Semantic Service Integration & Metropolitan Medical Network

A Thesis

Submitted to the Faculty

of

Indiana University

by

Nikeshbhai Patel

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science

August 2005

DEDICATION

To Papa (DAD), Arun Uncle and Meena Auntie.

ACKNOWLEDGMENTS

I will like to express my gratitude for Dr. Chung-Kuo Chang my advisor for his valuable guidance. He has been constantly inspiring me for my thesis and graduate studies. His interest and ideas to explore new realms of research had constantly motivated me through my thesis. I am very much thankful to him to encourage me and provide me facilities as well as knowledge and support to complete my graduate studies and thesis.

I would here take this opportunity to thank Dr. Rajeev Raje for being on my advisory committee and who has imparted me rich knowledge for one of the graduate course.

I would like to impart special thanks to Dr. Jeffrey Huang for not only being on my advisory committee but also for giving me valuable guidance.

I want to impart my special thanks to staff at Department of Computer and Information Science for guiding me during my graduate studies and support. Also I sincerely thank to Vickie Bucker & School of Informatics for providing me facility and support.

At this time I also want to thank Jennifer Stewart and Jeffrey Allen for being very helpful in my work.

I also cannot forget my family who has been encouraging me and also providing their full support and wishes so that I can become successful, my heartiest thanks to all of my family members.

Friends of mine who has acted as my neighbor and comrades; has filled in me cheer and strength to complete my graduate studies also deserves my sincere thanks.

Lastly I would say that this had been a wonderful experience of lifetime; I am very much thankful to everyone who has made this day possible for me.

DECLARATION

This research work is a part of Master’s thesis and Graduate Studies. Here use of any resources is for academic study only and does not hold any commercial goal. Moreover no portion of this work has been submitted for any another degree or qualification.

TABLE OF CONTENTS

Page

Page

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

Acronym	Term
caBIG	cancer Biomedical Informatics Grid
CLIPS	C Language Integrated Production System
DAML+OIL	DARPA Agent Markup Language + Ontology Inference Language
DGQL	Directed Graph Query Language
DIG	DL Implementation Group
DL	Description Logic
ebXML	electronic business eXtensible Markup Language
EMR	Electronic Medical Record
FACT	Fast Classification of Terminologies
GALEN	Generalized Architecture for Languages, Encyclopedias and Nomenclatures in Medicine
GRAIL	GALEN Representation and Integration Language
HL7	Health Level 7
ICD	International Codes for Diseases
ICIS	Information Conversion & Integration System
KBS	Knowledge Base System
KIF	Knowledge Interchange Format
KQML	Knowledge Query Manipulation Language
LARKS	Language for Advertisement and Request for Knowledge Sharing
LDAP	Lightweight Directory Access Protocol
LOINC	Logical Observer Identifier Naming Convention

MeSH	Medical Subject Headings
MMN	Metropolitan Medical network
NLM	National Library of Medicine
nRQL	new Racer Query Language
OMG	Object Management Group
OWL	Web Ontology Language
OWL-S	Web Ontology Language for Services
PCP	Primary Care Physician
RACER	Renamed ABox and Concept Expression Reasoner
RacerPro	Renamed ABox and Concept Expression Reasoner Professional
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
SNOMED-CT	the Systematized Nomenclature of Medicine Clinical Terms
TF-IDF	Term Frequency - Inverse Document Frequency
UDDI	Universal Description Discovery and Integration
UML	Unified Modeling Language
UMLS	Unified Medical Language System
XML	eXtensible Markup Language

ABSTRACT

Patel, Nikeshbhai. M.S., Purdue University, August, 2005. Semantic Service Integration & Metropolitan Medical Network. Major Professor: Dr. Chung-Kuo Chang.

Medical health partners use heterogeneous data formats, legacy software and strictly licensed vocabularies which make it hard to integrate their data and work. Integration of services and data are the two main necessities. The current architecture used provides partial solution by providing one-to-one mapping wrappers. This thesis provides discussion on difficulties encountered by the co-existence of so many medical vocabularies and efforts to provide interoperation. Also other problems are listed which hinders the interoperation between health partners.

Solution is proposed for some of these problems by forming semantic network based on multi-agent technology. Service composition and integration stages are shown to develop future advance health services. Middle layer is implemented which performs integration and provides common platform for sharing information, using global ontology and local domain ontology. Inference-based matchmaking algorithm proposed in this thesis helps in mapping and achieving our goal. Six different filtering techniques are selected and used in matchmaking algorithm. Analysis of these filtering techniques is provided to understand the integration process. In the ending section an abstract idea is proposed on basis of network architecture and matchmaking algorithm to develop Open Terminological System.

CHAPTER 1. INTRODUCTION

1.1. Objectives

The overall goal of this thesis or research was to provide a solution to heterogeneity and implement a method for interoperation between various terminologies. Other goal is to propose the network architecture based upon which advance health services can be developed. The specific objectives were to:

- Analyze the problems which prevent collaboration of research work and makes hard to share data with other health partners, clinical research, etc.
- Propose solution to the heterogeneity problem by forming a semantic network of health service providers and provide integration of services and data in this network.
- Investigate and decide the suitability of the language to develop ontology.
- Create matchmaking algorithm to perform matching and thus provide search facilities for the mentioned semantic network and also help in achieving seamless integration between medical terminologies.
- Verify the suitability of filtering techniques used in matchmaking algorithm, the language used to develop ontology, as well as architecture and approach suggested for developing highly scalable systems.

1.2. Background

It can be found that in past 15 years several approaches were taken to bring the enterprises together. Languages such as XML, BPEL, ebXML and several others provided the platform to communicate and share information.

Advance search engines have been developed to satisfy the needs of searching the information needed. Many efforts are been carried out so that research and information from any part of world can be made available to others. Advance search engines have started making use of web ontology language to implement their internal database. New technologies such as Semantic Web have changed the looks of today's web. Semantic web can support both Business-to-Consumer (B2C) interaction and Business-to-Business (B2B) collaboration. XML-based frameworks, protocols and standards (e.g., Soap, BizTalk, RosettaNet, cXML, eCO, WSDL, UDDI) are providing services to develop interactions in business applications but do not provide advance semantics. Semantic web satisfies need of providing advance semantics to describe services and structure of enterprise as well as data. Semantic web is the essential component of building advance web [BHL01]. Agent architecture has been adopted to build highly fault-tolerant and resilient structure on the platform provided by semantic web. They provide human-independent environment for dynamic and critical applications. Business applications have highly adapted these new technologies. People from bioinformatics are also using web ontology language and semantic web to build tools for genomic research and collaborate their work. Certain health providers have built network to integrate their work and provide advance services using Semantic web. Research is been carried out to make use of Semantic web at Stanford medical, at Semantic web research group of Maryland, HP Labs, MIT and many more places. A complete list can be found at [SwWG]. Semantic Web and web ontology language are becoming de facto standards to implement web services.

1.3. Motivation

Data integration is one of the most complex & challenging task not only for health enterprises but also many other kind of enterprises. Sharing data between health partners is essential task for providing efficient services to patients. Sharing clinical data is also required for health surveillance and outbreak detection. Web has become one of the most convenient ways of exchanging data and finding services. But current web is not efficient for doing so.

The second challenging task is new technology web demands not only efficient search technique but also services should be automatically composed & executed. Web should be made machine-interpretable for composing services automatically.

Health Enterprises are the rich resources of services. Global view of enterprise is required for all the health enterprises to work cordially [HuS92]. Heterogeneity exists at several levels of enterprise. At the very basic level the machines used might be different. The operating system, data processing method, data-storing format, data sharing and encoding method vary. Applications program using such data will be incompatible with applications used in another domain. Additionally expert system, knowledge base and information repository of each resource vary substantially from other resources. The resolution to this heterogeneity is, there should be integrated (semantic) way of exchanging data and services.

Matchmaking facilities implemented to-date is not considering the meaning or the overall structure of concept while performing match. Hence the result does not include concepts which are similar in type and behavior. An algorithm needs to be reformed which also considers semantic meaning while performing matching.

Large amount of money has been invested by US Government for developing and modernizing the health strategies. But it was found that money has not been properly utilized due to lack of interoperation. Gradual adaptation of the use of EMR and other technology options is hindering the progress of health enterprises. Interoperation problem is becoming more and more acute as many health enterprises have started using EMR and other electronic methods to store and share data. European and Canadian health partners are miles ahead due to their advance health network. There are so many different approaches carried in different states of US. Merging of these approaches will save millions of dollars and also help to provide efficient health services.

Advantages of Open Source systems are increasing. Moreover due to combine effort they will be well supported. There is a need of developing Open terminological system in health domain to take advantage of openness.

1.4. Contributions

The research performed here provides following contributions:

1. Investigated different needs of health providers and methods to provide advance technology to improve health services.
2. Provided list of problems that resists interoperation between health partners.
3. Proposed the network architecture to provide advance health services to health partners that is well scalable and easy to maintain.
4. Found methods and tools suitable for developing semantic services.
5. Developed ontologies using web ontology language based upon LOINC and UMLS (medical vocabulary).
6. Investigated various syntactic and semantic filtering techniques and tested its suitability for semantic matching process.
7. Implemented inference-based matchmaking algorithm using selected filtering techniques that performs concept-matching. Applied match-making algorithm to develop semantic services in MMN.
8. Analyzed the working of matchmaking algorithm by providing examples.
9. An abstract idea is proposed on basis of network architecture and matchmaking algorithm to develop Open Terminological System.

1.5. Organization

This thesis covers numerous aspects of semantic integration process and is divided into two main parts which is again subdivided into 13 chapters. Part I covers the upper level details relating to MMN, whereas Part II provides lower level information about matchmaking process. . Part I consists of chapters from 2 to 8 and remaining are under Part II.

The remainder of this thesis is structured as follows: In chapter 2 we have provided related work by other groups; in chapter 3, details about ontology development cycle and ontology language is discussed; in chapter 4 information about problems faced by health partners and solutions provided by MMN can be found; chapter 5 shows multi-agent architecture of MMN; in chapter 6 a discussion about service composition and integration process is provided; chapter 7 provides details about architecture of MMN and its components; details about global ontology development is provided in chapter 8.

In chapter 9 we have shown various filtering techniques and implementation of inference based matching algorithm; chapter 10 details algorithm analysis using two different examples; in chapter 11 introduction about tools used for development is provided whereas chapter 12 provides a general discussion on Open terminological System and chapter 13 provides briefing about my work and possible future extensions.

PART I

Semantic Services
&
Metropolitan Medical Network

CHAPTER 2. RELATED WORK

2.1. InfoSleuth

InfoSleuth (Fowler et al., 1999) is a multi-agent system for semantic inter-operability in heterogeneous data sources. Agents are used for query and instance transformations between data schemas. An agent is aware of its own ontology and the mapping between that ontology and the data schema, it is aware of the shared ontologies and it can map its ontology to those of other agents. InfoSleuth uses several shared ontologies, made available through the ontology agents. Individual data sources have (through the resource agents) a mapping to these shared ontologies. The shared ontologies are linked together through one-to-one ontology mapping. Note that the user agents use the shared ontologies as their vocabulary and local ontologies are only maintained by the resource agents.

2.2. ONION

ONION (Mitra and Wiederhold, 2001) takes a centralized, hierarchical approach to ontology mapping, where the user views the (global) articulation ontologies. The source ontologies are mapped to each other via articulation ontologies that are in turn used by the user to express queries. The articulation ontologies are organized in a tree structure. An articulation ontology used for the mapping of two source ontologies can in turn be one of the sources for articulation ontology. The creation of a hierarchy can be seen as a form of ontology clustering. But while (Visser and Tamma, 1999) take a top-down approach (first the root application ontology is specified, then child ontologies are created as is necessary), ONION takes a bottom-up approach in the creation of the articulation ontologies; furthermore, there is no defined root ontology for the cluster.

2.3. COG

In the Corporate Ontology Grid [COG] project the aim is to overcome the problems in semantic heterogeneity between data sources in by semantic integration of the sources using a central Information Model (i.e. ontology). Information Model is built using existing applications, data sources (assets) and input from domain experts. A mapping is then created between each data asset and the central model, thereby assigning a well-understood meaning to the concepts in each asset. With the use of Information Model, the location of information can be discovered throughout the data sources in the enterprise. Furthermore, because the mappings are created in a formal way, the transformations are automatically generated between different sources.

2.4. LARKS

The LARKS [SWK$^+$02] is the language for agent advertisements and requests, and present a flexible and efficient matchmaking process. LARKS uses Multi-Agent infrastructure to provide services of advertising and searching among heterogeneous cyberspace. The Larks matchmaking process performs both syntactic and semantic matching, and in addition allows the specification of concepts (local ontologies) via ITL, a concept language. The matching process uses five different filters: context matching, profile comparison, similarity matching, signature matching and constraint matching. The Global ontology is dynamically built from computed subsumption relations between the concepts included in any advertisement.

CHAPTER 3. ONTOLOGY DEVELOPMENT

3.1. Introduction

Various knowledge based systems are usually built to store the domain knowledge. In certain environment, knowledge is accumulated from several sources. The task to manage and manipulate large KBS is becoming more complex as they are increasing in size. The structure and schema of data collected from disparate sources looses the overall structure and meaning, making it incomprehensible to perform searching. Mechanism has to be developed to store this data in well structured format along with the relations that exists between such data in form of rules.

Terminological systems are widely used to store information where data is accumulated from varied resources. A terminology is collection of terms with relations between them. The most traditional relation between terms is *is-a* relation. Ontology is interchangeably used in place of terminology in field of information retrieval and considered most important tool in artificial intelligence. Ontology is the answers to the above mentioned problems. The well known definition of Ontology provided by [NFF$^+$91] is provided below:

"*Ontology defines the basic terms and relations comprising the vocabulary of a topic area, as well as the rules for combining terms and relations to define extensions to the vocabulary*".

Another definition provided by [SPK$^+$97] is "*Ontology is a hierarchically structured set of terms for describing a domain that can be used as a skeletal foundation for a knowledge base*". We can say from these definitions that in simple language, ontology is a description of the concepts and relationship that can exist between these concepts. Thus it is a model of some portion of the world. Ontologies of same domain can vary depending on its structure and implementation [Den02].

3.2. Development Phase

Often it is confusing for knowledge base architect to decide the approach for developing ontology. There is a nice tutorial [NoM] which provides guidance to understand and develop the first ontology. The basic steps to develop ontology are as follows:

1. Collect Domain Knowledge: Collect information from various resources about concept terms that describe various entities in the domain. Also consistently note the relations between these concepts.
2. Arrange concept terms: Identify the concrete domain terms and their attributes that relates one concept to another. Organize these terms to form consistent overall structure. Create abstract concepts and provide reification that is needed to provide clear definition of domain knowledge. Also link instances to their respective classes. This step is to provide organization of ontology.
3. Provide structural definitions: Once you have hierarchical class structure; create classes, properties and relations as needed. Add constraints to these properties that will link one class to another.
4. Check Inconsistencies: Perform syntactic and semantic consistency check. Check can also be made on classes related by subsumption relation. Perform coherency check on the concepts.
5. Instantiating: Finally create individuals for classes. Instances are the first class objects which are used practically to express domain knowledge. Perform consistency check on instances.

The above steps are not strictly obeyed during development of ontology and are intermingled as per the ontology designer.

3.2.1. Language Selection

In past description logic language like KIF [GeF92] has been used to represent knowledge. It has high expressiveness and is able to represent objects such as symbols, numbers, lists, etc. Also it is able to express relations and functions of variable arity. KQML [FWW^{+}93] is a communication language used for building agents. Other effort in this direction is DGQL [Rey01] which was used to represent knowledge in form of triples. It used Resource Description framework (RDF) [LaS99] as the language to build up the knowledge structure.

The basic requirement of language used to describe concepts and relation between concepts is: It should be expressive enough to express all of the relations and constraints existing between this concepts. Important requirements of the language used for building a knowledge base or ontology are:

- Expressiveness: It should be able to express not only concepts but also meaning related with that concept. The concepts are not seen individually but all together form the complete hierarchy of concepts and objects. This structure reveals the domain knowledge. The expressiveness of language is very important to show how one concept is related to another concept in the concept hierarchy.
- Subsumption: The subsumption relation is important to reveal the hierarchy existing amongst the concepts. This hierarchy provides information about where the concept is lying and which are the parent and child concepts. Also it tells about the equivalent concepts. Thus the language should be able to reveal subsumption relations existing between concepts and its properties.
- Completeness: The language should be complete; it means it should be able to specify all the restrictions or constraints applicable on the concept. Axiom relating to concepts should be well specified by such language. Language is considered complete when it includes above two qualities and also be able to express other relations such as transitivity, disjoint, functional property and inverse relations. The other important part of

completeness is its ability to demonstrate logical assertion and logical inference.

Daml+Oil is the ontology development language which has integrated ontology inference capability. It was the most suitable candidate for building the knowledge base. As it supports many of the requirements specified above. Daml+Oil lacks full expressiveness which is necessary in some of the inference environment. The more complete language existing at present is OWL [McH03]. OWL is the frame based language which supports expressiveness to support logical inference [HPH03]. OWL is based upon DAML+OIL but it is more complete and expressive.

3.3. OWL: Web Ontology Language

There are three different flavors of OWL depending on the expressiveness and completeness [McH03]. OWL is top on the stack built using XML, XML-Schema, RDF [LaS99], RDF Schema, DAML+OIL. OWL provides constructs to add more vocabulary for describing properties, relation between classes (e.g. disjointness), cardinality (e.g. minCardinality, maxCardinality, exactly one) richer typing of properties, characteristics of properties (e.g. transitive, symmetric) and enumerated classes.

Varieties of OWL language:

1. OWL Lite: It fulfills primary need by providing minimum constructs for classification hierarchy and simple constraints.
2. OWL DL: OWL Description Logic provides all the constructs provided by OWL Lite. Additionally it provides maximum expressiveness, completeness and decidability.
3. OWL FULL: This language provides full expressiveness and the syntactic freedom of RDF. It is hardly possible for RACER [RAC] to support complete reasoning.

We will be interested in using OWL DL for our work as it is able to provide complete constructs needed to build ontology and expressiveness needed to provide inference capabilities.

Let us see what type of relations can be expressed by using OWL. If we want to say that, “A Mother is a person having gender as Female and with at-least one child”. This can be expressed in OWL using following syntax:

```
<owl:Class rdf:ID="Mother">
<owl:equivalentClass>
  <owl:Class>
    <owl:intersectionOf rdf:parseType="Collection">
         <owl:Restriction>
            <owl:onProperty>
              <owl:FunctionalProperty rdf:about="#hasSex"/>
            </owl:onProperty>
            <owl:hasValue rdf:resource="#Female"/>
         </owl:Restriction>
         <owl:Restriction>
           <owl:minCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
            >1</owl:minCardinality>
            <owl:onProperty>
              <owl:ObjectProperty rdf:ID="hasChild"/>
           </owl:onProperty>
         </owl:Restriction>
       </owl:intersectionOf>
     </owl:Class>
   </owl:equivalentClass>
 </owl:Class>
```

The OWL syntax adopts XML syntax format as base. The OWL Guide [MWM04] provides guidance about use of OWL with more examples. The OWL Semantics and Abstract Syntax provides all the advance information about OWL syntax [PHH04] for developing ontology. It can be shown that OWL provides construct to express each of the relation and specification listed in Knowledge Representation System Specification [PsS93]. A complete table showing one to one mapping between OWL Construct and Description Logic Axiom is provided in Table B.1 of Appendix A.

CHAPTER 4. MMN SERVICES

4.1. What is MMN?

MMN is a network between health partners, hospitals, clinical laboratories and medical researchers. This network can be compared to a market place, full of different service providers, and a new service provider can be integrated with other service providers with little effort. Authorized entity can use services and also provide services to others within network. The primary goal of forming MMN is to provide real time sharing of clinical and syndromic data amongst medical laboratories and other health entities. Seamless integration of data and services is possible by semantic mapping of respective domain knowledge. MMN provides facility of automatic service composition that will help health partners to fulfill higher-level goals such as health surveillance and disease outbreak detection.

There are multiple uses of MMN; primary to this is patient record collection and data sharing. Health provider specializing in one category of service may not be able to efficiently provide health services of other categories or specialization. This is due to lack of specialty care doctors for certain treatments and diseases. Appropriate advance facilities, laboratories and machinery for treating patients are not available or remotely shared. Location also plays a major role; certain uptown country places are devoid of health facilities. In such conditions they have to rely upon other health partners in health provider network. It is very helpful, if there is a semantic network of health service provider, which forms the services automatically and provides all the required services.

4.2. Problems & Solution

Health providers use their own legacy software provided by their preferred vendors. Different applications uses different vocabularies like DICOM [DIC04] is used by imaging center and LOINC [MHS+04] codes are used for laboratory findings. SNOMED Clinical Core terminology [SNO] provides a common language that enables a consistent way of capturing, sharing and aggregating health data such as electronic medical records, ICU monitoring, clinical decision support, medical research studies, clinical trials, computerized physician order entry, disease surveillance, image indexing and consumer health information services. HL7 [BHR+99] is the major effort to provided unified message format. MeSH [MSH] is used as medical meta-thesaurus. UMLS [UKS] is another major effort from NLM [NLM] that provides platform for medical data representation. ICD [ICD] families of terminologies are also widely used as medical terminology dictionary for various diseases. The above mentioned terminologies refer to their own corresponding underlying domain terms. There are so many standards or efforts that confusion is created while sharing data amongst medical enterprises. Thus, the care system cannot fully "understand" and properly file the results they receive unless they either adopt the producer's laboratory codes (which is impossible if they receive results from multiple sources), or invest in the work to map each result producer's code system to their internal code system [RHI].

Health partners share unique data, which cannot be replicated. These intricacies make it more difficult to share resources (Patient records, images, etc) or use services from other health providers. Issues like interoperability, scalability and heterogeneity are the major issues in any healthcare enterprise. Health care departments are distributed in terms of service administration and location. This makes more difficult to provide quick services.

Most of traditional health systems are found to have facility of transferring data through overnight batch process. Data in such system are analyzed after delay of several hours, which makes surveillance task less efficient. The real time data sharing capability shares disease data of the patients to the research institutes in real time and thus encourages research activities. Our idea is to integrate this facility in MMN through automated procedures for exchanging valuable information and services (e.g. clinical information).

There are countless efforts going on to bring technology in Medical field. One of the campaigns working in this direction is to use EMR. It was found that interoperability between EMR failed because of heterogeneous formats of data. Our effort will provide this semantic interoperability between EMR.

These issues can be solved by forming semantic homogenous network. Integration of services is done by providing common medium to interchange information. MMN implements this medium based on Semantic Web [BHL01]. Our effort is to provide method for interoperation between them and not to solve the confusion created by using many terminological systems together. HL7 is the de facto standard for message implementation and communication between medical partners. Software agents cannot process HL7 messages directly. For automatic service composition the message should be machine interpretable and so this message has been further mapped to OWL-S [MBH+04] constructs. Web ontology language (OWL) is used to make these messages interpretable by agents. Terms from one terminology are mapped to terms in global knowledge base. The method to perform this mapping is given in matchmaking section (9.4) of this thesis.

A method using multi agent network is proposed here to achieve the required goals. Common view of enterprises is very important and can be achieved through this network. We build ontology for each of the local information model which needs to be integrated and a “*global ontology*” [HuS92] from existing local ontologies.

CHAPTER 5. MMN MULTI-AGENTS

A multi-agent architecture is implemented to provide semantic services in MMN. Agents [Hen01] are the software entities (programs) which are semi-autonomous, pro-active and adaptive. They are intelligent programs that assist humans in several operations. We have defined three different types of agents Figure 5.1 depending on the work they perform. They are User agent, Mediator agent and Service agent.

5.1. User Agent

The software agent which performs which assists services users to perform several operations is known as User Agent. They perform operation such as building semantic query according to the specification of service seeker or user. A user agent hides the lower level details of agent communication and architecture from the users. They provide an easy-to-use platform or interface through which user can access advance web (health) services provided by MMN. User agent communicates with mediator agent to perform search for required services or communicates with service agent [with the help of mediator agent] to use service provided by them.

5.2. Mediator Agent

The middle layer application works cordially with mediator agent. The primary role of mediator agent is to provide platform using which two heterogeneous agents can understand each other and exchange services. When a query requiring some service comes from user agent, the mediator agent performs matching with the service profile of various service agent stored in service registry. Mediator agents then returns best matching results and other information related to service to the user agent.

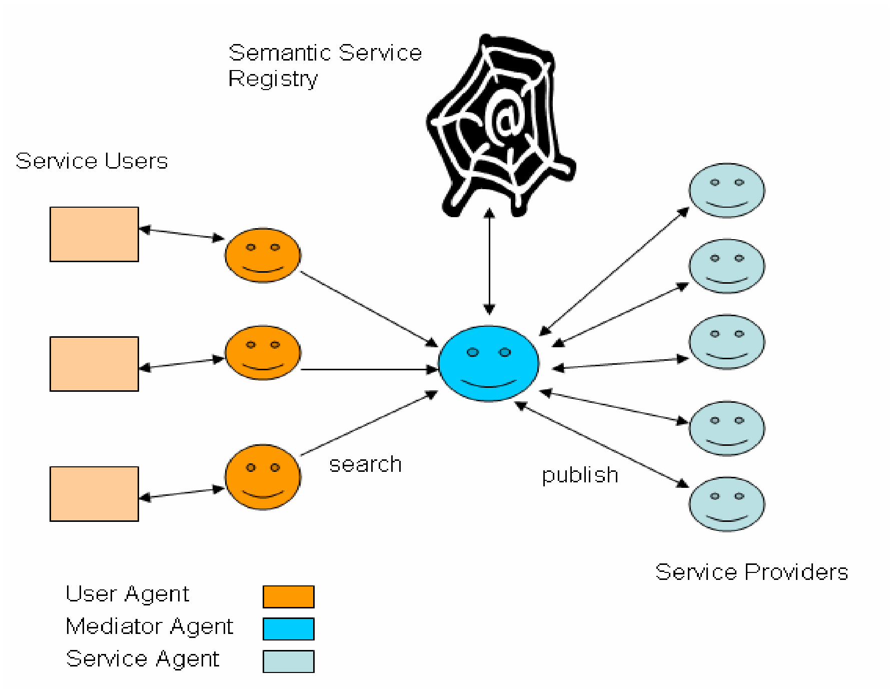


Figure 5.1 Multi-Agent Architecture

5.3. Service Agent

The agent which works for service provider is known as Service Agent. This agent registers the advertisement of service provider in the semantic service registry of middle layer. For this, service agent has to communicate with the mediator agent and provide its service profile. Many times one service provider will need to use services provided by other service provider. In such cases service agent of service user becomes user agent. Thus agents can change roles as per the need. Service can also be of interactive type in which service agent has to communicate with user agent through mediator agent through out the service execution process.

5.4. Agent role in MMN

MMN provides a pool of service providers. This type of network is very advantageous for health providers, because it not only provides faster method to search a service, but also provide service accurately matching to service query description. For e.g. Primary care physicians treating patient showing symptoms of Parkinson disease may require nuclear imaging scan service. The other requirement of the service-seeker is nuclear imaging center has to be nearer to the patient's residence. Service provider should be able to accept insurance from "General American Life". The user agent of the physician will automatically form the service query with appropriate service parameters or specification and using the vocabulary terms used by the physician. The service execution engine will then execute the service on behalf of physician and in return receives the list of service providers fulfilling the requirements. The list of service providers returned will be a near or exact match to the requirements of the physician. More about automatic service composition, semantic integration and agents is discussed in CHAPTER 6.

CHAPTER 6. SERVICE COMPOSITION & INTEGRATION

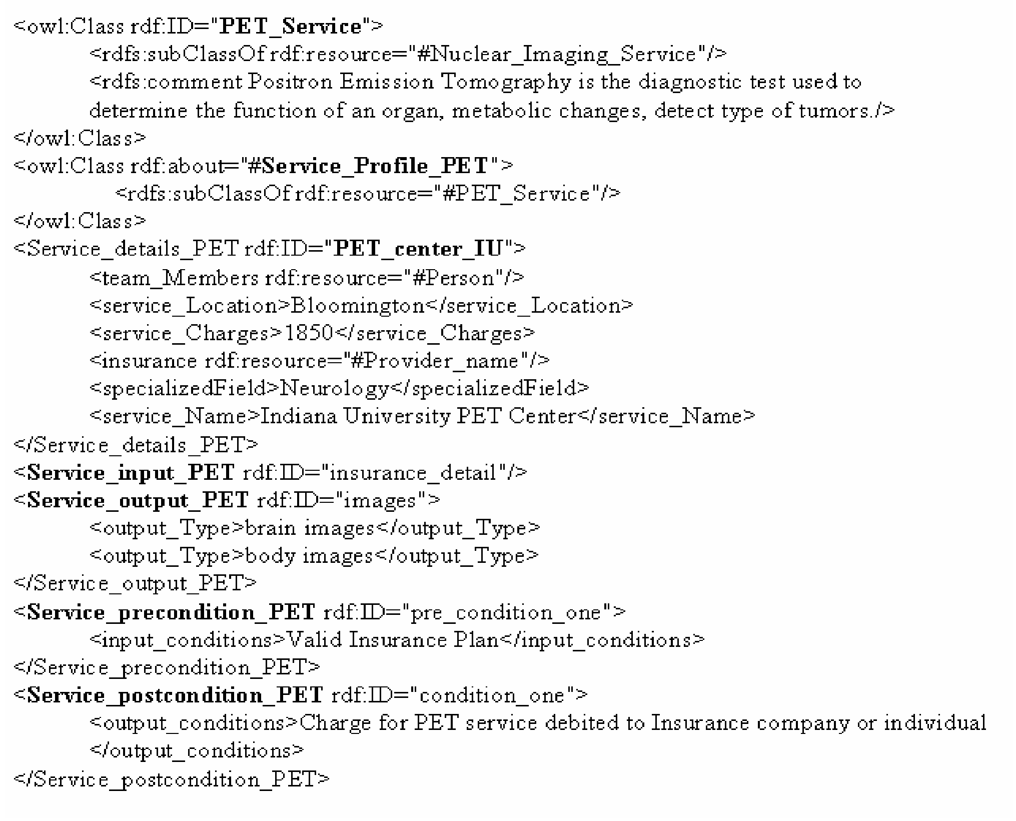
According to the workflow [WRM] model, services are created and placed on the web page as per the need and with appropriate specification. This model describes procedural steps, required input, output information and tools needed for each step in the integration process. In this section, method for service composition, agent formation and semantic matching is more deeply discussed.

6.1. Service Composition

Service formation is strictly procedural process. Description about the needed service has to be provided in the initial step. There are many ontology oriented efforts to provide service description. Some other initiatives like UDDI [Bou00], ebXML [ebX] have failed to attain the goal of providing service description [TBGc01]. The description terms used along with service profile to advertise about the capabilities of services is also known as “Service parameter”. Service parameter also includes the parameters needed as input to satisfy requirements of service provider. Moreover service parameter includes output parameters that results due to execution of service. Service description very much depends on type of service offered or required. In this thesis we focus on medical domain services. Service profile, service model and service grounding are the three type of information required for the formation of any type of service [OSC03].

6.1.1. Service profile

Service profile answers question like what does service provides. It also contains information about the requirements of service-seeking agent. Service profile contains definition of properties such as name of the service, contact information, quality of the service, and additional information that may help to evaluate the service [OSC03].



```
<owl:Class rdf:ID="PET_Service">
        <rdfs:subClassOf rdf:resource="#Nuclear_Imaging_Service"/>
        <rdfs:comment Positron Emission Tomography is the diagnostic test used to
        determine the function of an organ, metabolic changes, detect type of tumors./>
</owl:Class>
<owl:Class rdf:about="#Service_Profile_PET">
          <rdfs:subClassOf rdf:resource="#PET_Service"/>
</owl:Class>
<Service_details_PET rdf:ID="PET_center_IU">
        <team_Members rdf:resource="#Person"/>
        <service_Location>Bloomington</service_Location>
        <service_Charges>1850</service_Charges>
        <insurance rdf:resource="#Provider_name"/>
        <specializedField>Neurology</specializedField>
        <service_Name>Indiana University PET Center</service_Name>
</Service_details_PET>
<Service_input_PET rdf:ID="insurance_detail"/>
<Service_output_PET rdf:ID="images">
        <output_Type>brain images</output_Type>
        <output_Type>body images</output_Type>
</Service_output_PET>
<Service_precondition_PET rdf:ID="pre_condition_one">
        <input_conditions>Valid Insurance Plan</input_conditions>
</Service_precondition_PET>
<Service_postcondition_PET rdf:ID="condition_one">
        <output_conditions>Charge for PET service debited to Insurance company or individual
        </output_conditions>
</Service_postcondition_PET>
```

Figure 6.1 Service Profile for Nuclear Imaging Center

The main functional properties of service profile are input, output, precondition and effects that help with the specification of what the service provides. Compositional knowledge such as syntactic and pragmatic knowledge [LCG04] gives the logical meaning to service profile and also governs service formation. Syntactic knowledge consists of various inputs and outputs that are used in service composition. Various service parameters are required as an input for the execution of a service. When this service executes the outcome will be in form of output parameters. This type of knowledge is required by the agents for forming the service. Pragmatic knowledge deals with extra information that guides service formation process. It includes rules governing the use of service. Such rules incorporate restriction on condition in which this type of service can be used e.g. when a new service is provided by a health partner, they also include information about how to access that service (Registering resource or taking appointment with physician) and the restrictions (time frame allocated) in using that service. Any service-seeking agent interested in using this service has to abide it with the rules of the service-providing agent.

In Figure 6.1 we have shown Service Profile ontology (explained in more details in later section) for Nuclear Imaging service provided at “PET (Positron Emission Tomography) Service Center” of Indiana University, Bloomington. This ontology is composed using OWL-DL (details in later section). The first class tag is top-level class and stands for PET service. Service_Profile_PET class is the second level class. Service profile has sub-class as: “Service_details_PET”, “Service_input_PET”, “Service_output_PET”, “Service_precondition_PET” and Service_postcondition_PET. Service detail class holds several properties such as service_name, location, service charge, accepted insurance provider, field specializing in and team_members. Service_input class provides list of parameters required to form the service (insurance detail). Service output class shows what will be the feedback or service provided (PET images). Precondition class notes condition necessary for formation of service (Valid insurance plan) whereas post-condition class lists conditions after execution of service (Bill or charge). We have taken only single instance for each of them, there can be much more detail information which we have eliminated to keep example easy to understand.

6.1.2. Service model

Service model contains information about the process of execution of service. It tells about the consequences that arise due to service execution. The information gained from this type of knowledge is useful to the service-seeking agent for performing various analysis tasks or to get the feedback on the current status of service execution. Service model consist of domain specific semantic knowledge that helps in service execution.

6.1.3. Service grounding

Service grounding discusses low-level details such as how a service-seeking agent can use or access the service. This knowledge discusses about message formats, low-level service specific details and communication protocol, etc. Grounding performs the mapping from abstract service parameter to concrete service terms. This mapping has to be done quickly and thus need efficient semantic mapping method.

6.2. Agent formation

Using the service parameter appropriate query is build by the user software agents. Agent plays a major role in service composition, service discovery and automatic service execution. As describe above agent interacts with the other agents of service providers in the network. During these interaction agents performs semantic matching between terms in domain of service provider and in domain of service receiver with the help of common terminological system or by using matchmaking algorithm. In our case we have a special layer called 'ICIS' (section 7.3.1) which performs this operation. Service execution engine decides upon which results is to be called perfectly matching or partly matching service agents.

Let us see one more example to learn about agent's role in providing service. Patient record retrieval system is the service provided by central authoring system of MMN. When a primary care physician requires access to any of his patient record, then a software agent is automatically formed to provide this service. Agent uses input parameters such as patient ID and information like type of service (i.e. patient retrieval service) desired to compose the query. The service agent of patient retrieval system will process query parameters provided by physician’s user agent. Once service-providing agent verifies the input parameters, the execution of service will start. As a result of this service execution, physician’s user agent will return with the output parameters to the physician that will be in form of patient’s record. This is very basic example for simple services that can be provided by agents.

6.3. Semantic Matching

The two main concept of Semantic Web [BHL01] is service selection and semantic matching. Service selection is itself a challenging and complex task [ShS04]. In this thesis I am concentrating on the semantic matching process that is at lower level then service selection process. Semantic matching is also one integrated task in the process of service selection. Thus Semantic matching can be considered the most important process in Semantic Web. Formal task of semantic matching process is to map service parameters to existing local medical ontology and vocabularies. Ontology models the medical enterprise and provides conceptual information about that domain. Matchmaking process then performs mapping between this ontologies to provide interoperation service.

The terms from local ontology of service-seeking agent are mapped to terms in global ontology and finally global ontology is mapped to local ontology of service provider. Concept matching is inevitable process for any semantic integration. Terms (vocabulary) used to describe same underlying concept (meaning) may be different for two different domains. Interoperability between heterogeneous applications highly depends on performance of matchmaking algorithm. In MMN global ontology is stored in a layer known as Semantic Integration Layer or ICIS.

Traditional method performs this matching by comparing the terms syntactically. This is not efficient as well consumes large amount of time. Thus if there are 1000 service providing agent then matching their service profile with the query will consume more then few seconds. By performing semantic matching this process is made faster and more efficient. The matchmaking algorithm implemented in this thesis is based on inference-based matching (section 9.4).

Figure 6.2 shows two different approaches for forming connections between health partners. The vocabulary, medical reference system or terminologies used by each enterprise may vary. In one of the approach integration server is providing the common platform to communicate. Advantages achieved by this architecture are very obvious. To understand this let us consider that health network initially has ‘N’ number of health partners. One way of achieving interoperability between each application is to provide mapping from one application domain directly to another application domain. The user of the system needs to have knowledge of terms used by other service provider to form the appropriate query for that service provider. Thus total number of wrappers needed in this case is ($N \times N$). Also adding or modifying an application domain needs addition or modification of ‘N’ wrappers in the system. Lot of work indeed!!

Another approach is using local and global ontology. Now in the system with ‘N’ application and a global ontology, there are maximum ($N \times 1$) mappings in the network. Adding a new application or modifying an application needs adding or modifying only one connection (instead of ‘n’ wrappers in previous case). It is very obvious that use of global ontology will provide a highly scalable system. The advantage of this approach is it provides flexibility while adding or removing new enterprise domain and thus provides resilient network. This architecture along with proposed matchmaking algorithm can be together used to provide the platform to develop Open Terminological System (discussed in section 12.2.3).

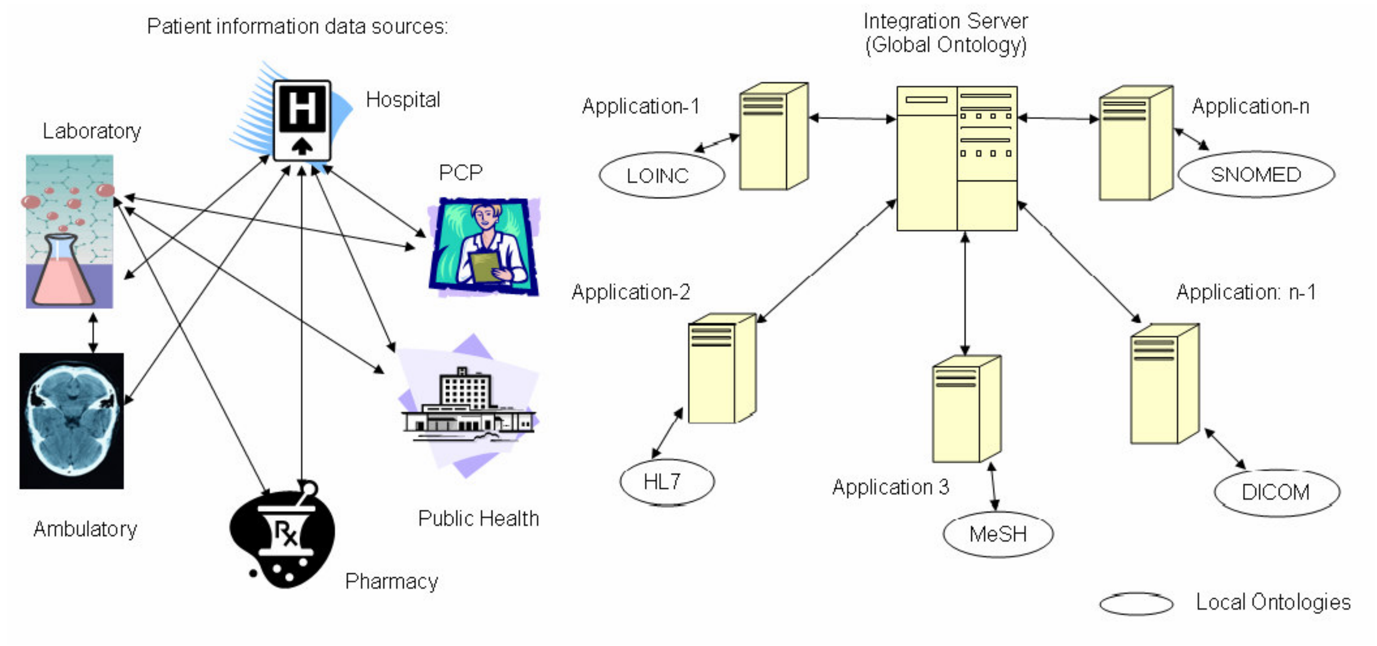


Figure 6.2 N×N wrappers vs. N mappings of MMN

CHAPTER 7. WORKING & ARCHITECTURE OF MMN

7.1. MMN Architecture

Based upon architecture proposed using global ontology the simple view of Metropolitan Medical Network is provided in Figure 7.1. It is made up of health partners like Indiana University Medical Group (IUMG), clinical research centers (e.g. Regenstrief Health Services), family physicians or primary care physicians (PCP) and other health providers. MMN provides the platform for sharing services, facilities and interaction between different health partners. Each hospital may specialize in specific service and thus patients could benefit by such network. Moreover Clinical research and laboratories also benefits by obtaining health related information from large number of hospitals.

7.2. Service Flow

In this section we will like to discuss about flow of service from patient to family physicians and to all MMN system elements, consider Figure 7.2. Taking example of a patient suffering from Parkinson disease, he first visits family physician nearest to his residence. Family physician in general case will perform basic check-up and provides basic pathology services such as blood/urine tests. Physician can also refer to advanced pathology services provided by any other health provider within the network.

If physician needs other special services then he can inform his user agent to find suitable service. This user agent will then automatically form search query and send it to mediator agent. Mediator agent performs semantic matching between advertisements stored in semantic service registry by health service providers. Outcome of this search will be list of service providers whose service profile matches service query. Physician can even specify user agent to search as well as register the most suitable service automatically. In this case agent performs negotiation with other agents and registers required resources and services.

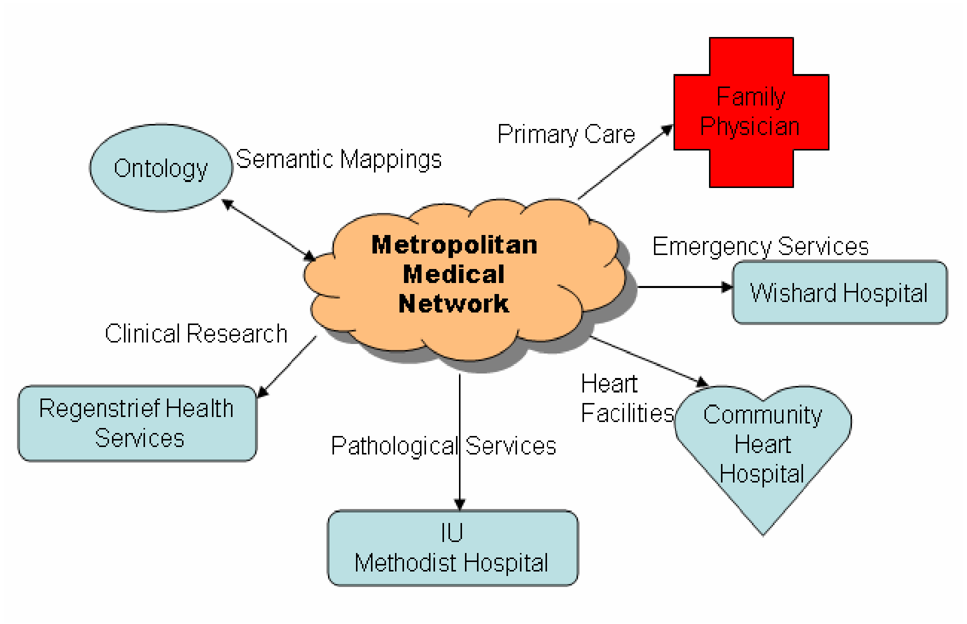


Figure 7.1 Metropolitan Medical Network

The flow of services is shown by arrows. Mediator agent discovers that IU-Methodist is providing advance pathological facilities nearby to patient's residence and fulfills all the given requirements. Here rating based matching strategy [ShS04] can be incorporated to break tie between service providers. Reports of the test performed by IU-Methodist are made available to family physician through web services provided by MMN.

PCP may see the need of advance diagnosis of the patient. He then recommends further treatment under specialty care physician such as a neurologist. User agent performs the search and finds that specialty care center specializing in neurosurgery located in Riley hospital fulfills all the requirements of the patient. This center provides several DICOM based services like CT scan, nuclear imaging, etc. necessary for taking images of brain. The results of these images may reveal need of advance service such as neurosurgery. User agent once again recommends “Neurology and Surgery Center” in the same hospital. Family physician can access any CT scan report through Image Retrieval System. PCP can track all the advance treatments his patient is diagnosed with using MMN. This information will help him in present as well as future diagnosis of same patient as well as other patient showing similar symptoms.

A similar project was done at UMKC known as SMS [LCG04] which provides similar architecture but different matchmaking algorithm. We found that their matchmaking algorithm is not efficient as they are not performing matching of meaning achieved through relations between concepts and property constraints, which are the most important for performing semantic matching.

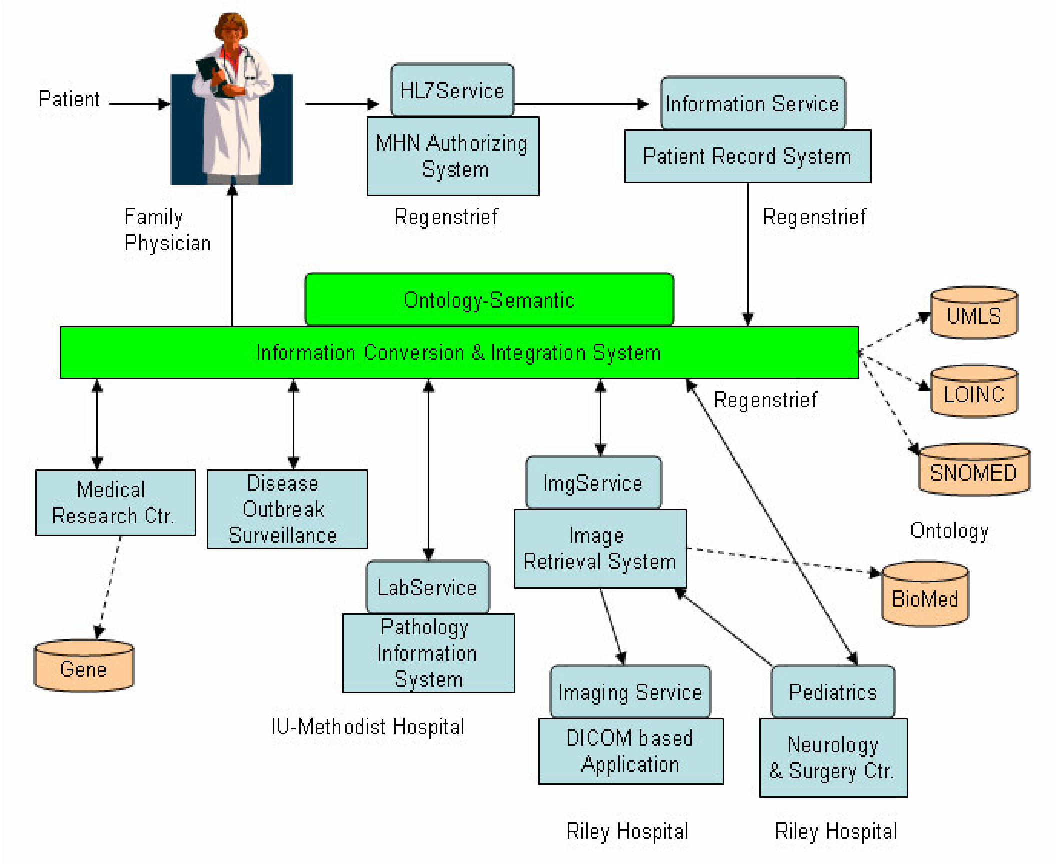


Figure 7.2 Service Flow between Health Providers

7.3. Components of MMN

Some of the important components of MMN are shown in Figure 7.2. Patient record system is the central database system, which keeps patient identity information. For using all this services person should be authorized by MHN Authorizing System. Once authorized, physician can use any semantic services provided by central Information Conversion & Integration System (ICIS), patient record system and services from any other provider in the network.

A detailed explanation is necessary for understanding implementation of this component. In this thesis our main concern is providing information about ICIS implementation and working (section 7.3.1). Some of the facilities integrated with ICIS are Medical Research Center, Lab Services, Imaging center, etc.

Different Hospitals and other sources can transfer data in real-time to Medical research center. Data from various sources can be feeded to Medical research center by mapping each of them to Global ontology. Lab Services are integrated in this network and others can use these services using ICIS. Disease Outbreak Surveillance can get results from laboratories, medical research centers and other public health partners. All of this heterogeneous information can now be well understood by the institutes with the help of ICIS. The components shown here are provided for example and not the complete list of services that can be integrated in MMN.

7.3.1. Information Conversion and Integration System

ICIS is the most important component of MMN as it provides interoperation between heterogeneous applications and data sources. It provides this by building global ontology from existing local domain ontologies. The efficiency of integration highly depends on the accuracy in formation of global ontology and mapping algorithm. When a query from domain ‘A’ wants to send message to domain ‘B’, the message in form of query first goes to ICIS, it maps terms from this query to terms in global ontology ‘G’. If there are no matching term available in global ontology for domain ‘A’ then matchmaking algorithm is applied on these terms. During matchmaking, conceptually matching term from domain ontology ‘B’ is found for term in query. This mapping is then added to global ontology. Also converted message is sent to domain ‘B’.

If query from domain ‘A’ is to find a service provider providing certain service ‘S’ then ICIS itself provides the service of a registry. UDDI [Bou00] was designed for providing registry and is implemented using industry standard language XML, but it is incapable of providing higher-level details. ICIS registers new services and stores service profile (advertisement of capabilities) of all such service providers in machine interpretable form using web ontology language. Matchmaking is performed by mediator agents in ICIS when query is available from ‘A’. It matches this query against the service profiles in the registry.

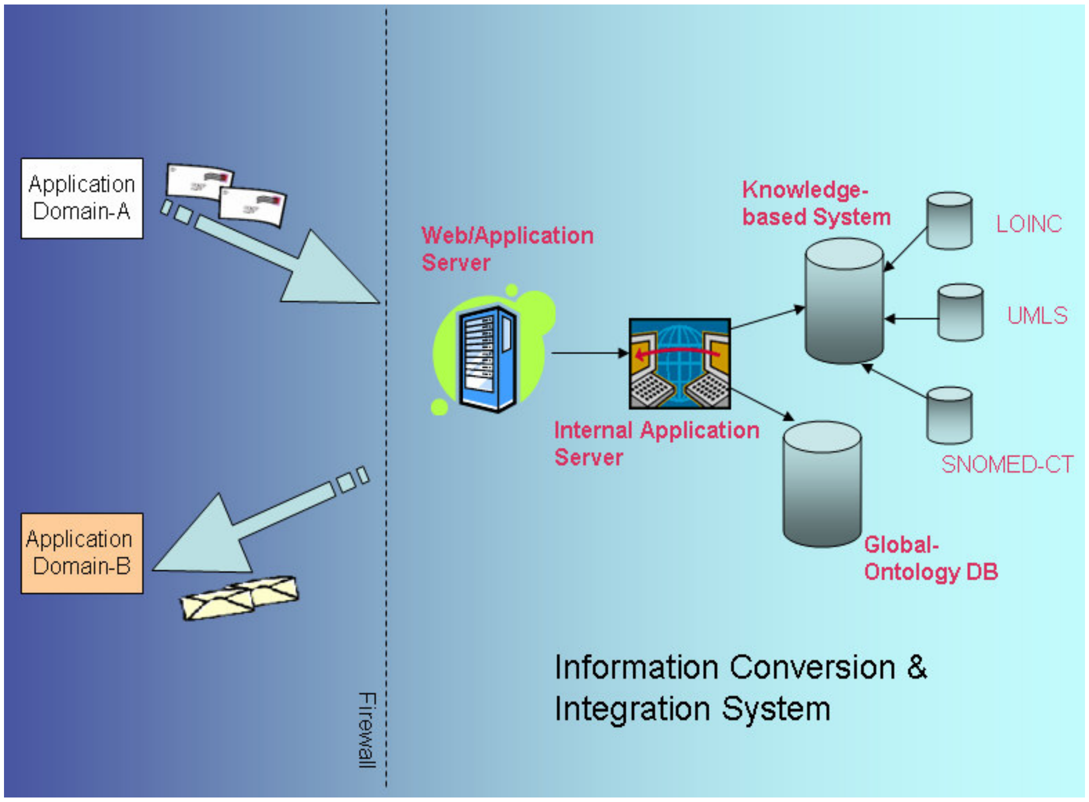


Figure 7.3 Information Conversion & Integration System

Thus all the tasks related to semantic integration are concentrated in one place. It has shortcoming such as ICIS becomes very complex and maintenance thus becomes a complicated process. But this is important task and will provide base for future open terminological medical system.

The flow of information integration is best understood by Figure 7.3. Application Domain ‘A’ wants to communicate with Application Domain ‘B’ and both are using different vocabulary. It can communicate with the help of ICIS which acts as mediator. Let us see how ICIS provides integration and conversion facilities. The data sent to ICIS by Application Domain ‘A’ is wrapped in HL7 message format. ICIS is the group of application server and database system that processes the incoming HL7 messages. Interface to ICIS from the outside world is provided by web/application server. This server provides the ports that are listening for the incoming request for exchanging the data. This server then sends this request to the internal server of ICIS system. Internal application server is running RacerPro [RAC] server, matchmaking algorithm as well as other applications useful for conversion process. Application running at this server will process the incoming request of sending this message to Domain ‘B’. ICIS performs mapping between two concepts (one from Domain A and the second is picked from Domain B), by looking if there is information related to those concepts already available in Global Ontology (Auxiliary DB). If not then this concepts are passed as argument to matchmaking algorithm. Matchmaking algorithm will then perform matching using the filtering techniques explained in section 9.4.1. The results returned by the matchmaking algorithm will be stored in the global ontology and also a copy will be send back to the web/application server. Ultimately web/application server will send the converted message to Application Domain ‘B’.

7.4. Service Selection Process

Service selection is a challenging task and requires multi-step semantic matching process. Several techniques are discovered in past which provides rating to commodities and services [HuS92] which helps in selection process.

Ratings can be provided by the consumer of service or/and by the matchmaking algorithm.

Our matchmaking algorithm provides rating of the service provider from overall score calculated at each step of matching process (E.g. 0 to 5 stars). For a given new query and list of service providers available in registry, mediator agent will perform matching between query and service profile of a service provider. The provider rated highest is declared as the best match and the provider having rating below threshold are rejected. Tie between providers can be further broken by the confidence level of the users in the provider established by past transactions. According to this, service provider is rated based upon the history of services and quality they have provided.

CHAPTER 8. GLOBAL ONTOLOGY

Matchmaking algorithm is time consuming process. In the dynamic environment such as internet search engine, each new query might be for a new word or the word already queried in past. If matchmaking algorithm is applied for each new query regarding of word queried in past then this process will be very time consuming. The process of matchmaking can be made faster if the concepts encountered in the past are stored in an auxiliary database along with the information related to matching. This would help from avoiding redundant execution of matchmaking process in the case where same concepts are queried frequently. Certain enterprise domain is less dynamic, and so frequency of querying new concepts is very less. Even in such cases storing the complete information about matchmaking of concepts in auxiliary database would outperform any other fastest matchmaking algorithm. This auxiliary database could store the concepts along with restrictions and properties associated with it. Auxiliary database storing lexical knowledge about several domains thus can be termed as Global Ontology.

Global ontology is not necessarily the union set of local ontological system, but stores partial domain knowledge from each of the local ontology. For e.g. In medical enterprise domain; terms used for message implementation are referenced from available medical dictionaries. Hence the mapping of two concept or terms necessarily remains the same, unless medical dictionaries are updated frequently. In this case storing mapping results between terms in global ontology will make data mapping process much faster as needed.

8.1. Implementation details

Global ontology development is a gradual process. In initial stage global ontology is empty as there is no concept matching performed. When new concepts are available for mapping; the matchmaking algorithm is applied on them to find the degree of matching. Results of this matchmaking process along with complete information about this concepts is then stored in global ontology. A result of matchmaking is also sent back to the user. Global ontology can be formed by placing these concepts under the root concept and providing a link between them which also stores complete rating information. Each new concept will form new subsumption relation in the existing concept hierarchy. Snapshot of global ontology at certain point in time will consist of concepts along with constraints and attributes acquired from different local domain knowledge bases.

8.2. Ontology for Medical domain

Using OWL we have developed partial ontology for LOINC [MHS$^+$04] database. LOINC database provides set of universal names and ID codes for identifying laboratory and clinical test results. LOINC consists of 34000 terms representing different LOINC codes. Moreover efforts are undergoing to add a field in LOINC message to cross-reference terms in SNOMED-CT [SNO] vocabulary and HL7 [BHR$^+$99] messages. LOINC ontology prepared by us currently has 128 direct classes, and 56 slots (attributes). In the future work more classes and attributes will be added to LOINC.owl file. We have added sample ontology in APPENDIX C.

We also have UMLS [USN] ontology in OWL format. This UMLS ontology was initially developed and maintained by NLM [NLM]. This Ontology which is stored as a terminology in simple text file was later converted to DAML+OIL. Our efforts was to convert DAML+OIL format file to OWL file format, so that we can perform mapping between different ontologies in OWL format, one of which is LOINC.owl. Our idea is to provide mapping initially between few medical terminologies such as LOINC, UMLS, SNOMED-CT and DICOM [DIC04]. This will help to provide interoperation between them and provide platform to develop Open Terminological System (detail discussion is available in section CHAPTER 12).

PART II

Semantic Mapping & Inference-Based Matchmaking Algorithm

CHAPTER 9. INFERENCE BASED MATCHING ALGORITHM

9.1. Background

Matchmaking is an important process for integration of business applications, and in heterogeneous environment where interoperability is the principal concern. This problem is long known and several new techniques are realized to perform efficient integration. [Trastour, D et al] Advance matchmaking requires rich and flexible metadata that are not supported by current available e-commerce standards such as UDDI [Bou00] and ebXML [ebX]. Success of integration of data and application depends on language used to describe services and data. Secondly performance of this process highly depends on matchmaking algorithm.

Matchmaking process can be described as a process through which a semantically similar value or concept is found for the given concept. This can be carried out by matching features of one concept to that of another. In most cases middle layer or mediator provides this service. Software agents based technology is widely adopted for integration of services [NFK+00, NBN99].

Basically matchmaking techniques can be divided as: Syntactic matching and semantic matching. Although there are different levels of matching such as exact, plug-in, subsume, intersection and disjoint, all of them fall under the one of the basic matchmaking type. Most of the search engines use syntactic method based on string distance for finding the match between words along with some semantic matching. Such techniques can be considered to find a match but at the same time performs poorly by missing out complex concept structure which reveals behavior and meaning of the concept. Also the search results missed out those search results where syntactic words are unlike but concepts related to them are alike. If we don't want to miss such results then word context as a whole should also be compared.

In this thesis we propose inference based technique based upon description Logic $\mathcal{AL}$ for performing matchmaking. Our main goal is to achieve interoperation between heterogeneous data existing in medical domain. The complete information of language which supports inference capabilities is discussed in "Ontology Development" section of this thesis.

9.2. Description Logic for Matchmaking

9.2.1. Introduction

Description Logic languages gives power to provide reasoning capability on the structured knowledge. They are considered an important formalism unifying and giving a logical basis to the well known traditions of frame-based systems, semantic networks and semantic data models. DL systems have been used in Information Integration, Query Processing and conceptual modeling.

The building blocks of DL is concepts, roles and individuals {C, R, I}. Concept is the generic entity that ensembles all the entities having similar behavior and attributes. Concepts can be considered as first class objects or unary predicates. Role is the binary relation between concepts. DL provides language constructs such as intersection, union etc which is used to define new role or concept. The main part of DL language is to provide classification, satisfiability and instance checking.

Subsumption relations are widely used in reference systems, search engine and match-making algorithms. A concept C subsumes another concept C' if the extension of C' is a subset of that of C. This means, that the logical constraints defined in the term of the concept C' logically imply those of the more general concept C. Thus subsumption relations specified *is-a* relation existing between concepts. Classification is the process of computing this subsumption relation. Computing the concepts and individual which satisfies certain constraints is known as satisfiability. Instance finding process finds all of the individuals of certain concept.

Selection of DL language is important to achieve full expressiveness and reasoning capability. Every DL language trades off representational expressiveness with computational tractability.

9.3. Description Logic Background

The Description Logic $\mathcal{AL}$ is the most preliminary language that can express relations or concept description formed with DL building blocks:

expr →	C	|	(Atomic Concept)
	⊥	|	(Universal Concept)
	T	|	(Bottom Concept)
	¬	|	(negation)
	⊓	|	(intersection)
	∀	|	(allValues Restriction)
	∃	|	(someValues Restriction)

Table 9.1 Description Logic $\mathcal{AL}$

The name for Description Logic language is given by the feature they provide. The reasoner RacerPro[1] we use for our purpose uses logic $\mathcal{ALCQHI}_{\mathcal{R}+}(\mathcal{D})$ [HST00] which was extended from $\mathcal{ALCNH}_{\mathcal{R}+}$[HaM00]. Let us how $\mathcal{AL}$ is augmented with the constructs used to build $\mathcal{ALCQHI}_{\mathcal{R}+}(\mathcal{D})$:

Symbol	Type of Construct
$\mathcal{C}$	negation of arbitrary concepts
$\mathcal{Q}$	qualifying number restriction
$\mathcal{H}$	role hierarchies
$\mathcal{I}$	inverse roles
$\mathcal{R}+$	transitive roles
$(\mathcal{D})$	restricted form of concrete domains;

Table 9.2 Symbolic Notation for $\mathcal{ALCQHI}_{\mathcal{R}+}(\mathcal{D})$

Restricted form of concrete domains includes facilities for algebraic reasoning including concrete domain such as:

- min/max restrictions over integers
- linear polynomial equations over the real or cardinals with order relations
- non-linear multivariate polynomial equation for complex numbers and
- equalities and inequalities of strings.

$\mathcal{ALCQHI}_{\mathcal{R}+}(\mathcal{D})$ is also termed as SHIQ [HST00] More information about evolution of description Logic, types of DL and its features is available at [HaM01a] [HaM01b] [BMN^{+}02]. The complete knowledge representation specification [concept syntax, role syntax, attribute syntax, statement syntax, assertion syntax and semantics] is available at [PsS93].

9.4. Matching Algorithm

Using the above described description logic language we have implemented a matching algorithm that will provide rating on the basis of computed degree of matching between two concepts. Different filtering techniques were studied to find the appropriateness and performance for this purpose. The techniques which poorly performed were then eliminated.

One of such technique is bigram approach where two strings are matched on the basis of sequences of two letters. This method is an example of static syntax based method. It is found that matching of character pairs is inefficient because it gives incorrect results if words are homographs. Moreover contextual information is lost when such matching is done. Second technique that was discarded is matching properties on the basis of its “types”. Because it was researched that two properties of same name can have different data type. Thus matching of data type is irrelevant and we have avoided the fact that property types are derived through solipsism.

Quality and speed are the two major attributes related to the matching process. A user demands the quality or speed desirable for their application. Different environment has different performance and speed requirement. Number of filtering levels used for matchmaking algorithm is decided by the speed and quality of matching desired. Based upon the number of levels of filtering used quality of matching can be differentiated as follows:

1. Best: As the name suggests the quality of matching acquired through this approach is the best. All the filtering techniques described later in this thesis are used which makes this approach most restrictive. Restrictive means the less matching concepts are eliminated and so at the end of the whole process only few best matching concepts will be available. It becomes very slow because of applying all the techniques. This approach is used in the environment such as health, military and domains where matching quality is the critical factor.
2. Better: The quality of matchmaking desired for this approach is less critical then “Best” approach. Also this approach performs matching quicker then the previous approach. Hence fewer filtering methods are applied and less number of concepts is eliminated. This type of approach is highly

preferred as it is less strict and achieves speed needed in most applications.

3. Good: In some applications speed is the most critical factor. In such cases results returned by faster syntactic matching with “Good” enough quality of matching is satisfactory. Least number of filtering techniques is applied to achieve speed and matching of concepts. This approach will return maximum number of matching results.

9.4.1. Filtering Techniques

The inference based matchmaking algorithm consists of six filtering techniques as described below:

1. Direct matching: Matching the words directly using middle layer reference system, also conditionally match labels (alternate name of the concept).
2. Description-based matching: The annotations of each concept are matched. Thus matching is based upon the description provided along with the concept definition.
3. Signature matching: Matching concepts[1] that are equivalent to concept C with concepts that are equivalent to concept C’.
4. Role-based matching: Matching roles associated with the concept C to the roles associated with concept C’.
5. Hierarchical Matching: Matching parent/child concepts of C which can be computed using subsumption relations with that of C’.
6. Axiomatic-rule matching: This technique computes concepts related to concept C by the assertions and constraints. Also it computes concepts

[1] Concept A is considered to be matching Concept B (denoted as A≡B) if for all interpretations $\mathcal{I}$; $A^{\mathcal{I}}=B^{\mathcal{I}}$ Interpretations $\mathcal{I}$ consist of a non-empty set Δ^{I} (the domain of the interpretation) and an interpretation function, which assigns to every atomic concept A, a set $A^{I} \subseteq \Delta^{I}$.

that are related to concept C’ with its corresponding constraints. Matching is done between these computed concepts.

Each of the above techniques provides the rating depending on the degree of matching calculated in the respective step. The accumulated rates from each filtering step are then normalized to calculate final star rating. This star rating is compared against the cut-off or threshold value. If they are out of threshold range then it will be concluded that concepts are distant concepts.

From the above given filtering techniques, first two techniques are syntax based matching as they directly matches words. Third technique falls under semantic matching category, as equivalent concepts are computed by logically reasoning the classified form of ontology. Role-based matching, hierarchical matching and axiomatic-rule matching are considered as semantic matching techniques. It involves matching of roles, concepts related with subsumption relation and matching of concepts related through logical assertions (constraints) respectively which are all computed by inference queries on the ontology.

For practical application and explanation of filtering techniques we compare concepts from family ontology [CHR] which we will refer as ont-A and generation ontology [Hor] as ont-B.

Let us match the concept ‘Father’ denoting it as C from ont-A and concept ‘Father’ denoting as C’ from ont-B. Although the name of the concepts looks alike it is possible that their underlying concepts might be distant concepts e.g. For homographs (two words with the same spelling) or polysemy and contronyms, the meaning of word depends on the context in which it is used. This meaning and context of the word can be well understood by taking into consideration the relation of this concept with other domain concept and comparing the constraints relating it to other concepts.

9.4.1.1. Direct matching

To perform this type of filtering concept C is matched against concept C' directly. Concepts can also be matched on the basis of labels which are the alternate name given to a concept. Let L = {l_1, l_2, l_3...} list of labels for concept C and L' = { l_1', l_2', l_3'…}. Then filtering technique operates in two steps:

1. Direct matching is done between concept C and concept C'
2. Direct matching is done between list of labels L and list of labels L'. Also it is possible to match concept C directly with the labels in L' and vice-versa.

If step 1 is fully successful then step 2 can be skipped as strong matching is found between concepts C and C'. If step 1 fails then step 2 is conditionally executed to match their corresponding labels. If in step 2 labels of these concepts are matching, then those concepts can be considered as near concepts. For e.g. if step 1 fails and both the concepts C and C' are having label as "DAD". In that case step 2 will be able to find the match and rate these concepts as near concepts. There cannot be direct matching of spelling and thus requires some technique to perform matching of words.

The strategy we are implementing here is to map the concept C to the concept in online lexical referencing system such as WORDNET® [WOR]. Find the best matching synset from WORDNET for concept C and match that synset with concept C'. By using middle layer consisting of open widely used dictionaries, heterogeneity between domains of concepts C and C' can be resolved. We term above process as "reference matching". As global dictionaries are complete mostly all terms related to concepts C and C' can be found.

Search of a word "Father" in WORDNET® resulted into 8 polysemy count or senses in noun category and 1 sense in verb category. Performing syntactic comparison between word "Father" and these senses, 1 sense found is a "male Parent" which can be considered nearest match. The word "Woman" was found as antonym to sense 1. Our algorithm works by finding all the related hypernyms, hyponyms, and words related with estimated frequency. This search is meaningful to map two concepts with each other.

There is another famous technique know as trigger-pair model [Ros94] which can be used to determine the real-valued word distance between roots and attached concepts of the word pairs does not exceed certain threshold value. Here if the word pairs are having the similar semantic domain then they are given the name as trigger-pairs. Stars are given depending on the real-valued distance.

Command Statements:

a) Query to fetch label names. Get the label (rdf:label) for the concept C and C'. Using method from Protégé-Owl API [POA]:

```
edu.stanford.smi.protegex.owl.model.impl.AbstractOWLModel
                    |
                    |__getRDFSLabelProperty( )
  Returns: DAD
```

b) All the Concepts from the OntB. Using TBOX [GiL96] query of RacerPro:
(all-atomic-concepts tbox-name)

[Note: When we want to a find matching concept for C among concepts in Ont-b, then the initial step is to get list of all the concepts in Ont-b which we is done by this command]

9.4.1.2. Description based matching

For this filtering technique annotation property is extracted. This annotation property provides textual description about the concepts, understandable by the humans. Annotation property can be considered as a document. The document matching technique that we implement is on the basis of famous technique from Information Retrieval area, known as Term frequency-inverse document frequency weighting (TF-IDF) [SaB87].

If we denote a word as α then frequency of α in all the documents together is termed as Document Frequency df(α). Similarly we can term *wf(α,d)* as the number of times word appears in a document 'd'. So now the relevance of document 'd' on basis of α is directly proportional to *wf(α,d)* and can be considered inversely proportional to df(α).

Each word α in document 'd' holds different significance. Significance of classification of word α for document 'd' from set of 'D' documents can be denoted as *wt*(α, d). It can be calculated as follows:

$$wt(\alpha,d) = wf(\alpha,d) * \left(\frac{\log|D|}{df(\alpha)} \right) \qquad \text{Equation 9.1}$$

If there is more then one document to be compared another formulae can be used to find the weight of the word as below:

$$wt(\alpha,d) = wf(\alpha,d) * \left(\log_2 \frac{N}{n} \right) \qquad \text{Equation 9.2}$$

[where 'N' is total number of documents and 'n' is number of documents where the word '*α*' is used at-least once.]

As a result for a document 'd' we can form weighted keyword representation *wkv(d,V)* contains for every word *α* in a given dictionary V the weight *wt(α,d)* as an element. Here V is the medical dictionary which can contain large vocabulary. We can cut down the dimension of the vector by heuristically deciding fixed set of words and set of words pertaining to particular medical domain [for e.g. considering words only for Neurology domain].

Finally the distance between document d_1 provided by annotation of concept C and document d_2 provided by annotation of concept C' can be calculated as:

Equation 9.3

$$dist_{(di,d2)} = 1 - \left(\frac{d_1 \cdot d_2}{|d_1| \cdot |d_2|} \right)$$

Here $d_1 . d_2$ is the inner product of weighted keyword vectors. If $dist_{d1,d2}$ increases beyond threshold value β then those concepts are considered distant concepts, otherwise those concepts are assigned stars depending on the value $dist_{d1,d2}$.

Command statements: Get the annotation property (rdfs:Comments) for the concept C and C'. Using method from Protégé-Owl API:

```
edu.stanford.smi.protegex.owl.model.impl.AbstractOWLModel
                         |
                         |__ getRDFSCommentProperty( )
```

Returns: Father is a Person having atleast 1 child and is a male.

9.4.1.3. <u>Signature Matching</u>

This filtering technique is categorized as semantic matchmaking technique. For concept C, all the equivalent concepts CL = {c_1, c_2, c_3…} are extracted using assertions. Similarly equivalent concepts for concept C' is computed using assertions applied on concept C'. If the assertions applied as necessary and sufficient condition to the concept C are same as assertions applied as necessary and sufficient condition to another concept then both these concepts are having same semantic structure. The concepts related to each other with similar assertion values are termed as equivalent concepts.

Each assertion consists of expression constructed using qualifier and/or cardinality constraints and/or value constraints. Role specifier and other concept can also be in specification of assertion.

Assertion →	*expr* | *C*
expr →	*C U C* | *C ∩ C* | *expr U expr* | *expr ∩ expr* | *{role}{hasValue}{value2}* | *{owl-restriction}{role}{C}* | *{role}{cardinality constraint}{value1}*
owl-restriction →	*∃* | *∀*
cardinality-constraint →	>= | <= | =
value1 →	*Integer*
value2 →	*C* | *Individual*

Table 9.3 Concept Assertion Expression

Command Statements: Get the equivalent concept for the concept C and C'. Using RacerPro Tbox query:

(concept-synonyms
|http://health.informatics.iupui.edu/ontology/matching/family.owl#Father|)

Results: (|http://health.informatics.iupui.edu/ontology/matching/family.owl#Dad|
|http://health.informatics.iupui.edu/ontology/matching/family.owl#Father|)

[Future extension: Comparing two concepts related by owl:sameAs (OWL construct)]

9.4.1.4. Role-based matching

All the concepts have their distinct features which distinguishes them from other concepts. These features are known as attributes of the concept. Role is the more generic term which includes attributes and “is-a” kind of relations. Role-based matching filtering technique is to match roles of concept C with roles of concept C’.

Different kinds of roles exists and can be applied to concept, such as Object-type properties, Datatype properties, properties inherited from parent concepts, sub-properties of directly applied property.

For this technique; direct role of concept C such as “hasChild” is matched against direct role of concept C’ such as “hasChild”. Matching attributes this way helps to check whether concept C is sibling concept of C’. Because it can be noticed that many times concept having uncommon parents or ancestors have substantial different properties e.g. concept “Person” have two sub-concepts as “Father” and “Mother”. Both of these concepts have same roles “hasChild” and “hasSex” so we can conclude that these concepts are near concepts. Whereas the concept “Brother” having super-concept as “Person” has attribute “hasSex” but not “hasChild”. We can find that “Father” and “Brother” have one common attribute and thus consider that they have partially near. If we consider other concept such as “Plant” it doesn’t contain attributes “hasChild” or “hasSex” attribute and thus we can say that concept “Plant” and “Father” are the far concepts.

Nearness of two concepts can also be proved by comparing property of one concept with sub-property of another concept. To understand let us consider concept C (Father) has property “hasChild” which has sub-property “hasSon". Let us consider that John is son of Dave. We can say Dave is related to John by “hasSon” role or alternatively can be related to John by “hasChild” relation. Thus it is sometimes important to perform match between properties and sub-properties of concepts.

This filtering technique can be applied in four steps:

1. Perform direct matching between direct roles of concept C with direct roles of concept C’.
2. Perform matching between direct roles of concept C and sub-roles of concept C’.
3. Perform matching between sub-roles of concept C and sub-roles of concept C’.
4. Perform match between equivalent roles of concept C and equivalent roles of concept C’.

Command Statements: Get all the direct properties for concept C. Using method from Protégé-OWL API:

```
edu.stanford.smi.protegex.owl.model
                    |
                    |__RDFSNamedClass.getUnionDomainProperties(Boolean
transitive)
```

Results: Inherited Properties (transitive = true):- hasChild, hasSex, hasAunt, hasBrother, hasConsort, hasDaughter, hasFather, hasMother, hasNephew, hasNiece, hasParent, hasSibling, hasSister, hasSon, hasUncle, name

[Future extension: Perform matching between concepts by matching equivalent properties of direct properties]

9.4.1.5. <u>Hierarchical Matching</u>

This filtering technique is categorized as semantic matching strategy. Ontology is mainly organized into concept hierarchy. Large quantity of information can be inferred from such class-subclass relationship. This hierarchy provides information about where the concept stands in the hierarchy. Such categorization provides view of information related with particular category or sub-category. Hierarchy provides information about the parent concepts and child concepts for the given concept

This subsumption relation addresses the semantic meaning associated with the given concept hierarchy. It can be easily inferred that concepts having strong match between their structures are necessarily near concepts. The concept hierarchy for concept C can be computed using Tbox commands provided by RacerPro.

This filtering technique is executed in two steps:

1. For the concept C; parent concepts are obtained and matched along the parent concepts of C'.
2. For the concept C; child concepts are obtained and matched along the child concepts of C'.

The above filtering steps computes the degree of matching and rates accordingly. The depth until which such matching has to be done is the matter of observation and performance quality required.

Command Statements:

Step-1: Get all the concepts subsuming the concept Father. Using RacerPro Tbox Query:

```
(concept-parents
|http://health.informatics.iupui.edu/ontology/matching/family.owl#Father|)
```

Results:

```
((|http://health.informatics.iupui.edu/ontology/matching/family.owl#Parent|)
(|http://health.informatics.iupui.edu/ontology/matching/family.owl#Man|))
```

Step-2: Get all the concepts “directly” subsumed by the concept Father

(concept-children
|http://health.informatics.iupui.edu/ontology/matching/family.owl#Father|)

Results: ((*BOTTOM* BOTTOM))
BOTTOM is the lower most concept attached automatically to the concept hierarchy.

9.4.1.6. Axiomatic Matching

This is the most complex level of filtering technique and should be applied when speed is not the major concern. Restrictions or constraints are the assertions applied to the properties of concept to restrict the range of values that property can have. These values can be cardinal, XML Datatype, another concept or individual. Let us say set of properties related to concept C:

$X = \{x_i \mid x_i(d) \in C, x_i(r) \in V\}$
V = {XML Datatype | String | Concept | Individual}
where C is list of concepts, V is the list of values.
$x_i(d)$= domain of property x_i
$x_i(r)$ = range of property x_i

This filtering technique finds out all the values related to the given concept C using constraint applied on the given restricted property of the concept. Constraint relates one concept in the concept hierarchy to another concept in the concept hierarchy. The importance of this step can be well understood by the following example: Concept “Father” has the properties such as: “hasChild” and “hasSex”. The property “hasChild” is restricted by cardinality constraint “>=” and “hasSex” is restricted by “$\ni$”(hasValue). Concept “Mother” also holds the same attributes and restrictions, but what really distinguishes both of these concepts is the value related with the constraint “hasSex”. For the concept “Father” this value is another concept “Male” whereas for the concept “Mother” value of the constraint is “Female”.

The following sequence should be followed for this technique:

1. Get all the properties or roles for given class C.
2. Get Collection of all Restrictions that are defined on a given property.
3. For each Restriction on given property get the corresponding filler value which can be RDFSDatatype, OWLDataRange or RDFS Class (concept).
4. Compare values obtained in step 3 with corresponding values obtained for Class C’.

Command Statements:

```
Step 1: edu.stanford.smi.protegex.owl.model
                 |
                 |__RDFSNamedClass.getUnionDomainProperties(Boolean
transitive)
Step 2: edu.stanford.smi.protegex.owl.model
                 |
                 |__
                 getOWLRestrictionsOnProperty(RDFProperty property)
Step: 3
         If restriction is of type ‘϶’ (hasValue) then it can be obtained by:
                 edu.stanford.smi.protegex.owl.model
                         |
                         public Object getHasValue()
         else restrictions are Quantifier restrictions such as ∃ or ∀
                 edu.stanford.smi.protegex.owl.model
                         |
                         public RDFResource getFiller()
```

The performance of the above mentioned filtering techniques can be measured by applying these techniques on different ontology models. We haven't showed any quality analysis results in this thesis as that will be one of the future tasks. In all of the above filtering techniques a need arises for matching concept words, roles or constraints syntactically in final stage of matching. Whenever such matching has to be done, we use the reference matching technique described in our first filtering technique.

The important aspect of our inference based filtering technique is there is no need of human intervention. Previous methods such as LARKS [SWK+02] needed human presence to initialize the matrix and thus were not been able to perform well. Speed can be achieved by if this task is executed automatically in machine interpretable manner. All the above techniques are applied directly on ontology without any infrastructure to be initialized. Our goal is to make matchmaking process capable of mapping concept from one terminological system into other terminological system of medical domain.

Note: A detail example is given in CHAPTER 10, which shows matching process between two similar concepts. Also we have provided an example that demonstrates matching process of two far concepts.

9.4.2. Normalization

The final degree of matching depends on the normalized value computed by combining rates from all the above mentioned filtering techniques. Each filtering technique can be assigned weight depending on its importance in matchmaking process. The weight considered is strictly heuristic value and mostly found through observation. Equation 9.4 calculates final rating $\mathbf{R}_{(c,c')}$ by multiplying rating from different filtering technique with its corresponding weight from weight vector. The final summation is then normalized by dividing it with the number of filtering steps used.

$$R_{(c,c')} = \frac{1}{j} * \sum_{i}^{j} \Phi_i \lambda_i$$ Equation 9.4

where Φ_i = weight assigned to filtering technique i

λ_i = degree of matching after applying technique i

j = number of filtering steps (mode)

The value for $\mathbf{R}_{(c,c')}$ lies between 0 and 5. If the value of $\mathbf{R}_{(c,c')}$ is above threshold value β then concepts are considered as matching concepts. Value for threshold can be heuristically decided. The best way to select suitable threshold value is, to initially allocate a very high value to β. So the number of matches found will be very less, and then slowly decrease this value to include more concepts. Final value of β will be the one which returns required number of matching concepts.

$$\sum_{i=1}^{j} \Phi_i \cong j$$ Equation 9.5

The Equation 9.5 is the necessary condition for rating. It checks that the sum of weights used in Equation 9.4 is not less then or greater then total number of filtering techniques used for rating the matchmaking of concepts.

9.5. Matchmaking Modes

Matchmaking process can be categorized in several modes depending on the severity of filtering done. Severity of filtering is directly related to quality of matching and inversely related to process speed. Different environment demands different quality and speed as discussed earlier. We broadly derive matchmaking modes from the approaches they follow and are presented below.

9.5.1. Aggressive mode

This mode adopts the “BEST” approach and so the number of matching results returned is quite few. This mode eliminates retains strongly matching concepts and so the result of matching is highly qualified. This mode of operation is used where quality of matching is the major concern whereas speed is the least concern. Here all the above discussed filtering techniques are applied. This technique is most suited for health enterprise domain. In medical domain, disease and surveillance data are mostly exchanged; such data are very critical and should be properly mapped. Our main focus is to implement this strategy to provide service like interoperation between health enterprises.

9.5.2. Normal mode

It is found that aggressive mode is too strict for certain uses and such modes are rarely used. The more frequent need is of less strict mode which can perform well along with quick matching. At the same time such filtering technique should not be too lenient and thus should work efficiently. We term such mode as Normal mode which can provide both quality and speed performance. “Better” approach is been obeyed by this mode and thus only selective filtering techniques are applied. This mode is most suitable for dynamic environment such as Internet where both quality and speed is of concern. Majority of search engines implements this mode.

Techniques selected to be used for this mode is decided by observation of speed and filtering quality. We use direct filtering, description-based filtering, signature matching, role-based matching and hierarchical matching in normal mode. As axiomatic matching technique was dealing with behavioral aspect matching it was more complex to calculate and consumes lot of time.

9.5.3. Lenient mode

This mode of matchmaking is needed to be less time consuming and only few concepts are filtered out. Thus the amount of matches found is large. We haven’t tried to test the working of this mode of operation in case of concepts which are homographs and tautology. As less number of filtering techniques is used; it obeys “Good” strategy which is explained earlier. The filtering techniques selected for this mode has to be quick. Hence we use direct matching, description-base matching and signature matching techniques. This mode can be used in document search methods, file based searching as well as search facility provided by desktop PCs.

CHAPTER 10. ALGORITHM ANALYSIS

Considering the same example of “father” used to explain different filtering techniques, we will provide detail comparison results and rates provided on the basis of degree of matching. Here concept C is from “father.owl” file ontology and concept C’ is from “generations.owl” ontology. Also we will show how the matchmaking results can change across different modes. The first example is for showing working of filtering techniques on concepts which are similar. In the Second example we will test matchmaking algorithm on distant concepts.

10.1. Example 1

We are considering concept “Father” from father.owl as C, and concept “Father” from generations.owl as C’.

10.1.1. Aggressive Mode

Under this mode we will be using all the six filtering techniques.

Step: - 1 Direct matching

Command	Concept C	Concept C’
Direct Match	Father	Father
getRDFSLabelProperty( )	Pater	Daddy

Results: Concept C matches strongly with concept C’ using direct match and so this step succeeds fully.

Stars allocated: ★★★★★

Step: - 2 Description-based matching

Command	Concept C	Concept C’
getRDFSCommentProperty()	Father is a person having at-least one child and has gender as male.	Father is a male and also person having a child who is a person too.

Results: The complete inverse document frequency weighting (TF-IDF) strategy is not included or tested and will be the one of the topic of future extension.

Note: There is alternate solution to obtain the comments associated with concepts using TBOX-Retrieval query and data-substrate layer in RacerPro.

Step: - 3 Signature matching

Command	Concept C		Concept C’
(concept-synonyms |http://health.informatics.iupui.edu/ontology/matching/family.owl#Father|)	(|http://health.informatics.iupui.edu/ontology/matching/family.owl#Dad|)	(concept-synonyms |http://health.informatics.iupui.edu/ontology/matching/generations.owl#Father|)	-

Results: It can be seen that there are no equivalent concept for concept C’. In such cases this filtering step can be skipped from final normalization. Alternately the rating is done heuristically or a special value can be allocated which denotes that no decision was taken. The approach taken is implementation specific.

Step: - 4 Role-based matching

Command	Concept C	Concept C’
Father.getUnionDomainProperties (true)	hasChild, hasSex, □as Aunt, hasBrother, hasConsort, hasDaughter, hasFather, hasMother, hasNephew, hasNiece, hasParent, hasSibling, hasSister, hasSon, hasUncle, name	hasChild, hasSex

Results: matching direct properties succeeds, while some properties are not found.
Stars allocated: ★★★★

Step: - 5 Hierarchical Matching

Command	Concept C		Concept C’
(concept-parents |http://health.informatics.iupui.edu/ontology/matching/family.owl#Father|)	((|http://health.informatics.iupui.edu/ontology/matching/family.owl#Parent|) (|http://health.informatics.iupui.edu/ontology/matching/family.owl#Man|))	(concept-parents |http://health.informatics.iupui.edu/ontology/matching/generations.owl#Father|)	((|http://health.informatics.iupui.edu/ontology/matching/generations.owl#Parent|) (|http://health.informatics.iupui.edu/ontology/matching/generations.owl#Man|))

(concept-children |http://health.informatics.iupui.edu/ontology/matching/family.owl#Father|)	-	(concept-children |http://health.informatics.iupui.edu/ontology/matching/generations.owl#Father|)	((|http://health.informatics.iupui.edu/ontology/matching/generations.owl#GrandFather|))

Results: The first row computes the concepts subsuming concept C and C’. The match is found between computed concepts and so this filtering step fully succeeds. In the second row we are fetching concepts subsumed by concept C and C’. For concept C there are no subsuming concepts and so results from first row is only considered.

Stars: ★★★★

Step: - 6 Axiomatic Matching

Command	Concept C	Concept C’
Father.getUnionDomainProperties(true) (STEP-1)	hasChild, hasSex, □as Aunt, hasBrother, hasConsort, hasDaughter, hasFather, hasMother, hasNephew, hasNiece, hasParent, hasSibling, hasSister, hasSon, hasUncle, name	hasChild, hasSex

getOWLRestrictionsOnProperty(hasChild) (STEP-2)	>=	∃
getOWLRestrictionsOnProperty(hasSex) (STEP-2)	∍	∍
getHasValue( ) for ∍ restriction (STEP-3)	Male	MaleSex

Results: It is found in step 2 that the restriction applied on “hasChild” property of C differs from restriction on “hasChild” property of C’ and thus step-3 is not executed for such properties. Restriction on “hasSex” property of each concept is same and as a result step-3 provides the values related to this restriction. The match is found as Male ≅ MaleSex.
Stars: ★★★★

Normalization:

Let us use the Equation 9.4 derived earlier to find the final rating of matchmaking algorithm on the given concepts.

$$R_{(Father,Father)} = \frac{1}{4}[(1*5)+(1*4)+(1*4)+(1*4)] = {}^{17}\!/_{4} = 4.25$$

Here stars from direct, role-based, hierarchical and axiomatic matching are considered. For simplicity we assign same weight-age (Φ_i) to all of the filtering techniques and that is equal to 1. Heuristically we can decide the threshold of matchmaking algorithm. Let us for our purpose consider 2.75 as threshold value. Here we get final rating value as 4.25. Final rating value is certainly greater then threshold limit and thus concepts C and C’ are considered matching.

10.1.2. Normal Mode

Using filtering results of direct matching, role-based matching and hierarchical matching, the final rating value of matchmaking algorithm used under normal mode is:

$$R_{(Father,Father)} = \frac{1}{3}\left[(1*5)+(1*4)+(1*4)\right] = {}^{13}\!/_{3} = 4.33$$

The result R(Father,Mother) for normal mode is showing higher rates then aggressive modes because normal mode is less strict and so doesn't consider the partial matching of behavior.

10.1.3. Lenient Mode

Using filtering results of direct matching, the final rating value of matchmaking algorithm used under lenient mode is:

$$R_{(Father,Father)} = \frac{1}{1}\left[(1*5)\right] = {}^{5}\!/_{1} = 5.0$$

This mode gives 5 stars to the matching process of given concepts. It can be seen that results are consistent with the theory provided earlier. As this mode is lenient it only considers the syntactic matching.

10.2. Example 2

Now we take two concepts which might be distant concepts to demonstrate working of matchmaking algorithm. The first concept C' is “Woman” from generations.owl and let the second concept C be “Father” from family.owl

10.2.1. Aggressive Mode

Under this mode we will be using all the six filtering techniques.

Step: - 1 Direct matching

Command	Concept C	Concept C’
Direct Match	Father	Mother
getRDFSLabelProperty( )	Pater	Mater

Results: Direct match fails as Father and Mother are considered antonym by our reference system.
Stars allocated: ★

Step: - 2 Description-based matching

Command	Concept C	Concept C’
getRDFSCommentProperty()	Father is a person having at-least one child and has gender as male.	Mother is a female and also person having a child who is a person too.

Results: The complete inverse document frequency weighting (TF-IDF) strategy is not included or tested completely and will be the one of the topic of future extension.

Note: There is alternate solution to obtain the comments associated with concepts using TBOX-Retrieval query and data-substrate layer in RacerPro

Step: - 3 Signature matching

Command	Concept C	Command	Concept C’
(concept-synonyms |http://health.informatics.iupui.edu/ontology/matching/family.owl#Father|)	(|http://health.informatics.iupui.edu/ontology/matching/family.owl#Dad|)	(concept-synonyms |http://health.informatics.iupui.edu/ontology/matching/generations.owl#Mother|)	(|http://health.informatics.iupui.edu/ontology/matching/generations.owl#Mummy|)

Results: “Dad” is the equivalent concept of “father” and “Mummy” is the equivalent concept of “mother”. Match cannot be found between “Dad” and “Mummy”. Hence this step fails.

Stars allocated: ★

Step: - 4 Role-based matching

Command	Concept C	Concept C’
Father.getUnionDomainProperties (true)	hasChild, hasSex, hasAunt, hasBrother, hasConsort, hasDaughter, hasFather, hasMother, hasNephew, hasNiece, hasParent, hasSibling, hasSister, hasSon, hasUncle, name	hasChild, hasSex

Results: matching direct properties succeeds, while some properties are not found.

Stars allocated: ★★★★

Step: - 5 Hierarchical Matching

Command	Concept C	Command	Concept C’

(concept-parents |http://health.informatics.iupui.edu/ontology/matching/family.owl#Father|)	((|http://health.informatics.iupui.edu/ontology/matching/family.owl#Parent|) (|http://health.informatics.iupui.edu/ontology/matching/family.owl#Man|))	(concept-parents |http://health.informatics.iupui.edu/ontology/matching/generations.owl#Mother|)	((|http://health.informatics.iupui.edu/ontology/matching/generations.owl#Parent|) (|http://health.informatics.iupui.edu/ontology/matching/generations.owl#Woman|))
(concept-children |http://health.informatics.iupui.edu/ontology/matching/family.owl#Father|)	-	(concept-children |http://health.informatics.iupui.edu/ontology/matching/generations.owl#Mother|)	((|http://health.informatics.iupui.edu/ontology/matching/generations.owl#GrandMother|))

Results: The command in first row computes the concepts subsuming concept C and C’. The match is not found as “Man” and “Woman” is disjoint concepts and so this step fails. In the second row we are fetching concepts subsumed by concept C and C’. For concept C there are no subsuming concepts and so results from first row is only considered.

Stars: ★

Step: - 6 Axiomatic Matching

Command	Concept C	Concept C’

Father.getUnionDomainProperties(true) (STEP-1)	hasChild, hasSex, hasAunt, hasBrother, hasConsort, hasDaughter, hasFather, hasMother, hasNephew, hasNiece, hasParent, hasSibling, hasSister, hasSon, hasUncle, name	hasChild, hasSex
getOWLRestrictionsOnProperty(hasChild) (STEP-2)	>=	∃
getOWLRestrictionsOnProperty(hasSex) (STEP-2)	∍	∍
getHasValue( ) for ∍ restriction (STEP-3)	Male	FemaleSex

Results: It is found in step 2 that the restriction applied on “hasChild” property of C differs from restriction on “hasChild” property of C’ and thus step-3 is not executed for such properties. Restriction on “hasSex” property of each concept is same and as a result step-3 provides the values related to this restriction. “Male” is related to “Father” and “FemaleSex” is related to “Mother”. As “Male” and “FemaleSex” are distant concepts this filtering step fails.
Stars: ★

Normalization:

Let us use the Equation 9.4 derived earlier to find the final rating of matchmaking algorithm on the given concepts.

$$R_{(Father,Mother)} = \frac{1}{5}\left[(1*1)+(1*1)+(1*4)+(1*1)+(1*1)\right] = 9/5 = 1.8$$

Here stars from Direct, signature-based, role-based, hierarchical and axiomatic matching is considered. For simplicity we assign same weight-age (Φ_i) to all of the filtering techniques and that is equal to 1. Heuristically we can decide the threshold of matchmaking algorithm. Let us for our purpose consider 2.75 as threshold value. Here we get final rating value as 1.8. Final rating value is certainly smaller then threshold limit and thus concepts are considered distant concepts.

10.2.2. Normal Mode

Using filtering results of direct matching, signature-based, role-based matching and hierarchical matching, the final rating value of matchmaking algorithm used under normal mode is:

$$R_{(Father,Mother)} = \frac{1}{4}\left[(1*1)+(1*1)+(1*4)+(1*1)\right] = 7/4 = 1.75$$

The result R(Father,Mother) for normal mode is lower rates then aggressive modes because normal mode gives equal importance to syntactic matching as semantic matching, whereas in aggressive mode we are stressing on semantic meaning of word and relations.

10.2.3. Lenient Mode

Using the results obtained by applying filtering technique 1, 2 and 3 is used here.

$$R_{(Father,Mother)} = \frac{1}{2}\left[(1*1)+(1*1)\right] = 2/2 = 1.0$$

This mode gives 1 star to the matching process of given concepts. It can be seen that results are consistent with the theory provided earlier. As this mode only considers the syntactic matching the words “Father” and “Mother” are treated as totally disjoint.

CHAPTER 11. KNOWLEDGE REPRESENTATION TOOLS

In this section we give details about tools that provide interface and facilities to develop knowledge base and perform several operations on it. The two primary tools used in our work are Protégé [Pro] and RacerPro [RAC].

11.1. Protégé

There are several tools available for building and editing ontologies. Few well-known amongst them are OilEd [HOE] and Protégé. The complete survey and capabilities of tools can be found at [Den04].

Protégé is an easy-to-use graphical interface for creating and editing ontologies and knowledge base. Primary feature important to our work is:

- It provides support for RDF [LaS99], RDFS [BrG00], DAML+OIL [CHH$^+$01], XML, OWL, CLIPS [Gia02] and UML [UML].
- Concept subsumption and satisfiability via a DIG-compliant [Bec03] reasoner such as RacerPro or FaCT.
- It has facilities to provide full, extensible metamodel and metaclass support, multiple inheritance. OWL language elements including named classes, properties, restrictions, logical class expressions, enumerations, individuals, metaclasses, ontology metadata and other annotations. Some of them are shown in Table 4.
- Protégé is available in Java API class format which allows user to integrate Protégé with their application.

- Protégé can be widely extended by Protégé plug-ins. The complete listing of such plug-ins can be found here [PPL].

Description Logic constructs	Protégé elements
Concept C	Class
Role R	Slot
Individual I	Individuals
Number, integer, string	XMLSchema#Datatype
(and $C_1 \ldots C_n$)	($C_1 \sqcap C_2 \sqcap \ldots. C_n$)
(or $C_1 \ldots. C_n$)	($C_1 \cup C_2 \ldots\ldots. C_n$)
(not C)	($\neg C$)
(all R C)	$\forall$ R:C
(some R C)	$\exists$ R.C
(at-least n R)	$\geq$ n R
(at-most n R)	$\leq$ n R
(exactly n R)	= n R

Table 11.1 Description Logic Axioms vs. Protégé Axioms

The plug-ins developed on top of Protégé, provides connection between Protégé and other tools. RacerPro server is one of such tool which can work with Protégé client. The RQL tab available in Protégé enables it to load owl ontology in RacerPro. It also enables to send the inference queries to the server.

Protégé is written completely in Java and is maintained by Open Source community which makes is extensible and powerful.

11.2. RacerPro

RACER [RAC] and FaCT [FaC98] are two well known DL reasoners available. RacerPro is the commercial form of RACER systems which is very well known from long time for providing inference system facilities. It is a knowledge representation system that implements a highly optimized tableau calculus for very expressive description logic. It provides reasoning facilities for Tbox and Abox querying. The new extended query language nRQL [HMW04] provides large library of commands to query Abox.

RACER implements the description logic $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D})$ as described in DL section of this thesis. RacerPro implements the HTTP-based quasi-standard DIG for interconnecting DL systems with interfaces and applications using an XML-based protocol. The primary types of inference [HaM01a] provided are:

- Consistency of Abox and Tbox
- Subsumption relation
- Classification of Tbox
- Coherence check for Tbox
- Instance checking
- Retrieval of individuals

RacerPro internally converts OWL code into DIG code. This code can be viewed from Protégé editor. RacerPro server supports multi-connection capabilities and thus can accept queries from many clients running on different machines. Each client can send query to RacerPro Server.

11.2.1. Shortcoming of RacerPro

Problem with RACER is if the ontology or owl file is modified then ontology has to be reloaded into RacerPro server [GcTB01]. This reloading results into re-classification of the taxonomy. This limitation can be resolved externally by generating an application thread that will perform full reset on RacerPro server as soon as ontology is modified. Also this application thread will command RacerPro to re-classify the taxonomy.

CHAPTER 12. OPEN TERMINOLOGICAL SYSTEM

12.1. Introduction

'Open Source' software is now ubiquitous. The notion that software can be both free, reliable, and supported, is no longer at issue. But how does this relate to clinical terminology?

In today’s world abundant research is been done on patient’s health data, syndromic outbreak data and in clinical laboratories in different parts of world. Many European countries and Asian countries have developed a strong network amongst the health providers. This network ties up all the medical health partners and also different medical domains together.

A terminological system which has capability of integrating any medical terminology seamlessly and performs virtual conversion of data to be send from one domain to another is termed as Open Terminological System. Need for this system is increasing as research is performed in many parts of world and can prove important to other medical researcher and public health providers and so it is important that they should be able to share their research. By collaboration of work, amount of redundant research efforts can be significantly reduced.

Currently investigators from one medical domain are unable to get important results and data from different medical domain, because the vocabularies used by both domains are different. Most of the medical vocabularies are strictly licensed; some of them allow partial use of this vocabulary. Moreover legacy systems used by institutes make it harder to share data. Our concept is to develop a middle layer which performs conversion of data transparently.

Making this system open to everyone will allow medical investigators to integrate terminology used by them with others and thus form a virtual “Global” terminology. Let us see some of the projects and initiatives carried out in this direction.

12.2. Background and related work

12.2.1. BioMOBY

The BioMOBY Project is an open-source, simple, extensible platform to enable the discovery, representation, integration, and retrieval of biological data from widely disparate data hosts and analysis services [BIO]. They use MOBY-S (MOBY-Services) central registry server for services. MOBY-S registry uses ontologies to determine the structure and relationships between data-types and services to provide service discovery. The S-MOBY (Semantic-MOBY) branch of BioMOBY encompasses a minimal set of reserved-word assertions to allow the construction of ontological relationships. Clients and providers communicate through middle-layer vocabulary.

12.2.2. OpenGalen

OpenGalen [OGa] is a new approach to the development of clinical systems and the sharing of medical knowledge. GALEN [RSN+94] has developed a Terminology Server to support the development and integration of clinical systems through a range of key *terminological services,* built around a language-independent, re-usable, shared system of concepts - the CORE model. The focus is on supporting applications for medical records, clinical user interfaces and clinical information systems, but also includes systems for natural language understanding, clinical decision support, management of coding and classification schemes, and bibliographic retrieval. The Terminology Server integrates three modules: the Concept Module which implements the GRAIL [RBG96] formalism and manages the internal representation of concept entities, the Multilingual Module which manages the mapping of concept entities to natural language, and the Code Conversion Module which manages the mapping of concept entities to and from existing coding and classification schemes. The OpenGalen model contains a well defined set of relationships between medical concepts based on description logic (DL) theories of generation and subsumption of composite concepts.

12.2.3. The Open Terminology Services (OTS) project

The Open Terminology Services (OTS) [SAC03] project provides a common, well-specified mechanism to access terminological content in a vendor and platform neutral fashion. The project includes a freely available API specification and an open source reference implementation. The API specification defines mechanisms for browsing, querying and import terminological content. The Java-based reference implementation uses the LDAP [WHK97] for a back end, and provides a mechanism to query and distribute heterogeneous terminological content using a common format. The project includes the CTS (Central Terminology Services) subset under HL7.

12.2.4. caBIG

The cancer Biomedical Informatics Grid [caBIG], is a voluntary network or grid connecting individuals and institutions to enable the sharing of data and tools, creating a World Wide Web of cancer research. The goal is to speed the delivery of innovative approaches for the prevention and treatment of cancer. Researchers from around the world will have open access to the common platform of caBIG, be able to use common tools, and rapidly convert, relate, and analyze data from different sources. The Globus Tool Kit and the Open Grid Services Architecture-Data Access Integration (OGSA-DAI) were selected as the basis for the development of a prototype system that satisfied simple data integration and sharing use cases.

12.3. MMN Approach for OTS

MMN architecture is scalable and has the capability to include new terminological system easily. The Matchmaking algorithm explained in this thesis can be used to achieve immense interoperability.

The main need of an Open Terminological System is one vocabulary should be able to map to any other vocabulary. This can be achieved by concept of Global Ontology and mapping using matchmaking algorithm. From the experience earned from learning above mentioned system, it is found that the language used for building global ontology should be able to structure these terminologies effectively together. This can be achieved by using OWL and upcoming version of OWL. Also there is need of backbone which is scalable and this requirement is well fulfilled by architecture of MMN.

CHAPTER 13. CONCLUSION & FUTURE WORKS

13.1. RECAPITULATION

In this thesis the work done; to investigate about different needs and problems of health providers; develop solution to provide integration of services and data. We have discussed here about multi-agent based Metropolitan Medical Network which is abstract model for providing advance health services. In this research we have developed a inference based matchmaking algorithm to perform mapping between various vocabularies and provide matching services for MMN. Lastly a discussion on Open Terminological System is provided.

13.2. CONCLUSION

The investigation about the needs and problems is very helpful and gives in-depth idea about hurdles obstructing the development of health care industry. The proposed architecture and the lower level details of components of this architecture is well thought system that can remove some of the hurdles.

Study of web ontology language is very helpful for building large knowledge base system or domain ontology. Also suitability of web ontology language about expressing and openness capability was explored; which helps to inference knowledge and fetch metadata. The complete cycle of building and using ontology is developed.

Using the ontologies along with the description logic system such as RacerPro helped us to implement inference-based matchmaking algorithm. This algorithm is applied on sample ontology which provides proof of its working. Suitability of inference-based method for matching concepts to provide advance health services and interoperation is proved here. Also we have shown how the proposed network architecture, language and matchmaking algorithm will help to achieve our goal of developing Open Terminological System.

13.3. <u>FUTURE WORK</u>

There is plenty of room for development and future extension. Such as:

- Development of complete LOINC ontology containing all LOINC codes representing laboratorial results is needed.
- Also developing ontologies for other medical domain vocabularies is needed to perform mapping between them.
- Provide mapping between HL7 tags and OWL language constructs. This is needed as currently most of the health entities use this message format to exchange data.
- While explaining about some of the filtering techniques we have provided a note which says which other domain knowledge can be compared. Hence refine this matchmaking algorithm.
- Simulate the MMN network with the real data from health partners. Implement registry to register all the health service providers and develop communication links between them and middle layer.
- Providing implementation of multi-agent architecture i.e. develop an user interface application and the agents (discussed in this thesis).
- Provide facilities of automatic service composition and execution.
- Implementing the matchmaking algorithm provided here using the Protégé API and RacerPro commands. Performing matching between various different kind of ontologies to perform quality and performance analysis of this algorithm.
- After implementation phase, performing model and performance analysis on complete architecture.

LIST OF REFERENCES

[Bec03] Bechhofer, S., The DIG Description Logic Interface: DIG/1.1, February 2003.

[BIO] Wilkinson, MD., Gessler, D., Farmer, A., Stein, L., BIOMOBY, Proceedings of the Virtual Conference on Genomics and Bioinformatics (3):16-26, 2003.

[BHL01] Berners-Lee, T., Hendler, J., Lassila, O., “The Semantic Web.” Scientific American, 284(5), pp. 34-43, May 2001.

[BHR+99] Beeler, G. W., Huff, S., Rishel, W., Shakir, A-M., Walker, M., Mead, C., Schadow, G., HL7 Version 3 Message Development Framework, December 1999.

[BMN+02] Baader, F., McGuinness, D., Nardi, D., Patel-Schneider, P. F., editors, Description Logic Handbook: Theory, Implementation and Applications, Cambridge University Press, 2002.

[BrG00] Brickley, D., Guha, R. V., Resource Description Framework Schema Specification (RDFS) 1.0. W3C Candidate Recommendation, 27th March 2000.

[Bou00] Boubez, T., et al., UDDI Data Structure Reference V1.0. September 2000. HTTP://WWW.OASIS-OPEN.ORG.

[caBIG] cancer Biomedical Information Grid, HTTPS://CABIG.NCI.NIH.GOV/GUIDELINES_DOCUMENTATION/CAGRIDWHITEPAPER.PDF.

[CGS00] COHN, A. G., Giunchiglia, F., Selman, B., editors, “International Conference on Knowledge Representation and Reasoning (KR’ 2000)”, April 2000.

[CHH+01] Connolly, D., Harmelen, F. V., Horrocks, I., McGuinness, D., Patel-Schneider, P., Stein, L. A., DAML+OIL (March 2001) reference description. W3C Note, 18th December 2001. Available at HTTP://WWW.W3.ORG/TR/2001/NOTE-DAML+OIL-REFERENCE-20011218.

[CHR] Golbreich, C., A SWRL/OWL demo ontology about family relationships, HTTP://PROTEGE.STANFORD.EDU/PLUGINS/OWL/OWL-LIBRARY/INDEX.HTML.

[COG] Corporate Ontology Grid Project, HTTP://WWW.COGPROJECT.ORG/.

[Den02] Denny, M., Ontology Building: A Survey of Editing Tools, November 2002.

[Den04] Denny, M., Ontology Tools Survey, HTTP://WWW.XML.COM/PUB/A/2004/07/14/ONTO.HTML, July 2004.

[DIC04] Digital Imaging and Communications in Medicine (DICOM), Part 6: Data Dictionary, Published by National Electrical Manufacturers Association, 2004.

[ebX] ebXML: electronic business eXtensible Markup Language, HTTP://WWW.EBXML.ORG/ .

[FaC98] I. Horrocks, FaCT: DL Classifier, HTTP://WWW.CS.MAN.AC.UK/~HORROCKS/FACT/, 1998.

[FEL98] Fellbaum, C., Wordnet: An electronic lexical database for the English language. The MIT Press, 1998.

[FWW$^+$93] Finin, T., Weber, J., et al., Specification of the KQML Agent Communication Language, by The DARPA Knowledge Sharing Initiative External Interfaces Working Group, June 1993.

[GcTB01] Gonzalez-Castillo, J., Trastour, D., Bartolini, C., Description Logics for Matchmaking of Services, HP Labs Technical Report, 2001.

[GeF92] Genesereth, M., Fikes, R., Knowledge Interchange Format Version 3.0 Reference Manual. Report Logic 92-1 Computer Science Department. Stanford University, 1992.

[Gia02] Giarratano, J., CLIPS: Users Guide, v6.20, March 2002.

[GiL96] Giacomo, G., Lenzerini, M., TBox and ABox reasoning in expressive description logics. In Luigia C. Aiello, John Doyle, and Stuart C. Shapiro, editors, Proc. of the 5th Int. Conf. on the

Principles of Knowledge Representation and Reasoning (KR-96), pp. 316-327. Morgan Kaufmann, Los Altos, 1996.

[HaM00] Haarslev, V., Möller, R., “Expressive ABox reasoning with number restrictions, role hierarchies and transitivity closed roles”, In Cohn et al. [CGS00], pp 273-284. [Hen01], James Hendler, “Agents and the Semantic Web”; IEEE Intelligent systems, vol. 16, pp. 30-37, March 2001.

[HaM01a] Haarslev, V., Möller, R., Description of the racer system and its applications, In Proceedings of the International Workshop in Description Logics 2001 (DL2001), Stanford, USA, August 2001.

[HaM01b] Haarslev, V., Möller, R., Turhan, A.-Y., Exploiting pseudo models for TBox and ABox reasoning in expressive description logics. Lecture notes, [Mas01].

[HeM00] Hendler, J., McGuinness, D.L., The DARPA Agent Markup Language, IEEE Intelligent Systems, vol. 16, no. 6, Jan/Feb 2000, pp. 67-73.

[HMW04] Haarslev, V., Möller, R., Wessel, M., new Racer Query Language, 2004.

[HOE] Horrocks, I., OiLed Ontology Editor, HTTP://OILED.MAN.AC.UK/.

[Hol95] Hollingsworth, D., The Workflow Management Coalition: The Workflow Reference Model, document number TC00-1003, issue 1.1, January 1995.

[Hor] Horridge, M., An ontology about family relationships, HTTP://PROTEGE.STANFORD.EDU/PLUGINS/OWL/OWL-LIBRARY/INDEX.HTML.

[HPH03] Horrocks, I., Patel-Schneider P. F., Harmelen F. V., From SHIQ and RDF to OWL: The Making of a Web Ontology Language, Journal of Web Semantics, Volume 1, 2003.

[HST00] Horrocks, I., Slatter U., Tobis, S., “Reasoning with the individuals for description logic SHIQ”, In David MacAllester, editor, Proceedings of the 17th International Conference on Automated Deduction (CADE-17), number 1831 in lecture notes in Computer Science, Germany, 2000, Springer-Verlag.

[HuS92] Huhns, M., Singh, M., "The Semantic Integration of Information Models", AAAI Workshop on Cooperation among Heterogeneous Intelligent Agents, Washington-D.C., July 1992.

[ICD] ICD: International Code for Diseases, HTTP://WWW.WOLFBANE.COM/ICD/.

[LaS99] Lassila, O., Swick, R., Resource Description Framework (RDF) Model and Syntax specification. W3C Recommendation, 22nd February 1999.

[LCG04] Lee, Y., Patel, C., Chun, S., Geller, J., "Compositional Knowledge Management for medical services on Semantic web", International World Wide Web Conference, Proceedings of the 13th international World Wide Web conference, pp. 498-499, May 2004.

[Mas01] Massaci, F., editor, International Conference on Automated Reasoning (IJCAR' 2001), June 18-23 2001, Siena, Italy, Lecture Notes in Artificial Intelligence, 9(2):135-196, 1977.

[McH03] McGuinness D. L., Harmelen, F. V., "OWL Web Ontology Language Overview, HTTP://WWW.W3.ORG/TR/2003/PR-OWL-FEATURES-20031215/, December 2003.

[MBH+04] Martin, D., Burstein, M., Hobbs, J., et al., OWL-S: Semantic Markup for Web Services – Overview, v1.1, HTTP://WWW.DAML.ORG/SERVICES/OWL-S/1.1/OVERVIEW/, December 2004

[MHS+04] McDonald, C., Huff, S., Suico, J., Mercer, K., Logical Observation Identifier Names and Codes (LOINC) User's Guide, Regenstrief Institute, Indianapolis, 2004.

[MSH] MeSH: Medical Subject Headings, HTTP://WWW.NLM.NIH.GOV/MESH/MESHHOME.HTML

[MWM04] Smith M. K., Welty, C., McGuinness D. L., OWL Web Ontology Language Guide, HTTP://WWW.W3.ORG/TR/OWL-GUIDE/, February 2004.

[NFF+91] Neches, R., Fikes, R., Finin, T., Gruber, T., Patil, R., Senator, T., Swartout, W.R., Enabling Technology for Knowledge Sharing. AI Magazine, pp. 36-56, 1991.

[NFK+00] Nodine, M., Fowler, J., Ksiezyk, T., Perry, B., Taylor, M., Unruh. A., Active information gathering in Infosleuth. International Journal of Cooperative Information Systems, 9(1-2):3–28, 2000.

[NoM] Noy, N., McGuinness, D. L., Ontology development 101: a guide to creating your first ontology, HTTP://PROTEGE.STANFORD.EDU/PUBLICATIONS/ONTOLOGY_DEVELOPMENT/ONTOLOGY101-NOY-MCGUINNESS.HTML.

[NLM] NLM: National Library of Medicine, HTTP://WWW.NLM.NIH.GOV/.

[OGa] OpenGalen, HTTP://WWW.OPENGALEN.ORG/.

[OSC03] The OWL Services Coalition, editors, "OWL-S: Semantic Markup for web services", HTTP://WWW.DAML.ORG/SERVICES/OWL-S/1.0/OWL-S.HTML, December 2003.

[PHH04] Patel-Schneider P. F., Hayes, P., Horrocks, I., OWL Web Ontology Language Semantics and Abstract Syntax, HTTP://WWW.W3.ORG/TR/OWL-SEMANTICS/, February 2004.

[POA] The Protégé Owl API. Created and maintained by: Stanford Medical Informatics, HTTP://PROTEGE.STANFORD.EDU/PLUGINS/OWL/API/INDEX.HTML.

[PPL] Protégé Plug-in Library, HTTP://PROTEGE.STANFORD.EDU/DOWNLOAD/PLUGINS.HTML.

[Pro] The Protégé ontology editor and Acquisition system, Stanford Medical Informatics, HTTP://PROTEGE.STANFORD.EDU/.

[PsS93] Patel-Schneider P. F., Swartout, B., Description Logic Knowledge Representation System Specification from the KRSS Group of the ARPA Knowledge Sharing Effort, November 1993.

[RAC] Racer Systems, HTTP://WWW.RACER-SYSTEMS.COM/.

[RBG96] Rector, A. L., Bechofer, S., Goble, C., Horrocks, I., Nowlan, W., Solomon, W., The GRAIL Concept Modeling Language for Medical Terminology. In: Medical Informatics Group, Department of Computer Science, University of Manchester, 1996.

[Rey01] Reynolds, F., An RDF Framework for Resource Discovery, Proceedings of the Second International Workshop on the Semantic Web (SemWeb'2001), May 2001.

[RHI] Regenstrief Health Institute, HTTP://WWW.REGENSTRIEF.ORG/.

[Ros94] Rosenfield, R., Adaptive statistic language model, Ph.D. thesis, Carnegie Mellon University, 1994.

[RRM05] RacerPro Reference Manual, v1.8, April 2005.

[RSN+94] Rector A. L., Solomon W. D., Nowlan W. A., Rush T. W., A Terminology Server for Medical Language and Medical Information Systems", IMIA Proceedings, Geneva, May 1994.

[SaB87] Salton, G., Buckley, C., Text weighting approaches in automatic text retrieval, Cornell University Technical Report, pp. 87-881, 1987.

[SAC03] Solbrig H.R., Armbrust D. C., Chute C. G., The Open Terminology Services (OTS) project, AMIA Annu Symp Proc. 2003:1011.

[ShS04] Shreenath, R., Singh, M., Agent-Based Service Selection, Journal on Web Semantics (JWS), vol. 1, number 3, pp. 261-279, April 2004.

[SNO] SNOMED-CT, HTTP://WWW.SNOMED.ORG/SNOMEDCT/.

[SPK+97] Swartout, B., Patil, R., Knight, K., Russ, T., Toward Distributed Use of Large-Scale Ontologies Ontological Engineering, AAAI-97, Spring Symposium Series, pp. 138-148, 1997.

[SwWG] Semantic Web Working Group, HTTP://WWW.W3.ORG/2001/SW/.

[SWK+02] Sycara, K., Widoff, S., Klusch, M., Lu, J., "Larks: Dynamic Matchmaking Among Heterogeneous Software Agents in Cyberspace," ACM Portal, Source: Autonomous Agents and Multi-Agent Systems, v5, issue 2, pp. 173–203, June 2002.

[TBGc01] Trastour, D., Bartolini, C., Gonzalez-Castillo, J., "A Semantic Web Approach to Service Description for Matchmaking of Services", Journal of Semantic Web, sec. Services and Application, 2001.

[UKS] UMLS Knowledge Source, HTTP://WWW.NLM.NIH.GOV/RESEARCH/UMLS/UMLSDOC.HTML.

[UML] OMG Unified Modeling Language Specification (draft) Feb 2001, For further info see: HTTP://WWW.OMG.ORG/UML [USN], The UMLS Semantic Network, HTTP://SEMANTICNETWORK.NLM.NIH.GOV/.

[WHK97] Wahl, M., Howes, T., Kille, S., Lightweight Directory Access Protocol (v3), rfc2251, Network Working Group, December 1997.

Appendix A.

Method to start and initialize RacerPro server

1. Start the RacerPro server and load Ontology ‘A’ into RacerPro server either through command line options or using racer client (Protégé).
2. Load Ontology ‘B’ in racer-client such as protégé editor.
3. Configure Protégé editor to show RQL tab. This tab is used to query RacerPro.
4. Apply matchmaking algorithm: For concept in ‘B’ find if there is corresponding concept in ‘A’.

Command Statements	Purpose
owl-read-document	Load owl file in RacerPro server and generate t-box and a-box with name generations
Full-reset	Clears all Tboxes and Aboxes, perform complete reset of Server

Table A.1 Racer Commands

Appendix B.

OWL and DL Constructs

The table below shows the constructs in OWL used for relations and constructs in Description Logic Specification. More complete list of all possible type of relations in DL and its corresponding OWL syntax can be found at [HPH03].

DL predicates	Abstract	OWL DL Constructs
TOP	$\top$	OWL:Thing
BOTTOM	$\perp$	OWL:Nothing
Number		rdf:Datatype (XMLSchema#int)
Integer		XMLSchema#decimal
String		XMLSchema#string
(and $C_1 \ldots C_n$)	($C_1 \sqcap C_2 \sqcap \ldots C_n$)	OWL:intersectionOf
(or $C_1 \ldots C_n$)	($C_1 \cup C_2 \ldots C_n$)	OWL: unionOf
(not C)	($\neg$C)	OWL:complementOf
(all R C)	$\forall$ R:C	owl:allValuesFrom
(some R C)	$\exists$ R.C	owl:someValuesFrom
(at-least n R)	$\geq$ n R	owl:minCardinality
(at-most n R)	$\leq$ n R	owl:maxCardinality
(exactly n R)	= n R	owl:cardinality
(equal R1 R2)	R1= R2	owl:equivalentProperty
(not-equal R2 R2)	R1$\neq$ R2	owl:differentFrom
(subset R1 R2)	R1 $\subseteq$ R2	rdfs:subPropertyOf
(fillers R L1…Ln	{L1…Ln}	owl:oneOf

Table B.1 Full list of DL vs. OWL Constructs

Now we will provide complete list of OWL Constructs for class and roles in Table B.2.

owl:AllDifferent	owl:incompatibleWith	owl:unionOf
owl:allValuesFrom	owl:intersectionOf	owl:versionInfo
owl:AnnotationProperty	owl:InverseFunctionalProperty	
owl:backwardCompatibleWith	owl:inverseOf	
owl:cardinality	owl:maxCardinality	
owl:Class	owl:minCardinality	
owl:complementOf	owl:Nothing	
owl:DataRange	owl:ObjectProperty	
owl:DatatypeProperty	owl:oneOf	
owl:DeprecatedClass	owl:onProperty	
owl:DeprecatedProperty	owl:Ontology	
owl:differentFrom	owl:OntologyProperty	
owl:disjointWith	owl:priorVersion	
owl:distinctMembers	owl:Restriction	
owl:equivalentClass	owl:sameAs	
owl:equivalentProperty	owl:someValuesFrom	
owl:FunctionalProperty	owl:SymmetricProperty	
owl:hasValue	owl:Thing	
owl:imports	owl:TransitiveProperty	

Table B.2 Complete List of OWL Constructs

Appendix C.

SAMPLE ONTOLOGIES

Here we provide family ontology used for explaining matchmaking algorithm.

Figure C.1 Family.OWL

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rss="http://purl.org/rss/1.0/"
    xmlns="http://health.informatics.iupui.edu/ontology/matching/family.owl#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
   xml:base="http://health.informatics.iupui.edu/ontology/matching/family.owl">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="Child">
    <owl:equivalentClass>
      <owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
          <owl:Restriction>
            <owl:minCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
            >1</owl:minCardinality>
            <owl:onProperty>
              <owl:ObjectProperty rdf:ID="hasParent"/>
            </owl:onProperty>
          </owl:Restriction>
          <owl:Class rdf:ID="Person"/>
        </owl:intersectionOf>
      </owl:Class>
    </owl:equivalentClass>
  </owl:Class>
  <owl:Class rdf:about="#Person">
    <owl:equivalentClass>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:ID="Man"/>
          <owl:Class rdf:ID="Woman"/>
        </owl:unionOf>
      </owl:Class>
    </owl:equivalentClass>
```

```
</owl:Class>
<owl:Class rdf:ID="Relative">
  <owl:equivalentClass>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Child"/>
        <owl:Class rdf:ID="Parent"/>
        <owl:Class rdf:ID="Aunt"/>
        <owl:Class rdf:ID="Nephew"/>
        <owl:Class rdf:ID="Niece"/>
        <owl:Class rdf:ID="Uncle"/>
        <owl:Class rdf:ID="Sibling"/>
      </owl:unionOf>
    </owl:Class>
  </owl:equivalentClass>
  <rdfs:subClassOf rdf:resource="#Person"/>
</owl:Class>
<owl:Class rdf:ID="Father">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Parent"/>
        <owl:Class rdf:about="#Man"/>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
  <owl:sameAs>
    <owl:Class rdf:ID="Dad"/>
  </owl:sameAs>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Father is a Person having atleast 1 child and is a male.</rdfs:comment>
</owl:Class>
<owl:Class rdf:ID="Son">
  <owl:disjointWith>
    <owl:Class rdf:ID="Daughter"/>
  </owl:disjointWith>
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Man"/>
        <owl:Class rdf:about="#Child"/>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
```

```
<owl:Class rdf:about="#Daughter">
  <owl:disjointWith rdf:resource="#Son"/>
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Child"/>
        <owl:Class rdf:about="#Woman"/>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
<owl:Class rdf:ID="Gender">
  <owl:equivalentClass>
    <owl:Class>
      <owl:oneOf rdf:parseType="Collection">
        <Gender rdf:ID="Female"/>
        <Gender rdf:ID="Male"/>
      </owl:oneOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
<owl:Class rdf:ID="Mother">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Parent"/>
        <owl:Class rdf:about="#Woman"/>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
<owl:Class rdf:about="#Nephew">
  <owl:disjointWith>
    <owl:Class rdf:about="#Niece"/>
  </owl:disjointWith>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:hasValue rdf:resource="#Male"/>
      <owl:onProperty>
        <owl:FunctionalProperty rdf:ID="hasSex"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <owl:equivalentClass>
    <owl:Class>
```

```
        <owl:intersectionOf rdf:parseType="Collection">
          <owl:Class>
            <owl:unionOf rdf:parseType="Collection">
              <owl:Restriction>
                <owl:onProperty>
                  <owl:ObjectProperty rdf:ID="hasUncle"/>
                </owl:onProperty>
                <owl:minCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
                >1</owl:minCardinality>
              </owl:Restriction>
              <owl:Restriction>
                <owl:onProperty>
                  <owl:ObjectProperty rdf:ID="hasAunt"/>
                </owl:onProperty>
                <owl:minCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
                >1</owl:minCardinality>
              </owl:Restriction>
            </owl:unionOf>
          </owl:Class>
          <owl:Class rdf:about="#Man"/>
        </owl:intersectionOf>
      </owl:Class>
    </owl:equivalentClass>
    <rdfs:subClassOf rdf:resource="#Relative"/>
  </owl:Class>
  <owl:Class rdf:ID="Sister">
    <owl:equivalentClass>
      <owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#Sibling"/>
          <owl:Class rdf:about="#Woman"/>
        </owl:intersectionOf>
      </owl:Class>
    </owl:equivalentClass>
    <owl:disjointWith>
      <owl:Class rdf:ID="Brother"/>
    </owl:disjointWith>
  </owl:Class>
  <owl:Class rdf:about="#Sibling">
    <owl:equivalentClass>
      <owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
          <owl:Restriction>
            <owl:minCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
            >1</owl:minCardinality>
            <owl:onProperty>
              <owl:SymmetricProperty rdf:ID="hasSibling"/>
            </owl:onProperty>
          </owl:Restriction>
          <owl:Class rdf:about="#Person"/>
        </owl:intersectionOf>
      </owl:Class>
    </owl:equivalentClass>
  </owl:Class>
  <owl:Class rdf:about="#Woman">
    <owl:equivalentClass>
      <owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
          <owl:Restriction>
            <owl:onProperty>
              <owl:FunctionalProperty rdf:about="#hasSex"/>
            </owl:onProperty>
            <owl:hasValue rdf:resource="#Female"/>
          </owl:Restriction>
          <owl:Class rdf:about="#Person"/>
        </owl:intersectionOf>
      </owl:Class>
    </owl:equivalentClass>
  </owl:Class>
  <owl:Class rdf:about="#Niece">
    <owl:disjointWith rdf:resource="#Nephew"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:hasValue rdf:resource="#Female"/>
        <owl:onProperty>
          <owl:FunctionalProperty rdf:about="#hasSex"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>

    <owl:equivalentClass>
      <owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
          <owl:Class>
            <owl:unionOf rdf:parseType="Collection">
              <owl:Restriction>
                <owl:onProperty>
```

```
                <owl:ObjectProperty rdf:about="#hasUncle"/>
              </owl:onProperty>
              <owl:minCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
              >1</owl:minCardinality>
            </owl:Restriction>
            <owl:Restriction>
              <owl:onProperty>
                <owl:ObjectProperty rdf:about="#hasAunt"/>
              </owl:onProperty>
              <owl:minCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
              >1</owl:minCardinality>
            </owl:Restriction>
          </owl:unionOf>
        </owl:Class>
        <owl:Class rdf:about="#Woman"/>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
  <rdfs:subClassOf rdf:resource="#Relative"/>
</owl:Class>
<owl:Class rdf:about="#Brother">
  <owl:disjointWith rdf:resource="#Sister"/>
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Sibling"/>
        <owl:Class rdf:about="#Man"/>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
<owl:Class rdf:about="#Aunt">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Woman"/>
        <owl:Class>
          <owl:unionOf rdf:parseType="Collection">
            <owl:Restriction>
              <owl:minCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
              >1</owl:minCardinality>
              <owl:onProperty>
```

```
                <owl:ObjectProperty rdf:ID="hasNephew"/>
              </owl:onProperty>
            </owl:Restriction>
            <owl:Restriction>
              <owl:minCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
              >1</owl:minCardinality>
              <owl:onProperty>
                <owl:ObjectProperty rdf:ID="hasNiece"/>
              </owl:onProperty>
            </owl:Restriction>
          </owl:unionOf>
        </owl:Class>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
  <owl:disjointWith>
    <owl:Class rdf:about="#Uncle"/>
  </owl:disjointWith>
  <rdfs:subClassOf rdf:resource="#Relative"/>
</owl:Class>
<owl:Class rdf:about="#Dad">
  <owl:sameAs rdf:resource="#Father"/>
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Man"/>
        <owl:Class rdf:about="#Parent"/>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
<owl:Class rdf:about="#Parent">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Restriction>
          <owl:minCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
          >1</owl:minCardinality>
          <owl:onProperty>
            <owl:ObjectProperty rdf:ID="hasChild"/>
          </owl:onProperty>
        </owl:Restriction>
        <owl:Class rdf:about="#Person"/>
        </owl:intersectionOf>
      </owl:Class>
    </owl:equivalentClass>
  </owl:Class>
  <owl:Class rdf:about="#Uncle">
    <owl:equivalentClass>
      <owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#Man"/>
          <owl:Class>
            <owl:unionOf rdf:parseType="Collection">
              <owl:Restriction>
                <owl:minCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
                >1</owl:minCardinality>
                <owl:onProperty>
                  <owl:ObjectProperty rdf:about="#hasNephew"/>
                </owl:onProperty>
              </owl:Restriction>
              <owl:Restriction>
                <owl:minCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
                >1</owl:minCardinality>
                <owl:onProperty>
                  <owl:ObjectProperty rdf:about="#hasNiece"/>
                </owl:onProperty>
              </owl:Restriction>
            </owl:unionOf>
          </owl:Class>
        </owl:intersectionOf>
      </owl:Class>
    </owl:equivalentClass>
    <owl:disjointWith rdf:resource="#Aunt"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:FunctionalProperty rdf:about="#hasSex"/>
        </owl:onProperty>
        <owl:hasValue rdf:resource="#Male"/>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="#Relative"/>
  </owl:Class>
  <owl:Class rdf:about="#Man">
    <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Restriction>
          <owl:hasValue rdf:resource="#Male"/>
          <owl:onProperty>
            <owl:FunctionalProperty rdf:about="#hasSex"/>
          </owl:onProperty>
        </owl:Restriction>
        <owl:Class rdf:about="#Person"/>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
<owl:ObjectProperty rdf:ID="hasSon">
  <rdfs:subPropertyOf>
    <owl:ObjectProperty rdf:about="#hasChild"/>
  </rdfs:subPropertyOf>
  <rdfs:domain rdf:resource="#Person"/>
  <rdfs:range rdf:resource="#Man"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasConsort">
  <rdfs:domain rdf:resource="#Person"/>
  <rdfs:range rdf:resource="#Person"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#SymmetricProperty"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasNephew">
  <rdfs:domain rdf:resource="#Person"/>
  <rdfs:range rdf:resource="#Man"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasParent">
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#hasChild"/>
  </owl:inverseOf>
  <rdfs:domain rdf:resource="#Person"/>
  <rdfs:range rdf:resource="#Person"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasBrother">
  <rdfs:domain rdf:resource="#Person"/>
  <rdfs:subPropertyOf>
    <owl:SymmetricProperty rdf:about="#hasSibling"/>
  </rdfs:subPropertyOf>
  <rdfs:range rdf:resource="#Man"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasFather">
```

```
  <rdfs:domain rdf:resource="#Person"/>
  <rdfs:subPropertyOf rdf:resource="#hasParent"/>
  <rdfs:range rdf:resource="#Man"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasDaughter">
  <rdfs:domain rdf:resource="#Person"/>
  <rdfs:subPropertyOf>
    <owl:ObjectProperty rdf:about="#hasChild"/>
  </rdfs:subPropertyOf>
  <rdfs:range rdf:resource="#Woman"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasChild">
  <rdfs:range rdf:resource="#Person"/>
  <owl:inverseOf rdf:resource="#hasParent"/>
  <rdfs:domain rdf:resource="#Person"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasSister">
  <rdfs:domain rdf:resource="#Person"/>
  <rdfs:range rdf:resource="#Woman"/>
  <rdfs:subPropertyOf>
    <owl:SymmetricProperty rdf:about="#hasSibling"/>
  </rdfs:subPropertyOf>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasNiece">
  <rdfs:domain rdf:resource="#Person"/>
  <rdfs:range rdf:resource="#Woman"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasAunt">
  <rdfs:range rdf:resource="#Woman"/>
  <rdfs:domain rdf:resource="#Person"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasUncle">
  <rdfs:domain rdf:resource="#Person"/>
  <rdfs:range rdf:resource="#Man"/>
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:ID="name">
  <rdfs:domain rdf:resource="#Person"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
<owl:SymmetricProperty rdf:about="#hasSibling">
  <rdfs:range rdf:resource="#Person"/>
  <rdfs:domain rdf:resource="#Person"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
```

```
</owl:SymmetricProperty>
<owl:FunctionalProperty rdf:about="#hasSex">
  <rdfs:domain rdf:resource="#Person"/>
  <rdfs:range rdf:resource="#Gender"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="hasMother">
  <rdfs:domain rdf:resource="#Person"/>
  <rdfs:range rdf:resource="#Woman"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <rdfs:subPropertyOf rdf:resource="#hasParent"/>
</owl:FunctionalProperty>
<Woman rdf:ID="F10">
  <name rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Whitney</name>
  <hasSex rdf:resource="#Female"/>
</Woman>
<Woman rdf:ID="F02">
  <hasParent>
    <Man rdf:ID="M01">
      <hasConsort>
        <Woman rdf:ID="F01">
          <name rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
          >Mary</name>
          <hasSex rdf:resource="#Female"/>
        </Woman>
      </hasConsort>
      <hasChild>
        <Woman rdf:ID="F03">
          <name rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
          >Elizabeth</name>
          <hasParent rdf:resource="#M01"/>
          <hasSex rdf:resource="#Female"/>
        </Woman>
      </hasChild>
      <hasSex rdf:resource="#Male"/>
      <hasChild rdf:resource="#F02"/>
      <name rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >Bill</name>
      <hasChild>
        <Man rdf:ID="M02">
          <name rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
          >Adam</name>
          <hasSex rdf:resource="#Male"/>
          <hasChild>
  <Man rdf:ID="M03">
    <hasParent rdf:resource="#M02"/>
    <hasChild>
      <Woman rdf:ID="F09">
        <hasSex rdf:resource="#Female"/>
        <hasParent rdf:resource="#M03"/>
        <name rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
        >Surrey</name>
      </Woman>
    </hasChild>
    <hasSex rdf:resource="#Male"/>
    <name rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >George</name>
    <hasConsort>
      <Woman rdf:ID="F08">
        <name rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
        >Emily</name>
        <hasSex rdf:resource="#Female"/>
      </Woman>
    </hasConsort>
  </Man>
</hasChild>
<hasParent rdf:resource="#M01"/>
<hasChild>
  <Woman rdf:ID="F05">
    <name rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Anna</name>
    <hasSex rdf:resource="#Female"/>
    <hasParent rdf:resource="#M02"/>
  </Woman>
</hasChild>
<hasConsort>
  <Woman rdf:ID="F04">
    <name rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Marilyn</name>
    <hasSex rdf:resource="#Female"/>
  </Woman>
</hasConsort>
<hasChild>
  <Man rdf:ID="M05">
    <name rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Michael</name>
    <hasParent rdf:resource="#M02"/>
    <hasSex rdf:resource="#Male"/>
  </Man>
```

```
          </hasChild>
        </Man>
      </hasChild>
    </Man>
  </hasParent>
  <hasSex rdf:resource="#Female"/>
  <name rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Catherine</name>
</Woman>
<Man rdf:ID="M10">
  <name rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Jack</name>
  <hasParent>
    <Man rdf:ID="M08">
      <name rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >Jimmy</name>
      <hasConsort>
        <Woman rdf:ID="F06">
          <hasSex rdf:resource="#Female"/>
          <name rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
          >Eva</name>
          <hasParent>
            <Man rdf:ID="M04">
              <name rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
              >Phillipe</name>
              <hasChild rdf:resource="#F06"/>
              <hasChild>
                <Man rdf:ID="M06">
                  <hasSex rdf:resource="#Male"/>
                  <hasChild>
                    <Man rdf:ID="M09">
                      <name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
                      >Ronald</name>
                      <hasSex rdf:resource="#Male"/>
                      <hasParent rdf:resource="#M06"/>
                    </Man>
                  </hasChild>
                  <hasParent rdf:resource="#M04"/>
                  <hasConsort rdf:resource="#F10"/>
                  <name rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
                  >Tom</name>
                </Man>
              </hasChild>
              <hasConsort rdf:resource="#F03"/>
              <hasParent>
                <Man rdf:ID="M07">
                  <hasSex rdf:resource="#Male"/>
                  <name rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
                  >John</name>
                  <hasConsort>
                    <Woman rdf:ID="F07">
                      <name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
                      >Audrey</name>
                      <hasSex rdf:resource="#Female"/>
                    </Woman>
                  </hasConsort>
                  <hasChild rdf:resource="#M04"/>
                </Man>
              </hasParent>
              <hasSex rdf:resource="#Male"/>
            </Man>
          </hasParent>
        </Woman>
      </hasConsort>
      <hasChild rdf:resource="#M10"/>
      <hasSex rdf:resource="#Male"/>
    </Man>
  </hasParent>
  <hasSex rdf:resource="#Male"/>
 </Man>
</rdf:RDF>
```

Figure C.2 **Generations.OWL**

```
<?xml version="1.0"?>
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
   xmlns="http://health.informatics.iupui.edu/ontology/matching/generations.owl#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:owl="http://www.w3.org/2002/07/owl#"

xml:base="http://health.informatics.iupui.edu/ontology/matching/generations.owl"
>
 <owl:Ontology rdf:about="">
   <owl:versionInfo rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
```

```
  >An example ontology created by Matthew Horridge</owl:versionInfo>
</owl:Ontology>
<owl:Class rdf:ID="Offspring">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:ID="Person"/>
        <owl:Restriction>
          <owl:someValuesFrom rdf:resource="#Person"/>
          <owl:onProperty>
            <owl:ObjectProperty rdf:ID="hasParent"/>
          </owl:onProperty>
        </owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
<owl:Class rdf:ID="Daughter">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Restriction>
          <owl:onProperty>
            <owl:ObjectProperty rdf:about="#hasParent"/>
          </owl:onProperty>
          <owl:someValuesFrom rdf:resource="#Person"/>
        </owl:Restriction>
        <owl:Restriction>
          <owl:onProperty>
            <owl:FunctionalProperty rdf:ID="hasSex"/>
          </owl:onProperty>
          <owl:hasValue>
            <Sex rdf:ID="FemaleSex"/>
          </owl:hasValue>
        </owl:Restriction>
        <owl:Class rdf:about="#Person"/>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
<owl:Class rdf:ID="Male">
  <owl:equivalentClass>
    <owl:Restriction>
      <owl:hasValue>
        <Sex rdf:ID="MaleSex"/>
      </owl:hasValue>
      <owl:onProperty>
        <owl:FunctionalProperty rdf:about="#hasSex"/>
      </owl:onProperty>
    </owl:Restriction>
  </owl:equivalentClass>
</owl:Class>
<owl:Class rdf:ID="Grandmother">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Person"/>
        <owl:Restriction>
          <owl:someValuesFrom>
            <owl:Class>
              <owl:intersectionOf rdf:parseType="Collection">
                <owl:Class rdf:about="#Person"/>
                <owl:Restriction>
                  <owl:someValuesFrom rdf:resource="#Person"/>
                  <owl:onProperty>
                    <owl:ObjectProperty rdf:ID="hasChild"/>
                  </owl:onProperty>
                </owl:Restriction>
              </owl:intersectionOf>
            </owl:Class>
          </owl:someValuesFrom>
          <owl:onProperty>
            <owl:ObjectProperty rdf:about="#hasChild"/>
          </owl:onProperty>
        </owl:Restriction>
        <owl:Restriction>
          <owl:onProperty>
            <owl:FunctionalProperty rdf:about="#hasSex"/>
          </owl:onProperty>
          <owl:hasValue rdf:resource="#FemaleSex"/>
        </owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
<owl:Class rdf:ID="Mater">
  <owl:versionInfo rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >TODO: Find command in nRQL to get the owl:sameAs
class</owl:versionInfo>
  <owl:sameAs>
```

```
      <owl:Class rdf:ID="Mother"/>
    </owl:sameAs>
  </owl:Class>
  <owl:Class rdf:ID="Sex">
    <owl:equivalentClass>
      <owl:Class>
        <owl:oneOf rdf:parseType="Collection">
          <Sex rdf:about="#MaleSex"/>
          <Sex rdf:about="#FemaleSex"/>
        </owl:oneOf>
      </owl:Class>
    </owl:equivalentClass>
  </owl:Class>
  <owl:Class rdf:ID="Father">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Father is a male and also person having a child who is a person
too.</rdfs:comment>
    <rdfs:label>Daddy</rdfs:label>
    <owl:equivalentClass>
      <owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#Person"/>
          <owl:Restriction>
            <owl:hasValue rdf:resource="#MaleSex"/>
            <owl:onProperty>
              <owl:FunctionalProperty rdf:about="#hasSex"/>
            </owl:onProperty>
          </owl:Restriction>
          <owl:Restriction>
            <owl:someValuesFrom rdf:resource="#Person"/>
            <owl:onProperty>
              <owl:ObjectProperty rdf:about="#hasChild"/>
            </owl:onProperty>
          </owl:Restriction>
        </owl:intersectionOf>
      </owl:Class>
    </owl:equivalentClass>
  </owl:Class>
  <owl:Class rdf:about="#Mother">
    <owl:equivalentClass>
      <owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#Person"/>
          <owl:Restriction>
            <owl:someValuesFrom rdf:resource="#Person"/>
            <owl:onProperty>
              <owl:ObjectProperty rdf:about="#hasChild"/>
            </owl:onProperty>
          </owl:Restriction>
          <owl:Restriction>
            <owl:hasValue rdf:resource="#FemaleSex"/>
            <owl:onProperty>
              <owl:FunctionalProperty rdf:about="#hasSex"/>
            </owl:onProperty>
          </owl:Restriction>
        </owl:intersectionOf>
      </owl:Class>
    </owl:equivalentClass>
    <owl:sameAs rdf:resource="#Mater"/>
  </owl:Class>
  <owl:Class rdf:ID="Grandfather">
    <owl:equivalentClass>
      <owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#Person"/>
          <owl:Restriction>
            <owl:onProperty>
              <owl:ObjectProperty rdf:about="#hasChild"/>
            </owl:onProperty>
            <owl:someValuesFrom>
              <owl:Class>
                <owl:intersectionOf rdf:parseType="Collection">
                  <owl:Class rdf:about="#Person"/>
                  <owl:Restriction>
                    <owl:someValuesFrom rdf:resource="#Person"/>
                    <owl:onProperty>
                      <owl:ObjectProperty rdf:about="#hasChild"/>
                    </owl:onProperty>
                  </owl:Restriction>
                </owl:intersectionOf>
              </owl:Class>
            </owl:someValuesFrom>
          </owl:Restriction>
          <owl:Restriction>
            <owl:onProperty>
              <owl:FunctionalProperty rdf:about="#hasSex"/>
            </owl:onProperty>
            <owl:hasValue rdf:resource="#MaleSex"/>
          </owl:Restriction>
        </owl:intersectionOf>
      </owl:Class>
    </owl:equivalentClass>
  </owl:Class>
  <owl:Class rdf:ID="Sister">
    <owl:equivalentClass>
      <owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#Person"/>
          <owl:Restriction>
            <owl:someValuesFrom rdf:resource="#Person"/>
            <owl:onProperty>
              <owl:SymmetricProperty rdf:ID="hasSibling"/>
            </owl:onProperty>
          </owl:Restriction>
          <owl:Restriction>
            <owl:onProperty>
              <owl:FunctionalProperty rdf:about="#hasSex"/>
            </owl:onProperty>
            <owl:hasValue rdf:resource="#FemaleSex"/>
          </owl:Restriction>
        </owl:intersectionOf>
      </owl:Class>
    </owl:equivalentClass>
  </owl:Class>
  <owl:Class rdf:ID="Brother">
    <owl:equivalentClass>
      <owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#Person"/>
          <owl:Class>
            <owl:intersectionOf rdf:parseType="Collection">
              <owl:Restriction>
                <owl:onProperty>
                  <owl:SymmetricProperty rdf:about="#hasSibling"/>
                </owl:onProperty>
                <owl:someValuesFrom rdf:resource="#Person"/>
              </owl:Restriction>
              <owl:Restriction>
                <owl:onProperty>
                  <owl:FunctionalProperty rdf:about="#hasSex"/>
                </owl:onProperty>
                <owl:hasValue rdf:resource="#MaleSex"/>
              </owl:Restriction>
            </owl:intersectionOf>
          </owl:Class>
        </owl:intersectionOf>
      </owl:Class>
    </owl:equivalentClass>
  </owl:Class>
  <owl:Class rdf:ID="Woman">
    <owl:equivalentClass>
      <owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#Person"/>
          <owl:Restriction>
            <owl:onProperty>
              <owl:FunctionalProperty rdf:about="#hasSex"/>
            </owl:onProperty>
            <owl:hasValue rdf:resource="#FemaleSex"/>
          </owl:Restriction>
        </owl:intersectionOf>
      </owl:Class>
    </owl:equivalentClass>
  </owl:Class>
  <owl:Class rdf:ID="Man">
    <owl:equivalentClass>
      <owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#Person"/>
          <owl:Restriction>
            <owl:hasValue rdf:resource="#MaleSex"/>
            <owl:onProperty>
              <owl:FunctionalProperty rdf:about="#hasSex"/>
            </owl:onProperty>
          </owl:Restriction>
        </owl:intersectionOf>
      </owl:Class>
    </owl:equivalentClass>
  </owl:Class>
  <owl:Class rdf:ID="Parent">
    <owl:equivalentClass>
      <owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#Person"/>
          <owl:Restriction>
            <owl:someValuesFrom rdf:resource="#Person"/>
            <owl:onProperty>
              <owl:ObjectProperty rdf:about="#hasChild"/>
            </owl:onProperty>
          </owl:Restriction>
        </owl:intersectionOf>
      </owl:Class>
    </owl:equivalentClass>
  </owl:Class>
  <owl:Class rdf:ID="Mummy">
    <owl:equivalentClass>
      <owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
          <owl:Restriction>
            <owl:someValuesFrom rdf:resource="#Person"/>
            <owl:onProperty>
              <owl:ObjectProperty rdf:about="#hasChild"/>
            </owl:onProperty>
          </owl:Restriction>
          <owl:Restriction>
            <owl:onProperty>
              <owl:FunctionalProperty rdf:about="#hasSex"/>
            </owl:onProperty>
            <owl:hasValue rdf:resource="#FemaleSex"/>
          </owl:Restriction>
          <owl:Class rdf:about="#Person"/>
        </owl:intersectionOf>
      </owl:Class>
    </owl:equivalentClass>
  </owl:Class>
  <owl:Class rdf:ID="Son">
    <owl:equivalentClass>
      <owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#Person"/>
          <owl:Restriction>
            <owl:onProperty>
              <owl:ObjectProperty rdf:about="#hasParent"/>
            </owl:onProperty>
            <owl:someValuesFrom rdf:resource="#Person"/>
          </owl:Restriction>
          <owl:Restriction>
            <owl:onProperty>
              <owl:FunctionalProperty rdf:about="#hasSex"/>
            </owl:onProperty>
            <owl:hasValue rdf:resource="#MaleSex"/>
          </owl:Restriction>
        </owl:intersectionOf>
      </owl:Class>
    </owl:equivalentClass>
```

```
</owl:Class>
<owl:Class rdf:ID="Sibling">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Person"/>
        <owl:Restriction>
          <owl:onProperty>
            <owl:SymmetricProperty rdf:about="#hasSibling"/>
          </owl:onProperty>
          <owl:someValuesFrom rdf:resource="#Person"/>
        </owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
<owl:Class rdf:ID="Grandparent">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Person"/>
        <owl:Restriction>
          <owl:onProperty>
            <owl:ObjectProperty rdf:about="#hasChild"/>
          </owl:onProperty>
          <owl:someValuesFrom>
            <owl:Class>
              <owl:intersectionOf rdf:parseType="Collection">
                <owl:Class rdf:about="#Person"/>
                <owl:Restriction>
                  <owl:someValuesFrom rdf:resource="#Person"/>
                  <owl:onProperty>
                    <owl:ObjectProperty rdf:about="#hasChild"/>
                  </owl:onProperty>
                </owl:Restriction>
              </owl:intersectionOf>
            </owl:Class>
          </owl:someValuesFrom>
        </owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
<owl:Class rdf:ID="Female">
  <owl:equivalentClass>
```

```
    <owl:Restriction>
      <owl:onProperty>
        <owl:FunctionalProperty rdf:about="#hasSex"/>
      </owl:onProperty>
      <owl:hasValue rdf:resource="#FemaleSex"/>
    </owl:Restriction>
  </owl:equivalentClass>
</owl:Class>
<owl:ObjectProperty rdf:about="#hasParent">
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#hasChild"/>
  </owl:inverseOf>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasChild">
  <owl:inverseOf rdf:resource="#hasParent"/>
</owl:ObjectProperty>
<owl:SymmetricProperty rdf:about="#hasSibling">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
</owl:SymmetricProperty>
<owl:FunctionalProperty rdf:about="#hasSex">
  <rdfs:range rdf:resource="#Sex"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
</owl:FunctionalProperty>
<Person rdf:ID="Gemma">
  <hasSex rdf:resource="#FemaleSex"/>
</Person>
<Person rdf:ID="Peter">
  <hasSex rdf:resource="#MaleSex"/>
  <hasParent>
    <Person rdf:ID="William">
      <hasSex rdf:resource="#MaleSex"/>
      <hasChild rdf:resource="#Peter"/>
    </Person>
  </hasParent>
  <hasChild>
    <Person rdf:ID="Matt">
      <hasSibling rdf:resource="#Gemma"/>
      <owl:sameAs>
        <Person rdf:ID="Matthew">
          <owl:sameAs rdf:resource="#Matt"/>

        </Person>
      </owl:sameAs>
      <hasParent rdf:resource="#Peter"/>
      <hasSex rdf:resource="#MaleSex"/>
    </Person>
  </hasChild>
 </Person>
</rdf:RDF>
```

We are including only some part of LOINC structure below, as LOINC.OWL is too big to fit here. The class hierarchy below is part of LOINC.OWL file.

Figure C.3 **LOINC.OWL**

```
<?xml version="1.0"?>
<rdf:RDF
   xmlns="http://health.informatics.iupui.edu/ontology/LOINCOWL.owl#"
   xmlns:j.0="http://protege.stanford.edu/plugins/owl/protege#"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
 xml:base="http://health.informatics.iupui.edu/ontology/LOINCOWL.owl">
 <owl:Ontology rdf:about="">
   <owl:imports rdf:resource="http://protege.stanford.edu/plugins/owl/protege"/>
 </owl:Ontology>
 <owl:Class rdf:ID="LOINC_NM_SYS">
   <rdfs:subClassOf>
     <owl:Class rdf:ID="LOINC_NM"/>
   </rdfs:subClassOf>
   <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
   >Fourth subpart from six subparts of fully specified name of test result or
clinical observation. This provides information about sample or system
type.</rdfs:comment>
 </owl:Class>
 <owl:Class rdf:ID="LOINC_SV_QUE_SRC">
   <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
   >Exact name of the survey instrument and the item/question
number.</rdfs:comment>
   <rdfs:subClassOf>
     <owl:Class rdf:ID="LOINC"/>
   </rdfs:subClassOf>
 </owl:Class>
 <owl:Class rdf:ID="SYS_CD">
   <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
   >System type codes used in fully specified name.</rdfs:comment>
 </owl:Class>
 <owl:Class rdf:ID="LABORATORYCLASS">
```

```
    <rdfs:subClassOf>
      <owl:Class rdf:ID="LOINC_CLS"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="LOINC_KB"/>
  <owl:Class rdf:ID="LOINC_MOL_ID">
    <rdfs:subClassOf rdf:resource="#LOINC"/>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Molecular structure ID, usually CAS number.</rdfs:comment>
  </owl:Class>
  <owl:Class rdf:ID="LOINC_CDC_CD">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Code from CDC Complexity file that maps laboratory tests to the instruments
used to perform them. These codes are at the analyte level, not the test
instrument level.</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#LOINC"/>
  </owl:Class>
  <owl:Class rdf:ID="ATTACHMENTCLASS">
    <rdfs:subClassOf>
      <owl:Class rdf:about="#LOINC_CLS"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="TA_TAM">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Time aspect modifier is optional subpart of the time component. It allows an
indication of some sub-selection of the measures taken over the defined period
of time.</rdfs:comment>
    <rdfs:subClassOf>
      <owl:Class rdf:ID="LOINC_NM_TA"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="LOINC_FLA">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Regression equation details for many OB.US calculated
terms.</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#LOINC"/>
  </owl:Class>
  <owl:Class rdf:ID="LOINC_MPH_CD">
    <rdfs:subClassOf rdf:resource="#LOINC"/>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >MetPath Code for future use.</rdfs:comment>
  </owl:Class>
  <owl:Class rdf:ID="PPT">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Properties</rdfs:comment>
  </owl:Class>
  <owl:Class rdf:ID="LOINC_PNL_ELE">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >List of individual tests that comprise a panel.</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#LOINC"/>
  </owl:Class>
  <owl:Class rdf:ID="LOINC_SET_RT">
    <rdfs:subClassOf rdf:resource="#LOINC"/>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Used for claims attachments. </rdfs:comment>
  </owl:Class>
  <owl:Class rdf:ID="LOINC_REF">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Contains references to medical literature, product announcements, or other
written sources of information on the test or measurement described by the
LOINC record.</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#LOINC"/>
  </owl:Class>
  <owl:Class rdf:ID="CH_POST">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >The challenge that is referred at time of testing component or
analyte.</rdfs:comment>
    <rdfs:subClassOf>
      <owl:Class rdf:ID="CPT_CH_POST"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="LOINC_SMD_CD">
    <rdfs:subClassOf rdf:resource="#LOINC"/>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >SNOMED Code for future use.</rdfs:comment>
  </owl:Class>
```

Note: UMLS ontology is not listed here because of its large size. It can be found at http://health.informatics.iupui.edu/ontology/UMLSOWL.owl

LIST OF REFERENCES

APPENDICES