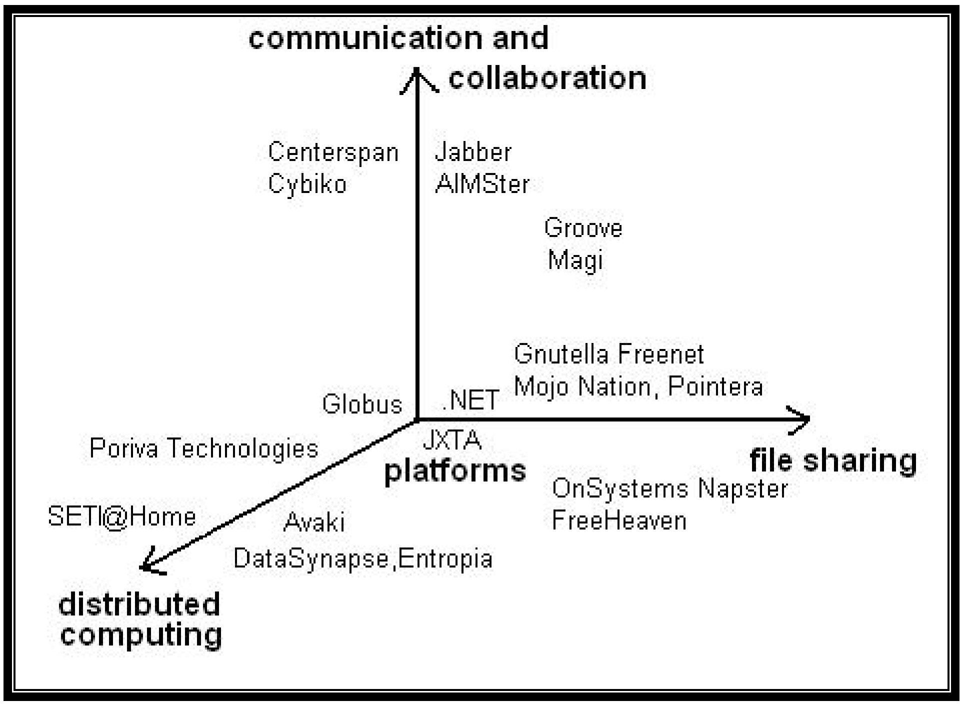


Figure 2.11: A classification of P2P Systems Based on the taxonomy in Figure 2.10

2.4.3 Characteristics of P2P Systems

The followings are the most important characteristics that P2P Systems support:

- **Decentralization:** In pure P2P systems, every peer is an equal participant, and such systems are said fully decentralized. The node joins the network of peers by establishing a connection with at least one peer that is currently in the network. However implementation of pure P2P systems is difficult since there does not exist a centralized server that views all the peers in the network or the files they provide. So many P2P systems are built as hybrid approach.
- **Scalability:** Scalability is limited by factors such as the amount of centralized operations (e.g., synchronization and coordination) that needs to be performed, the amount of state that needs to be maintained etc. Napster solved scalability problem by having the peers directly download music files from the peers that possess the requested document. As a result, Napster was able to scale up to over 6 million users at the peak of its service.
- **Anonymity:** An important goal of anonymity is to allow people to use systems without concern for issues.
- **Self-Organization:** In P2P systems, self-organization is needed because of scalability, fault resilience, intermittent connection of resources, and the cost of ownership. P2P systems can scale unpredictably in terms of the number of systems, number of users, and the load.
- **Cost of Ownership:** One of the premises of P2P computing is shared ownership. Shared ownership reduces the cost of owning the systems and the content, and the cost of maintaining them.
- **Ad-Hoc Connectivity:** P2P systems and applications in distributed computing need to be aware of ad-hoc nature and be able to handle systems joining and withdrawing from the pool of available P2P systems. While in traditional distributed systems, this was an exceptional event, in P2P systems it is considered as usual.
- **Performance:** P2P systems aim to improve performance by aggregating distributed storage capacity (e.g., Napster, Gnutella) and computing cycles (e.g., SETI@Home) of devices spread across a network. Because of the decentralized nature of these models, performance is influenced by three types of resources: processing, storage, and networking.

- **Fault Resilience:** One of the primary design goals of a P2P system is to avoid a central point of failure (which is obvious in client-server architecture).A challenging aspect of P2P systems is that the system maintenance responsibility is completely distributed and needs to be addressed by each peer to ensure availability. This is quite different from client-server systems, where availability is a server-side responsibility.
- **Interoperability:** Although many P2P systems already exist there is still no support to enable these P2P systems to interoperate with each other. The P2P Working Group [42] is an attempt to gather the community of P2P developers together and establish common ground by writing reports and white papers that would enable common understanding among P2P developers.

2.4.4 JXTA Platform

JXTA is a set of open, generalized peer-to-peer (P2P) protocols that allow any connected device on the network; from cell phone to PDA, from PC to server, to communicate and collaborate as peers [45]. The JXTA protocols are independent of any programming language, and multiple implementations (called bindings in Project JXTA) exist for different environments. The JXTA project was created by Sun on April 25, 2001 and was intended to be a platform on which to develop a wide range of distributed computing applications.

The vision of the JXTA project is to provide an open, innovative collaboration platform that supports a wide range of distributed computing applications and enables them to run on any device with a digital heartbeat [44].

Primary goal of JXTA is to provide a platform with the basic functions necessary for a P2P network. In addition, JXTA technology seeks to overcome potential shortcomings in many of the existing P2P systems:

- *Interoperability*: JXTA technology is designed to enable peers providing various P2P services to locate each other and communicate with each other.
- *Platform independence*; JXTA technology is designed to be independent of programming languages, transport protocols, and deployment platforms.
- *Ubiquity*; JXTA technology is designed to be accessible by any device with a digital heartbeat, not just PCs or a specific deployment platform.

JXTA provides a common set of open protocols and an open source reference implementation for developing peer-to-peer applications. The JXTA protocols standardize the manner in which peers:

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

The JXTA protocols are designed to be independent of programming languages, and independent of transport protocols. The protocols can be implemented in the Java programming language, C/C++, Perl, and numerous other languages. They can be implemented on top of TCP/IP, HTTP, Bluetooth, HomePNA, or other transport protocols. Because the protocols are independent of both programming language and transport protocols, heterogeneous devices with completely different software stacks can interoperate with one another.

2.4.4.1 JXTA Architecture

The Project JXTA software architecture is divided into three layers, as shown in Figure 2.12 [44];

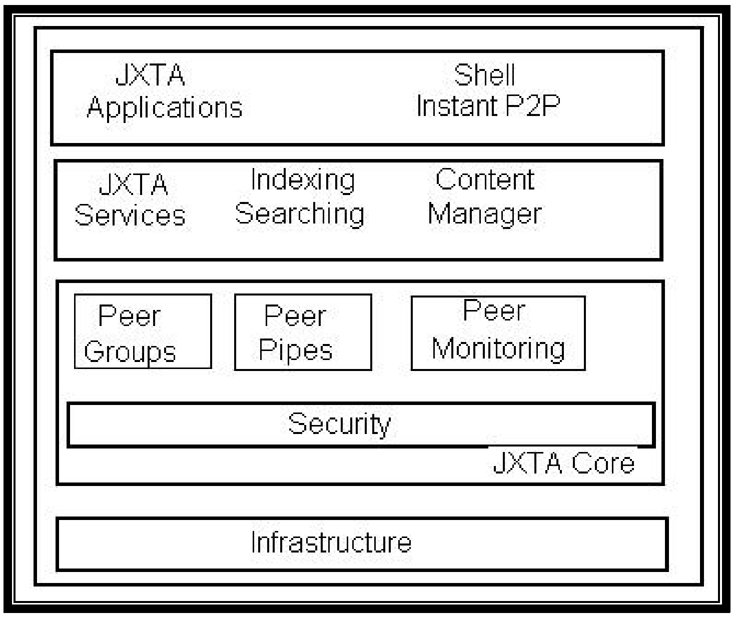


Figure 2.12: JXTA Architecture

- *Platform Layer (JXTA Core)*: Encapsulates minimal and essential primitives that are common to P2P networking. It includes building blocks to enable key mechanisms for P2P applications, including discovery, transport (including firewall handling), the creation of peers and peer groups, and associated security primitives.
- *Services Layer*: Includes network services that may not be absolutely necessary for a P2P network to operate, but are common or desirable in the P2P environment. Examples of network services include searching and indexing, directory, storage systems, file sharing, distributed file systems, resource aggregation and renting, protocol translation, authentication, and PKI (Public Key Infrastructure) services.
- *Applications Layer*: Includes implementation of integrated applications, such as P2P instant messaging, document and resource sharing, entertainment content management and delivery, P2P Email systems, distributed auction systems, and many others.

The boundary between services and applications is not rigid. An application to one customer can be viewed as a service to another customer. The entire system is designed to be modular, allowing developers to pick and choose a collection of services and applications that suits their needs.

2.4.4.2 JXTA Concepts

The JXTA terminology and the primary components of the JXTA platform is described in this section.

2.4.4.2.1 Peers

A *peer* is any networked device that implements one or more of the JXTA protocols. Peers can include sensors, phones, and PDAs, as well as PCs, servers, and supercomputers. Each peer operates independently and asynchronously from all other peers, and is uniquely identified by a Peer ID.

Peers publish one or more network interfaces for use with the JXTA protocols. Each published interface is advertised as a *peer endpoint,* which uniquely identifies the network interface. Peer endpoints are used by peers to establish direct point-to-point connections between two peers.

2.4.4.2.2 Peer Groups

A *peer group* is a collection of peers that have agreed upon a common set of services. Peers self-organize into peer groups, each identified by a unique peer group ID. Each peer group can establish its own membership policy from open (any peer can join) to highly secure and protected (sufficient credentials are required to join).

Peers may belong to more than one peer group simultaneously. By default, the first group that is instantiated is the **Net Peer Group** [44]. All peers belong to the Net Peer Group. Peers may elect to join additional peer groups.

A peer group provides a set of services called *peer group services*. JXTA defines a core set of peer group services. Additional services can be developed for delivering specific services. In order for two peers to interact via a service, they must both be part of the same peer group.

The core peer group services include the following [44]:

- *Discovery Service*; used by peer members to search for peer group resources, such as peers, peer groups, pipes and services.
- *Membership Service* used by current members to reject or accept a new group membership application. Peers wishing to join a peer group must first locate a current member, and then request to join. The application to join is either rejected or accepted by the collective set of current members. The membership service may enforce a vote of peers or elect a designated group representative to accept or reject new membership applications.
- *Access Service;* used to validate requests made by one peer to another. The peer receiving the request provides the requesting peers credentials and information about the request being made to determine if the access is permitted. (Note: not all actions within the peer group need to be checked with the access service; only those actions which are limited to some peers need to be checked.)
- *Pipe Service* used to create and manage pipe connections between the peer group members.
- *Resolver Service;* used to send generic query requests to other peers. Peers can define and exchange queries to find any information that may be needed (e.g., the status of a service or the state of a pipe endpoint).

- *Monitoring Service;* used to allow one peer to monitor other members of the same peer group.

Not all the above services must be implemented by every peer group. A peer group is free to implement only the services it finds useful, and rely on the default net peer group to provide generic implementations of non-critical core services.

2.4.4.2.3 Modules

JXTA modules are an abstraction used to represent any piece of "code" used to implement behaviour in the JXTA world. Network services are the most common example of behaviour that can be instantiated on a peer. For instance, modules can be used to represent different implementations of a network service on different platforms, such as the Java platform [29], Microsoft Windows [34], or the Solaris Operating Environment [55].

The ability to describe and publish platform independent behaviour is essential to support peer groups composed of heterogeneous peers. The module advertisements enable JXTA peers to describe a behaviour in a platform-independent manner. The JXTA platform uses module advertisements to self-describe itself.

2.4.4.2.4 Pipes

JXTA peers use *pipes* to send messages to one another. Pipes are an asynchronous and unidirectional message transfer mechanism used for service communication. Pipes are indiscriminate; they support the transfer of any object, including binary code, data strings, and objects.

The pipe endpoints are referred to as the *input pipe* (the receiving end) and the *output pipe* (the sending end). Pipe endpoints are dynamically bound to peer endpoints at runtime. Pipes are virtual communication channels and may connect peers that do not have a direct physical link. In this case, one or more intermediary peer endpoints are used to relay messages between the two pipe endpoints.

2.4.4.2.5 Messages

A message is an object that is sent between JXTA peers; it is the basic unit of data exchange between peers. Messages are sent and received by the Pipe Service and by the Endpoint Service. Typically, applications use the Pipe Service to create, send, and receive messages.

There are two representations for messages: XML and binary. The use of XML messages to define protocols allows many different kinds of peers to participate in a protocol. Because the data is tagged, each peer is free to implement the protocol in a manner best-suited to its abilities and role.

2.4.4.2.6 Advertisements

All JXTA network resources, such as peers, peer groups, pipes, and services, are represented by an *advertisement*. Advertisements are language-neutral metadata structures represented as XML documents. The JXTA protocols use advertisements to describe and publish the existence of peer resources. Peers discover resources by searching for their corresponding advertisements, and may cache any discovered advertisements locally.

Each advertisement is published with a lifetime that specifies the availability of its associated resource. Lifetimes enable the deletion of obsolete resources without requiring any centralized control. An advertisement can be republished (before the original advertisement expires) to extend the lifetime of a resource.

2.4.4.2.7 Security

JXTA peers operate in a role-based trust model, in which an individual peer acts under the authority granted to it by another trusted peer to perform a particular task. Five basic security requirements must be provided:

- *Confidentiality; g*uarantees that the contents of a message are not disclosed to unauthorized individuals.
- *Authentication;* guarantees that the sender is who he or she claims to be.
- *Authorization*; guarantees that the sender is authorized to send a message.
- *Data integrity;* guarantees that the message was not modified accidentally or deliberately in transit.

- *Refutability;* guarantees that the message was transmitted by a properly identified sender and is not a replay of a previously transmitted message.

XML messages provide the ability to add metadata such as credentials, certificates, digests, and public keys to JXTA messages, enabling these basic security requirements to be met.

2.4.4.2.8 IDs

Peers, peer groups, pipes and other JXTA resources need to be uniquely identifiable. A JXTA ID uniquely identifies an entity and serves as a canonical way of referring to that entity. Currently, there are six types of JXTA entities which have JXTA ID types defined: peers, peer group, pipes, contents, module classes, and module specifications.

An example JXTA peer ID is:
Urn:jxta:uuid-
59616261646162614A78746150325033F3BC76FF13C2414CBC0AB663666DA53903

An example JXTA pipe ID is:
Urn:jxta:uuid-
59616261646162614E504720503250338E3E786229EA460DADC1A176B69B731504

Unique IDs are generated randomly by the corresponding JXTA platform binding. There are two special reserved JXTA IDs: the NULL ID and the Net Peer Group ID.

2.4.4.3 JXTA Network Architecture

The JXTA network is an ad hoc, multi-hop, and adaptive network composed of connected peers. Connections in the network may be transient, and message routing between peers is nondeterministic. Peers may join or leave the network at any time, and routes may change frequently.

The organization of the network is not mandated by the JXTA framework, but in practice four kinds of peers are typically used:

- *Minimal edge peer;* A minimal edge peer can send and receive messages, but does not cache advertisements or route messages for other peers. Peers on devices with limited resources (e.g., a PDA or cell phone) would likely be minimal edge peers.
- *Full-featured edge peer;* a full-featured peer can send and receive messages, and will typically cache advertisements. A simple peer replies to discovery requests with information found in its cached advertisements, but does not forward any discovery requests. Most peers are likely to be edge peers.
- *Rendezvous peer;* A rendezvous peer is like any other peer, and maintains a cache of advertisements. However, rendezvous peers also forward discovery requests to help other peers discover resources. When a peer joins a peer group, it automatically seeks a rendezvous peer if no rendezvous peer is found, it dynamically becomes a rendezvous peer for that peer group. Each rendezvous peer maintains a list of other known rendezvous peers and also the peers that are using it as a rendezvous.

  Each peer group maintains its own set of rendezvous peers, and may have as many rendezvous peers as needed. Only rendezvous peers that are a member of a peer group will see peer group specific search requests.

  Edge peers send search and discovery requests to rendezvous peers, which in turn forward requests they cannot answer to other known rendezvous peers. The discovery process continues until one peer has the answer or the request dies. Messages have a default time-to-live (TTL) of seven hops. Loopbacks are prevented by maintaining the list of peers along the message path.
- *Relay peer;* A relay peer maintains information about the routes to other peers and routes messages to peers. A peer first looks in its local cache for route information. If it isn't found, the peer sends queries to relay peers asking for route information. Relay peers also forward messages on the behalf of peers that cannot directly address another peer (e.g., NAT environments), bridging different physical and/or logical networks

CHAPTER 3

DESIGNING SEMANTICALLY ENRICHED WEB SERVICES IN THE HEALTHCARE DOMAIN

Most of the existing healthcare information systems serve only one specific department or hospital, using different operating systems, object models and programming languages. Communication between these different systems is often difficult. These problems must be overcome to enable smooth transactions between medical institutes. On the other hand Web service technology is the natural way for application to application interoperability and information exchange across enterprises. So it can be deduced that by introducing Web services in the healthcare domain it may be possible for healthcare units to share clinic data among other healthcare providers. However it would offer limited use unless the semantics of services are described and exploited [10]. In this study, it is aimed to show how healthcare centers can make use of the possibilities Web services offer to solve problems of integrating disparate systems.

During the thesis, a real-life running healthcare information system is examined in Turkish Armed Forces (TAF) Rehabilitation and Care Center (REHAB) [60], founded for the treatment of war veterans and disabled military people in Turkey. The software used in REHAB, corTTex [6], is an integrated hospital management information system environment, which brings most of the advanced technologies of medical informatics, information processing, data communication and clinical systems. Although corTTex fullfills the Center's local information system needs, many problems arise when it comes to the exchange of information with other military hospitals, which is a frequent situation in the Center.

In this Chapter; the need for medical information system interoperability and advantages gained when the operability achieved is explained briefly in Section 3.1. CorTTex information system is introduced in Section 3.2. The problems that exist in the Center are described in Section 3.3. In order to clarify the concepts, sample healthcare Web services are developed during the study to show how they can be made use to solve some of the problems on the running system. In Section 3.4, simple ontologies are developed based on prominent healthcare standards such as HL7, GEHR and CEN ENV 13606-2. Finally, in the last Section, the sample services that are developed during the study are explained. It should be noted that sample services are developed to show how they can be used to solve some of the problems in the system; the aim of the study is not to generate complete solutions for the problems of the examined healthcare center.

3.1 Motivation

In the past two decades, healthcare institutions, and hospitals in particular, have begun to automate aspects of their information management. Initially, such efforts have been geared towards reducing paper processing, improving cash flow, and improving management decision making. In later years a distinct focus on streamlining and improving clinical and ancillary services has evolved, including bedside (in hospitals and other inpatient environments) and "patient-side" systems (in ambulatory settings). Within the last few years, interest has increased in integrating all information related to the delivery of healthcare to a patient over his or her lifetime (i.e., an electronic medical record).

Most of the problems in medical information systems can be solved by a common understanding and sharing of clinical information related to patient. In order to clarify concepts and to show the benefits gained by interoperability among medical units, two examples are introduced in this section.

3.1.1 Patient History Information

The traditional method during patient treatment is to get the past information (patient history) by the help of local information management system. This helps clinician in both diagnoses and treatment steps. Exchanging information among medical information systems with semantic negotiation would enable clinicians to get past information even in the first patient visit.

Patient related information is spread around different medical institutes, such as hospitals, village clinics, pharmacy, governmental and private health units. It is important to know the past information of patient during medical treatment. Information related to a patient, not only in the local care unit but also other medical information systems will ease the clinicians' work in many ways. Integrating medical information systems using Web service technology would solve this deficiency in the healthcare domain.

For example, lots of emergency events occur resulting from traffic accidents, earthquakes or such disasters. In such cases, patients are taken to the nearest healthcare center where the event happens, and rarely any information even the blood type of the patient is known. Only the identification information of patient might be available to care system. In that case, the actions previously performed to the patient such as operations, lab test and other patient centric information, such as allergies, is unavailable to the health unit, which results in longer treatment times. The past information of patient helps the clinician and makes shorter intervention times, which is an important factor for life saving in emergency rooms.

In addition to the past information of patient, it is also important for healthcare units to know the insurance information of patient for billing issues. Insurance companies may force the health institute to be informed prior to any operations performed. As a result the required confirmations should be obtained before the treatment, depending on the severity of the event.

Interoperability among medical information systems would enable partners; to query patient-related information such as clinical data, the identification information, insurance information, allergies and so on, which helps to solve the underlined problems.

3.1.2 Health Expenses and Provision Systems

The increasing medication expenses in Turkey forced insurance companies to control the pharmacy information. Pioneered by Retired State Employees (ES) [51], a governmental health insurance system, Pharmacy Provision System put in practise in November 1997 [50]. System requires all medication information to be entered by the pharmacy to the information center of ES. Pharmacy Provision System enabled ES to control the following properties prior to the approval of the medication payment:

- Determination of the non-eligible,
- Medication use time control,

- Automatic dose calculation (5 day for inpatient,10 day for outpatient),
- Medication price control,
- Determination of medication payment status (some medications are not paid by the company),
- The equivalent medication determination,
- Controlling the medication appropriateness (using patient age, sex, illness relation etc)

These are the main titles, which includes several more specific controls and checks. The system work as follows; drugstores enter the doctor's written prescription information, medications are given to patient only after the system approves the doctor’s written prescription.

Approval process includes the controls some of which stated above. For example; each medication has a usage time depending on some parameters such as medication type, use and treatment method. Prior to the provision system, patients can possibly visit different healthcare units and same type of medications can be written before the use time of already taken medication finishes. By checking the medication use time during approval process, such drawbacks are blocked by the provision system.

After the system put in practise the medication expenses has dramatically decreased. ES declared 405.1 million $ gain at average of four years, after the provision system started running [50].

The provision system explained here works fine with medication consumption and written prescriptions. But during treatment process, many laboratory tests (microbiology, radiology, biochemistry etc) are performed by different institutes to patients in short time periods, especially when patient is sent to other hospital for further analysis. In fact, some of the previous test results can be used in the next steps of treatment or by other medical centers if the results were made publicly available. Providing information sharing and semantic interoperability among healthcare units will help the partners in the medical domain to prevent such cases.

It is also important that a similar provision system can be developed by insurance companies with the help of Web service technology in the healthcare domain. Querying the steps, actions performed on patient, controlling the process appropriateness against the patient disease and similar checks would result a similar decrease in health expenses and prevent possible misuses.

3.2 Corttex (An Integrated Hospital Management Information System)

The hospital information system used throughout this study is corTTex [6]; an integrated hospital management information system environment, an advanced approach to the healthcare informatics. It brings together many advanced technologies of medical informatics, information processing, resource planning, data communication, and clinical systems to form a solution to everyday problems of health and patient care organizations. The objectives of corTTex are to provide a comprehensive and integrated support to the whole range of healthcare.

The integrated system includes many subsystems, such as registration management, scheduling management, inpatient/outpatient clinical systems, laboratory information system, radiology information system, operation room management system, pharmacy management system. These subsystems make up the local information management of the care center.

The programming model used in the application software is client/server. It requires a centred database server, such as Oracle [38] or MS SQL Server [33]. The clients directly access to database via the application programming interface (API) of the software.

Prior to the application put in practise in a health unit, system and user parameters, work processes, resources, and other infrastructure parameters must be defined by the system administrator of corTTex. This procedure is done using Enterprise Manager (EM) [6], work process management software of corTTEx.

EM is a powerful tool for describing and manipulating the system and the user parameters on the basis of medical unit. This modelling tool works integrated with the application program in the environment which users, rights, work processes, appointments, data entry parameters, rules, reports, resources, interconnection of resources and all functions including stock management can be defined. Many of the administrative tasks, such as creating users, assigning roles to users, defining hospital structure, action definitions, and step definitions can be rapidly achieved using EM.

The system architecture consists of the main of classes shown in Figure 3.1.

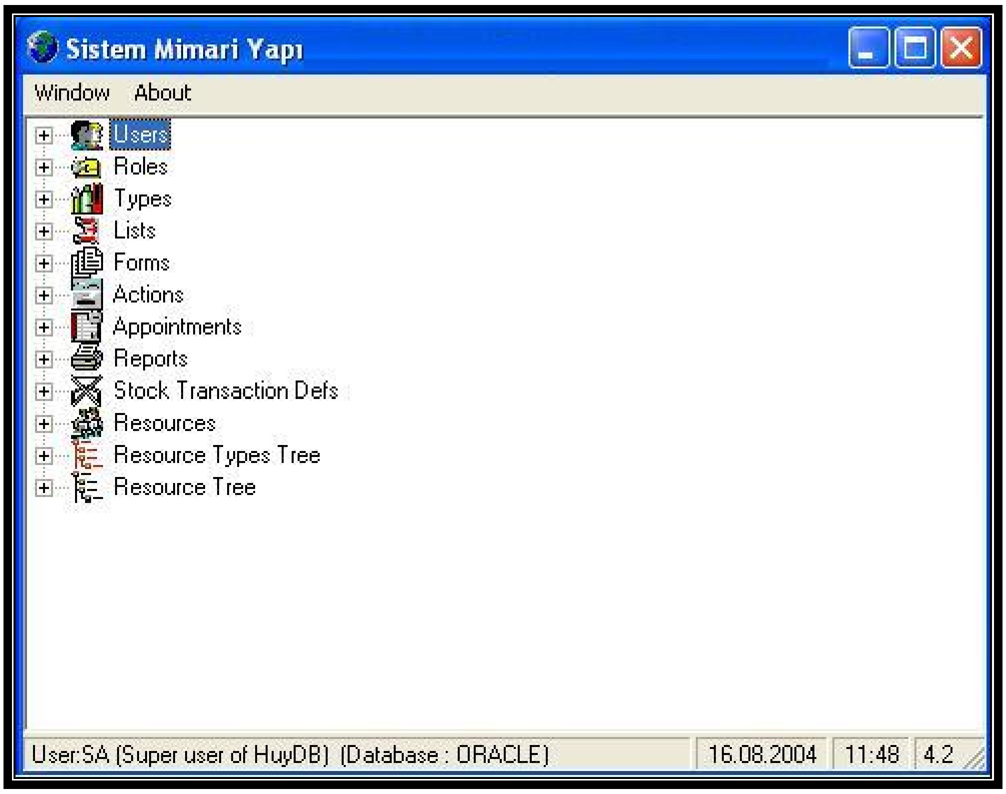


Figure 3.1: Enterprise Manager API

The manager is designed using an object-oriented model, which makes system administration and maintenance very simple. For example, creating a new user includes several steps such as entering user identity information, user department definition and role assigning. All these tasks can be done using EM in seconds.

The business logic in corTTex is event-driven, i.e., when a trigger event occurs, the related actions are fired depending on the event type and some duties are automatically assigned to related users by the information system. For example, after a doctor requests for a microbiology test, the request information such as test codes, patient account number, request date/time and request methods are automatically assigned to the microbiology laboratory work list. Besides depending on the patient type the billing system records the request and required actions are performed. The required verifications and billing issues are controlled by the system. Neither the test requester nor the provider is concerned with the intermediate steps during the action.

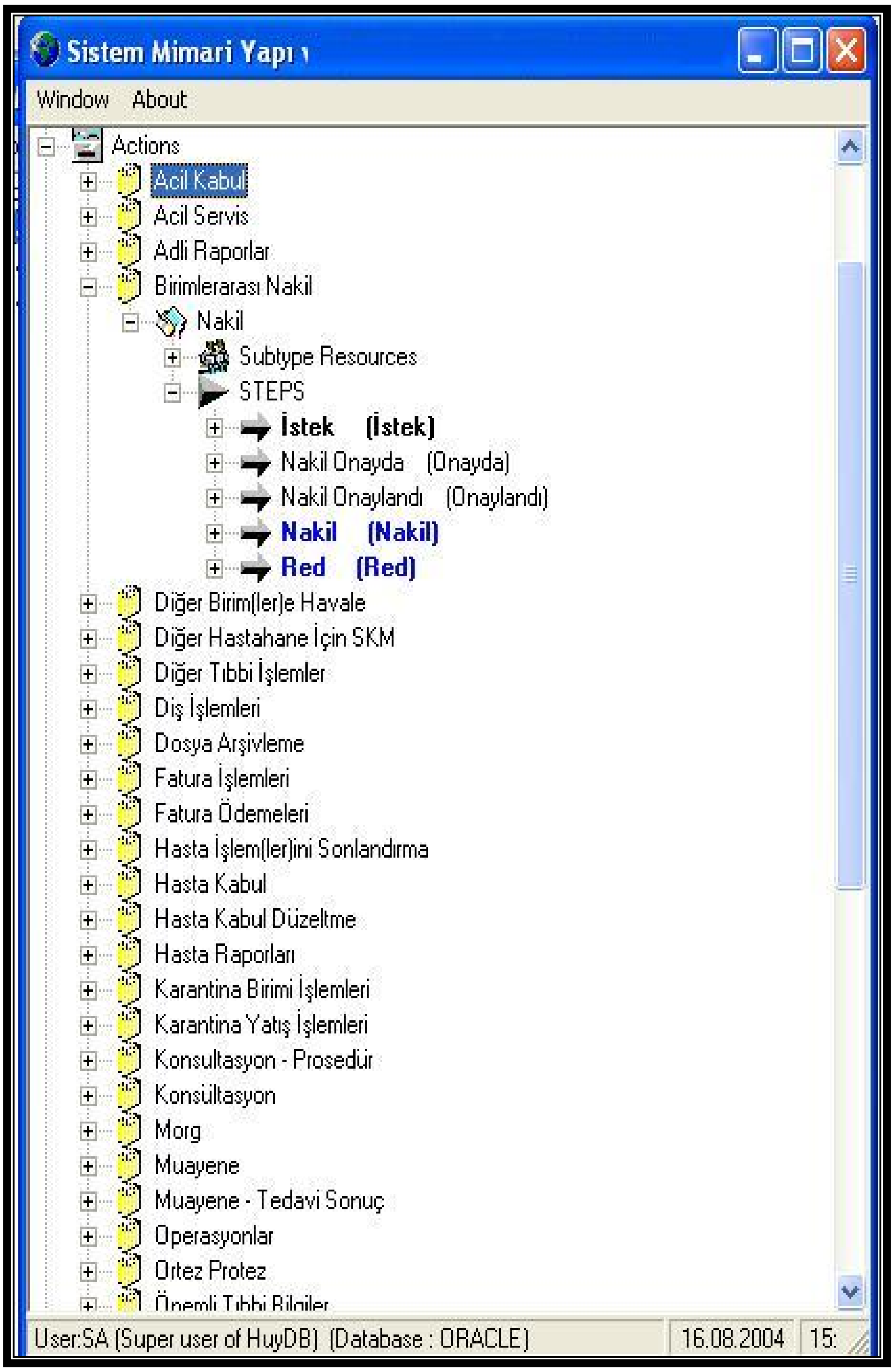
All users follow the jobs, regulations and works assigned to them using their work list periodically. The access to system resources is restricted by the system and only authorized users can have access.

Patients are given a unique patient identification (PID) number on their first visits. Patient specific information is related to that PID number. For each patient visit a protocol number (visit ID) is generated in the system, to which all actions performed on that visit are recorded.

All past clinical information of patient such as reports, test results, allergies can be viewed and queried by the system. System uses ICD-10 and CPT 4 standards in clinical information processes.

All laboratory devices are integrated to the information system. The generated lab results are transferred to the information system automatically. All previous lab test results can also be queried by the system and comparisons among results can be made easily.

All possible events ("actions" in programme terminology), the steps in these events (action steps) are defined via EM API as shown in Figure 3.2.


Sistem Mimari Yapı

Window About

Actions
- Acil Kabul
- Acil Servis
- Adli Raporlar
- Birimlerarası Nakil
  - Nakil
    - Subtype Resources
    - STEPS
      - **İstek (İstek)**
      - Nakil Onayda (Onayda)
      - Nakil Onaylandı (Onaylandı)
      - **Nakil (Nakil)**
      - **Red (Red)**
- Diğer Birim(ler)e Havale
- Diğer Hastahane İçin SKM
- Diğer Tıbbi İşlemler
- Diş İşlemleri
- Dosya Arşivleme
- Fatura İşlemleri
- Fatura Ödemeleri
- Hasta İşlem(ler)ini Sonlandırma
- Hasta Kabul
- Hasta Kabul Düzeltme
- Hasta Raporları
- Karantina Birimi İşlemleri
- Karantina Yatış İşlemleri
- Konsultasyon - Prosedür
- Konsültasyon
- Morg
- Muayene
- Muayene - Tedavi Sonuç
- Operasyonlar
- Ortez Protez
- Önemli Tıbbi Bilgiler

User:SA (Super user of HuyDB) (Database : ORACLE) 16.08.2004 15:

Figure 3.2: Detailed View of System Actions

For example, the steps of "laboratory test request" action are shown in Figure 3.3.

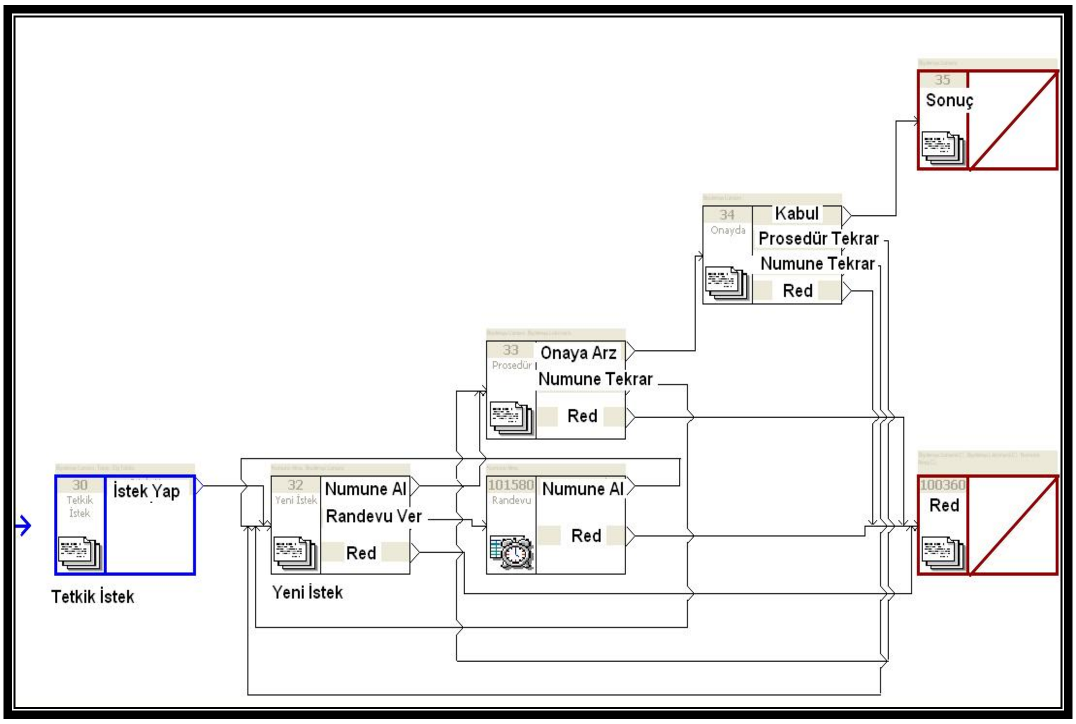


Figure 3.3: Action Steps (Lab Test Request)

After a “İstek Yap” (make request) action is fired (Figure 3.3) in “Tetkik İstek” (lab test request) event, a new work process is created by the system named “Yeni İstek” (new request). 3 possible choices are given to user in this stage of the action which can be viewed by microbiology lab technician:

- “Numune Al” (take sample)
- “Randevu Ver” (schedule)
- “Red” (deny request)

Depending on the action chosen in this step, a new event is created. For example if “Numune Al” action is chosen by microbiology lab technician, a new “Procedure” event is created, which has also 3 possible next steps depending on the results of the test;

- “Onaya Arz” (submit to approval)
- “Numune Tekrar” (retake sample)
- “Red” (deny test)

Starting from patient registration, all transaction flow operates similarly. This event-based architecure has many advantages essentially providing modularity, where users accomplish their tasks independent of each other.

As a conclusion, corTTex focuses on flow of patients, information and resources throughout the health continuum in a logical, strategic and efficient way. CorTTex synchronizes workflow across the entire enterprise, orchestrates patient care, and improves operational efficiency. The management of the system is also held easily using EM. As stated previously, even-based mechanism provides many advantages on the system and reduces the process time.

3.3 Problems in the Existing System

REHAB is a department of a bigger military hospital in Turkey; Gülhane Military Medicine Academy (GATA) [16]. REHAB and GATA are founded on different locations and both have different hospital applications, network structures and database systems. Integrating the two sides requires extensive site-specific programming in a network environment.

Many problems exist between REHAB and GATA hospitals as a result of having two different information systems. Patient information in one center is unreachable from the other site. The main problem of the existing system is the need for keeping two different records for same patient. This leads to problems during treatment processes and billing tasks.

Another problem occurs during laboratory test requests between two hospitals. Since some lab devices do not exist at both sites, especially for REHAB center, patients are sent to GATA for some lab tests. In such cases, transmitting observations and results of diagnostic studies from the producing system (e.g., clinical laboratory system, EKG system), to the ordering system (e.g., HIS order entry, physician's office system) constitutes problem between the two information systems. Besides patient specific information such as protocol number, insurance information, visit number is unreachable from the test provider site. This information is required by the local information system of the provider site in order to process the required tests. In addition test results can not be sent to the requester site electronically which results huge manual operations and waste of time.

Patient management constitutes another problem for two sites. Although REHAB is a department of GATA, duplicate records exist for some patients. This results in mistakes during statistical studies, which is an important factor for future decisions and studies.

Hand devices such as PDAs, palms are increasingly used by physicians, nurses and other clinicians, on which hospital management software can not be run directly. This causes another deficiency in the Center, especially during doctor visits, patient online data is unaccessible for inpatients. If patient data and results generated by the information system is served as an XML payload in a Web service environment, that payload can easily be transformed to whatever the end-user presentation needs to be. This is much more easier than tearing out the hospital management software and rewriting it from the ground up to work on a PDA or what clinician has.

CorTTex as described in previous section is an integrated hospital management software, i.e., brings many applications together and serves to users in a unique view. However, it is possible in future that multiple medical applications might run on different platforms, so the doctor can not access them in one unified view. Web services can also handle this situation easily.

The underlined problems focused in this section has many drawbacks on the overall health services. These obstacles hinder health quality, decrease both personal and patient satisfaction resulting from unnecessary actions between hospitals during treatment. This results in waste of time and money as well as work loss in both hospitals.

3.4 Service Functionality and Message Ontologies

In order to facilitate the discovery of the Web services, an ontology is required to describe the semantics of medical Web services. The service ontology is to be used in describing the service functionality in the healthcare domain. For example, when a Web service instance, say “HastaKlinikBilgisiSorgula” is annotated with the “RequestPatientClinicalInformation” node of such an ontology, its operational meaning becomes clear that this service can be used for querying patient clinical information in a hospital.

An electronic healthcare record may get very complex with data coming from diverse systems such as lab tests, diagnosis, and prescription of medications which may be in different formats. Service functionality semantics is not enough in real life medical information services, it is also necessary to define the semantics of documents exchanged through Web services [10].

When considering the sample Web services introduced in this chapter, the semantics of an action is usually clear from the functional ontology, but the format of service parameters and contents of those parameters are not clear. Therefore another ontology is required so that the structure of the message parameters should also be understandable at the receiving end.

In the following subsections; in order to classify sample medical Web services, a service functionality ontology is introduced. In addition, in order to annotate the services retrieving meaningful EHR components, a message ontology is also generated. It should be noted that the aim of this thesis is not to propose such healthcare ontologies but to show how such ontology, once developed, can be used in semantic mediation.

3.4.1 Functionality Ontology

As described in Chapter 2, HL7 has already categorized the events by considering service functionality which reflects the business logic in the healthcare domain. This categorization is used as a basis for defining the service action semantics through Service Functionality Ontology.

The HL7 standard [17] groups the HL7 events into the following clusters: Patient Administration, Order Entry, Query, Financial Management, Observation Reporting, Master Files, Medical Records/Information Management, Scheduling, Patient Referral, and Patient Care. These clusters also have sub clusters. A partial Web service Functionality Ontology based on HL7 events is already provided in the Artemis project (Figure 3.4) [10].

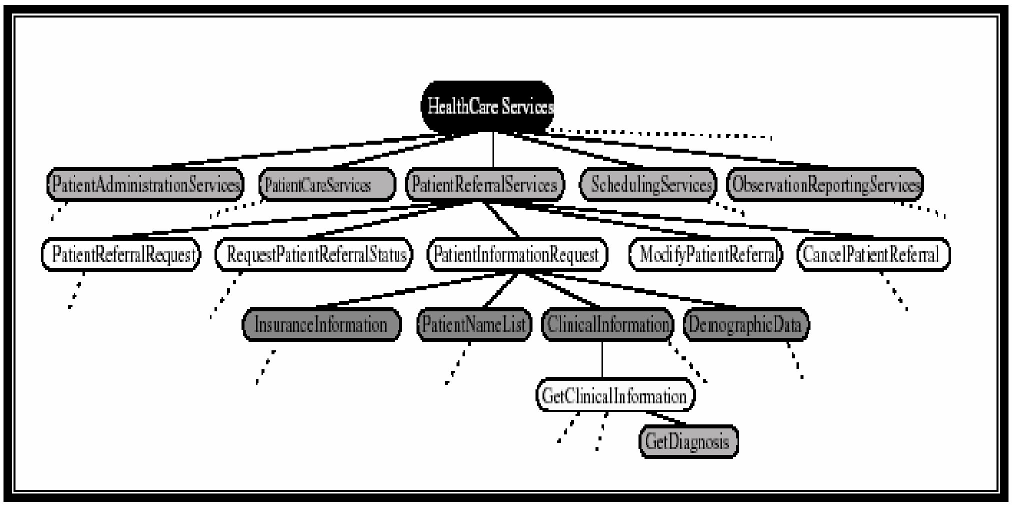


Figure 3.4: A Service Functionality Ontology based on HL7

The Service Functionality Ontology introduced in the Artemis Framework is extended considering the needs of the running system in this study. This extension is also based on HL7 events (Figure 3.5). The RDF-Schema of the whole Functionality Ontology is provided in Appendix-A.

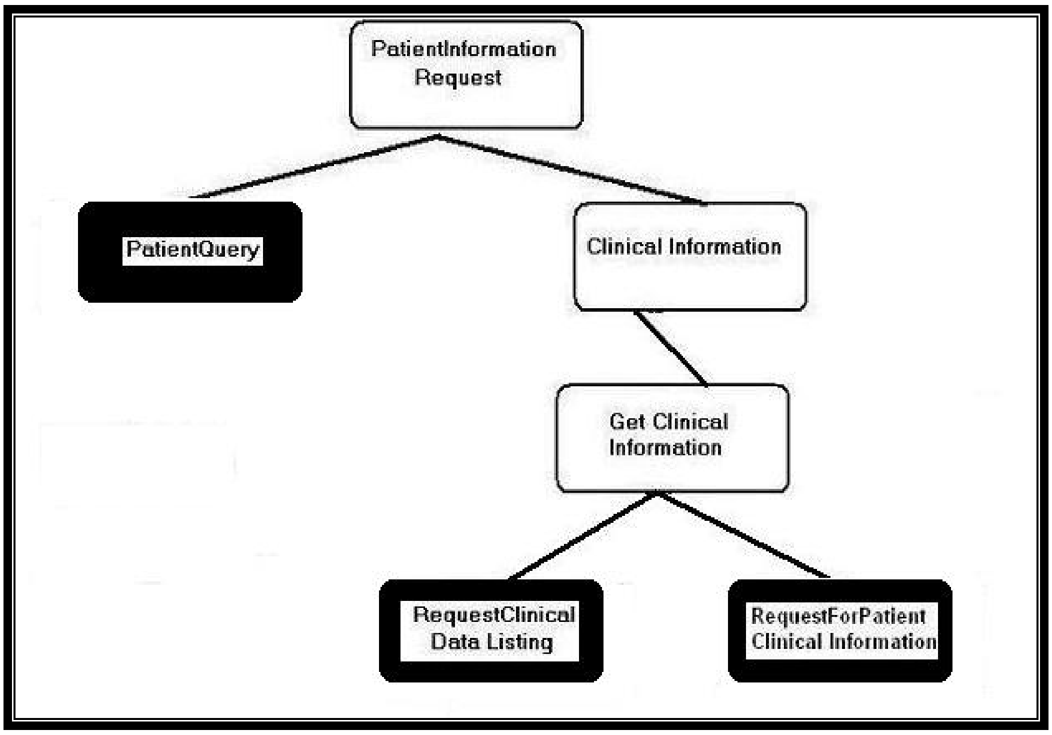

PatientInformation Request
PatientQuery
Clinical Information
Get Clinical Information
RequestClinical Data Listing
RequestForPatient Clinical Information

Figure 3.5: Extended Functionality Ontology

When searching for the right Web services, consumers can consult this ontology to find out the semantics of the service they are looking for. For example, a Web service retrieving "Patient Identification Information" can be classified under "PatientQuery" node as shown in Figure 3.5. How this is achieved in UDDI and ebXML registries, is explained in [10].

3.4.2 Message Ontology

As stated previously, service functionality semantics is not enough in real life medical information services. Since there can be quite complex service parameters coming from different information systems.

Consider a Web service named "RequestPatientVisitInformation" which is annotated to the "RequestForPatientClinicalInformation" node of the ontology given in Figure 3.5. Although the semantic action of the service is clear from the functionality ontology (i.e., it is retrieving the visit-specific information about a patient), it is not clear what the content and format of service parameters like "PatientID" and "VisitInformation" are. Therefore the structure of the message parameters should also be meaningful at the ends, producer and consumer, of the medical information systems.

As an example, in Figure 3.6, two partial ontologies are presented based on the "building blocks" of HL7 and CEN ENV-13606 [10].

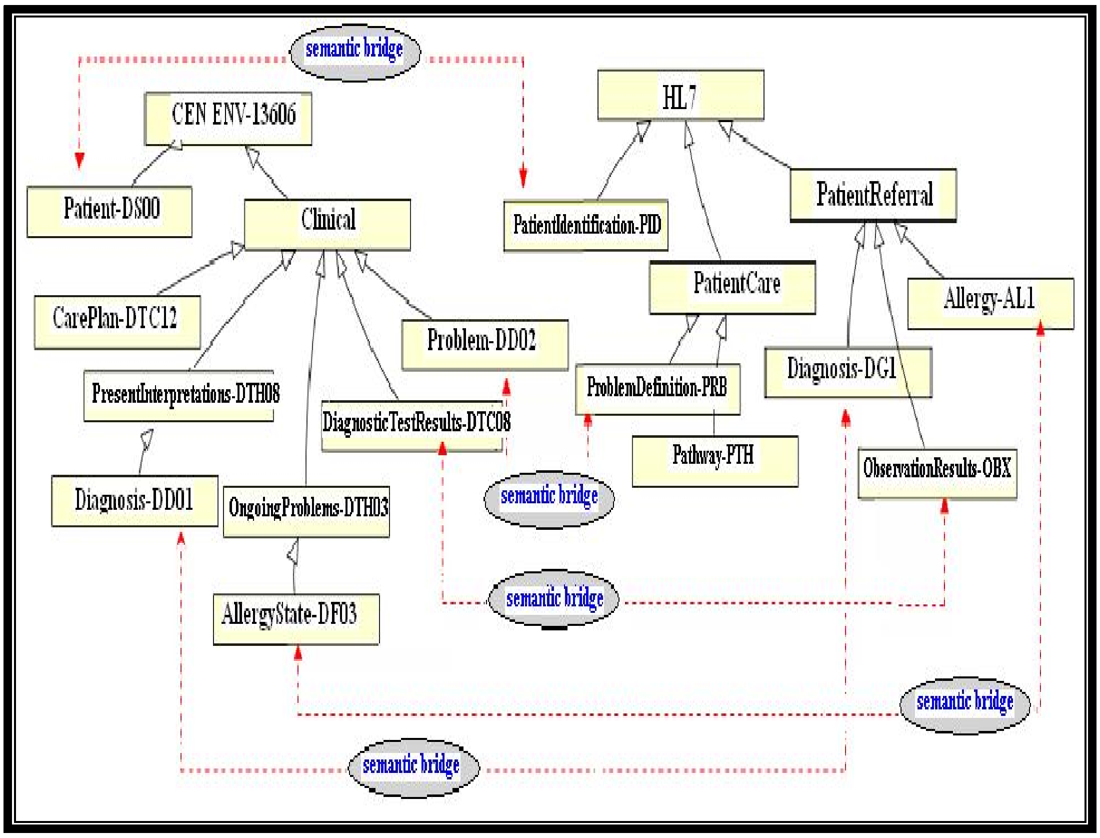


Figure 3.6: CEN ENV-13606 and HL7 Clinical Concept Ontologies

In order to provide information integration, additional message semantics is essential and EHR based standards like HL7 CDA [52], GOM (GEHR Object Model) [4] and CEN's ENV 13606 [5] are exploited in this respect. These standards provide conceptual "building blocks" or "meaningful components" by which any clinical model can be represented within the standardized framework.

Same building blocks can be composed differently by different health units, which results in different message structures. Although this provides flexibility, semantic and structural mapping is required in order to interoperate these systems. How this mapping can be made in the Artemis Framework is explained in [10].

In order to understand the sample medical Web services, generated through the study, the building blocks shown in Figure 3.7 are defined based on HL7 to generate message blocks used in sample services.

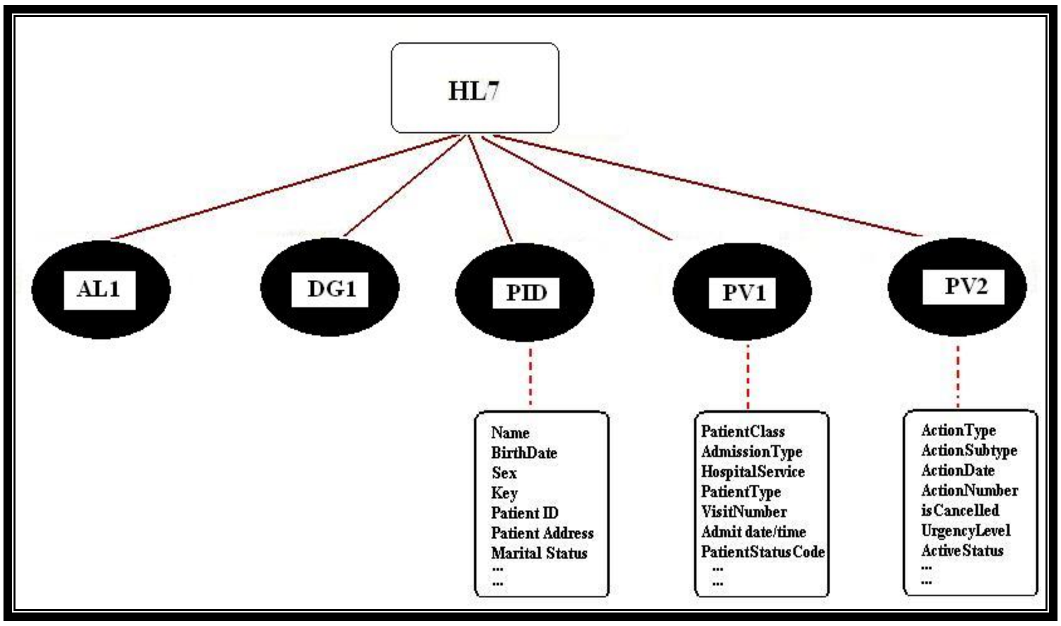


Figure 3.7: Message Building Blocks

The details of these building blocks (segments) are defined in the rest of this section. The RDF-Schema of the whole Message Ontology is provided in Appendix-B.

3.4.2.1 Patient Identification (PID)

The PID segment is used by information management system as the primary means of communicating patient identification information. This segment contains permanent patient identifying and demographic information that, for the most part, is not likely to change frequently. The fields are;

- *Patient ID* : This field contains the unique number given to the patient in the first visit.

- *Patient name* : This field contains the legal name of the patient.
- *Birthdate*: This field contains the patient's date and time of birth.
- *Sex*: This field contains the patient's sex.
- *Patient address* : This field contains the communication address of the patient.
- *Marital status* : This field contains the patient's marital status.

3.4.2.2 Patient Visit (PV1)

This segment is used by the information system to communicate information on a visit-specific basis. The properties of the fields in this segment are;

- *PatientClass*: This field is used by systems to categorize patients. Possible values in the system are; emergency, inpatient, outpatient etc.
- *AdmissionType*: This field indicates the circumstances under which the patient was or will be admitted. Some suggested values (provided by HL7) are; Accident, Emergency, Labor, Routine. It is subject to site-specific variations.
- *HospitalService*: This field contains the treatment or type of service that the patient is scheduled to receive.
- *PatientType*: This field contains information that identifies the patient type. (i.e., retired, official, civil etc.)
- *VisitNumber*: This field contains the unique number assigned to the visit.
- *Admit date/time*: This field contains the admit date/time.
- *PatientStatusCode*: The status of the patient which the visit belongs to.

3.4.2.3 Patient Visit Additional Information (PV2)

The PV2 segment is a continuation of visit-specific information contained on the PV1 segment. This segment is mainly used for the actions performed on patient. Each action performed on the patient during the care is uniquely identified in clinical information using the PV2 segment. The details of the fields are;

- *ActionType*: The type of the action performed (admission, discharge etc.)
- *ActionSubtype*: The sub category of the action type (microbiology test, etc.).
- *ActionDate*: The date/time of the action.
- *ActionNumber*: The unique number assigned to the action.

- *isCancelled*: Indicates whether the action is cancelled or not.
- *UrgencyLevel*: The urgency information of the action.
- *ActiveStatus*: Indicates whether the action is still in progress. (for long term actions such as therapies)

3.5 Sample Healthcare Web Services

As stated in the introduction part of this chapter, a number of Web services are generated from corTTex. These sample services are developed to show how such services, once available, can be used by other institutes and how some problems can be solved in this way. The study does not aim to develope complete solutions to the existing problems of the hospital that are mentioned in the previous sections.

It is important to note that, the Web services developed through this study is mainly designed for information supply (read-only), i.e., the services are not designed for updating any patient data, such as adding new records, changing patient clinical data or cancelling previous actions on the patient files. The major goal is to enable other health units to reach and query information related to patients.

The services introduces in this section provide the consumers to query whether a patient has information in the health unit, if a patient visited the health center before; these services enable consumers to search for visit-specific information such as the actions performed on the patient, the date of the actions, and the status of the patient during the treatment and so on.

Once the semantic negotiation between health institutes is achieved using Service Functionality and Message Ontologies, both ends can consume each others services. In Figure 3.8, the annotation of the generated Web services to the functionality ontology (Figure 3.5) is shown. As mentioned before, the annotation of the real-life services to an ontology is essential in that the consumer would be able to search and use the correct service.

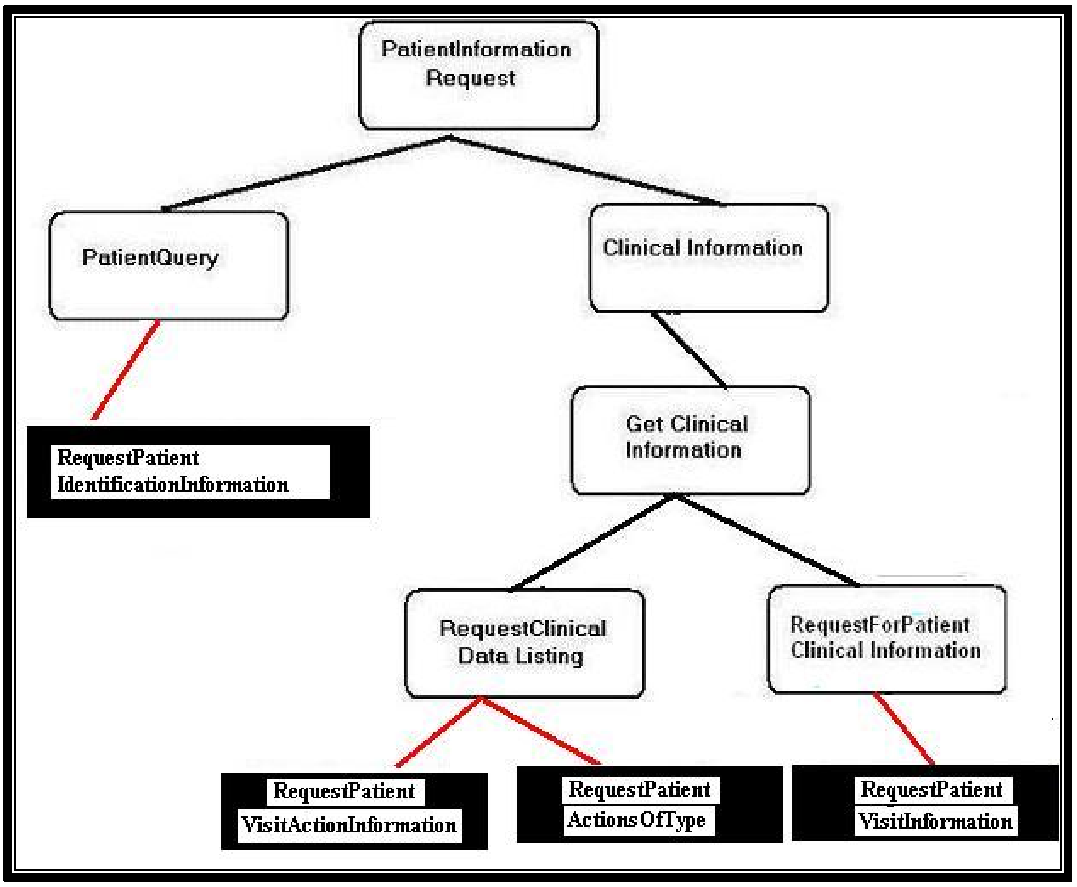

PatientInformation Request
PatientQuery
Clinical Information
RequestPatient IdentificationInformation
Get Clinical Information
RequestClinical Data Listing
RequestForPatient Clinical Information
RequestPatient VisitActionInformation
RequestPatient ActionsOfType
RequestPatient VisitInformation

Figure 3.8: Annotation of Services

The sample Web services generated throughout the study are described in the remaining parts of this section.

3.5.1 RequestPatientIdentificationInformation

Since any system attached to the health domain requires information about patients, the *RequestPatientIdentificationInformation* service is one of the most commonly used.

When patient visits a health unit for the first time, the patient may or may not be able to provide positive ID information (i.e., in emergency rooms). The admission system needs to register the patient; in doing so, it is desired to record the patient's demographic data in the hospital management information system. The registrar issues a patient query request to the service provider, with some basic patient demographics data as search criteria. In the returned patient list, an appropriate record for the patient is picked up, including the hospital's patient ID, to enter the information system.

Search criteria entered by the consumer might include one or more of the following;

- Partial or complete patient name
- Patient ID (if already known)
- Date of birth/age range

This service returns a list of patients showing the full name, age, sex, and admission date (PID segment). The consumer then selects the appropriate record to enter the patient identity information into the local information system.

This service is also essential for consumers to use other services, since all services require patient ID information, to identify the patient, as one service parameter. Using this service, the patient ID can be fetched and used in required services.

3.5.2 RequestPatientVisitInformation

This service is primarily about patient-visit based queries. In the running software, for each patient visit (both inpatient and outpatient) a unique visit number and file is assigned to hold the actions performed on the visits. The patient clinic data in the hospital can be queried using this visit information. This service is essential in that the details of actions performed on visits can be queried once the action number is supplied. Using this service, the required actions and fields can be queried.

Visit information covers;

- diagnoses
- operations
- healthcare resource groups
- length of stay
- waiting times
- admission methods
- age of patients
- sex of patients
- ethnic group of patients
- maternity care
- psychiatric care

This service is provided so that, when a health unit needs specific clinical information related to a patient, the visit information will be supplied, including start date, end date, patient identification information, the status of the visit (open or close), the bed patient assigned (for inpatient).

In fact this service is a generic or auxiliary service which is used to query more specific information requests. This service requires the patient ID (PID) information as input parameter, and returns PV1 data segment.

3.5.3 RequestPatientVisitActionInformation

The previous service returns the major files of patient in visit basis. The specific information related to a patient can be queried by *RequestPatientVisitActionInformation* service. This service is primarily used for supplying all the actions performed on the same patient visit. As stated in previous chapters, the running application in the hospital relates all actions and steps performed on patients in visit based manner.

This services requires the ID of the visit (PV1 segment) that is queried (each visit of patient is uniquely identified) and returns PV2.

Using this service, all of the actions that were performed during a visit, the steps of the actions and a general view of the treatment can be obtained. This service is essential in that it provides visit specific results, so for example when a consumer needs the most recent actions performed on a patient at the provider hospital, it may use this service providing the required input parameters. Note that this service does not provide details of actions, such as the results of a lab test request

3.5.4 RequestPatientActionsOfType

The previous web service returns the actions in a visit based manner, i.e., all the actions on one visit. It is sometimes useful to provide actions of certain type such as, all the radiology tests performed on the patient or all the admissions done before. Such action-type based queries can be queried using this service.

It is similar to the RequestPatientVisitActionInformation service in that this service also returns action related information as result (PV2 segment). However as stated, this service is not limited to one specific patient visit. Using this service it is possible to query all of the actions of some type (i.e., the microbiology tests).

The service requires patient ID that the actions is to be queried (PID segment) and the type of the action to be queried and returns PV2.

The sample services introduced in this chapter enables exchange of patient clinic information between REHAB and GATA hospitals. Although the access via the services is limited, i.e., no updates are available yet, new services can be designed for further operations. Keeping two distinct patient identification records in sites can be avoided using RequestPatient-IdentificationInformation service during patient admission process (for new patients). Before assigning a new identifier, two hospitals may invoke the other's service to check whether the patient has arrived to the other hospital previously. If the patient has already been given an identification number that can be used during registration process.

Another improvement provided with the services is during test requests. As described in the previous sections, patients may be sent from one hospital to the other during the treatment. The requester site, once implemented the services, can supply the required information in network environment. The provider site, use the RequestPatientVisitInformation service to query the most recent visit (i.e., it is already open episode), then it may query the actions performed on that visit and the required information such as protocol number, patient ID etc. can be obtained. Transmitting the results electronically is not possible using the sample services, but new services can be implemented so that the consumer sends the result of actions such as laboratory tests.

A clinician at GATA hospital, may query all actions of certain type such as discharges, admissions etc., those were previously performed on a patient at REHAB by using RequestPatientActionsOfType web service.

The demonstration of the generated sample services in Artemis platform is presented in Chapter 4.

3.6 Summary of the System

In order to achieve the interoperability and information exchange among healthcare information systems, Web service technology seems to be very suitable. However using Web services effectively requires well known and accepted domain ontologies.

In this chapter, sample web services are generated from a real-life hospital information system. By providing the services and describing the service semantics through Service Functionality and Message Ontologies based on a prominent healthcare standard, HL7 in our case, it is made available for other systems to make use these services.

In the next chapter a sample scenario is given in order to illustrate how the generated services can be used once they are made available. The technologies used during the development of services and the Artemis P2P Platform are also introduced in the next chapter.

CHAPTER 4

IMPLEMENTATION OF SEMANTICALLY ENRICHED WEB SERVICES IN THE HEALTHCARE DOMAIN

Web services depend on the ability of parties to communicate with each other even if they are using different information systems. XML [15], making data portable, is a key technology in addressing this need. The Java platform [29], which makes code portable, is a natural choice for developing Web services. As a result, the healthcare web services are coded using Java programming language.

In addition to data portability and code portability, Web services need to be scalable, secure, and efficient. The Java 2 Platform, Enterprise Edition (J2EE) [26] defines the standard for developing multitier enterprise applications. J2EE simplifies enterprise applications by basing them on standardized, modular components, by providing a complete set of services to those components, and by handling many details of application behaviour automatically, without complex programming.

This chapter is organised as follows; Web service implementation using the J2EE Platform is described in Section 4.1. In Section 4.2, Artemis architecture is introduced briefly. Section 4.3 presents a sample demonstration of the overall system on the Artemis framework. Finally the tools used in the implementation are described in Section 4.4.

4.1 Building and Deploying Sample Web Services

The sample healthcare Web services developed during the study are built using Java API for XML-based RPC (JAX-RPC) [27] on J2EE platform. JAX-RPC is a technology for building Web services and clients that use *remote procedure calls* (RPC) and XML [15]. With JAX-RPC, the developer does not generate or parse SOAP messages. It is the JAX-RPC runtime system that converts the API calls and responses to and from SOAP messages.

J2EE platform provides the XML APIs and tools to quickly design, develop, test, and deploy Web services and clients that fully interoperate with other Web services and clients running on Java-based or non-Java-based platforms.

JAX-RPC is not restrictive; a JAX-RPC client can access a Web service that is not running on the Java platform, and vice versa [25]. This flexibility is possible because JAX-RPC uses technologies defined by the World Wide Web Consortium (W3C) [65]: HTTP [19], SOAP [54], and WSDL [66].

The starting point for developing a JAX-RPC Web service is the service endpoint interface. A *service endpoint interface* (SEI) [25] is a Java interface that declares the methods that a client can invoke on the service.

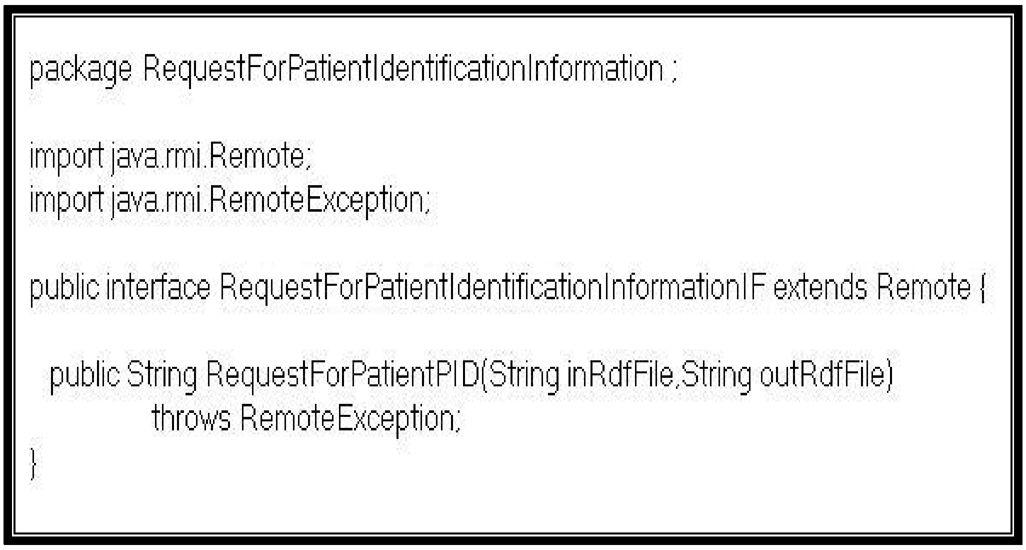
The following steps are followed during the service generation;

- Coding the SEI, implementation classes and interface configuration file.
- Compiling the SEI and implementation classes.
- Using **wscompile** tool to generate the files required to deploy the service.
- Using **deploytool** to package the files into a WAR file.
- Deploy the WAR file on the J2EE application server.

In the following sections the details of these steps are defined. The following examples include code snippets about *RequestPatientIdentificationInformation* service, introduced in Chapter 3. The steps done for the other services are similar.

4.1.1 Coding the Service Endpoint Interface and Implementation Class

A server side service endpoint definition of RequestPatientIdentificationInformation service is given in Figure 4.1.



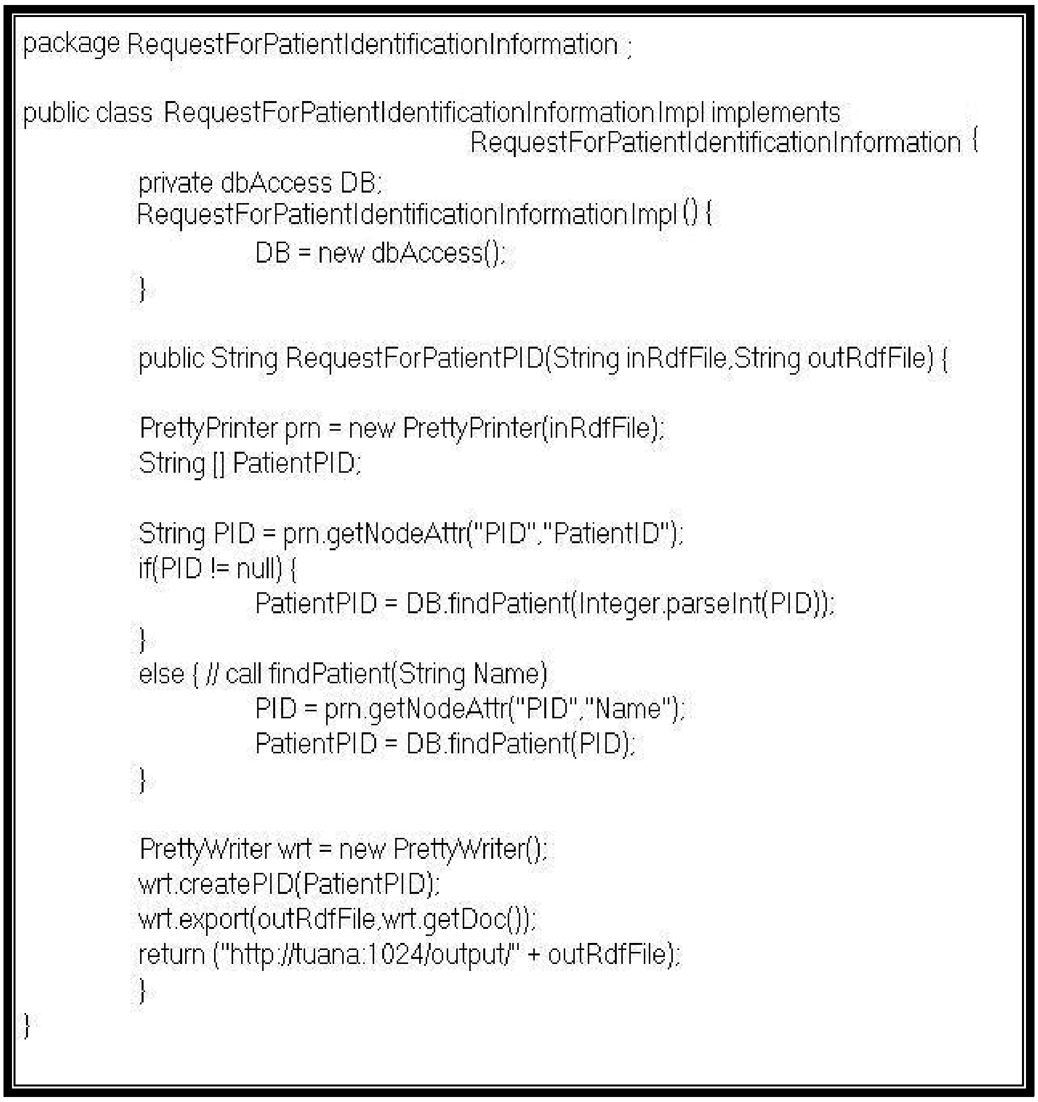
```
package RequestForPatientIdentificationInformation ;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface RequestForPatientIdentificationInformationIF extends Remote {

  public String RequestForPatientPID(String inRdfFile,String outRdfFile)
            throws RemoteException;
}
```

Figure 4.1: Service Endpoint Definition

The service endpoint interface must conform to the following rules [25];

- It extends the java.rmi.Remote interface.
- It must not have constant declarations, such as public final static.
- The methods must throw the java.rmi.RemoteException or one of its subclasses. (The methods may also throw service-specific exceptions.)
- Method parameters and return types must be supported JAX-RPC types.

The implementation class that implements the above SEI is given in Figure 4.2.



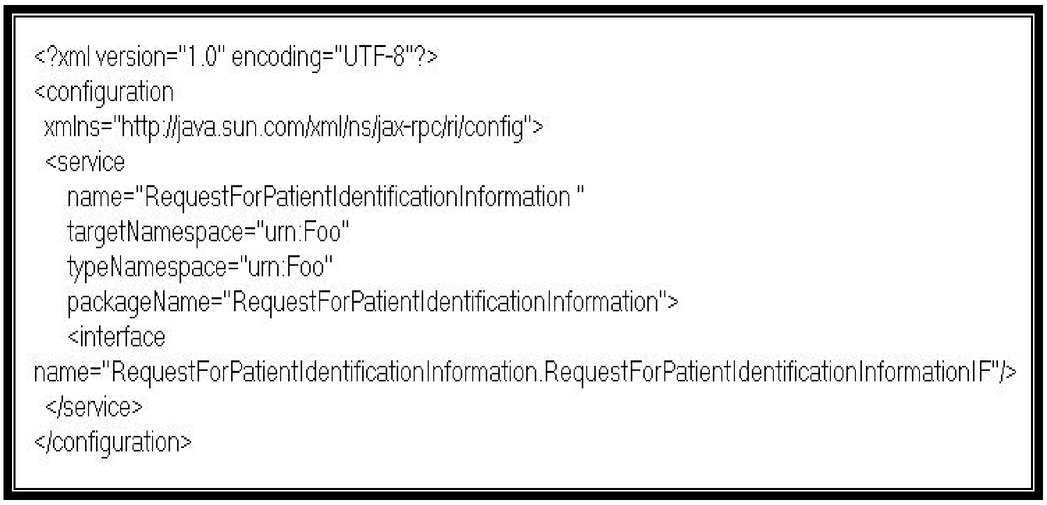
```
package RequestForPatientIdentificationInformation ;

public class  RequestForPatientIdentificationInformationImpl implements
                                        RequestForPatientIdentificationInformation {
        private dbAccess DB;
        RequestForPatientIdentificationInformationImpl () {
                DB = new dbAccess();
        }

        public String RequestForPatientPID(String inRdfFile,String outRdfFile) {

        PrettyPrinter prn = new PrettyPrinter(inRdfFile);
        String [] PatientPID;

        String PID = prn.getNodeAttr("PID","PatientID");
        if(PID != null) {
                PatientPID = DB.findPatient(Integer.parseInt(PID));
        }
        else { // call findPatient(String Name)
                PID = prn.getNodeAttr("PID","Name");
                PatientPID = DB.findPatient(PID);
        }

        PrettyWriter wrt = new PrettyWriter();
        wrt.createPID(PatientPID);
        wrt.export(outRdfFile,wrt.getDoc());
        return ("http://tuana:1024/output/" + outRdfFile);
        }
}
```

Figure 4.2: Service Implementation Class

4.1.2 Building *RequestPatientIdentificationInformation* Service

In order to build the service, service class files (service interface, implementation and other auxiliary classes) should be compiled. After the compilation, the WSDL file is generated. The **wscompile** [25] tool creates the WSDL and mapping files. The mapping file contains information that correlates the mapping between the Java interfaces and the WSDL definition.

The wscompile tool reads an interface configuration file that specifies information about the SEI. An example configuration file is given in Figure 4.3.



```
<?xml version="1.0" encoding="UTF-8"?>
<configuration
 xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
  <service
    name="RequestForPatientIdentificationInformation "
    targetNamespace="urn:Foo"
    typeNamespace="urn:Foo"
    packageName="RequestForPatientIdentificationInformation">
    <interface
name="RequestForPatientIdentificationInformation.RequestForPatientIdentificationInformationIF"/>
  </service>
</configuration>
```

Figure 4.3: Interface Configuration File

This configuration file tells wscompile to create a WSDL file named RequestPatient-IdentificationInformation *.wsdl* with the following information;

- The service name is RequestPatientIdentificationInformation.
- The WSDL target and type namespace is urn:Foo.
- The SEI is; RequestPatientIdentificationInformation.RequestPatientIdentificationInformationIF.

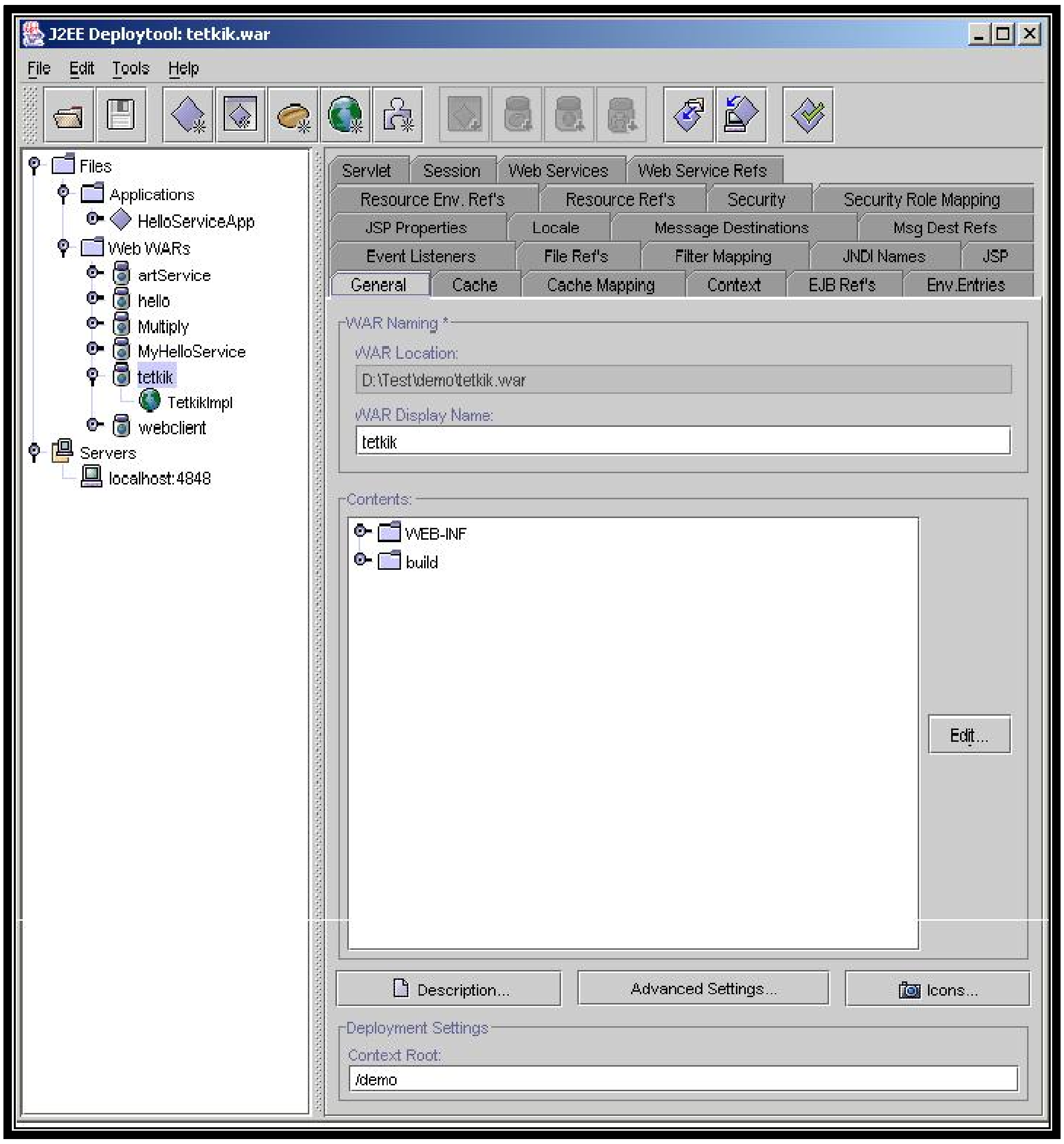
4.1.3 Packaging and Deploying RequestPatientIdentificationInformation Service

Behind the scenes, a JAX-RPC Web service is implemented as a servlet [28]. Servlets are Java programming language classes that dynamically process requests and construct responses. The **deploytool** [25] utility is used to package the service (Figure 4.4). During this process the wizard performs the following tasks:

- Creates the Web application deployment descriptor,
- Creates a WAR file,
- Adds the deployment descriptor and service files to the WAR file.

Service deployment is done again using deploytool. Now the sample service is ready and can be used by client programmes.

During the deployment of a JAX-RPC service endpoint, the deployment tool configures one or more protocol bindings for this service endpoint. A binding ties an abstract service endpoint definition to a specific protocol and transport. An example of a binding is SOAP protocol over HTTP.



WAR Naming *
WAR Location: D:\Test\demo\tetkik.war
WAR Display Name: tetkik
Contents:
Deployment Settings
Context Root: /demo

Figure 4.4: J2EE Deploytool API

After completing the above tasks, the Web service is ready for clients (i.e., consumers) to be invoked.

4.2 Artemis Architecture

Artemis Project [3], a semantic web service-based P2P infrastructure for the interoperability of medical information systems, supported by the European Commission, aims to provide interoperability in the healthcare domain. In this section the architecture of Artemis is explained.

In Artemis, healthcare institutes communicate with each other through mediators which resolve their differences bilaterally [2]. 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.

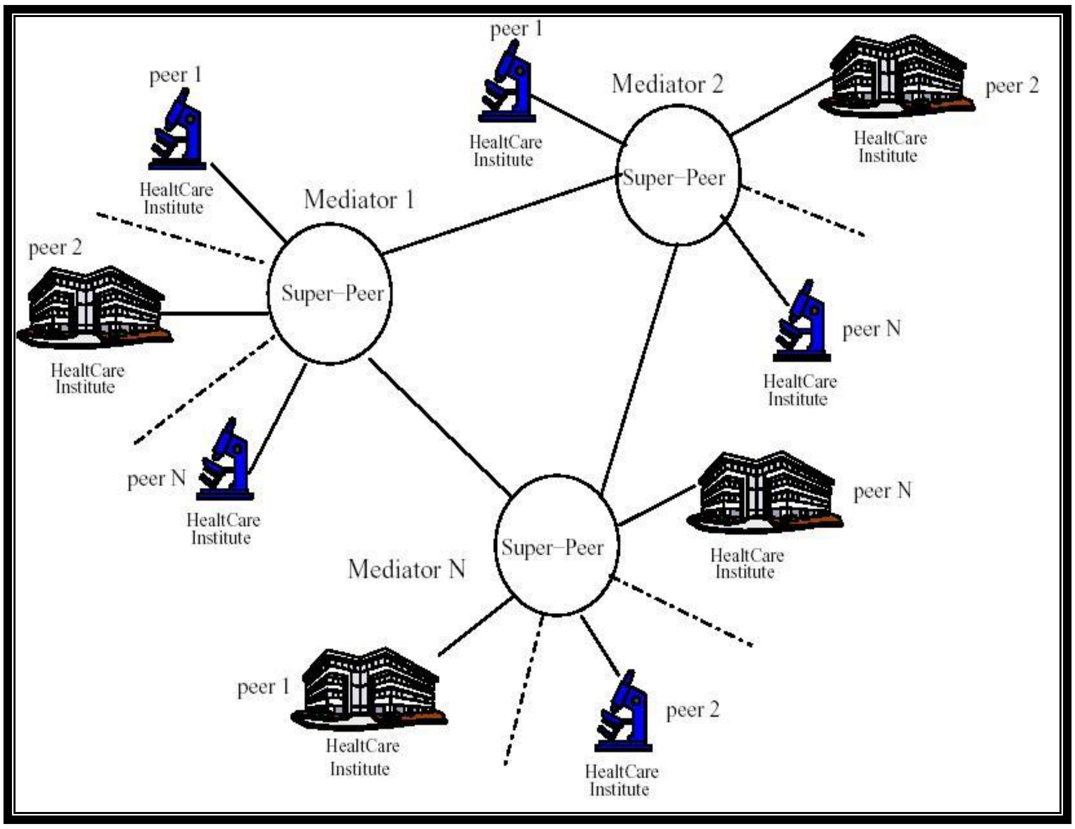


Figure 4.5: Artemis P2P Architecture

Artemis Web service architecture does not rely on globally agreed ontologies; rather healthcare institutes may 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. Artemis architecture then helps to reconcile the semantic differences among healthcare institutes through the mediator component. Artemis has P2P communication architecture in order to provide scalability and discovery of other mediators [2].

In Artemis, the source and target healthcare institutes willing to exchange information may have different ontologies. However, the mapping of these different ontologies is achieved through the reference ontologies stored in the mediator: the generic *Service Functionality* and *Service Message* ontologies. The mediator resolves the semantic differences between source and target ontologies by using these ontologies. 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.

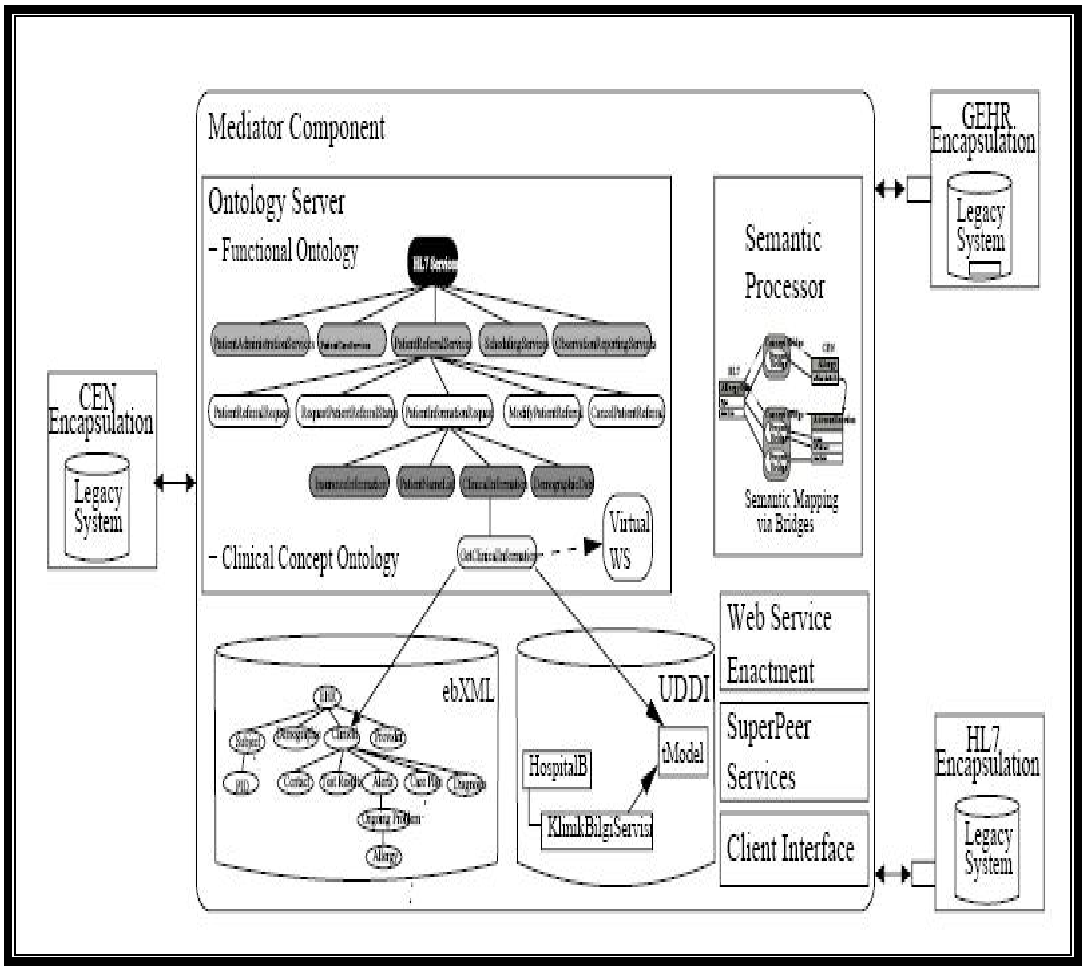


Figure 4.6: An Overview of the Mediator

The mediator architecture shown in Figure 4.6 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. 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.
- *Semantic Processor*: There may be more than one *Service Functionality* and *Service Message* ontology in the mediator and the mediator generates the mappings between them using its own reference ontologies based on the healthcare standards. In Artemis, MAFRA [32] is used to represent the mappings and to transform the ontology instances. At runtime the source ontology instances are transformed into target ontology instances by providing the source instance and the RDF [49] 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 [31] is used to deploy and invoke Web services in JXTA [45] environment.
- *Super peer Services Component* contains the services that provide 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.

4.3 Demonstration of Healthcare Web Services on Artemis Framework (A Sample Scenario)

The following scenario is provided to show the usage of semantically enriched web services in the healthcare domain. The scenario is a demonstration of the overall system described in this thesis. As previously stated, the Artemis P2P Framework is used for annotation. In the scenario *RequestPatientActionsOfType* service, described in Chapter 3, is used in the figures and the description is based on this service.

Suppose that a healthcare institute (hospital A) decides to generate the RequestPatient-ActionsOfType service described in Chapter 3. The institute also constructs the Message and Functionality Ontologies based on prominent healthcare standards, HL7 in our case. After generating the services, to be able to deploy and make the services available to other units, the hospital should register itself to Artemis P2P network.

Figure 4.7, given in the Artemis Demo Outline [2], shows the steps of demo on Artemis Framework.

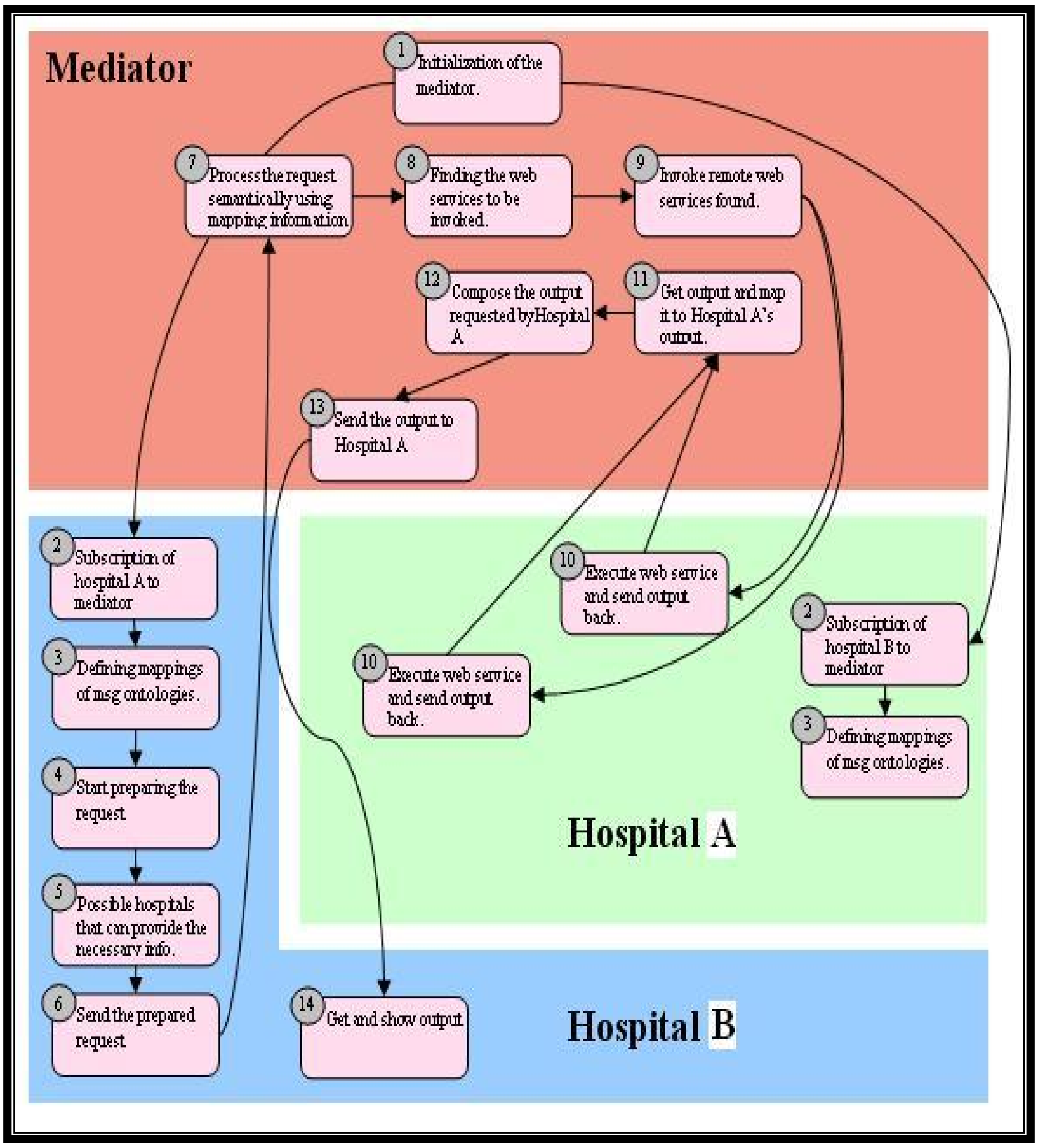

Mediator
1 Initialization of the mediator.
7 Process the request semantically using mapping information
8 Finding the web services to be invoked.
9 Invoke remote web services found.
12 Compose the output requested by Hospital A
11 Get output and map it to Hospital A's output.
13 Send the output to Hospital A
2 Subscription of hospital A to mediator
3 Defining mappings of msg ontologies.
4 Start preparing the request
5 Possible hospitals that can provide the necessary info.
6 Send the prepared request
14 Get and show output
10 Execute web service and send output back.
10 Execute web service and send output back.
2 Subscription of hospital B to mediator
3 Defining mappings of msg ontologies.
Hospital A
Hospital B

Figure 4.7: Artemis Demo Outline

4.3.1 Mediator Initialization

The initialization of the mediator component has following steps [2];

- Registry Client Construction (in order to access services in a UDDI registry)
- Ontology Server and Ontology Client Preparation (to store ontologies and access them respectively)

- Semantic Processor Construction (in order to parse and in some situations map ontologies and service composition or decomposition processes)
- Cache Initialization (local cache)

After creating and initializing required components, the mediator tries to join into P2P network. It searches for the ‘Artemis’ peer group, the peer group name used in demonstration, comparing the group names for a while (if the peer group is not created yet, mediator also creates the ‘Artemis’ group). After joining/creating the Artemis group, the mediator peer registers ‘request’ and ‘register’ handlers to the resolver service of the ‘Artemis’ group. These services are used by other peers in the Artemis group to register themselves to the mediator peer.

4.3.2 Subscription of Hospitals to Mediator

Upon executing client component of Artemis Demo from the command line, the main GUI appears (Figure 4.8). The registration phase is started by pressing Register (R) button in the main GUI window.

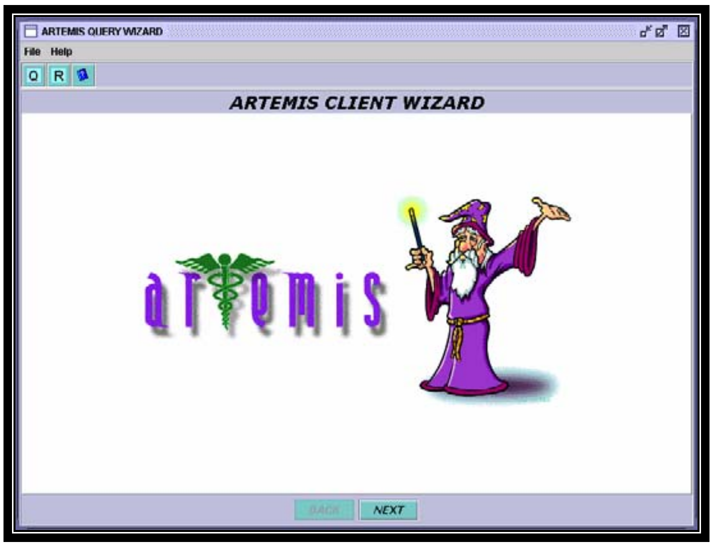


Figure 4.8: Artemis Client Wizard

The registration process has two parts;

- Setting Functionality Ontology of the subscriber
- Setting Message Ontology of the subscriber

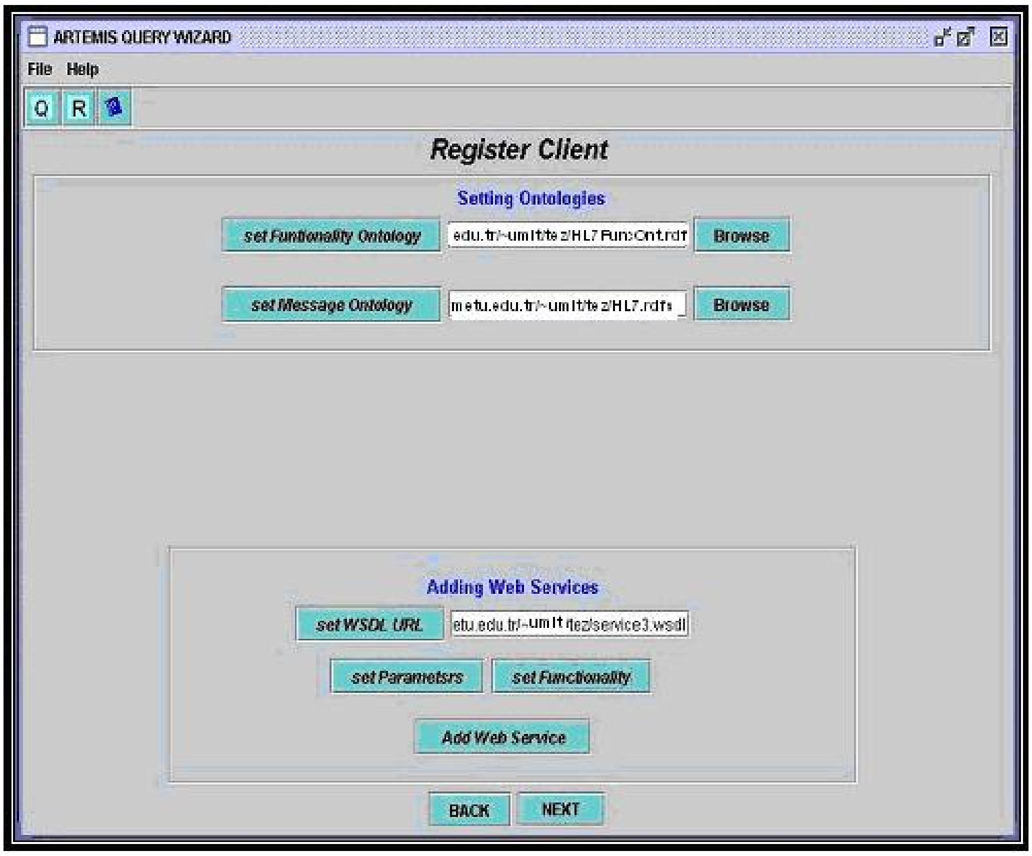


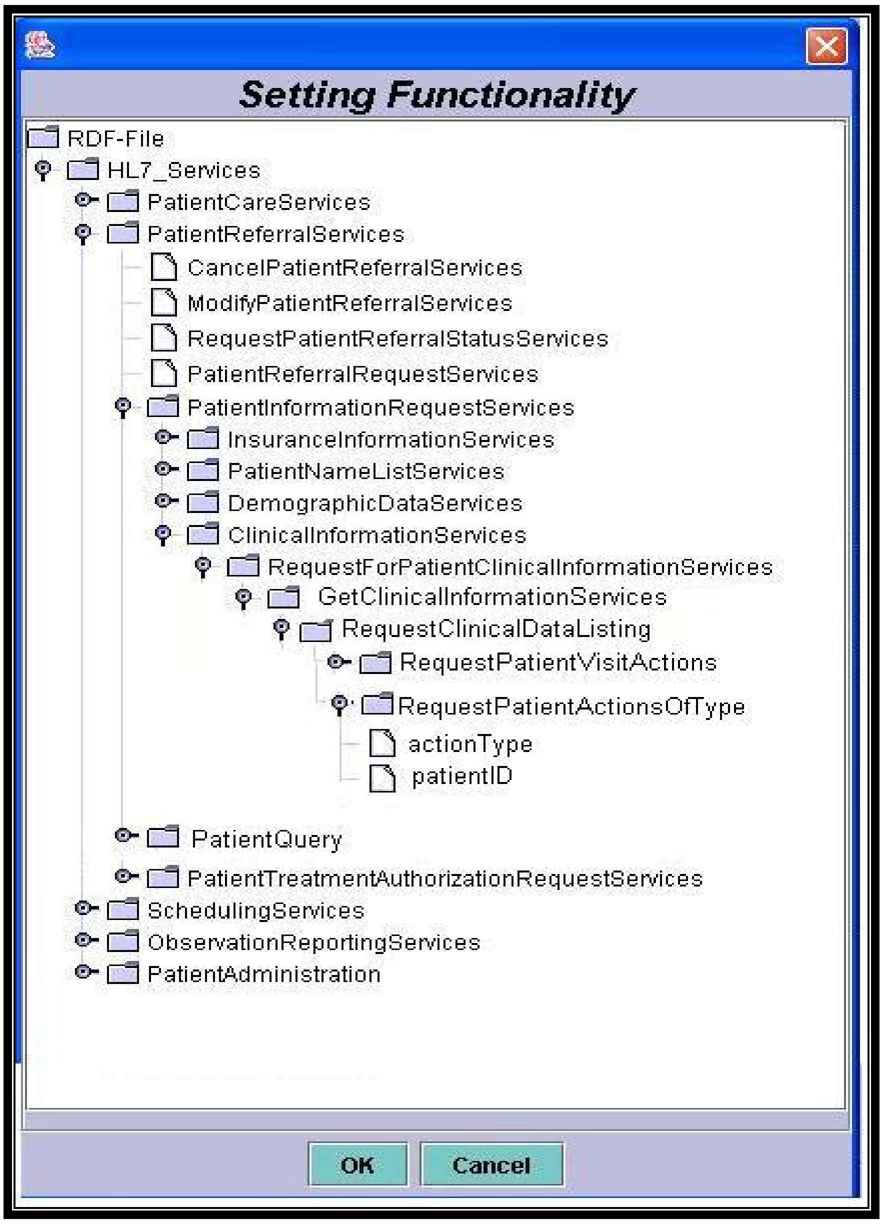
Figure 4.9: Artemis Client Registration API

The URLs of functionality and message ontologies are browsed and entered into the related text boxes (Figure 4.9). This action creates a connection with the mediator then the specified ontologies are sent to the mediator. The ontologies set during the registration phase are inserted into the ontology server of the mediator component for further processes (queries, mapping etc.).

4.3.3 Adding RequestPatientActionsOfType Service

At this step, the example service (RequestPatientActionsOfType) is added. First the interface file (WSDL description of the service) of the service is set using the main API shown in Figure 4.9. It should be noted that the WSDL description of the services are put as JXTA advertisements in the Artemis P2P network. These advertisements are located in mediator peers for others to easily find the requested information.

Next, to enable semantic search and processing in the mediator, the service is annotated with the related node of the functionality ontology. By pressing the *setFunctionality* button in the main API (Figure 4.9), a new window is opened (Figure 4.10) in which the functionality ontology can be viewed and set.



```
RDF-File
  HL7_Services
    PatientCareServices
    PatientReferralServices
      CancelPatientReferralServices
      ModifyPatientReferralServices
      RequestPatientReferralStatusServices
      PatientReferralRequestServices
      PatientInformationRequestServices
        InsuranceInformationServices
        PatientNameListServices
        DemographicDataServices
        ClinicalInformationServices
          RequestForPatientClinicalInformationServices
            GetClinicalInformationServices
              RequestClinicalDataListing
                RequestPatientVisitActions
                RequestPatientActionsOfType
                  actionType
                  patientID
      PatientQuery
      PatientTreatmentAuthorizationRequestServices
    SchedulingServices
    ObservationReportingServices
    PatientAdministration
```

Figure 4.10: Functionality Setting API

The *setParameters* button in the main API (Figure 4.9) opens another window (Fig 4.11) to set the input and output parameters of the web service. The message ontology of the related institute is parsed and shown in a tree view model. Input and output annotation is done by pressing the related buttons.

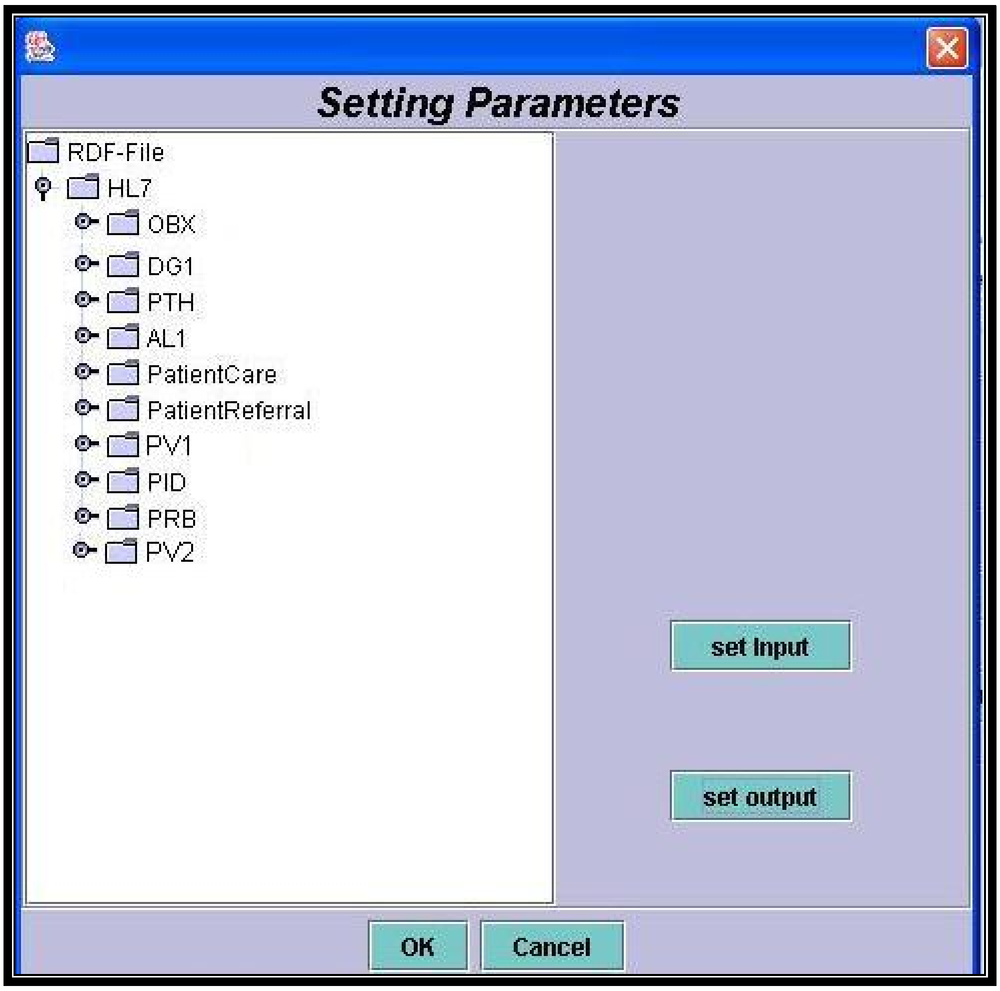


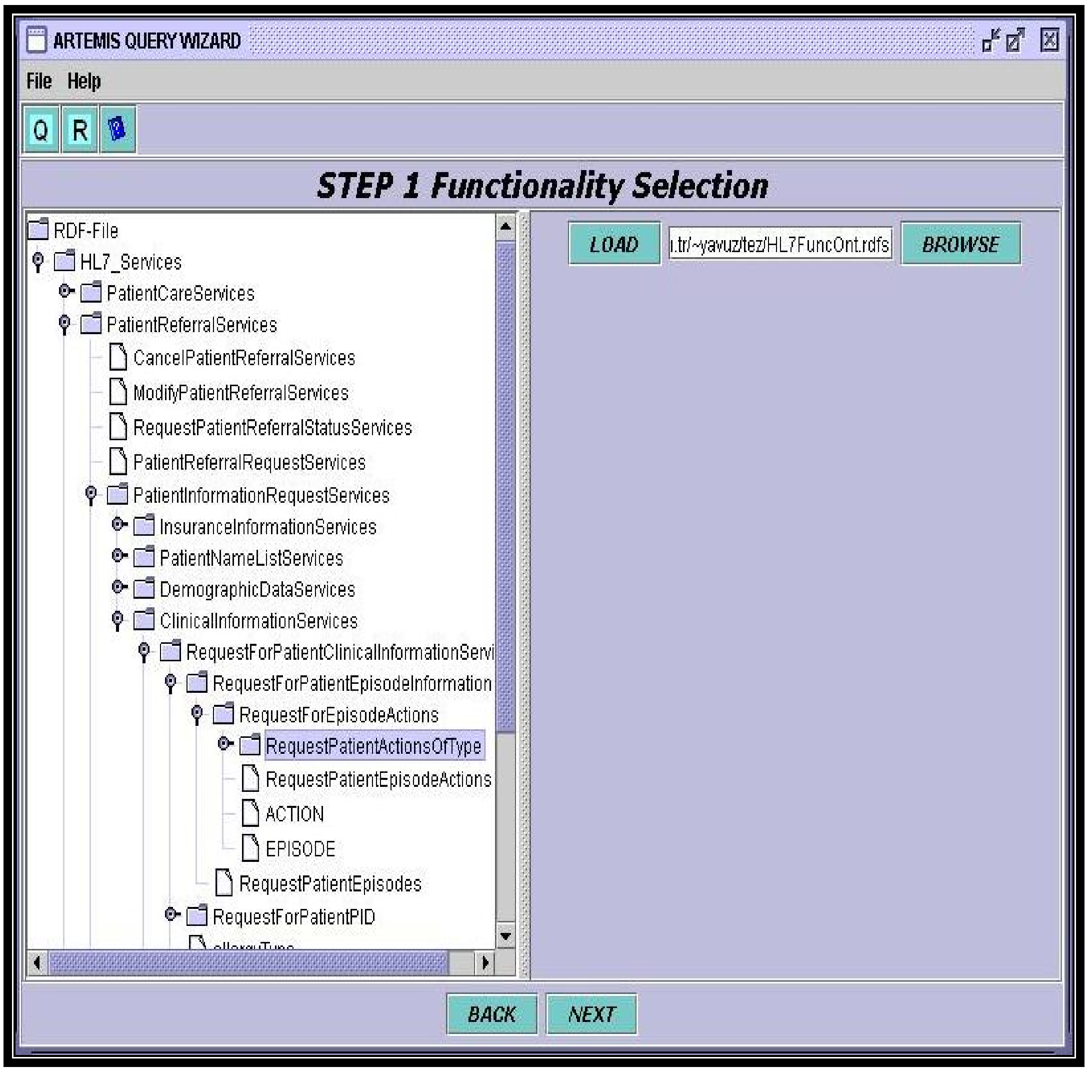
Figure 4.11: Parameter Setting API

After annotating the input and output parameters, the web service is ready to be registered. Related information is sent to mediator. The mediator publishes the web service to its local UDDI registry.

Adding RequestPatientActionsOfType service process is finished by pressing the "*Add Web Service*" button in the registration wizard (Figure 4.9).

4.3.4 Preparation and Sending of the Request (Consumer Part)

Hospital B initiates the scenario by requesting the action information of a specified patient from its mediator. The first task to be done is to specify the classification of the service requested according to its functionality. While registering itself to the Artemis system, Hospital B has already specified its functionality and message ontologies which are CEN compliant in our case. When Hospital B needs to search a service through the Artemis network, the CEN functionality ontology is presented to the clinicians in Hospital B. The user selects related node of the ontology to specify that he/she is searching for a service providing the actions performed on a patient. (The functionality selection window for HL7 ontology is given in Figure 4.12)



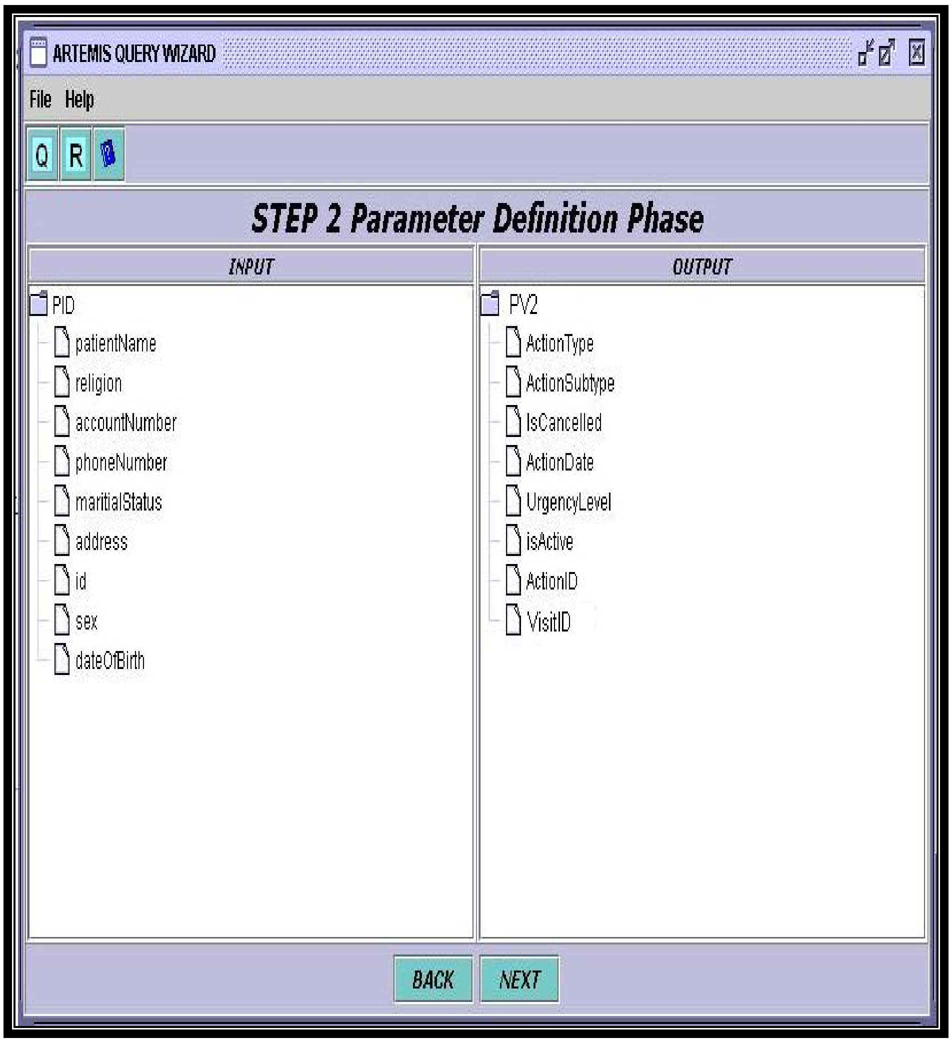
i.tr/~yavuz/tez/HL7FuncOnt.rdfs

```
RDF-File
  HL7_Services
    PatientCareServices
    PatientReferralServices
      CancelPatientReferralServices
      ModifyPatientReferralServices
      RequestPatientReferralStatusServices
      PatientReferralRequestServices
      PatientInformationRequestServices
        InsuranceInformationServices
        PatientNameListServices
        DemographicDataServices
        ClinicalInformationServices
          RequestForPatientClinicalInformationServi
            RequestForPatientEpisodeInformation
              RequestForEpisodeActions
                RequestPatientActionsOfType
                RequestPatientEpisodeActions
                ACTION
                EPISODE
              RequestPatientEpisodes
            RequestForPatientPID
```

Figure 4.12: Functionality Selection API

The mediator then consults to the Functionality Ontology of Hospital B to find the input and output parameters of the requested service in the CEN terminology. The CEN functionality ontology gives references to the Message Ontology in order to describe the structures of the input and output parameters.

In Figure 4.13, the input and output parameters of a "RequestPatientActionsOfType" service are depicted to the user.



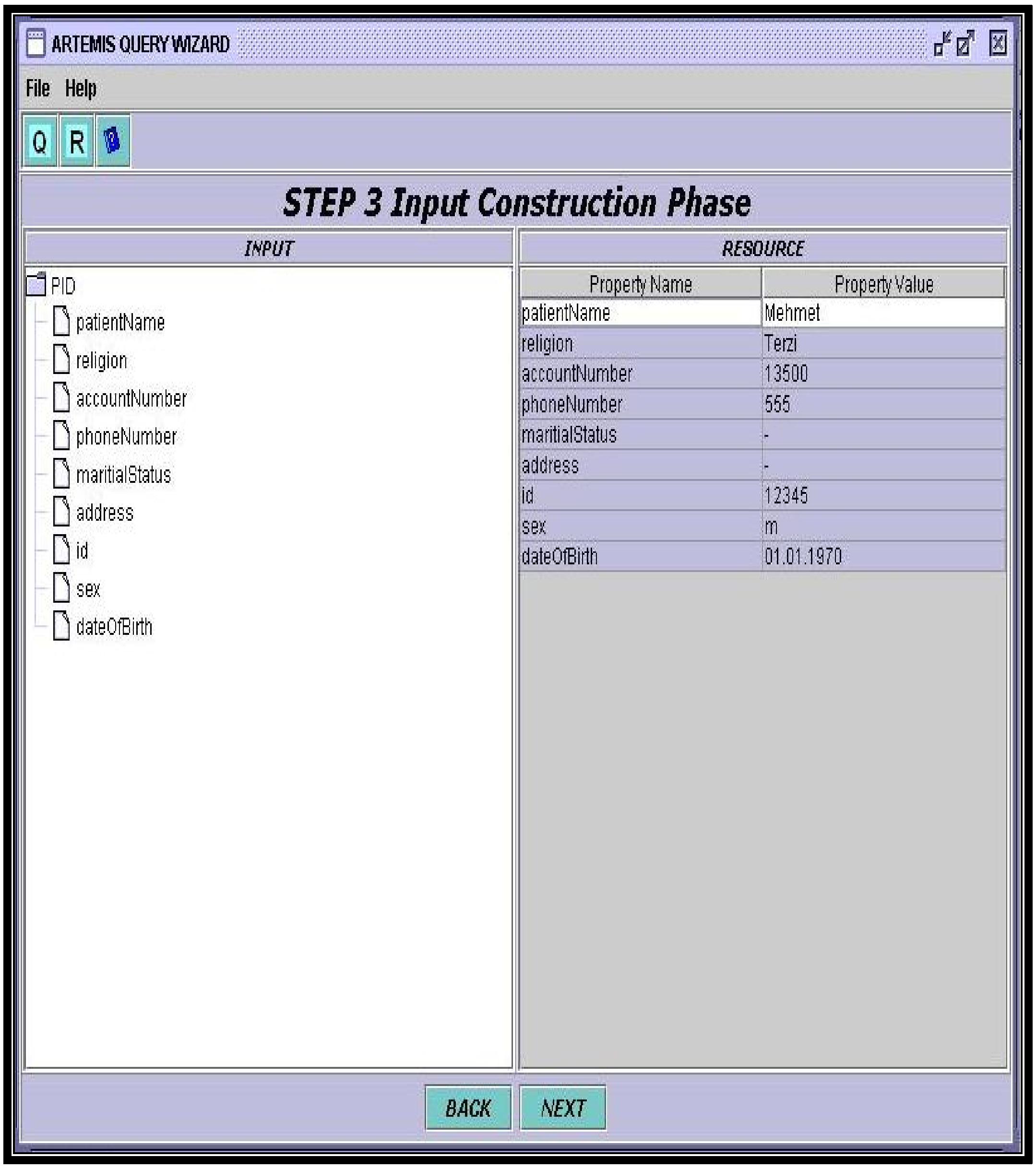
PID
- patientName
- religion
- accountNumber
- phoneNumber
- maritialStatus
- address
- id
- sex
- dateOfBirth

PV2
- ActionType
- ActionSubtype
- IsCancelled
- ActionDate
- UrgencyLevel
- isActive
- ActionID
- VisitID

Figure 4.13: Parameter Definition API

In the next step the user fills in the input parameters (Figure 4.14). Hospital B specifies the identification of the patient whose clinical information (actions performed) is sought.

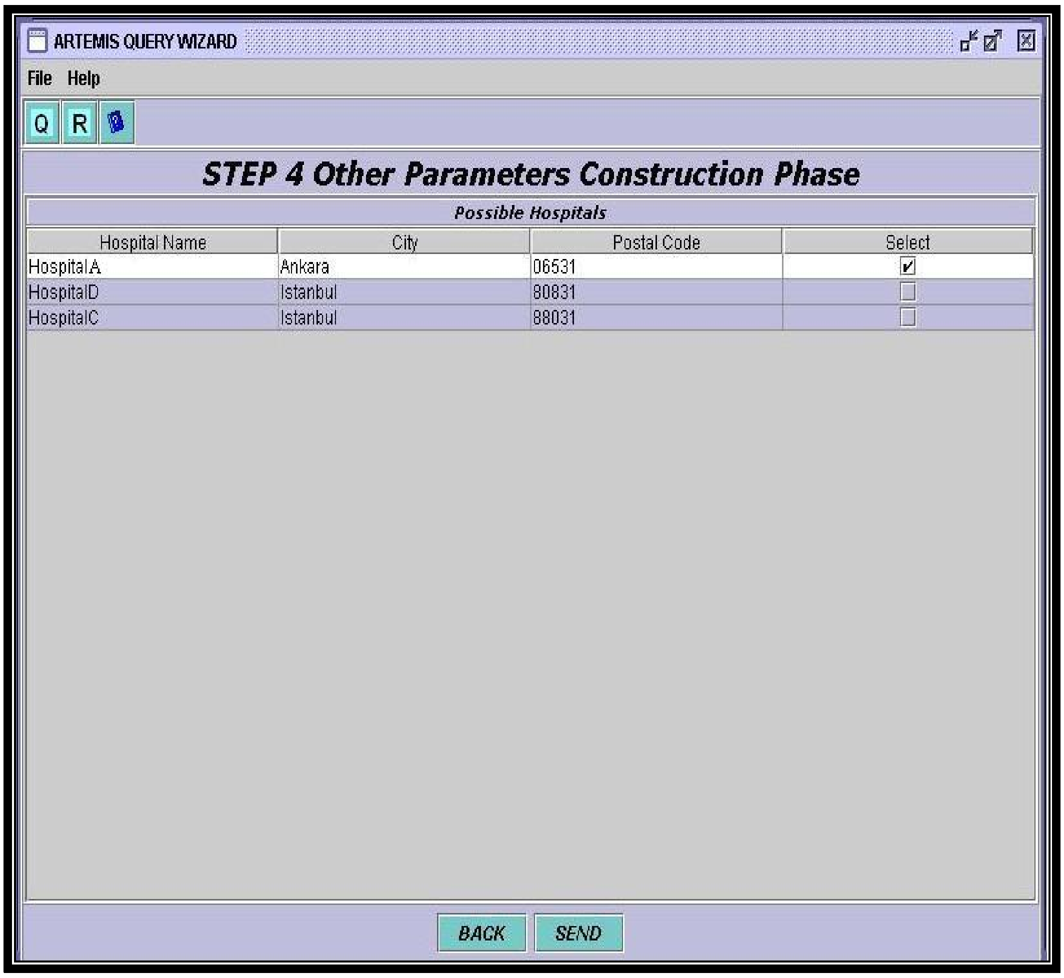
When the required service is found in the Artemis Network, these input parameters is mapped to the input parameters of the found Web service through the Ontology Mapping definitions [3].



Property Name	Property Value
patientName	Mehmet
religion	Terzi
accountNumber	13500
phoneNumber	555
maritialStatus	-
address	-
id	12345
sex	m
dateOfBirth	01.01.1970

Figure 4.14: Input Construction API

In this step, Hospital B is presented with a list of available Hospitals in the Artemis Network (Figure 4.15). Hospital B may have a prior knowledge about where to look for the clinical information of a specified patient, according to his clinical history, or it may ask each of the available hospitals whether they have the clinical information of the required patient. In our case, Hospital A is chosen; as a result the request will be sent to Hospital A only.



STEP 4 Other Parameters Construction Phase

Possible Hospitals

Hospital Name	City	Postal Code	Select
HospitalA	Ankara	06531	✓
HospitalD	Istanbul	80831	☐
HospitalC	Istanbul	88031	☐

Figure 4.15: List of Available Hospitals

4.3.5 Semantic Mediation

The two hospitals have different message ontologies and different message structures, for example, Hospital A requires the Patient ID information as input parameter in order to return the result of "RequestPatientActionsOfType service. Hospital A's message ontology is based on HL7 whereas Hospital B is CEN compliant. At this stage, semantic mapping between the two ontologies is required. Semantic mapping is the process of semantically relating two ontologies at conceptual level and transforming the source ontology instances into the target ontology entities according to those semantic relations.

The mediator resolves the semantic differences between source and target ontologies by using these ontologies. MAFRA [32] is used to represent the mappings and to transform the ontology instances in Artemis framework. The mediator stores the previously defined mappings via 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.

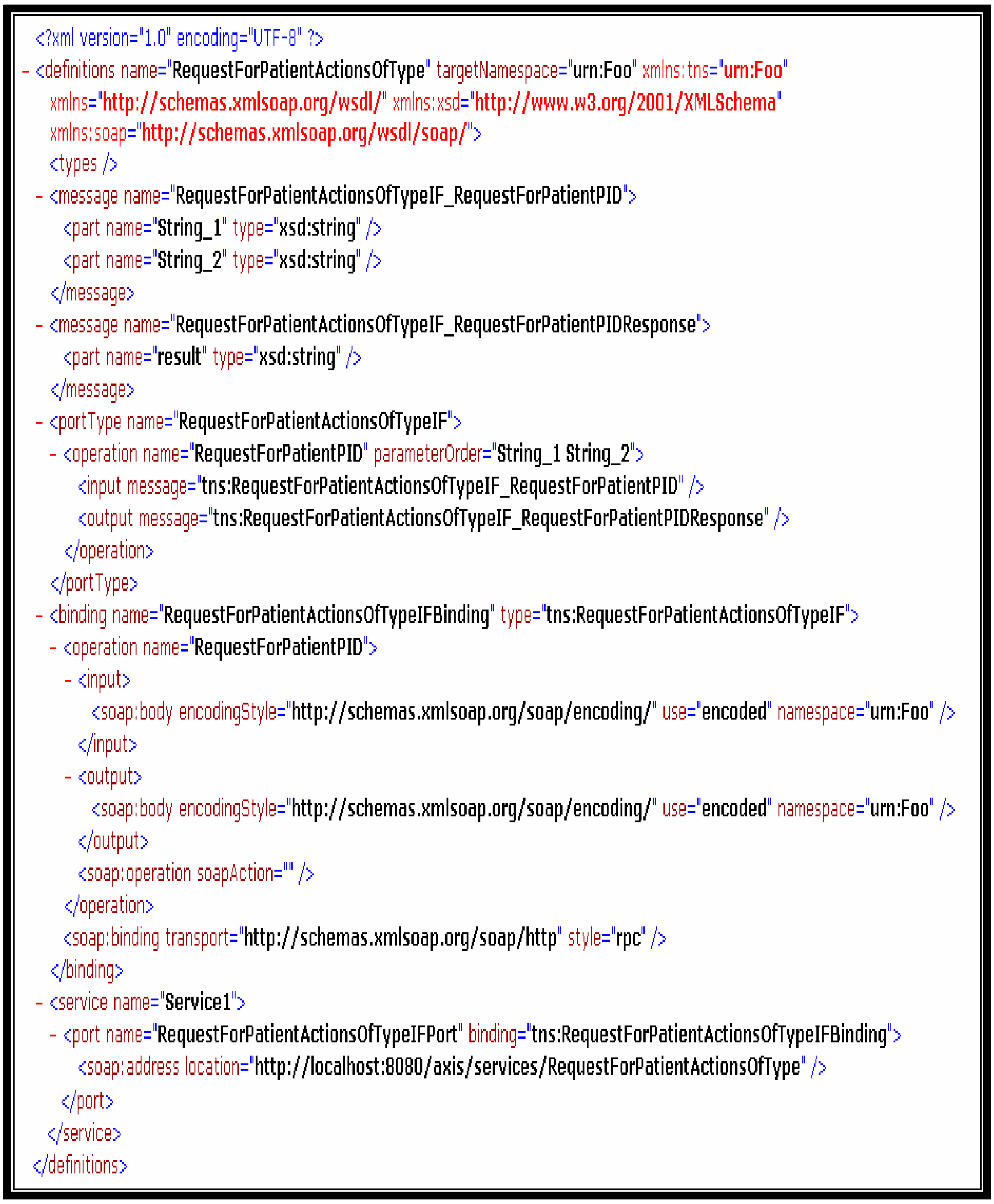
So the Semantic Processor, which resides in mediator peer, converts the CEN related nodes (DS00; for patient identification) of Hospital B, into corresponding HL7 related nodes (PID) of Hospital A. This semantic conversion and mediation enables Hospital B to invoke the service. The results of the service should also be transformed into the related nodes of the consumer peer so that it can be meaningful at the receiving end.

4.3.6 Invocation of RequestPatientActionsOfType Service

In order to make the Web service available in the JXTA P2P environment, the generated Web service must be a part of the P2P environment. For this purpose, the web services on Artemis framework are deployed using Bridge (Axis and the JXTA-SOAP project) [31].

JXTA has its own set of services like the Discovery Service, and after deploying the Web services to JXTA world, they can also be considered as JXTA services. Both, the JXTA services and Web services, are advertised in the same way. As described in Chapter 2, JXTA uses advertisements for communication and information, and JXTA services have their own set of advertisements [45]. However, Web services are identified with their WSDL files. Therefore, the WSDL information should be merged into the ordinary service advertisements. In order to illustrate how this is done in the demonstration, the advertisement process for *RequestPatientActionsOfType* service is described in this section.

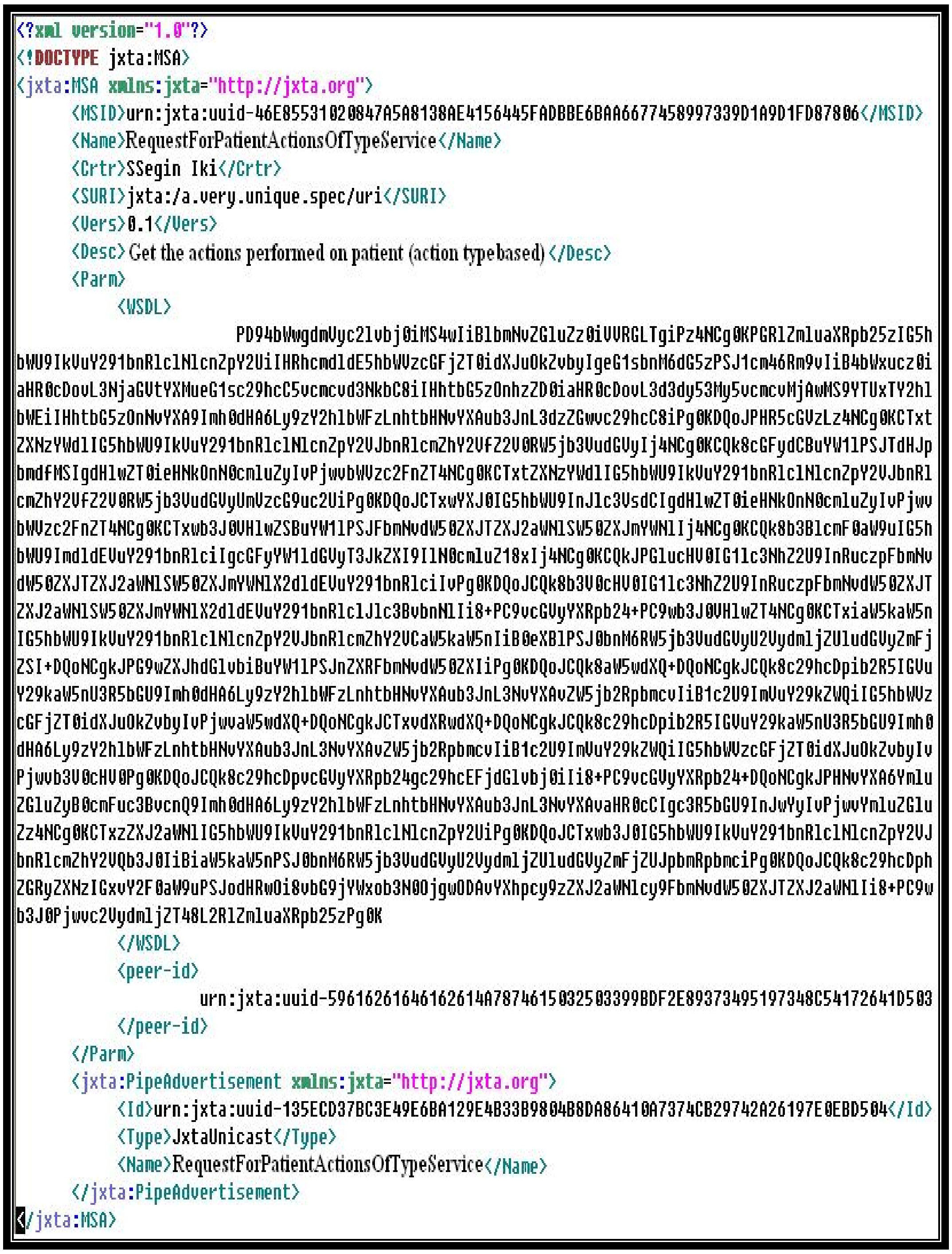
The WSDL file of the RequestPatientActionsOfType Service is given in Figure 4.16.



```
  <?xml version="1.0" encoding="UTF-8" ?>
- <definitions name="RequestForPatientActionsOfType" 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="RequestForPatientActionsOfTypeIF_RequestForPatientPID">
      <part name="String_1" type="xsd:string" />
      <part name="String_2" type="xsd:string" />
    </message>
  - <message name="RequestForPatientActionsOfTypeIF_RequestForPatientPIDResponse">
      <part name="result" type="xsd:string" />
    </message>
  - <portType name="RequestForPatientActionsOfTypeIF">
    - <operation name="RequestForPatientPID" parameterOrder="String_1 String_2">
        <input message="tns:RequestForPatientActionsOfTypeIF_RequestForPatientPID" />
        <output message="tns:RequestForPatientActionsOfTypeIF_RequestForPatientPIDResponse" />
      </operation>
    </portType>
  - <binding name="RequestForPatientActionsOfTypeIFBinding" type="tns:RequestForPatientActionsOfTypeIF">
    - <operation name="RequestForPatientPID">
      - <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="Service1">
    - <port name="RequestForPatientActionsOfTypeIFPort" binding="tns:RequestForPatientActionsOfTypeIFBinding">
        <soap:address location="http://localhost:8080/axis/services/RequestForPatientActionsOfType" />
      </port>
    </service>
  </definitions>
```

Figure 4.16: WSDL File (RequestPatientActionsOfType)

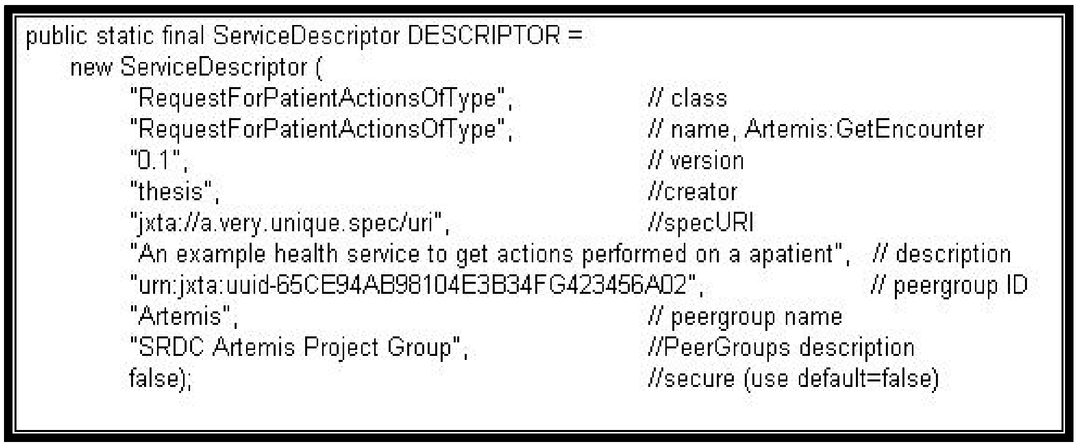
The WSDL information is merged into the Module Spec Advertisement [44] under the Parameters tag <Parm> as the <WSDL> tag (Figure 4.17). The information seen under the WSDL tag is not human understandable. However, from the Module Spec Advertisement (MSA), it is possible to parse the WSDL file all together.



```xml
<?xml version="1.0"?>
<!DOCTYPE jxta:MSA>
<jxta:MSA xmlns:jxta="http://jxta.org">
	<MSID>urn:jxta:uuid-46E85531020847A5A8138AE4156445FADBBE6BAA6677458997339D1A9D1FD87806</MSID>
	<Name>RequestForPatientActionsOfTypeService</Name>
	<Crtr>SSegin Iki</Crtr>
	<SURI>jxta:/a.very.unique.spec/uri</SURI>
	<Vers>0.1</Vers>
	<Desc>Get the actions performed on patient (action typebased)</Desc>
	<Parm>
		<WSDL>
			PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0iVVRGLTgiPz4NCg0KPGRlZmluaXRpb25zIG5h
bWU9IkVuY291bnRlclNlcnZpY2UiIHRhcmdldE5hbWVzcGFjZT0idXJuOkZvbyIgeG1sbnM6dG5zPSJ1cm46Rm9vIiB4bWxucz0i
aHR0cDovL3NjaGVtYXMueG1sc29hcC5vcmcvd3NkbC8iIHhtbG5zOnhzZD0iaHR0cDovL3d3dy53My5vcmcvMjAwMS9YTUxTY2hl
bWEiIHhtbG5zOnNvYXA9Imh0dHA6Ly9zY2hlbWFzLnhtbHNvYXAub3JnL3dzZGwvc29hcC8iPg0KDQoJPHR5cGVzLz4NCg0KCTxt
ZXNzYWdlIG5hbWU9IkVuY291bnRlclNlcnZpY2VJbnRlcmZhY2VfZ2V0RW5jb3VudGVyIj4NCg0KCQk8cGFydCBuYW1lPSJTdHJp
bmdfMSIgdHlwZT0ieHNkOnN0cmluZyIvPjwvbWVzc2FnZT4NCg0KCTxtZXNzYWdlIG5hbWU9IkVuY291bnRlclNlcnZpY2VJbnRl
cmZhY2VfZ2V0RW5jb3VudGVyUmVzcG9uc2UiPg0KDQoJCTxwYXJ0IG5hbWU9InJlc3VsdCIgdHlwZT0ieHNkOnN0cmluZyIvPjwv
bWVzc2FnZT4NCg0KCTxwb3J0VHlwZSBuYW1lPSJFbmNvdW50ZXJTZXJ2aWNlSW50ZXJmYWNlIj4NCg0KCQk8b3BlcmF0aW9uIG5h
bWU9ImdldEVuY291bnRlciIgcGFyYW1ldGVyT3JkZXI9IlN0cmluZ18xIj4NCg0KCQkJPGlucHV0IG1lc3NhZ2U9InRuczpFbmNv
dW50ZXJTZXJ2aWNlSW50ZXJmYWNlX2dldEVuY291bnRlciIvPg0KDQoJCQk8b3V0cHV0IG1lc3NhZ2U9InRuczpFbmNvdW50ZXJT
ZXJ2aWNlSW50ZXJmYWNlX2dldEVuY291bnRlclJlc3BvbnNlIi8+PC9vcGVyYXRpb24+PC9wb3J0VHlwZT4NCg0KCTxiaW5kaW5n
IG5hbWU9IkVuY291bnRlclNlcnZpY2VJbnRlcmZhY2VCaW5kaW5nIiB0eXBlPSJ0bnM6RW5jb3VudGVyU2VydmljZUludGVyZmFj
ZSI+DQoNCgkJPG9wZXJhdGlvbiBuYW1lPSJnZXRFbmNvdW50ZXIiPg0KDQoJCQk8aW5wdXQ+DQoNCgkJCQk8c29hcDpib2R5IGVu
Y29kaW5nU3R5bGU9Imh0dHA6Ly9zY2hlbWFzLnhtbHNvYXAub3JnL3NvYXAvZW5jb2RpbmcvIiB1c2U9ImVuY29kZWQiIG5hbWVz
cGFjZT0idXJuOkZvbyIvPjwvaW5wdXQ+DQoNCgkJCTxvdXRwdXQ+DQoNCgkJCQk8c29hcDpib2R5IGVuY29kaW5nU3R5bGU9Imh0
dHA6Ly9zY2hlbWFzLnhtbHNvYXAub3JnL3NvYXAvZW5jb2RpbmcvIiB1c2U9ImVuY29kZWQiIG5hbWVzcGFjZT0idXJuOkZvbyIv
Pjwvb3V0cHV0Pg0KDQoJCQk8c29hcDpvcGVyYXRpb24gc29hcEFjdGlvbj0iIi8+PC9vcGVyYXRpb24+DQoNCgkJPHNvYXA6Ymlu
ZGluZyB0cmFuc3BvcnQ9Imh0dHA6Ly9zY2hlbWFzLnhtbHNvYXAub3JnL3NvYXAvaHR0cCIgc3R5bGU9InJwYyIvPjwvYmluZGlu
Zz4NCg0KCTxzZXJ2aWNlIG5hbWU9IkVuY291bnRlclNlcnZpY2UiPg0KDQoJCTxwb3J0IG5hbWU9IkVuY291bnRlclNlcnZpY2VJ
bnRlcmZhY2VQb3J0IiBiaW5kaW5nPSJ0bnM6RW5jb3VudGVyU2VydmljZUludGVyZmFjZUJpbmRpbmciPg0KDQoJCQk8c29hcDph
ZGRyZXNzIGxvY2F0aW9uPSJodHRwOi8vbG9jYWxob3N0OjgwODAvYXhpcy9zZXJ2aWNlcy9FbmNvdW50ZXJTZXJ2aWNlIi8+PC9w
b3J0Pjwvc2VydmljZT48L2RlZmluaXRpb25zPg0K
		</WSDL>
		<peer-id>
			urn:jxta:uuid-59616261646162614A787461503250339BDF2E89373495197348C54172641D503
		</peer-id>
	</Parm>
	<jxta:PipeAdvertisement xmlns:jxta="http://jxta.org">
		<Id>urn:jxta:uuid-135ECD37BC3E49E6BA129E4B33B9804B8DA86410A7374CB29742A26197E0EBD504</Id>
		<Type>JxtaUnicast</Type>
		<Name>RequestForPatientActionsOfTypeService</Name>
	</jxta:PipeAdvertisement>
</jxta:MSA>
```

Figure 4.17: Module Spec Advertisement

The Axis Web service [67] also requires a descriptor for every Web service. The descriptor contains information of the web service and information about the deployment process. The descriptor of the service is given in Figure 4.18.



```
public static final ServiceDescriptor DESCRIPTOR =
    new ServiceDescriptor (
        "RequestForPatientActionsOfType",           // class
        "RequestForPatientActionsOfType",           // name, Artemis:GetEncounter
        "0.1",                                      // version
        "thesis",                                   //creator
        "jxta://a.very.unique.spec/uri",            //specURI
        "An example health service to get actions performed on a apatient",   // description
        "urn:jxta:uuid-65CE94AB98104E3B34FG423456A02",          // peergroup ID
        "Artemis",                                  // peergroup name
        "SRDC Artemis Project Group",               //PeerGroups description
        false);                                     //secure (use default=false)
```

Figure 4.18: Service Descriptor

The information used in the creation of the Module Spec Advertisement are “name”, “version”, “creator”, “specURI”, and “description”. The information related to the peer group is used in the deployment process. The descriptor (Figure 4.18) shows that the web service will be deployed to a peer group named *Artemis* with the given peer group ID. In case the peer group is not found a new group is created using the information in the descriptor and the web service is deployed to the newly created group.

The Web services of Hospital A will be deployed periodically and their advertisements are deleted due to timeout every minute. Once deployed, the services are ready to be invoked by any peer having its MSA or WSDL.

The Web services are invoked using Axis’ Call class [67]. A call instance is created from the CallFactory with the service descriptor, pipe advertisement, peer group name, WSDL, service name, and service port name as inputs. The operation name and time out of the call is set and with the invoke method; the web service is invoked with the required inputs, in our case **PID** and **actionType**.

4.3.7 Building the Output

The results of the web services that have been chosen conform to the output part of the Message Ontology of service provider (Hospital A). At this point, *Semantic Processor* has the information related to the results returned from the RequestPatientActionsOfType service.

After extracting the related parts from the RequestPatientActionsOfType service of Hospital A, they are mapped to the related message form of Hospital B by using the *semantic bridges* (i.e., PID → DS00, DG1→ DD01 etc.).

Finally, the action information that is queried by Hospital B is constructed and returned to Hospital B.

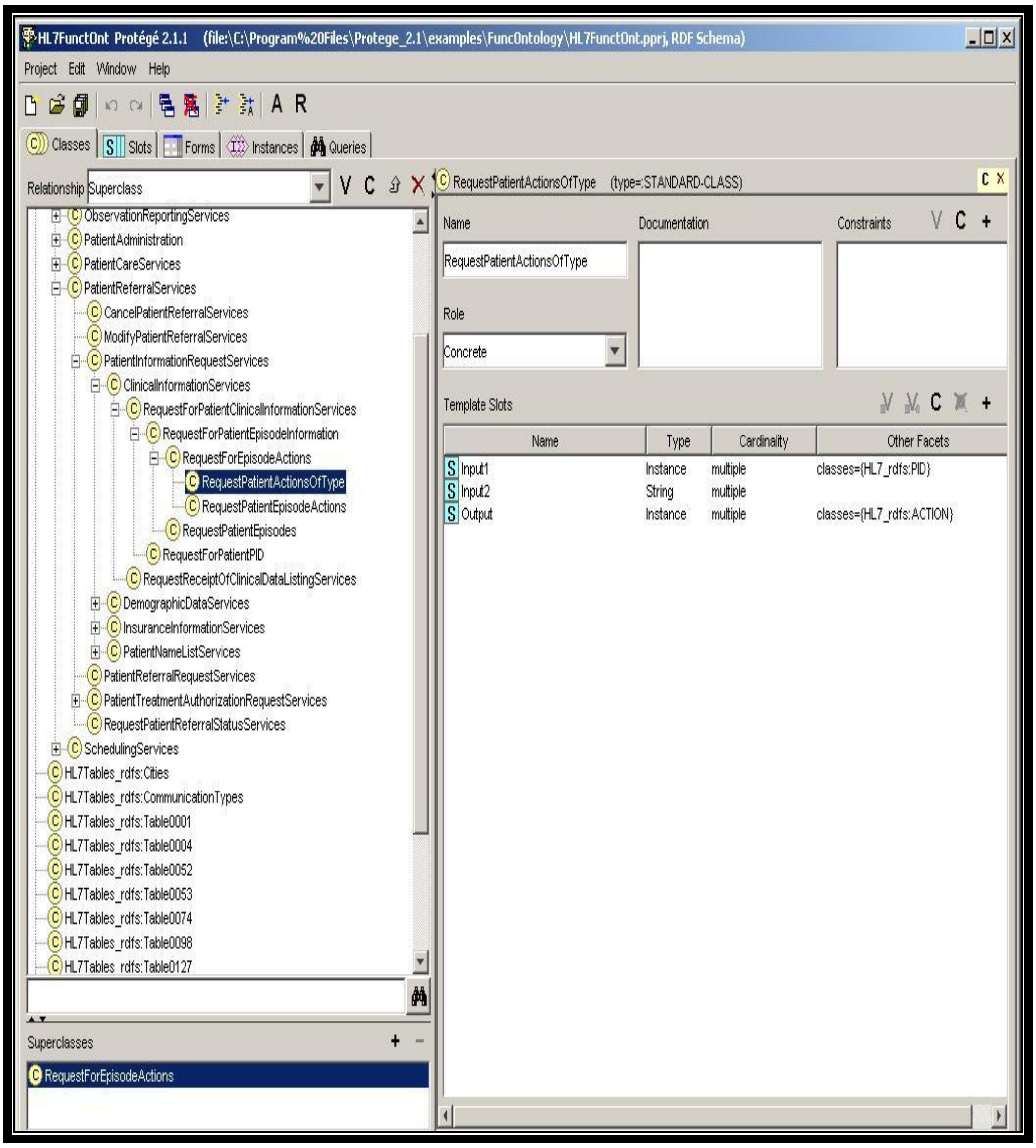
4.4 Tools Used

4.4.1 Protégé

In order to generate the Service Functionality Ontology described in Chapter 3, the Protégé-2000 ontology tool is used (Figure 4.3). Protégé-2000 allows users to construct a domain ontology, customize knowledge-acquisition forms and enter domain knowledge [46].

Protégé-2000 is designed to allow developers to reuse domain ontologies and problem-solving methods, thereby shortening the time needed for development and program maintenance. Several applications can use the same domain ontology to solve different problems, and the same problem-solving method can be used with different ontologies.

The Protégé OWL Plug-in enables to load and save OWL and RDF ontologies, edit and visualize OWL classes and their properties. Using the OWL plug-in, the Functionality Ontology of generated web services is created. It is also possible to create sample input RDF files from the generated RDF-Schemas, which makes it easy to get sample input for testing the services.


HL7FunctOnt Protégé 2.1.1 (file:\C:\Program%20Files\Protege_2.1\examples\FuncOntology\HL7FunctOnt.pprj, RDF Schema)

Relationship: Superclass

```
+ ObservationReportingServices
+ PatientAdministration
+ PatientCareServices
- PatientReferralServices
    CancelPatientReferralServices
    ModifyPatientReferralServices
  - PatientInformationRequestServices
    - ClinicalInformationServices
      - RequestForPatientClinicalInformationServices
        - RequestForPatientEpisodeInformation
          - RequestForEpisodeActions
              RequestPatientActionsOfType
              RequestPatientEpisodeActions
            RequestPatientEpisodes
          RequestForPatientPID
        RequestReceiptOfClinicalDataListingServices
    + DemographicDataServices
    + InsuranceInformationServices
    + PatientNameListServices
    PatientReferralRequestServices
  + PatientTreatmentAuthorizationRequestServices
    RequestPatientReferralStatusServices
+ SchedulingServices
HL7Tables_rdfs:Cities
HL7Tables_rdfs:CommunicationTypes
HL7Tables_rdfs:Table0001
HL7Tables_rdfs:Table0004
HL7Tables_rdfs:Table0052
HL7Tables_rdfs:Table0053
HL7Tables_rdfs:Table0074
HL7Tables_rdfs:Table0098
HL7Tables_rdfs:Table0127
```

Superclasses: RequestForEpisodeActions

RequestPatientActionsOfType (type=:STANDARD-CLASS)

Name: RequestPatientActionsOfType

Documentation

Constraints

Role: Concrete

Template Slots

Name	Type	Cardinality	Other Facets
Input1	Instance	multiple	classes={HL7_rdfs:PID}
Input2	String	multiple	
Output	Instance	multiple	classes={HL7_rdfs:ACTION}

Figure 4.19: The Protégé -2000 tool for ontology generating

4.4.2 JENA

Jena is a Java framework for building Semantic Web [53] applications. It provides a programmatic environment for RDF [49], RDFS [47] and OWL [39], including a rule-based inference engine [30].

The Jena Framework includes;

- A RDF API

- Reading and writing RDF in RDF/XML, N3 and N-Triples
- An OWL API
- In-memory and persistent storage
- RDQL – a query language for RDF

Since the generated web services uses message ontology instances, which are actually RDF files, input/output operations are done using the Jena API. Jena provides classes for reading/writing RDF in different formats, such as RDF/XML or N-Triples.

4.5 Summary

The overall goal of the system is the interoperability of healthcare systems. It is a hard process to make and share common understandings in healthcare domain since there are many standards that defines the same information.

In the proposed system and the given scenario, a hospital generates its services and registers itself to the P2P Artemis network, which is designed to provide the interoperability of enterprises in health informatics. By registering it and adding the intended services through their functionalities, both message-based and functionality-based, it becomes possible for other involved parties to search and invoke the services by the help of mediator components.

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

In this thesis, how semantically enriched web services can enable healthcare institutes to exchange information among them is described. Introducing the interoperability of medical information systems through Web services will make it possible to improve the quality of care and reduce costs in healthcare systems.

In order to achieve the interoperability among medical information systems, the Artemis project [3] is taken as basis. A real-life healthcare center is examined [60] and some of the major problems of the running system are determined. To show how the existing problems of the system can be solved, sample healthcare web services are developed considering the software in the running system. HL7 compliant functionality and message ontologies are generated to ease the semantic mediation and the services are annotated to the corresponding nodes of the functionality ontology. A sample demonstration platform is initiated using Artemis P2P Framework, on which peers can resolve each other and use their services.

As a future work;

Interoperability and information exchange is a major need in medical informatics. The process of integrating distributed applications that run on different information systems is challenging. One way that overcomes this difficulty is the use of Web service technology. So the healthcare institutes should start generating their Web services considering their local information management systems.

Besides developing medical services, the effective usage of them depends on finding and locating correct service instances by consumer institutes. The required message parameters should also be provided correctly in order to invoke and get the desired results. This is the semantics part of the mediation which requires ontology descriptions.

Medical Informatics is a complex domain in which it is not realistic to expect globally accepted ontologies. This necessitates ontology mappings between these different healthcare based domain ontologies. Although ontology mapping can be done using some methods, the health area has many different standards and data structures which complicate the things. As a result efficient mapping techniques should be developed for healthcare world.

As emphasized before, healthcare is one of the wealthiest areas which have many standards (coding, classification etc.). To cope with the interoperability issue, the healthcare standards must also be revised.

REFERENCES

[1] Artemis Deliverable D3.1.1.4: Review of the State of the Art: Healthcare Standards

[2] Artemis Demo Report, May 2004, http://www.srdc.metu.edu.tr/webpage/projects/ artemis/documents/

[3] Artemis Project, May 2004, http://www.srdc.metu.edu.tr/webpage/projects/artemis

[4] Beale, T., GEHR Object Model Architecture (GOM), June 2003, http://www.gehr. org/technical /model architecture/model architecture 4.1E.pdf

[5] CEN TC/251 (European Standardization of Health Informatics) ENV 13606, June 2003, Electronic Health Record Communication, http://www.centc251.org/

[6] Corttex Nederland, October 2004, http://www.corttex.nl/

[7] DAML Ontology Library, June 2003, http://www.daml.org/ontologies

[8] DAML Services Coalition (A.Ankolekar,M.Burstein,J.Hobbs,O.Lassila, D.Martin, S.McIlraith, S.Narayanan, M.Paolucci, T.Payne, K.Syara, H.Zeng),DAML-S: Semantic Markup for Web Services, In Proceeding of the International Semantic Web Working Symposium (SWWS), July 2001.

[9] Degoulet P, Sauquet S, Jaulent MC, Zapletal E, Lavril M, Rational and Design Considerations for a Semantic Mediator in Health Information Systems, Meth Inform Med, 1998; 37: 518-526.

[10] Dogac, A., Laleci, G., Kirbas S., Kabak Y., Sinir S., Yildiz A., Gurcan Y. "Artemis: Deploying Semantically Enriched Web Services in the Healthcare Domain", Information Systems Journal (Elsevier).

[11] Dogac, A., Kabak, Y., Laleci, G., "Enriching ebXML Registries with OWL Ontologies for Efficient Service Discovery", 14th International Workshop on Research Issues on Data Engineering, Boston, USA , March 28-29, 2004.

[12] ebXML Registry Information Model v2.1, June 2003, http://www.ebxml.org/specs/ ebRIM.pdf

[13] ebXML Registry Services Specification v2.1, June 2003, http://www.ebxml.org/ specs/ebiRS.pdf

[14] ebXML, June 2003, http://www.ebxml.org

[15] Extensible Markup Language (XML), June 2003, http://www.w3.org/XML/

[16] Gulhane Military Medicine Academia, October 2004, http://www.gata.edu .tr

[17] Health Level 7 (HL7), May 2004, http://www.hl7.org

[18] HL7 Reference Information Model, May 2004, http://www.hl7.org/rim

[19] Hypertext Transfer Protocol (HTTP), June 2003, http://www.w3.org/Protocols/

[20] IBM UDDI registry, July 2003, http://www3.ibm.com/services/uddi/

[21] IBM's tutorial on Web Services, July 2003, http://www.ibm.com/software/solutions/ webservices

[22] IHE IT Infrastructure Technical Framework, vol. 1 (ITI TF-1)

[23] ISO TC/215, International Organization for Standardization, Health Informatics Technical Committee, http://www.iso.ch/iso/en/stdsdevelopmenttc/tclist/Technical-CommitteeDetailPage.TechnicalCommitteeDetail?COMMID=4720

[24] ISO/TS 13808 Health Informatics – Requirements for an Electronic Health Record Architecture, Final Draft, 23.04.2003

[25] J2EE Tutorial, January 2003, http://java.sun.com/j2ee/1.4/docs/tutorial/doc/

[26] Java 2 Platform, Enterprise Edition, June 2004, http://java.sun.com/j2ee/

[27] Java API for XML-Based RPC, June 2004, http://java.sun.com/xml/jaxrpc/index.jsp

[28] Java Servlet Technology, June 2004, http://java.sun.com/products/servlet/

[29] Java Technology , June 2003, http://java.sun.com/

[30] JENA: A Resource Description Framework (RDF)Application Programming Interface (API), June 2004, http://www.hpl.hp.om/semweb/do /tutorial/RDF API/

[31] JXTA Bridge, July 2003, http://www.peerfear.org/jxta-bridge/

[32] MAFRA Toolkit, March 2004, http://sourceforge.net/projects/mafra-toolkit/

[33] Microsoft SQL Server, June 2003, http://www.microsoft.com/sql/

[34] Microsoft Windows, November 2003, http://www.microsoft.com/windows/

[35] Microsoft, Web Service Platform definitions, June 2004, http://msdn.microsoft.com/ library/default.asp?url=/library/enus/dnWebsrv/html/Websvcs_platform.asp

[36] Microsoft, Web Services definition, June 2003, http://msdn.microsoft.com/library /default.asp?url=/nhp/Default.asp?contentid=28000442

[37] OASIS ebXML Registry Reference Implementation Project (ebxmlrr), October 2003, http//ebxmlrr .sourceforge.net

[38] Oracle Corporation, November 2004, http://www.oracle.com

[39] OWL Web Ontology Language Overview,W3C Proposed Recommendation 15 December 2003, http://www.w3.org/TR/2003/PR-owl-features-20031215

[40] OWL-S 1.0 Release, January 2003, http://www.daml.org/services/owl-s/1.0

[41] Peer-to-Peer Computing , Dejan S. Milojicic, Vana Kalogeraki, Rajan Lukose, HP Laboratories

[42] Peer-To-Peer Working Group, June 2003, http://www.p2pwg.org

[43] Peer-To-Peer, Harnessing the Power of Disruptive Technology. O'Reilly

[44] Project JXTA 2.0: Java Programmers Guide, March 2004, http://www.jxta.org/docs/ JxtaProg- Guide_v2.pdf

[45] Project JXTA, June 2003, http://www.jxta.org

[46] Protégé Tool, June 2004, http://protege.stanford.edu

[47] RDF Schema: Resource Description Framework Schema Specification, W3C Proposed Recommendation, June 2003, http://www.w3.org/TR/PR-rdf-schema

[48] RDF Syntax: Resource Description Framework Model and Syntax Specifcation,W3C Recommendation, June 2003, http://www.w3.org/TR/REC-rdf-syntax

[49] RDF, June 2003, http://www.w3.org/RDF/

[50] Retired State Employees, Project Announcements; November 2004, http://www. emekli.gov.tr/duyuru_saglikprojesi.html

[51] Retired State Employees, Turkey, October 2004 ; http://www.emekli.gov.tr

[52] R. H. Dolin, L. Alschuler, S. Boyer, and C.Beebe, "An Update on HL7's XML-based Document Representation Standards", June 2004, http://www.amia.org/pubs/ symposia/ D200113.PDF

[53] Semantic Web, June 2003, http://www.w3.org/2001/SW/

[54] SOAP: Simple Object Access Protocol, June 2003, http://www.w3.org/TR/SOAP

[55] Solaris Operating System, July 2004, http://wwws.sun.com/software/solaris/

[56] SunOne, June 2003, http://www.sun.com/software/sunone/faq.html#2

[57] T.Berners-Lee,J.Hendler, and O.Lassila.The Semantic Web, Scientific American, May 2001

[58] The Good Electronic Health Record, June 2003, http://www.gehr.org

[59] Thomas Beale, "The GEHR System Architectures"

[60] Turkish Armed Forces Rehabilitation Center, November 2004, http://www.rehab.gata.edu.tr

[61] UDDI: Universal Description, Discovery and Integration, June 2003, http://www.uddi.org

[62] UMLS, July 2004, http://www.nlm.nih.gov/research/umls/

[63] UN/CEFACT, July 2004, http://www.unece.org/cefact

[64] Version 3 Standard: Clinical Document Architecture Framework, Release 1.0, July 2004, http://www.hl7.org/

[65] W3C: World Wide Web Consortium, June 2003, http://www.w3.org

[66] Web Service Description Language (WSDL), June 2003, http://www.w3.org/TR/wsdl

[67] Web Services, Axis Project; July 2004, http://ws.apache.org/axis/

APPENDIX A

SERVICE FUNCTIONALITY ONTOLOGY

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<!DOCTYPE uridef[
<!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns">
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema">
  <!ENTITY tables "http://www.srdc.metu.edu.tr/~banu/ws/HL7Tables.rdfs">
  <!ENTITY mo "http://www.srdc.metu.edu.tr/~banu/ws/HospitalAMO.rdfs">
<!ENTITY hl7cco "http://www.srdc.metu.edu.tr/~banu/ws/HL7.rdfs">
  <!ENTITY DEFAULT "http://www.srdc.metu.edu.tr/~banu/ws/HL7FuncOnt.rdfs">
  ]>
  <rdf:RDF
            xmlns:rdf="&rdf;#"
            xmlns:rdfs="&rdfs;#"
            xmlns:tables= "&tables;#"
            xmlns:mo= "&mo;#"
            xmlns:hl7cco="&hl7cco;#"
            xmlns="&DEFAULT;#">

            <rdfs:Class rdf:ID="HL7_Services">
                        <rdfs:comment>
                                    Class that represent all the HL7 Services
                        </rdfs:comment>
            </rdfs:Class>
            <rdfs:Class rdf:ID="PatientReferralServices">
                        <rdfs:subClassOf rdf:resource="#HL7_Services" />
                        <rdfs:comment>
                                    Class that represent all the Patient_Referral Services
                        </rdfs:comment>
            </rdfs:Class>
            <rdfs:Class rdf:ID="PatientInformationRequestServices">
                        <rdfs:subClassOf rdf:resource="#PatientReferralServices" />
                        <rdfs:comment>
                                    Class that represent all the PatientInformationRequest Services
                        </rdfs:comment>
            </rdfs:Class>
            <rdfs:Class rdf:ID="PatientReferralRequestServices">
                        <rdfs:subClassOf rdf:resource="#PatientReferralServices" />
                        <rdfs:comment>
                                    Class that represent all the I12 Services
                        </rdfs:comment>
            </rdfs:Class>
```

```
 <rdfs:Class rdf:ID="ModifyPatientReferralServices">
            <rdfs:subClassOf rdf:resource="#PatientReferralServices" />
                       <rdfs:comment>
                                   Class that represent all the I13 Services
                        </rdfs:comment>
            </rdfs:Class>
            <rdfs:Class rdf:ID="CancelPatientReferralServices">
                        <rdfs:subClassOf rdf:resource="#PatientReferralServices" />
                        <rdfs:comment>
                                   Class that represent all the I14 Services
                        </rdfs:comment>
            </rdfs:Class>
            <rdfs:Class rdf:ID="RequestPatientReferralStatusServices">
                        <rdfs:subClassOf rdf:resource="#PatientReferralServices" />
                        <rdfs:comment>
                                   Class that represent all the I15 Services
                        </rdfs:comment>
            </rdfs:Class>
            <rdfs:Class rdf:ID="PatientTreatmentAuthorizationRequestServices">
                        <rdfs:subClassOf rdf:resource="#PatientReferralServices" />
                        <rdfs:comment>
                                   Class that represent all the PatientTreatmentAuthorizationRequest
                                   Services
                        </rdfs:comment>
            </rdfs:Class>
            <rdfs:Class rdf:ID="RequestForTreatmentAuthorizationInformationServices">
                       <rdfs:subClassOf
                       rdf:resource="#PatientTreatmentAuthorizationRequestServices" />
                        <rdfs:comment>
                                   Class that represent all the I08 Services
                        </rdfs:comment>
            </rdfs:Class>
            <rdfs:Class rdf:ID="RequestForModificationToAnAuthorizationServices">
                        <rdfs:subClassOf
                       rdf:resource="#PatientTreatmentAuthorizationRequestServices" />
                        <rdfs:comment>
                                   Class that represent all the I09 Services
                        </rdfs:comment>
            </rdfs:Class>
            <rdfs:Class rdf:ID="RequestForResubmissionToAnAuthorizationServices">
                        <rdfs:subClassOf
rdf:resource="#PatientTreatmentAuthorizationRequestServices" />
                        <rdfs:comment>
                                   Class that represent all the I10 Services
                        </rdfs:comment>
            </rdfs:Class>
 <rdfs:Class rdf:ID="RequestForCancellationToAnAuthorizationServices">
                        <rdfs:subClassOf
                       rdf:resource="#PatientTreatmentAuthorizationRequestServices" />
                        <rdfs:comment>
                                   Class that represent all the I11 Services
                        </rdfs:comment>
            </rdfs:Class>
```

```
  <rdfs:Class rdf:ID="InsuranceInformationServices">
            <rdfs:subClassOf rdf:resource="#PatientInformationRequestServices" />
            <rdfs:comment>
                      Class that represent all the InsuranceInformation Services
            </rdfs:comment>
  </rdfs:Class>
 <rdfs:Class rdf:ID="PatientQueryServices">
            <rdfs:subClassOf rdf:resource="#PatientInformationRequestServices" />
            <rdfs:comment>
                      Class that represent all the PatientQuery Services
            </rdfs:comment>
  </rdfs:Class>

  <rdfs:Class rdf:ID="RequestForInsuranceInformationServices">
            <rdfs:subClassOf rdf:resource="#InsuranceInformationServices" />
            <rdfs:comment>
                      Class that represent all the I01 Services
            </rdfs:comment>
  </rdfs:Class>
  <rdfs:Class rdf:ID="UnsolicitedInsuranceInformationServices">
            <rdfs:subClassOf rdf:resource="#InsuranceInformationServices" />
            <rdfs:comment>
                      Class that represent all the I07 Services
            </rdfs:comment>
  </rdfs:Class>
  <rdfs:Class rdf:ID="PatientNameListServices">
            <rdfs:subClassOf rdf:resource="#PatientInformationRequestServices" />
            <rdfs:comment>
                      Class that represent all the PatientNameList Services
            </rdfs:comment>
  </rdfs:Class>
  <rdfs:Class rdf:ID="RequestReceiptOfPatientSelectionDisplayListServices">
            <rdfs:subClassOf rdf:resource="#PatientNameListServices" />
            <rdfs:comment>
                      Class that represent all the I02 Services
            </rdfs:comment>
  </rdfs:Class>
  <rdfs:Class rdf:ID="RequestReceiptOfPatientSelectionListServices">
            <rdfs:subClassOf rdf:resource="#PatientNameListServices" />
            <rdfs:comment>
                      Class that represent all the I03 Services
            </rdfs:comment>
  </rdfs:Class>
  <rdfs:Class rdf:ID="DemographicDataServices">
            <rdfs:subClassOf rdf:resource="#PatientInformationRequestServices" />
            <rdfs:comment>
                      Class that represent all the DemographicData Services
            </rdfs:comment>
  </rdfs:Class>
  <rdfs:Class rdf:ID="RequestForPatientDemographicDataServices">
```

```
			<rdfs:subClassOf rdf:resource="#DemographicDataServices" />
			<rdfs:comment>
				Class that represent all the I04 Services
			</rdfs:comment>
		</rdfs:Class>
		<rdfs:Class rdf:ID="ClinicalInformationServices">
			<rdfs:subClassOf rdf:resource="#PatientInformationRequestServices" />

			<rdfs:comment>
				Class that represent all the  ClinicalInformation Services
			</rdfs:comment>
		</rdfs:Class>
		<rdfs:Class rdf:ID="GetClinicalInformationServices">
			<rdfs:subClassOf rdf:resource="#ClinicalInformationServices" />
			<rdfs:comment>
				Class that represent all the I05 Services
			</rdfs:comment>
		</rdfs:Class>
	<rdfs:Class rdf:ID="RequestPatientVisitInformation">
			<rdfs:subClassOf rdf:resource="#RequestForPatientClinicalInformation" />
			<rdfs:comment>
				Class that represent visit specific information.
			</rdfs:comment>
		</rdfs:Class>

	<rdfs:Class rdf:ID="RequestForPatientClinicalInformation">
			<rdfs:subClassOf rdf:resource="#GetClinicalInformationServices" />
			<rdfs:comment>
				Class that represent patient clinical information services.
			</rdfs:comment>
		</rdfs:Class>

	<rdfs:Class rdf:ID="RequestPatientIdentificationInformation">
			<rdfs:subClassOf rdf:resource="#PatientQuery" />
			<rdfs:comment>
				Class that represent patient identification services.
			</rdfs:comment>
		</rdfs:Class>

	<rdfs:Class rdf:ID="RequestPatientActionsOfType">
			<rdfs:subClassOf rdf:resource="#RequestClinicalDataListing" />
			<rdfs:comment>
				Class that represent type-based actions performed on patient.
			</rdfs:comment>
		</rdfs:Class>
	<rdfs:Class rdf:ID="RequestPatientVisitActionInformation">
			<rdfs:subClassOf rdf:resource="#RequestClinicalDataListing" />
			<rdfs:comment>
				Class that represent all actions performed during a visit.
			</rdfs:comment>
		</rdfs:Class>
```

```
<rdfs:Class rdf:ID=" RequestClinicalDataListing ">
                <rdfs:subClassOf rdf:resource="#GetClinicalInformationServices" />
                <rdfs:comment>
                        Class that represent all clinic data request services.
                </rdfs:comment>
        </rdfs:Class>

        <rdf:Property rdf:ID="PID">
                <rdfs:domain rdf:resource="#RequestPatientActionsOfType"/>  <!-- as inp
-->
                <rdfs:domain rdf:resource="#RequestPatientVisitInformation"/>          <!-
- as inp -->
                <rdfs:domain rdf:resource="#RequestPatientIdentificationInformation"/>
<!-- as inp && outp -->
                <rdfs:range rdf:resource="&mo;#PID"/>
        </rdf:Property>

               <rdf:Property rdf:ID="PV2">
                <rdfs:domain rdf:resource="#RequestPatientActionsOfType"/> <!--  as
outp -->
                <rdfs:domain rdf:resource="#RequestPatientVisitActionInformation"/>
<!-- as outp -->
                <rdfs:range rdf:resource="&mo;#PV2"/>
        </rdf:Property>

        <rdf:Property rdf:ID="PV1">
                <rdfs:domain rdf:resource="#RequestPatientVisitInformation"/>  <!-- as
outp -->
                <rdfs:domain rdf:resource="# RequestPatientVisitActionInformation "/>
<!-- as inp -->
                <rdfs:range rdf:resource="&mo;#PV1"/>
        </rdf:Property>
        <rdfs:Class rdf:ID="RequestReceiptOfClinicalDataListingServices">
                <rdfs:subClassOf rdf:resource="#ClinicalInformationServices" />
                <rdfs:comment>
                        Class that represent all the I06 Services
                </rdfs:comment>
        </rdfs:Class>
        <!--
        Observation Reporting Part
        -->
        <rdfs:Class rdf:ID="ObservationReportingServices">
                <rdfs:subClassOf rdf:resource="#HL7_Services" />
                <rdfs:comment>
                        Class that represent all the Reporting and Scientific Trial services
                </rdfs:comment>
        </rdfs:Class>
        <rdfs:Class rdf:ID="ObservationReportingOfClinicalTrialsServices">
                <rdfs:subClassOf rdf:resource="#ObservationReportingServices" />
                <rdfs:comment>
                        Class that represent all the Scientific Tria Services
                </rdfs:comment>
        </rdfs:Class>
        <rdfs:Class rdf:ID="RegisterToClinicalTrialForObservationReporting">
                <rdfs:subClassOf
               rdf:resource="#ObservationReportingOfClinicalTrialsServices" />

                <rdfs:comment>
                        Class that represent all the C01 Services
                </rdfs:comment>
        </rdfs:Class>
        <rdfs:Class rdf:ID="CancelRegistrationToClinicalTrialForObservationReporting">
                <rdfs:subClassOf
               rdf:resource="#ObservationReportingOfClinicalTrialsServices" />
                <rdfs:comment>
                        Class that represent all the C02 Services
                </rdfs:comment>
        </rdfs:Class>
        <rdfs:Class rdf:ID="UpdateRegistrationToClinicalTrialForObservationReporting">
                <rdfs:subClassOf
               rdf:resource="#ObservationReportingOfClinicalTrialsServices" />
                <rdfs:comment>
                        Class that represent all the C03 Services
                </rdfs:comment>
        </rdfs:Class>
        <rdfs:Class rdf:ID="GoOffRegistrationToClinicalTrialForObservationReporting">
                <rdfs:subClassOf
               rdf:resource="#ObservationReportingOfClinicalTrialsServices" />
                <rdfs:comment>
                        Class that represent all the C04 Services
                </rdfs:comment>
        </rdfs:Class>
        <rdfs:Class rdf:ID="CompleteClinicalTrialForObservationReporting">
                <rdfs:subClassOf
               rdf:resource="#ObservationReportingOfClinicalTrialsServices" />
                <rdfs:comment>
                        Class that represent all the C09 Services
                </rdfs:comment>
        </rdfs:Class>
        <rdfs:Class rdf:ID="PhaseOfClinicalTrialRelatedServices">
                <rdfs:subClassOf
               rdf:resource="#ObservationReportingOfClinicalTrialsServices" />
                <rdfs:comment>
                        Class that represents all the services related to phases of Scientific
                      Trials
                </rdfs:comment>
        </rdfs:Class>
        <rdfs:Class rdf:ID="RegisterToPhaseOfClinicalTrialForObservationReporting">
                <rdfs:subClassOf rdf:resource="#PhaseOfClinicalTrialRelatedServices" />
                <rdfs:comment>
```

```
                        Class that represent all the C05 Services
                </rdfs:comment>
        </rdfs:Class>
        <rdfs:Class
rdf:ID="CancelRegistrationToPhaseOfClinicalTrialForObservationReporting">
                <rdfs:subClassOf rdf:resource="#PhaseOfClinicalTrialRelatedServices" />
                <rdfs:comment>
                        Class that represent all the C06 Services
                </rdfs:comment>
        </rdfs:Class>
        <rdfs:Class
rdf:ID="UpdateRegistrationToPhaseOfClinicalTrialForObservationReporting">
                <rdfs:subClassOf rdf:resource="#PhaseOfClinicalTrialRelatedServices" />
                <rdfs:comment>
                        Class that represent all the C07 Services
                </rdfs:comment>
        </rdfs:Class>
        <rdfs:Class
rdf:ID="GoOffRegistrationToPhaseOfClinicalTrialForObservationReporting">
                <rdfs:subClassOf rdf:resource="#PhaseOfClinicalTrialRelatedServices" />
                <rdfs:comment>
                        Class that represent all the C08 Services
                </rdfs:comment>
        </rdfs:Class>
        <rdfs:Class rdf:ID="CompletePhaseOfClinicalTrialForObservationReporting">
                <rdfs:subClassOf rdf:resource="#PhaseOfClinicalTrialRelatedServices" />
                <rdfs:comment>
                        Class that represent all the C10 Services
                </rdfs:comment>
        </rdfs:Class>
        <rdfs:Class rdf:ID="ClinicalObservationReportingServices">
                <rdfs:subClassOf rdf:resource="#ObservationReportingServices" />
                <rdfs:comment>
                        Class that represent all the Clinical Observation Services
                </rdfs:comment>
        </rdfs:Class>
        <rdfs:Class rdf:ID="RequestForClinicalObservationReportingService">
                       <rdfs:subClassOf
                       rdf:resource="#ClinicalObservationReportingServices" />
                <rdfs:comment>
                        Class that represent all the InsuranceInformation Services
                </rdfs:comment>
        </rdfs:Class>
        <rdfs:Class rdf:ID="RequestDisplayForClinicalObservationReportingService">
                <rdfs:subClassOf
               rdf:resource="#RequestForClinicalObservationReportingService" />
                <rdfs:comment>
                        Class that represent all the R01 Services
                </rdfs:comment>
        </rdfs:Class>
        <rdfs:Class rdf:ID="ResponseForClinicalObservationReportingService">
```

```
                <rdfs:subClassOf
                rdf:resource="#ClinicalObservationReportingServices" />
        <rdfs:comment>
                Class that represent all the Clinical Observation Response Services
        </rdfs:comment>
</rdfs:Class>
<rdfs:Class rdf:ID="SolicitedResponseForClinicalObservationReporting">
        <rdfs:subClassOf
       rdf:resource="#ResponseForClinicalObservationReportingService" />
        <rdfs:comment>
                Class that represent all the R02 Services
        </rdfs:comment>
</rdfs:Class>
<rdfs:Class rdf:ID="UnsolicitedResponseForClinicalObservationReporting">
        <rdfs:subClassOf
       rdf:resource="#ResponseForClinicalObservationReportingService" />
        <rdfs:comment>
                Class that represent all the R03 Services
        </rdfs:comment>
</rdfs:Class>
<rdfs:Class rdf:ID="DisplayResponseForClinicalObservationReporting">
        <rdfs:subClassOf
       rdf:resource="#ResponseForClinicalObservationReportingService" />
        <rdfs:comment>
                Class that represent all the R02 Services
        </rdfs:comment>
</rdfs:Class>
<!--
Patient Care Part
-->
<rdfs:Class rdf:ID="PatientCareServices">
        <rdfs:subClassOf rdf:resource="#HL7_Services" />
        <rdfs:comment>
                Class that represent all the PatientCare Services
        </rdfs:comment>
</rdfs:Class>
<rdfs:Class rdf:ID="PatientProblemServices">
        <rdfs:subClassOf rdf:resource="#PatientCareServices" />
        <rdfs:comment>
                Class that represent all the Patient Problem Services
        </rdfs:comment>
</rdfs:Class>
<rdfs:Class rdf:ID="PatientGoalServices">
        <rdfs:subClassOf rdf:resource="#PatientCareServices" />
        <rdfs:comment>
                Class that represent all the Patient Goal Services
        </rdfs:comment>
</rdfs:Class>
<rdfs:Class rdf:ID="PatientPathwayServices">
        <rdfs:subClassOf rdf:resource="#PatientCareServices" />
        <rdfs:comment>
```

```
			Class that represent all the Patient Pathway Services
		</rdfs:comment>
	</rdfs:Class>
	<rdfs:Class rdf:ID="PatientProblemAddServices">
		<rdfs:subClassOf rdf:resource="#PatientProblemServices" />
		<rdfs:comment>
			Class that represent all the Patient Problem Add (PC1) Services
		</rdfs:comment>
	</rdfs:Class>
	<rdfs:Class rdf:ID="PatientProblemUpdateServices">
		<rdfs:subClassOf rdf:resource="#PatientProblemServices" />
		<rdfs:comment>
			Class that represent all the Patient Problem Update (PC2) Services
		</rdfs:comment>
	</rdfs:Class>
	<rdfs:Class rdf:ID="PatientProblemDeleteServices">
		<rdfs:subClassOf rdf:resource="#PatientProblemServices" />
		<rdfs:comment>
			Class that represent all the Patient Problem Delete (PC3) Services
		</rdfs:comment>
	</rdfs:Class>
	<rdfs:Class rdf:ID="PatientProblemQueryServices">
		<rdfs:subClassOf rdf:resource="#PatientProblemServices" />
		<rdfs:comment>
			Class that represent all the Patient Problem Query (PC4) Services
		</rdfs:comment>
	</rdfs:Class>
	<rdfs:Class rdf:ID="PatientProblemResponseServices">
		<rdfs:subClassOf rdf:resource="#PatientProblemServices" />
		<rdfs:comment>
			Class that represent all the Patient Problem Response (PC5)
		       Services
		</rdfs:comment>
	</rdfs:Class>
	<rdfs:Class rdf:ID="PatientGoalAddServices">
		<rdfs:subClassOf rdf:resource="#PatientGoalServices" />
		<rdfs:comment>
			Class that represent all the Patient Goal Add (PC6) Services
		</rdfs:comment>
	</rdfs:Class>
	<rdfs:Class rdf:ID="PatientGoalUpdateServices">
		<rdfs:subClassOf rdf:resource="#PatientGoalServices" />
		<rdfs:comment>
			Class that represent all the Patient Goal Update (PC7) Services
		</rdfs:comment>
	</rdfs:Class>
	<rdfs:Class rdf:ID="PatientGoalDeleteServices">
		<rdfs:subClassOf rdf:resource="#PatientGoalServices" />
		<rdfs:comment>
			Class that represent all the Patient Goal Delete (PC8) Services
		</rdfs:comment>
```

```
</rdfs:Class>
<rdfs:Class rdf:ID="PatientGoalQueryServices">
        <rdfs:subClassOf rdf:resource="#PatientGoalServices" />
        <rdfs:comment>
                Class that represent all the Patient Goal Query (PC9) Services
        </rdfs:comment>
</rdfs:Class>
<rdfs:Class rdf:ID="PatientGoalResponseServices">
        <rdfs:subClassOf rdf:resource="#PatientGoalServices" />


        <rdfs:comment>
                Class that represent all the Patient Goal Response (PCA) Services
        </rdfs:comment>
</rdfs:Class>
<rdfs:Class rdf:ID="PatientPathwayAddServices">
        <rdfs:subClassOf rdf:resource="#PatientPathwayServices" />
        <rdfs:comment>
                Class that represent all the Patient Pathway Add (PCB) Services
        </rdfs:comment>
</rdfs:Class>
<rdfs:Class rdf:ID="PatientPathwayUpdateServices">
        <rdfs:subClassOf rdf:resource="#PatientPathwayServices" />
        <rdfs:comment>
                Class that represent all the Patient Pathway Update (PCC) Services
        </rdfs:comment>
</rdfs:Class>
<rdfs:Class rdf:ID="PatientPathwayDeleteServices">
        <rdfs:subClassOf rdf:resource="#PatientPathwayServices" />
        <rdfs:comment>
                Class that represent all the Patient Pathway Delete (PCD) Services
        </rdfs:comment>
</rdfs:Class>
<rdfs:Class rdf:ID="PatientPathwayQueryServices">
        <rdfs:subClassOf rdf:resource="#PatientPathwayServices" />
        <rdfs:comment>
                Class that represent all the Patient Pathway Query (PCE) Services
        </rdfs:comment>
</rdfs:Class>
<rdfs:Class rdf:ID="PatientPathwayResponseServices">
        <rdfs:subClassOf rdf:resource="#PatientPathwayServices" />
        <rdfs:comment>
                Class that represent all the Patient Pathway Query Response (PCF)
               Services
        </rdfs:comment>
</rdfs:Class>
<!--   Scheduling Services   -->
<rdfs:Class rdf:ID="SchedulingServices">
        <rdfs:subClassOf rdf:resource="#HL7_Services" />
        <rdfs:comment>
```

```
                Class that represent all the Scheduling Services
        </rdfs:comment>
</rdfs:Class>
<rdfs:Class rdf:ID="PlacerServices">
        <rdfs:subClassOf rdf:resource="#SchedulingServices" />
        <rdfs:comment>
                Class that represent all the services that are initiated by the placer
        </rdfs:comment>
</rdfs:Class>
<rdfs:Class rdf:ID="FillerServices">
        <rdfs:subClassOf rdf:resource="#SchedulingServices" />
        <rdfs:comment>
                Class that represent all the services that are initiated by the filler
        </rdfs:comment>
</rdfs:Class>
<rdfs:Class rdf:ID="OtherServices">
        <rdfs:subClassOf rdf:resource="#SchedulingServices" />
        <rdfs:comment>
                Class that represent all the services that are caused by maintanence
               purposes
        </rdfs:comment>
</rdfs:Class>
<rdfs:Class rdf:ID="QueryServices">
        <rdfs:subClassOf rdf:resource="#SchedulingServices" />
        <rdfs:comment>
                Class that represent all the services that are used to query the
       scheduled appointments
        </rdfs:comment>
</rdfs:Class>
<rdfs:Class rdf:ID="NoChangeOnAppointmentFillerServices">
        <rdfs:subClassOf rdf:resource="#FillerServices" />
        <rdfs:comment>
                Class that represent all the services that do not cause any change on
               the scheduled appointments
        </rdfs:comment>
</rdfs:Class>
<rdfs:Class rdf:ID="ChangeOnAppointmentFillerServices">
        <rdfs:subClassOf rdf:resource="#FillerServices" />
        <rdfs:comment>
                Class that represent all the services that cause change on the
               scheduled appointments
        </rdfs:comment>
</rdfs:Class>
<rdfs:Class rdf:ID="NoChangeOnAppointmentPlacerServices">
        <rdfs:subClassOf rdf:resource="#PlacerServices" />
        <rdfs:comment>
                Class that represent all the services that do not cause any change on
       the scheduled appointments
        </rdfs:comment>
</rdfs:Class>
```

```
		<rdfs:Class rdf:ID="ChangeOnAppointmentPlacerServices">
			<rdfs:subClassOf rdf:resource="#PlacerServices" />
			<rdfs:comment>
				Class that represent all the services that cause change on the
			scheduled appointments
			</rdfs:comment>
		</rdfs:Class>
		<rdfs:Class rdf:ID="RequestNewAppointmentBookingServices">
			<rdfs:subClassOf
			rdf:resource="#NoChangeOnAppointmentPlacerServices" />
			<rdfs:comment>
				Class that represent all S01 Services
			</rdfs:comment>
		</rdfs:Class>
		<!--PatientAdministration-->
		<rdfs:Class rdf:ID="PatientAdministration">
			<rdfs:subClassOf rdf:resource="#HL7_Services"/>
		</rdfs:Class>
		<!--Service tanimlari -->
		<rdfs:Class rdf:ID="AdmitVisitService">
			<rdfs:subClassOf rdf:resource="#PatientAdministration"/>
			<rdfs:comment>
				Class that represents all the AdmitVisit Services; invoked while
applying to a hospital as inpatient
			</rdfs:comment>
		</rdfs:Class>
		<rdfs:Class rdf:ID="RegisterService">
			<rdfs:subClassOf rdf:resource="#PatientAdministration"/>
			<rdfs:comment>
				Class that represents all the Register Services; invoked while
applying to a hospital as a patient
			</rdfs:comment>
		</rdfs:Class>
		<rdfs:Class rdf:ID="TransferService">
			<rdfs:subClassOf rdf:resource="#PatientAdministration"/>
			<rdfs:comment>
				Class that represents all the Transfer Services; invoked when a
transfer happens in the situation of a patient
			</rdfs:comment>
		</rdfs:Class>
		<rdfs:Class rdf:ID="DischargeService">
			<rdfs:subClassOf rdf:resource="#PatientAdministration"/>
			<rdfs:comment>
				Class that represents all the Discharge Services; invoked while
leaving the hospital
			</rdfs:comment>
		</rdfs:Class>

		<rdfs:Class rdf:ID="ChangeService">
			<rdfs:subClassOf rdf:resource="#PatientAdministration"/>
			<rdfs:comment>
				Class that represents all the Change Services; invoked when the
status of the patient is changed
			</rdfs:comment>
		</rdfs:Class>
		<rdfs:Class rdf:ID="AdmitVisitProcess">
			<rdfs:subClassOf rdf:resource="#AdmitVisitService"/>
			<rdfs:comment>
				Class that represents AdmitVisitProcess Service; invoked while
applying to a hospital as an inpatient
			</rdfs:comment>
		</rdfs:Class>
		<rdfs:Class rdf:ID="AdmitVisitCancel">
			<rdfs:subClassOf rdf:resource="#AdmitVisitService"/>
			<rdfs:comment>
				Class that represents AdmitVisitCancel Service;
				invoked while canceling a registeration of an inpatient
			</rdfs:comment>
		</rdfs:Class>

		<rdfs:Class rdf:ID="AdmitVisitPending">
			<rdfs:subClassOf rdf:resource="#AdmitVisitService"/>
			<rdfs:comment>
				Class that represents AdmitVisitPending Service; invoked when the
registeration of a person
				is not certain and pending as an inpatient
			</rdfs:comment>
		</rdfs:Class>
		<rdfs:Class rdf:ID="TransferProcess">
			<rdfs:subClassOf rdf:resource="#TransferService"/>
			<rdfs:comment>
				Class that represents Transfer Service; invoked when a patient is
changing situation
			</rdfs:comment>
		</rdfs:Class>
		<rdfs:Class rdf:ID="TransferCancel">
			<rdfs:subClassOf rdf:resource="#TransferService"/>
			<rdfs:comment>
				Class that represents Transfer Service;
				invoked while canceling a transfer of an inpatient
			</rdfs:comment>
		</rdfs:Class>
		<rdfs:Class rdf:ID="TransferPending">
			<rdfs:subClassOf rdf:resource="#TransferService"/>
			<rdfs:comment>
				Class that represents Transfer Service; invoked when the transfer of
a person
				is not certain and pending as an inpatient
			</rdfs:comment>
		</rdfs:Class>
```

```
	<rdfs:Class rdf:ID="DischargeProcess">
		<rdfs:subClassOf rdf:resource="#DischargeService"/>
		<rdfs:comment>
			Class that represents Discharge Service; invoked while discharging
from a hospital as  a patient
		</rdfs:comment>
	</rdfs:Class>
	<rdfs:Class rdf:ID="DischargeCancel">
		<rdfs:subClassOf rdf:resource="#DischargeService"/>
		<rdfs:comment>
			Class that represents Discharge Service;
			invoked while canceling a discharge of an inpatient
		</rdfs:comment>
	</rdfs:Class>
	<rdfs:Class rdf:ID="DischargePending">
		<rdfs:subClassOf rdf:resource="#DischargeService"/>
		<rdfs:comment>
			Class that represents Discharge Service; invoked when the
discharge of a person
			is not certain and pending as an inpatient
		</rdfs:comment>
	</rdfs:Class>
	<rdfs:Class rdf:ID="ChangeOutToIn">
		<rdfs:subClassOf rdf:resource="#ChangeService"/>
		<rdfs:comment>
			Class that represents Change Service;
			invoked while changing a status of a patient from outpatient to
inpatient
		</rdfs:comment>
	</rdfs:Class>
	<rdfs:Class rdf:ID="ChangeInToOut">
		<rdfs:subClassOf rdf:resource="#ChangeService"/>
		<rdfs:comment>
			Class that represents Change Service;
			invoked while changing a status of a patient from inpatient to
outpatient
		</rdfs:comment>
	</rdfs:Class>
	<rdfs:Class rdf:ID="RegisterProcessService">
		<rdfs:subClassOf rdf:resource="#RegisterService"/>
		<rdfs:comment>
			Class that represents RegisterProcess Service; invoked while
applying to a hospital as a normal patient
		</rdfs:comment>
	</rdfs:Class>

	<!--  Admit Patient Properties   -->
	<rdf:Property rdf:ID="patientClass">
		<rdfs:domain rdf:resource="#PatientCareServices"/>
```

```
			<rdfs:domain rdf:resource="#AdmitVisitService"/>
			<rdfs:domain rdf:resource="#RegisterProcessService"/>
			<rdfs:range rdf:resource="&tables;#Table0004"/>
		</rdf:Property>
		<rdf:Property rdf:ID="patientSex">
			<rdfs:domain rdf:resource="#AdmitVisitService"/>
			<rdfs:domain rdf:resource="#RegisterProcessService"/>
			<rdfs:range rdf:resource="&tables;#Table0001"/>
		</rdf:Property>
		<rdf:Property rdf:ID="allergyType">
			<rdfs:domain rdf:resource="#AdmitVisitService"/>
			<rdfs:domain rdf:resource="#RegisterProcessService"/>
			<rdfs:domain
	rdf:resource="#RequestForPatientClinicalInformationServices"/>
			<rdfs:range rdf:resource="&tables;#Table0127"/>
		</rdf:Property>
		<rdf:Property rdf:ID="appointmentReason">
			<rdfs:domain rdf:resource="#RequestNewAppointmentBookingServices"/>
			<rdfs:range rdf:resource="&tables;#Table0276"/>
		</rdf:Property>
		<rdf:Property rdf:ID="appointmentType">
			<rdfs:domain rdf:resource="#RequestNewAppointmentBookingServices"/>
			<rdfs:range rdf:resource="&tables;#Table0277"/>
		</rdf:Property>
		<rdf:Property rdf:ID="serviceType">
			<rdfs:domain rdf:resource="#RequestNewAppointmentBookingServices"/>
			<rdfs:range rdf:resource="&rdfs;Literal"/>
		</rdf:Property>
		<rdf:Property rdf:ID="resourceType">
			<rdfs:domain rdf:resource="#RequestNewAppointmentBookingServices"/>
			<rdfs:range rdf:resource="&rdfs;Literal"/>
		</rdf:Property>
		<rdf:Property rdf:ID="locationResourceType">
			<rdfs:domain rdf:resource="#RequestNewAppointmentBookingServices"/>
			<rdfs:range rdf:resource="&rdfs;Literal"/>
		</rdf:Property>
		<rdf:Property rdf:ID="personnelRole">
			<rdfs:domain rdf:resource="#RequestNewAppointmentBookingServices"/>
			<rdfs:range rdf:resource="&rdfs;Literal"/>
		</rdf:Property>
		<rdf:Property rdf:ID="admissionType">
			<rdfs:domain rdf:resource="#PatientCareServices"/>
			<rdfs:range rdf:resource="&rdfs;Literal"/>
		</rdf:Property>
		<rdf:Property rdf:ID="patientType">
			<rdfs:domain rdf:resource="#PatientCareServices"/>
			<rdfs:range rdf:resource="&rdfs;Literal"/>
			<rdfs:comment>
				Contains site-specific values that identify the patient type
			</rdfs:comment>
		</rdf:Property>
		<rdf:Property rdf:ID="problemRanking">
			<rdfs:domain rdf:resource="#PatientProblemServices"/>
			<rdfs:range rdf:resource="&rdfs;Literal"/>
			<rdfs:comment> primary secondary </rdfs:comment>
		</rdf:Property>
		<rdf:Property rdf:ID="problemPersistence">
			<rdfs:domain rdf:resource="#PatientProblemServices"/>
			<rdfs:range rdf:resource="&rdfs;Literal"/>
			<rdfs:comment> acute, chronic </rdfs:comment>
		</rdf:Property>
		<rdf:Property rdf:ID="problemClassification">
			<rdfs:domain rdf:resource="#PatientProblemServices"/>
			<rdfs:range rdf:resource="&rdfs;Literal"/>
			<rdfs:comment> admission, final, post-operative, pre-operative, outpatient,
discharge </rdfs:comment>
		</rdf:Property>
		<rdf:Property rdf:ID="diagnosisType">
			<rdfs:domain rdf:resource="#AdmitVisitService"/>
			<rdfs:domain rdf:resource="#RegisterProcessService"/>
			<rdfs:domain
rdf:resource="#RequestForPatientClinicalInformationServices"/>
			<rdfs:domain
rdf:resource="#RequestForClinicalObservationReportingService"/>
			<rdfs:range rdf:resource="&tables;#Table0052"/>
		</rdf:Property>
		<rdf:Property rdf:ID="diagnosisCode">
			<rdfs:domain
rdf:resource="#RequestForPatientClinicalInformationServices"/>
			<rdfs:domain
rdf:resource="#RequestForClinicalObservationReportingService"/>
			<rdfs:range rdf:resource="&tables;#Table0053"/>
		</rdf:Property>
		<rdf:Property rdf:ID="diagnosisServiceID">
			<rdfs:domain
rdf:resource="#RequestForPatientClinicalInformationServices"/>
			<rdfs:range rdf:resource="&tables;#Table0074"/>
		</rdf:Property>
		<rdf:Property rdf:ID="providerRole">
			<rdfs:domain rdf:resource="#RequestForInsuranceInformationServices"/>
			<rdfs:range rdf:resource="&tables;#Table0286"/>
		</rdf:Property>
		<rdf:Property rdf:ID="providerLocation">
			<rdfs:domain rdf:resource="#RequestForInsuranceInformationServices"/>
			<rdfs:range rdf:resource="&tables;#Cities"/>
		</rdf:Property>
		<rdf:Property rdf:ID="providerCommunicationInformation">
			<rdfs:domain rdf:resource="#RequestForInsuranceInformationServices"/>
			<rdfs:range rdf:resource="&tables;#CommunicationTypes"/>
		</rdf:Property>
		<rdf:Property rdf:ID="preferredMethodOfContact">
```

```
			<rdfs:domain rdf:resource="#RequestForInsuranceInformationServices"/>
			<rdfs:range rdf:resource="&tables;#Table0185"/>
		</rdf:Property>
		<rdf:Property rdf:ID="typeOfAgreementCode">
			<rdfs:domain rdf:resource="#RequestForInsuranceInformationServices"/>
			<rdfs:range rdf:resource="&tables;#Table0098"/>
		</rdf:Property>
		<rdf:Property rdf:ID="coverageType">
			<rdfs:domain rdf:resource="#RequestForInsuranceInformationServices"/>
			<rdfs:range rdf:resource="&tables;#Table0309"/>
		</rdf:Property>
	</rdf:RDF>
```

APPENDIX B

SERVICE MESSAGE ONTOLOGY

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE rdf:RDF [
   <!ENTITY kaon 'http://kaon.semanticweb.org/2001/11/kaon-lexical#'>
   <!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
   <!ENTITY rdfs 'http://www.w3.org/2000/01/rdf-schema#'>
]>
<rdf:RDF xml:base="file:/C:/rdf/demoMap/HL7.kaon"
   xmlns:kaon="&kaon;"
   xmlns:rdf="&rdf;"
   xmlns:rdfs="&rdfs;">
<rdfs:Class rdf:ID="PatientReferral">
   <rdfs:label xml:lang="en">PatientReferral</rdfs:label>
   <rdfs:subClassOf rdf:resource="#HL7"/>
</rdfs:Class>
<rdfs:Class rdf:ID="AL1">
   <rdfs:label xml:lang="en">AL1</rdfs:label>
   <rdfs:subClassOf rdf:resource="#HL7"/>
</rdfs:Class>
<rdfs:Class rdf:ID="HL7">
   <rdfs:label xml:lang="en">HL7</rdfs:label>
</rdfs:Class>
<rdfs:Class rdf:ID="PID">
   <rdfs:label xml:lang="en">PID</rdfs:label>
   <rdfs:subClassOf rdf:resource="#HL7"/>
</rdfs:Class>
<rdfs:Class rdf:ID="PV2">
   <rdfs:label xml:lang="en">PV2</rdfs:label>
   <rdfs:subClassOf rdf:resource="#HL7"/>
</rdfs:Class>
<rdfs:Class rdf:ID="PV1">
   <rdfs:label xml:lang="en">PV1</rdfs:label>
   <rdfs:subClassOf rdf:resource="#HL7"/>
</rdfs:Class>
<rdfs:Class rdf:ID="DG1">
   <rdfs:label xml:lang="en">DG1</rdfs:label>
   <rdfs:subClassOf rdf:resource="#HL7"/>
</rdfs:Class>
<rdfs:Class rdf:ID="PatientCare">
   <rdfs:label xml:lang="en">PatientCare</rdfs:label>
```

```
    <rdfs:subClassOf rdf:resource="#HL7"/>
</rdfs:Class>
<rdfs:Class rdf:ID="PRB">
    <rdfs:label xml:lang="en">PRB</rdfs:label>
    <rdfs:subClassOf rdf:resource="#HL7"/>
</rdfs:Class>
<rdfs:Class rdf:ID="PTH">
    <rdfs:label xml:lang="en">PTH</rdfs:label>
    <rdfs:subClassOf rdf:resource="#HL7"/>
</rdfs:Class>
<rdfs:Class rdf:ID="OBX">
    <rdfs:label xml:lang="en">OBX</rdfs:label>
    <rdfs:subClassOf rdf:resource="#HL7"/>
</rdfs:Class>
<rdf:Property rdf:ID="Name">
    <rdfs:label xml:lang="en">Name</rdfs:label>
    <rdfs:range rdf:resource="&rdfs;Literal"/>
    <rdfs:domain rdf:resource="#PID"/>
</rdf:Property>
<rdf:Property rdf:ID="observerID">
    <rdfs:label xml:lang="en">observerID</rdfs:label>
    <rdfs:range rdf:resource="&rdfs;Literal"/>
    <rdfs:domain rdf:resource="#OBX"/>
</rdf:Property>
<rdf:Property rdf:ID="hasOBX">
    <rdfs:label xml:lang="en">hasOBX</rdfs:label>
    <rdfs:domain rdf:resource="#PatientReferral"/>
    <rdfs:range rdf:resource="#OBX"/>
</rdf:Property>
<rdf:Property rdf:ID="diagnosisCode">
    <rdfs:label xml:lang="en">diagnosisCode</rdfs:label>
    <rdfs:range rdf:resource="&rdfs;Literal"/>
    <rdfs:domain rdf:resource="#DG1"/>
</rdf:Property>
<rdf:Property rdf:ID="allergyReaction">
    <rdfs:label xml:lang="en">allergyReaction</rdfs:label>
    <rdfs:range rdf:resource="&rdfs;Literal"/>
    <rdfs:domain rdf:resource="#AL1"/>
</rdf:Property>
<rdf:Property rdf:ID="diagnosisDate">
    <rdfs:label xml:lang="en">diagnosisDate</rdfs:label>
    <rdfs:range rdf:resource="&rdfs;Literal"/>
    <rdfs:domain rdf:resource="#DG1"/>
</rdf:Property>
<rdf:Property rdf:ID="Sex">
    <rdfs:label xml:lang="en">Sex</rdfs:label>
    <rdfs:range rdf:resource="&rdfs;Literal"/>
    <rdfs:domain rdf:resource="#PID"/>
</rdf:Property>
<rdf:Property rdf:ID="hasPTH">
    <rdfs:label xml:lang="en">hasPTH</rdfs:label>
```

```
  <rdfs:domain rdf:resource="#PatientCare"/>
    <rdfs:range rdf:resource="#PTH"/>
</rdf:Property>
<rdf:Property rdf:ID="diagnosisDescription">
    <rdfs:label xml:lang="en">diagnosisDescription</rdfs:label>
    <rdfs:range rdf:resource="&rdfs;Literal"/>
    <rdfs:domain rdf:resource="#DG1"/>
</rdf:Property>
<rdf:Property rdf:ID="hasPRB">
    <rdfs:label xml:lang="en">hasPRB</rdfs:label>
    <rdfs:domain rdf:resource="#PatientCare"/>
    <rdfs:range rdf:resource="#PRB"/>
</rdf:Property>
<rdf:Property rdf:ID="hasAL1">
    <rdfs:label xml:lang="en">hasAL1</rdfs:label>
    <rdfs:domain rdf:resource="#PatientReferral"/>
    <rdfs:range rdf:resource="#AL1"/>
</rdf:Property>
<rdf:Property rdf:ID="observationIdentifier">
    <rdfs:label xml:lang="en">observationIdentifier</rdfs:label>
    <rdfs:range rdf:resource="&rdfs;Literal"/>
    <rdfs:domain rdf:resource="#OBX"/>
</rdf:Property>
<rdf:Property rdf:ID="diagnosisClinician">
    <rdfs:label xml:lang="en">diagnosisClinician</rdfs:label>
    <rdfs:range rdf:resource="&rdfs;Literal"/>
    <rdfs:domain rdf:resource="#DG1"/>
</rdf:Property>
<rdf:Property rdf:ID="BirthPlace">
    <rdfs:label xml:lang="en">BirthPlace</rdfs:label>
    <rdfs:range rdf:resource="&rdfs;Literal"/>
    <rdfs:domain rdf:resource="#PID"/>
</rdf:Property>
<rdf:Property rdf:ID="PatientType">
    <rdfs:label xml:lang="en">PatientType</rdfs:label>
    <rdfs:range rdf:resource="&rdfs;Literal"/>
    <rdfs:domain rdf:resource="#PID"/>
</rdf:Property>
<rdf:Property rdf:ID="religion">
    <rdfs:label xml:lang="en">religion</rdfs:label>
    <rdfs:range rdf:resource="&rdfs;Literal"/>
    <rdfs:domain rdf:resource="#PID"/>
</rdf:Property>
<rdf:Property rdf:ID="identificationDate">
    <rdfs:label xml:lang="en">identificationDate</rdfs:label>
    <rdfs:range rdf:resource="&rdfs;Literal"/>
    <rdfs:domain rdf:resource="#AL1"/>
</rdf:Property>
<rdf:Property rdf:ID="accountNumber">
    <rdfs:label xml:lang="en">accountNumber</rdfs:label>
```

```
   <rdfs:range rdf:resource="&rdfs;Literal"/>
   <rdfs:domain rdf:resource="#PID"/>
</rdf:Property>
<rdf:Property rdf:ID="address">


   <rdfs:label xml:lang="en">address</rdfs:label>
   <rdfs:range rdf:resource="&rdfs;Literal"/>
   <rdfs:domain rdf:resource="#PID"/>
</rdf:Property>
<rdf:Property rdf:ID="allergySeverity">
   <rdfs:label xml:lang="en">allergySeverity</rdfs:label>
   <rdfs:range rdf:resource="&rdfs;Literal"/>
   <rdfs:domain rdf:resource="#AL1"/>
</rdf:Property>
<rdf:Property rdf:ID="allergyCode">
   <rdfs:label xml:lang="en">allergyCode</rdfs:label>
   <rdfs:range rdf:resource="&rdfs;Literal"/>
   <rdfs:domain rdf:resource="#AL1"/>
</rdf:Property>
<rdf:Property rdf:ID="date">
   <rdfs:label xml:lang="en">date</rdfs:label>
   <rdfs:range rdf:resource="&rdfs;Literal"/>
   <rdfs:domain rdf:resource="#OBX"/>
</rdf:Property>
<rdf:Property rdf:ID="BirthDate">
   <rdfs:label xml:lang="en">BirthDate</rdfs:label>
   <rdfs:range rdf:resource="&rdfs;Literal"/>
   <rdfs:domain rdf:resource="#PID"/>
</rdf:Property>
<rdf:Property rdf:ID="allergyType">
   <rdfs:label xml:lang="en">allergyType</rdfs:label>
   <rdfs:range rdf:resource="&rdfs;Literal"/>
   <rdfs:domain rdf:resource="#AL1"/>
</rdf:Property>
<rdf:Property rdf:ID="observationMethod">
   <rdfs:label xml:lang="en">observationMethod</rdfs:label>
   <rdfs:range rdf:resource="&rdfs;Literal"/>
   <rdfs:domain rdf:resource="#OBX"/>
</rdf:Property>
<rdf:Property rdf:ID="observationValue">
   <rdfs:label xml:lang="en">observationValue</rdfs:label>
   <rdfs:range rdf:resource="&rdfs;Literal"/>
   <rdfs:domain rdf:resource="#OBX"/>
</rdf:Property>
<rdf:Property rdf:ID="hasDG1">
   <rdfs:label xml:lang="en">hasDG1</rdfs:label>
   <rdfs:domain rdf:resource="#PatientReferral"/>
   <rdfs:range rdf:resource="#DG1"/>
</rdf:Property>
<rdf:Property rdf:ID="ActionDate">
```

```
 <rdfs:label xml:lang="en">ActionDate</rdfs:label>
    <rdfs:domain rdf:resource="#PV2"/>
</rdf:Property>
<rdf:Property rdf:ID="ActionID">
    <rdfs:label xml:lang="en">ActionID</rdfs:label>
    <rdfs:domain rdf:resource="#PV2"/>
</rdf:Property>
<rdf:Property rdf:ID="ActionSubtype">
    <rdfs:label xml:lang="en">ActionSubtype</rdfs:label>
    <rdfs:range rdf:resource="#PV2"/>
</rdf:Property>
<rdf:Property rdf:ID="ActionType">
    <rdfs:label xml:lang="en">ActionType</rdfs:label>
    <rdfs:range rdf:resource="#PV2"/>
</rdf:Property>
<rdf:Property rdf:ID="IsCancelled">
    <rdfs:label xml:lang="en">IsCancelled</rdfs:label>
    <rdfs:range rdf:resource="#PV2"/>
</rdf:Property>
<rdf:Property rdf:ID="UrgencyLevel">
    <rdfs:label xml:lang="en">UrgencyLevel</rdfs:label>
    <rdfs:range rdf:resource="#PV2"/>
</rdf:Property>
<rdf:Property rdf:ID="isActive">
    <rdfs:label xml:lang="en">isActive</rdfs:label>
    <rdfs:range rdf:resource="#PV2"/>
</rdf:Property>
<rdf:Property rdf:ID="VisitNumber">
    <rdfs:label xml:lang="en"> VisitNumber </rdfs:label>
    <rdfs:range rdf:resource="#PV2"/>
    <rdfs:range rdf:resource="#PV1"/>
</rdf:Property>
<rdf:Property rdf:ID="ClosingDate">
    <rdfs:label xml:lang="en">ClosingDate</rdfs:label>
    <rdfs:range rdf:resource="#PV1"/>
</rdf:Property>
<rdf:Property rdf:ID="OpeningDate">
    <rdfs:label xml:lang="en">OpeningDate</rdfs:label>
    <rdfs:range rdf:resource="#PV1"/>
</rdf:Property>
<rdf:Property rdf:ID="PatientID">
    <rdfs:label xml:lang="en">PatientID</rdfs:label>
    <rdfs:range rdf:resource="#PID"/>
    <rdfs:range rdf:resource="#PV1"/>
</rdf:Property>
<rdf:Property rdf:ID="PatientStatusCode">
    <rdfs:label xml:lang="en">PatientStatusCode</rdfs:label>
    <rdfs:range rdf:resource="#PV1"/>
</rdf:Property>
<rdf:Property rdf:ID="PatientType">
```

```
  <rdfs:label xml:lang="en">PatientType</rdfs:label>
    <rdfs:range rdf:resource="#PV1"/>
</rdf:Property>
<rdf:Property rdf:ID="PatientClass">
    <rdfs:label xml:lang="en"> PatientClass </rdfs:label>
    <rdfs:range rdf:resource="#PV1"/>
</rdf:Property>
<rdf:Property rdf:ID="AdmissionType">
    <rdfs:label xml:lang="en"> AdmissionType </rdfs:label>
    <rdfs:range rdf:resource="#PV1"/>
</rdf:Property>
<rdf:Property rdf:ID="HospitalService">
    <rdfs:label xml:lang="en"> HospitalService </rdfs:label>
    <rdfs:range rdf:resource="#PV1"/>
</rdf:Property>

</rdf:RDF>
```