USING SEMANTIC WEB SERVICES FOR DATA INTEGRATION IN BANKING DOMAIN

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

BY

ÇAĞLAR OKAT

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

MAY 2010

Approval of the thesis:

USING SEMANTIC WEB SERVICES FOR DATA INTEGRATION IN BANKING DOMAIN

submitted by **ÇAĞLAR OKAT** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen
Dean, Graduate School of **Natural and Applied Sciences** ______________________

Prof. Dr. Adnan Yazıcı ______________________
Head of Department, **Computer Engineering**

Assoc. Prof. Dr. Ali H. Doğru ______________________
Supervisor, **Computer Engineering Dept., METU**

Examining Committee Members:

Asst. Prof. Dr. Pınar Şenkul ______________________
Computer Engineering Dept., METU

Assoc. Prof. Dr. Ali H. Doğru ______________________
Computer Engineering Dept., METU

Asst. Prof. Dr. Tolga Can ______________________
Computer Engineering Dept., METU

Celal Kavuklu (M.Sc.) ______________________
Akbank T.A.Ş.

Hacer Yalım (M.Sc.) ______________________
Computer Engineer

Date: 05.05.2010

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name : Çağlar Okat

Signature :

ABSTRACT

USING SEMANTIC WEB SERVICES FOR DATA INTEGRATION IN BANKING DOMAIN

Okat, Çağlar

M. S., Department of Computer Engineering

Supervisor: Assoc. Prof. Dr. Ali Hikmet Doğru

May 2010, 94 pages

A semantic model oriented transformation mechanism is developed for the centralization of intra-enterprise data integration. Such a mechanism is especially crucial in the banking domain which is selected in this study. A new domain ontology is constructed to provide basis for annotations. A bottom-up approach is preferred for semantic annotations to utilize existing web service definitions. Transformations between syntactic web service XML responses and semantic model concepts are defined in transformation files. Transformation files are stored and executed in a separate central transformation repository to enhance abstraction and reusability. An RDF-Store is implemented to store transformed RDF data. Inference power of semantic model is exposed by executing semantic queries in the RDF-Store.

Keywords: Semantic Web, Ontology, Semantic Web Service, Enterprise Application Integration, SAWSDL Specification.

ÖZ

BANKACILIK ALANINDA VERİ ENTEGRASYONU İÇİN ANLAMBİLİMSEL WEB HİZMETLERİNİN KULLANILMASI

Okat, Çağlar

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi: Doç. Dr. Ali Hikmet Doğru

Mayıs 2010, 94 sayfa

Kurum içi veri entegrasyonunun merkezileştirilmesi için anlambilimsel model odaklı bir dönüşüm mekanizması geliştirildi. Böyle bir mekanizma bankacılık alanında özellikle önemli olduğundan dolayı, bu çalışma için bankacılık alanı seçildi. İşaretlemelere temel oluşturması için yeni bir alan ontolojisi oluşturuldu. Mevcut web hizmeti tanımlarından faydalanabilmek için aşağıdan-yukarıya bir yaklaşım tercih edildi. Sözdizimsel web hizmetlerinin XML cevapları ile anlambilimsel model kavramları arasındaki dönüşümler dönüşüm dosyalarında tanımlandı. Soyutlama ve yeniden kullanılabilirliği artırmak için dönüşüm dosyaları ayrı bir merkezi dönüşüm havuzunda saklandı ve çalıştırıldı. Dönüştürülmüş RDF verisini saklamak için bir RDF-Deposu geliştirildi. Anlambilimsel modelin çıkarım yapma gücünü ortaya çıkarmak için RDF-Deposu üzerinde anlambilimsel sorgular çalıştırıldı.

Anahtar Kelimeler: Anlambilimsel Ağ, Ontoloji, Anlambilimsel Web Hizmetleri, Kurumsal Uygulama Entegrasyonu, SAWSDL.

To My Wife and Family

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Assoc. Prof. Dr. Ali Hikmet Doğru, for his guidance and encouragement throughout the research. I also would like to thank to my wife and family for their great support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In recent years, enterprises have increased the volume of interactions they have with other enterprises to catch up with the globalizing world markets. The business requirements and demands become more varied and complex. This necessitates the departments of the enterprises to be in tight relation with each other. Providing immediate and reliable responses to the rapidly changing world conditions become more important.

One of the main purposes of software systems used in enterprises is to get aligned with the changing business needs. Many different types of applications with different technologies can take place inside an enterprise system. To enable them better for critical software architecture parameters like adaptability, reliability and scalability, well-designed integration efforts are needed.

Recently, Web service enabled Service Oriented Architectures are gaining popularity for Enterprise Application Integration. Web services offer a flexible integration environment for the enterprise architecture since they abstract the background implementation details from the calling applications. However they offer only syntactic definitions for their service and input/output definitions. This increases the complexity to use the web services for designing complex business processes and performing complex data transformations between source systems.

Semantic web technologies are emerging in recent years. Semantic web provides semantic models, annotation frameworks and enhancements for service discovery and composition. These technologies are also applied to standard web services to express their operations and exchange structures in semantic concepts. Semantics makes web service discovery, composition and integration easier and more reliable.

1.1. Scope of this work

Purpose of this thesis is to semantically annotate standard web services and use them to perform enterprise information integration in a more reliable and elegant way.

A domain ontology is developed using the Protégé Ontology Editor. Based on this ontology, web services are annotated according to Semantic Annotations for WSDL (SAWSDL) specification by utilizing the Radiant Annotation Tool.

Semantic data transformations are defined in XSLT language which are stored and executed in a *Central Transformation Repository*.

An RDF-Store is utilized as the semantic storage unit for resulting semantic data so that inference mechanisms can be applied to extract information that is scattered through the enterprise systems and currently implicit.

Data integration is managed through a *Semantic Data Integrator Process*, which is responsible for collecting data from source systems, performing transformations using *transformation service* and persisting semantic data into the RDF-Store.

1.2. Organization of the Thesis

Beyond this introductory chapter, the thesis is organized as follows: In Chapter 2, necessary background on Enterprise Application Integration and Semantic Web Technologies are included. Chapter 3 describes the usage of semantic web services for Enterprise Application Integration. In Chapter 4, related work in the usage of semantic web services for Enterprise Application Integration is presented. Chapter 5 presents the design, implementation and test results of proposed semantic web services based data integration system. Chapter 6 provides conclusions and further work for this study.

CHAPTER 2

BACKGROUND

2.1. Enterprise Application Integration

2.1.1. Enterprise Application Integration Definition

Globalization forced enterprises to do business with firms from all around the world. Merger and acquisitions between corporations increased to adapt this new competitive environment. Information exchange of an enterprise was limited to suppliers and clients in the old times which now include partners, government, subsidiaries, and agencies [1]. Minimizing response times for new business requirements become more important in the rapidly changing world, which requires designing adaptable software systems. Technological improvements lead to wider usage of software applications and increase in data storage capabilities. These factors enlarged inter-enterprise communication and messaging which lead to development of new protocols and message exchange networks (RosettaNet, SWIFT). Information exchange between various systems of different companies is generally referred as inter-EAI or Business to Business (B2B) integration [29].

Large enterprises have many numbers of departments working in tight coordination with each other to fulfil complex business demands. Each department may need different types of software applications with different characteristics. These applications are designed at different time periods, by different teams using different technologies. They generally focus on doing their own job without caring for integration. Types of applications in an enterprise information system can be grouped as batch applications, transactional applications, client/server applications, web applications, real-time applications and software packages. Differentiating features of those application types are format of events and data they publish, volume of events they can deal with and their data exchange capacity [1]. Information exchange between various systems inside the same company is generally referred as intra-EAI or Application to Application (A2A) integration [29].

As a result, enterprise information systems suffer from heterogeneity in hardware platforms, having multiple interfaces for applications and incapability in exchanging information.

Enterprise Application Integration (EAI) is methods/tools/services used to bring heterogeneous applications into communication as part of a distributed enterprise [1].

In this definition, communication is defined as exchanging messages without knowing internal structure of each other but obeying their respective constraints. Communication inside an application is not within the interest of EAI. In addition, communication between similar applications (the applications built with similar or same technologies) is not a part of EAI studies.

2.1.2. Layers in Enterprise Application Integration

Any integration architecture can be modelled based on three basic levels:

- *Transport and Connectivity.* This layer captures events or information generated by source applications and delivers them to the receiving applications.

- *Information Adaptation:* Transport and delivery is not enough to complete integration. Adaptation of events or information is necessary for the consumption of receiving applications. Determination of the recipient applications is also another task of this layer.

- *Business Process Automation:* Multi-step processes are generally required by enterprises to accomplish business demands. Complementary functions are used for integration between those processes.

2.1.2.1. Transport and Connectivity

2.1.2.1.1 Data and Event Transport

A multi-channel communication bus is required for information transportation. Such a system should be designed so that it does not give more privilege to any of the participating channels to prevent expensive conversions. Main channels of an enterprise can be listed as the following:

- *Database Management Systems (DBMS):* Main responsibility of database management systems is creation, maintenance and storage of data. In addition they also offer some basic mechanisms for data replication. The target of replication is generally another DBMS.

- *File Transfer:* File transfer is the oldest method used for data integration in enterprises. File transfer systems support different types of networks (TCP/IP, X.25, SNA etc.). Many different file exchange protocols exist (FTP, PeSIT and ETEBAC in banking, OFTP in automotive etc.). Basic functionalities such as data compression, continuation without restarting transfer after interruption and guarantee of delivery are expected from a file exchange protocol. File Transfer Protocol (FTP) is the most commonly utilized one which is relatively simpler but is not secure enough and poorly performing in heavy volume transfers. Management services are developed to offer secure and totally automated transfers.

- *Message Oriented Middleware (MOM):* Message Oriented Middleware allows application-to-application integration by exchanging events as messages. A *queue* is the basic transportation infrastructure in this system. If the applications are running on different machines, installation of queues to each machine offers more functionality. MOM functionality can be divided into services as transport services, internal services and management services. Transport services are required to support different types of networks, message grouping, guarantee of delivery and message compression. Internal services involve message persistence, various types of access to queues (FIFO, direct access etc.) and restart on interruption. Management services provide APIs for sending/receiving messages and logging for administrative purposes. Publication/subscription mechanisms are also adapted to MOM systems so that an application can put a message to queue without caring which receiving applications will use it.

- *Internet:* Application integration architectures benefits widely from the advantages of internet protocols. The main standards are HTTP, SMTP and FTP and many complex protocols are built upon them. *HTTP* is developed by W3C (World Wide Web Consortium) [40] as a client/server protocol aimed for exchange of any type of data. Today, HTTP is primarily used in the implementation of web applications that manage exchanges of HTML (Hypertext Markup Language) pages through browsers. *SMTP* is a protocol used to send messages between servers over the Internet. A messaging client and a suitable protocol (Post Office Protocol 3 (POP3) or Internet Message Access Protocol (IMAP)) are used in the receiving server to retrieve the message. *SOAP* (Simple Object Access Protocol) is a protocol for data exchange in decentralized, distributed systems which utilizes XML standard. There are three parts of a SOAP message: an envelope section to define the messages and how to process them, a set of coding rules to express application-defined data types, a convention to represent remote procedure calls and their responses. SOAP specification only explains how to implement it on HTTP, but it is widely used with various other standards. SOAP can be used as a foundation layer for web service protocol stack [1].

2.1.2.1.2 Connectivity

The only responsibility of multi-channel bus communication is transportation of data and events to the receiving application with the original format of the source application. The format of the information is frequently required to be changed for the receiving applications understanding. Adapters (also known as connectors) are separate software units that are used for this process. Adapters can be divided into two different types:

- *Light (Technical) Adapters:* Light (Technical) Adapters utilize existing functionalities of the platform they run on, so do not need the installation of any additional units on the host application platform. Their role only includes accomplishment of the interaction between the integrated application and the integration solution. They ensure the appropriate transport of information for each participating application.

The platform which hosts the core of the integration solution provides the infrastructure for all transformation and routing processes.

There are a number of advantages of this type of adapters: transformation is done solely on the core of the integration solution, deployment is easy, it offers completely adaptable solutions and has the capacity to manage complex transformation rules.

- *Thick (Business) Adapters:* Thick (Business) Adapters are used to ensure decentralized transformation functions. This type of adapters requires components to be installed on the platform which hosts both the application to be integrated and the core of the integration solution. They transform the sending application's proprietary format into a canonical format, or transform the canonical format to the receiving application's proprietary format.

There are a number of advantages of this type of adapters: a repository of standardized business events can be constructed by the usage of a pivot or canonical format, gives possibility for a complete set of parameters delivered with the integration solution through the use of a canonical format [1].

2.1.2.2. Information Adaptation

The capture, transport and delivery of information to the recipients in their own format are not enough to address all the problems of integration. Transformation of the content of the messages and events to the recipient applications expectations should also be satisfied. In addition it must be possible to determine these receiver applications and deliver what they expect at the right time.

2.1.2.2.1 Transformation

A message or event should go under transformation under two conditions: when the syntax or format of the message or event is not expressed in a way that is directly understandable by the recipient applications, when the message or event includes information used to create other events.

Many different types of data formats exist in an enterprise. Data formats can be classified as following:

- *Flat Format (Fixed Length):* Data items have the same length and are always located at a prearranged position. This format has the best performance.

- *Flat Format (Variable Length):* In this format, the value and length of data for each data item should both exist in the data document. The position of a specific data item will depend on the actual length of the data that precedes it. Order of data items is always in the same way.

- *Flat Format (with Delimiters):* There are two subtypes for this format: sequenced and non-sequenced. In sequenced data, data order is always the same and separated by a fixed character (i.e. commas). In non-sequenced data, data order is not fixed therefore both delimiter and keywords are used to identify the data items.

- *Hierarchical (Tree Structure) Format:* A combination of above formats can be utilized to construct this format. Composite data structures can be created by assembling other data items which can be simple or composite. Additionally, data items can be repeated multiple times to constitute a composite item.

- *XML Formats:* XML (eXtensible Markup Language) is the most popular format among hierarchical formats and is a standard from the W3C (World Wide Web Consortium) [40]. XML language describes the logical structure by textual schema documents. A tag system is used to define the elements and their relationships with the other elements constituting the structure. The primary objectives of this language are separation of the form and the content of a document, formalization of their structure, standardization of the tags and making their computing treatment easier [1].

Transformations can be analyzed in two types:

- *Syntactical Transformation:* The main aim of syntactical transformation is the modification of the representation of a message or event in order to make it usable for the processing application. The whole message or any data item constituting the message may go under transformation.

The order of information inside the message or the form for an item representation can be modified.

- *Semantic Transformation:* The main aim of semantic transformation is the modification of the meaning of all or part of the information for an original message or event, or from that event or message, to infer other events.

2.1.2.2.2 Routing

Routing is performed for determination of the recipients for the messages or events produced by a source application. There are many different ways to accomplish this task. In a widely adopted *spaghetti system* the recipients of the information are directly reached by the source applications. In such a system, the source application should know the target applications and routing functionality should be performed by itself.

A publication/subscription mechanism can be used to provide the independence of participating applications. The events are *published* by the generating applications and events become available for the interested applications which are *subscribed* to receive them.

2.1.2.2.3 Defining the Rules

The key operations of message and event adaptation like transformation, routing and storage require rule definitions. The best approach for management of rules is centralizing them in a global dictionary and then delegating the distribution to the integration infrastructure. This dictionary may contain different types of objects like the definition of the events and their structure, rules for identification of these events, rules for applicable transformations, the rules for routing and the rules for caching mechanisms and publication and subscription information [1].

2.1.2.3. Business Process Automation

A business process is defined by the Workflow Management Coalition (WfMC) as follows:

"A business process is a collection of one or more linked procedures or activities that together accomplish a business objective, in the context of an organizational structure that define roles and relationships."

A single organizational unit or numerous organizational units can constitute a business process. Triggering conditions for the initialization and possible outputs are defined before the execution of a process.

Auto-execution of a business process can be carried out after modeling of the process. The model describes the activities which will be performed, the relationships between the activities, the conditions for initialization and the end of the process, and all participants included in each activity.

The automation of a process is achieved by defining the business rules which will be executed during the process lifetime. This allows externalization of the coding rules for the applications so that changes in the business rule do not lead to modifications in the application.

2.1.3. Problems in Enterprise Application Integration

Main problems in EAI can be classified as following:

- *Data Propagation and Consistency*: Frequently, data belonging to the same domain is stored redundantly in different software systems for organizational, technical or geographical reasons. Two methods can be used to achieve consistency. First one is the repository approach. In this approach data is directly copied from the original database tables or files to the database tables or files of the other applications. The second is the event based approach. In this approach, an event is transmitted from the original application to the other applications and each receiver application updates its own data store.

- *Creating Composite applications:* Composite applications are developed by using data or services of other applications. Client applications on the Web are typical examples for this type of integration.

- *Management of Multi-Step Processes*: Business processes are comprised of several operational steps which have asynchronous and uni-directional interactions with logical interdependence. However, processing performed by each step is also dependent on the processing performed in the upper levels by the previous steps [1].

2.2. Semantic Web Technologies

2.2.1. The Semantic Web

"The Semantic Web is a vision: the idea of having data on the Web defined and linked in such a way that it can be used by machines not just for display purposes, but for automation, integration and reuse of data across various applications." (Semantic Web Activity Statement)

The fundamental building block of the current World Wide Web (WWW) [40] is HTML (Hyper Text Markup Language) [41], which is useful for publishing information. HTML is made up of a symbol set contained in a web page intended for displaying information in a web browser. The main target of information delivery on the current web is human consumption. Humans read, analyze and evaluate the web pages to understand the content. The meaning of the content is not understandable by computers and software agents [10].

If the information on the Web can be described in a different way, so that it can be interpreted by software systems not just for browsing purposes, but also for interoperability and integration between systems and applications, more benefit can be obtained from distributed information. The objective of the Semantic Web is providing the information in such a way that computers can interpret it. This will enable automated processing of information exchanges between computers and software systems. The Semantic Web is the extension of the current one where information is distributed with unambiguous meaning, allowing computers a better understanding.

The primary aim of the development efforts for the creation of the Semantic Web is adding machine readable information to the data and documents. There are a number of different approaches for giving meaning to resources. New standards and languages are continuously being researched and developed [10].

The distinction between the syntax and semantics is important for the evaluation of Semantic Web concepts.

- *Syntax:* In computer science, syntactically correct program means that the program is valid with respect to the syntactical rules of the compiler and no error messages are generated when it is compiled. However, this is not adequate for semantic correctness. In the integration process, XML format only ensures the syntactical correctness of the exchanged data between information systems. Integration efforts based solely on syntactical methods do not take account of the study of concepts such as *meaning* and *truth* [10].

- *Semantics:* The main point in the study of semantics is association of meaning to data items. The objectives of semantics and syntax are completely different. Semantics is related with the meaning of something, while syntax is concerned with the formal structure in which something is expressed [10].

Heterogeneities frequently occur in software systems due to disagreement about the meaning, interpretation or intended use of data. Semantic technologies can be helpful in resolving those complexities. Information heterogeneity can be analyzed in four different categories, namely system heterogeneity, syntactic heterogeneity, structural heterogeneity, and semantic heterogeneity:

- *System heterogeneity :* System heterogeneity occurs due to applications and data residing in different hardware platforms and operating systems

- *Syntactic heterogeneity:* Syntactic heterogeneity arises due to using different representations and encodings for data. XML supports ability to deal with syntactic heterogeneity.

- *Structural heterogeneity:* Structural heterogeneity is encountered when different information systems store their data in different document layouts and formats, data models, data structures and schemas. Some technologies dealing with this type of heterogeneity are XML, XPath, and XQuery [50].

- *Semantic heterogeneity:* Semantic heterogeneity occurs due to expression of the meaning of data in different ways. Semantic heterogeneity focuses on the content of an information item and its intended meaning.

Sharing and exchanging information in a semantically consistent way is the key factor for successful resolution of this type of heterogeneity. Ontology languages like RDF and OWL constitutes the basis for semantic modelling [10].

Two different solutions can be applied to solve the mentioned semantic heterogeneities. First solution is the classical approach that relies on developing adapters for translation of information between the terminologies of pairs of systems. If the number of systems joining interactions is small, this solution may be useful. However, this solution suffers from scalability, as the number of participating systems becomes more and more, the cost for development and the degree of semantic heterogeneity increases. Assuming execution of bidirectional translations, the interoperability of *n* systems can only be satisfied with *(n-l)+(n-2)+...+l* translators. The translators needed for integration of four different systems are shown in Figure 2.1:

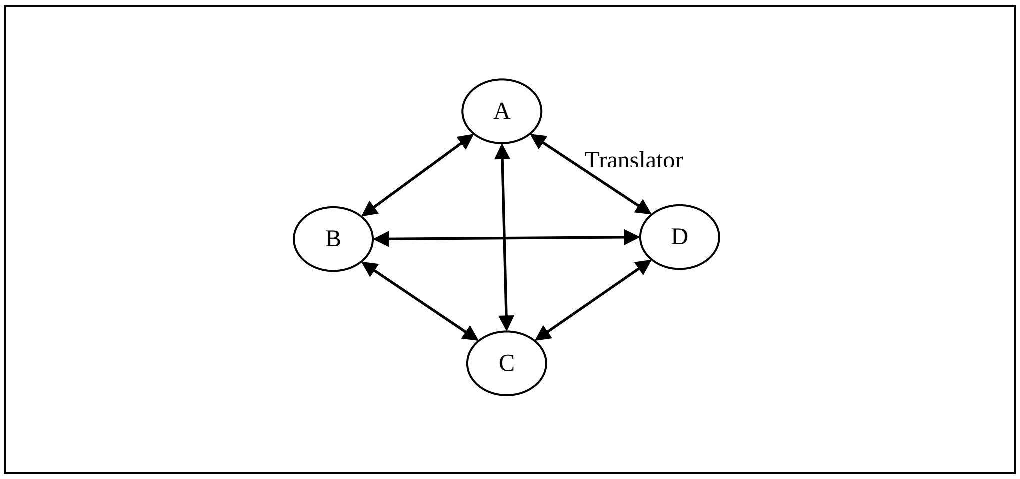


Figure 2.1 Translators to resolve semantic heterogeneity. [10]

Semantic technologies offer better solutions for resolution of semantic heterogeneities. Data provided by each distributed system is semantically defined with the concepts in a shared ontology. This solution only requires the development of 'n' links to interconnect systems. Links between systems in the existence of a shared ontology is shown in Figure 2.2:

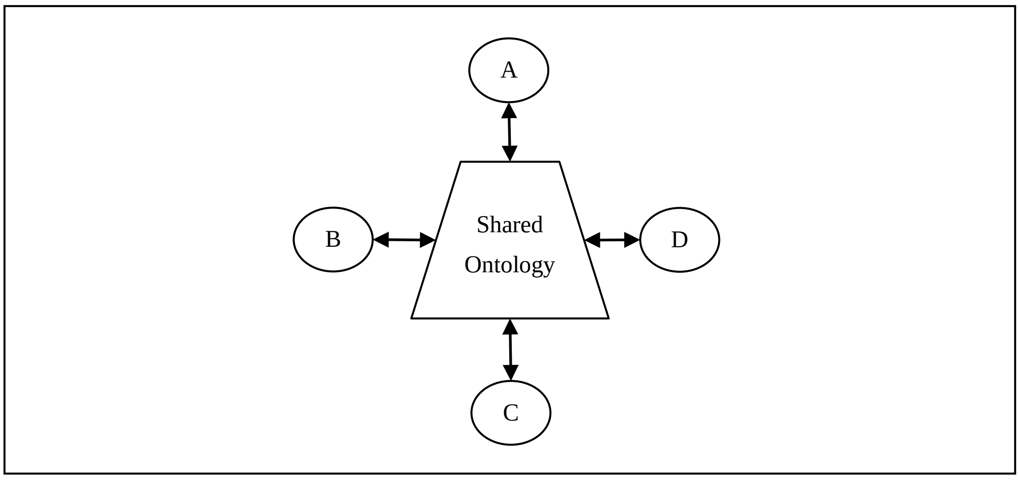


Figure 2.2 Shared ontology to resolve semantic heterogeneity.[10]

2.2.2. Ontologies

Computer Science adopts the *ontology* term from philosophy by slightly changing its meaning:

"An ontology is a formal explicit specification of a shared conceptualization." [16]

Ontologies constitute a bridge between human and machine understanding through formal and consensual terminologies. As a result of this key property, ontologies can be shared and reused between human and computers. Recent progress in the development of Semantic technologies increased the interest on ontology development.

Ontologies can be divided into three different types:

- *Generic Ontologies:* Generic Ontologies are used to model domain independent knowledge like time and space. Some examples for generic ontologies are CYC [43] and WordNet [42]. Software systems related with various domains make use of generic ontologies.

- *Domain Ontologies:* Domain Ontologies encapsulate the knowledge belonging to a certain domain. An example can be UNSPSC [44] which provides a scheme on product classification. Domain ontologies are shared between all the potential participants in that domain.

- *Application Ontologies:* Application Ontologies capture the knowledge necessary just for a specific application. Application ontologies are not considered as real ontologies by some authorities, because they are not really shared. An ontology developed for a particular Web site can be an example for this type ontologies.[16]

.

2.2.2.1. Resource Description Framework (RDF)

The Resource Description Framework (RDF) [17] was developed as the first language for adding machine-readable semantic metadata to existing data on the Web. Basic data model of RDF is the subject–predicate–object triple, commonly written as P(S, O). *Subject* is the thing that a statement is made about. *Predicate* is the property that is specified for the *subject. Object* is the value of this property for the *subject*. Uniform Resource Identifiers (URIs) are used to identify each resource uniquely.

There are several different serializations for representation of RDF triples. Notation 3 (N3) is a compact serialization of RDF triples. Notation 3 (N3) uses *qnames* for the representation of resources belonging to an RDF triple. A *qname* is a URI abbreviation scheme and composed of two parts: a namespace and an identifier. Namespace and identifier are separated with a column sign. So the *qname* representation for the identifier *Turkey* in the namespace *geo* is simply *geo:Turkey*.

Bindings for the namespaces should be declared before their usage in a triple. An example namespace declaration is shown as follows where *ex* is the namespace of *qname* and *http://example.org/#* is the actual URI of the namespace:

```
@prefix ex: <http://example.org/#>
```

Notation 3 (N3) serialization of an example RDF triple is shown below where *#ahmet* and *hasAge* are identifiers defined in the namespace *ex*:

```
ex:ahmet  ex:hasAge  "25" .
```

Although Notation 3 (N3) is suitable for human consumption, software systems need more structured representations. For this reason there is also an XML serialization of RDF called RDF/XML [51]. The *subject* is referenced using the XML attribute *rdf:about*, and the triples with this *subject* appear as subelements within this definition. The corresponding RDF/XML serialization for the above RDF triple is shown in Figure 2.3:

```
<rdf:RDF

  xmlns:rdf=”http://www.w3.org/1999/02/22−rdf−syntax−ns#”

   xmlns:ex=”http://example.org/#”>

  <rdf:Description about=”#ahmet”>

        <ex:hasAge>25</ex: hasAge >

  </rdf:Description>

</rdf:RDF>
```

Figure 2.3 The corresponding RDF/XML serialization.

RDF triples can also be represented as graphs where nodes symbolize *subjects* and *objects* (resources), and edges symbolize *predicates*. An *object* of a triple can be the *subject* of another triple which yields to a directed graph. Furthermore, RDF allows making *statements about other statements*, which means that any RDF statement can be utilized as a *subject* in a triple [16]. Suppose another RDF triple is added to the above semantic model as following:

ex:ayse ex:isMarriedWith ex:ahmet .

Resulting RDF graph representation for the semantic model is shown in Figure 2.4.

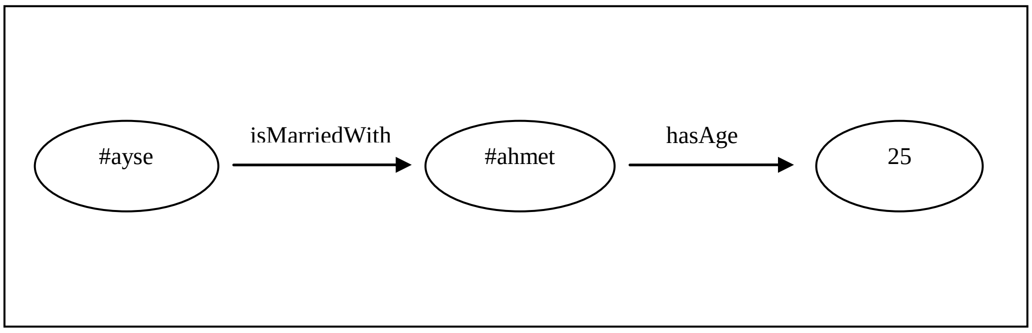


Figure 2.4 RDF graph representation for the semantic model.

2.2.2.2. Resource Description Framework Schema (RDFS)

RDF Schema (RDFS) [18] extends RDF with basic ontological modelling primitives. RDFS can be thought as an extension of RDF with a vocabulary for defining classes, class hierarchies, properties, property hierarchies, and property restrictions. RDFS classes and properties can be expressed in RDF. A sample RDFS semantic model is shown in Figure 2.5:

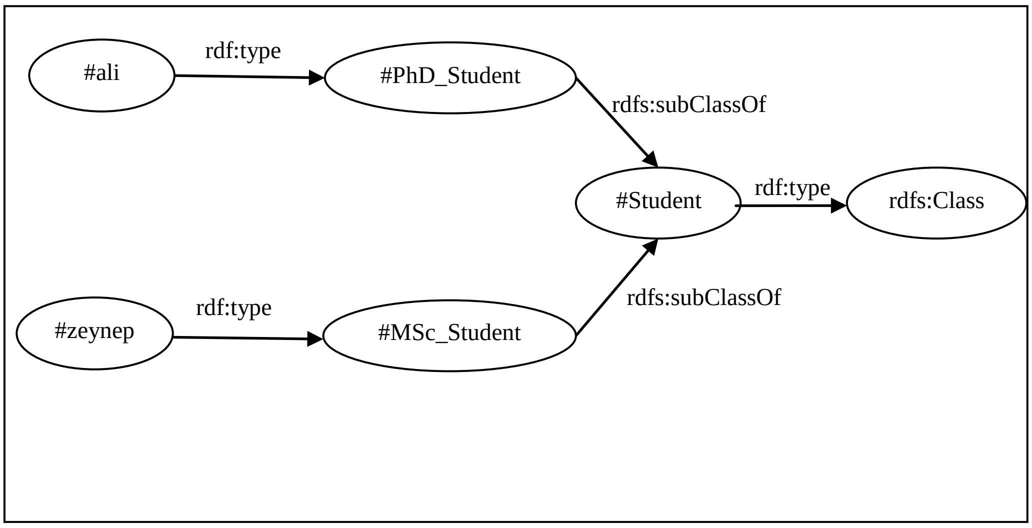


Figure 2.5 A sample RDFS semantic model.

RDF(S) (the combination of RDF and RDF Schema) has expressive limitations so more expressive languages are developed for the Semantic Web.

2.2.2.3. Web Ontology Language (OWL)

The Web Ontology Language OWL [19] is an expressive ontology language which is based upon RDFS. OWL has three species of increasing expressiveness:

- *OWL Lite:* OWL Lite is the least expressive of the OWL species. It adds simple cardinality restrictions, local range restrictions, existential restrictions, equality, and different types of properties (inverse, transitive and symmetric) on top of RDFS.

- *OWL DL:* OWL DL adds full support for negation, disjunction, enumerations, value restrictions, cardinality restrictions, compared with OWL Lite.

- *OWL Full:* OWL Full does not have restrictions on the use of vocabulary and the use of RDF statements in contrast with OWL Lite and OWL DL. OWL Full allows both the specification of classes-as-instances and the use of language constructs in the language itself [16].

2.2.3. Semantic Web Services

Web services are modular, self-describing, self-contained applications that are accessible over the Internet [11]. Currently, Web services are described using the Web Services Description Language [12], which provides syntactical information. The Web Services Description Language (WSDL) does not contain semantic descriptions, it only specifies the structure of message components using XML Schema constructs. Semantic Web services technologies intend to add semantic definitions to the web service descriptions besides syntactic definitions.

Relations between web, web services, semantic web and semantic web services are shown in Figure 2.6:

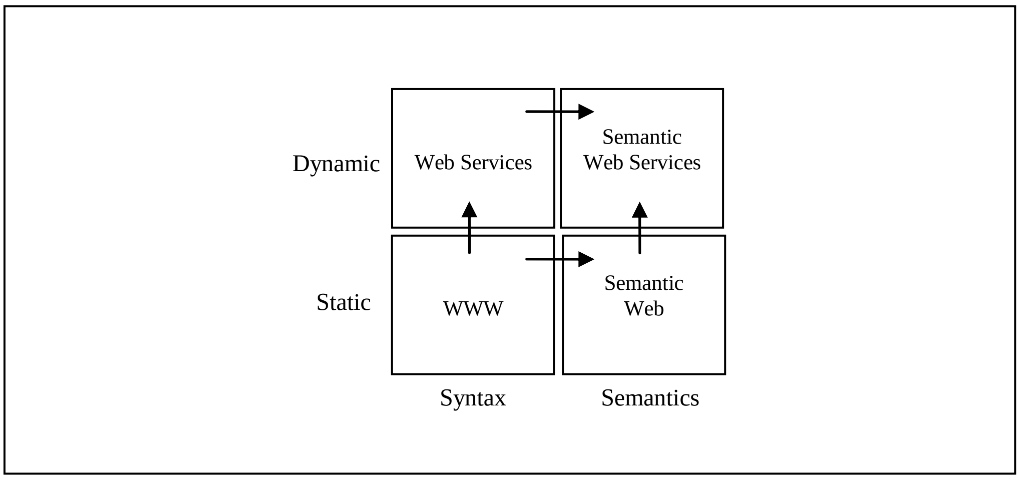


Figure 2.6 Relations between web, web services and semantics [10]

2.2.3.1. Semantic Annotation of Web Services

Semantically annotating a Web service implies explaining the exact semantics of the Web service data and functionality elements. Domain models and ontologies provides the necessary reference information for annotation. Web service elements are associated with the semantic concepts. Ambiguities in the interpretation of functionality or data of a Web service are eliminated by this way. The purpose of annotating Web services is to enable unambiguous and automated service discovery and composition. For example, two Web services may have the same names to represent their operations, inputs and outputs, but they could have totally different purposes in usage [10].

The parts of the web service to be annotated change due to application-specific requirements. Whole Web service or just a small part of it may be annotated. Operations, data input and output structures, preconditions and effects of the operations and non-functional aspects of the Web Service can be annotated.

There are four types of semantics associated with Web services:

- *Data Semantics:* Data semantics includes the formal definition of data input and output messages of a web service. It is used in service discovery and information integration.

- *Functional Semantics:* Functional semantics is the formal definition of the capabilities of a web service. It is used in discovery and composition of web services.

- *Non-functional Semantics:* Non-functional semantics is the formal definition of quantitative or non-quantitative constraints like QoS (Quality of Service), requirements like minimum cost, policy requirements like message encryption. It is used in discovery, composition and interoperability of web services.

- *Execution Semantics:* Formal definition of the execution or flow of services in a process or of operations within a service. It is used in process verification and exception handling [10].

The need for semi-automatic annotation of Web services increases with the widespread usage of Web Services and domain models. The main aim in the annotation process is the association of Web Service elements with the most appropriate concepts in a semantic model. Automatic generation of the mapping rules is still impossible [29]. Current mapping tools can only be used to validate the mappings or to suggest possible mappings. Human intervention is still necessary at some point in the mapping process [45].

The most well-known efforts in the semantic markup of Web services have been OWL-S (OWL-based Web Service Ontology) [13], WSMO (Web Services Modeling Ontology) [14], WSDL-S (WSDL-S, Web Service Semantics) [15] and SAWSDL (Semantic Annotations for WSDL) [4] specification. While WSMO and OWL-S define their own rich semantic models for Web services (top-down approach), WSDL-S and SAWSDL specifications work in a bottom-up fashion by preserving the information already present in the WSDL.

2.2.3.1.1 OWL-S

OWL-S presents an upper level ontology to describe web services. Web service providers use this ontology constructs to add semantic definitions to their Web services. The ontology comprises of a service profile (offerings of the service), service model (usage of the service) and service grounding (interaction ways with the service). Top level of the OWL-S service ontology is shown in Figure 2.7:

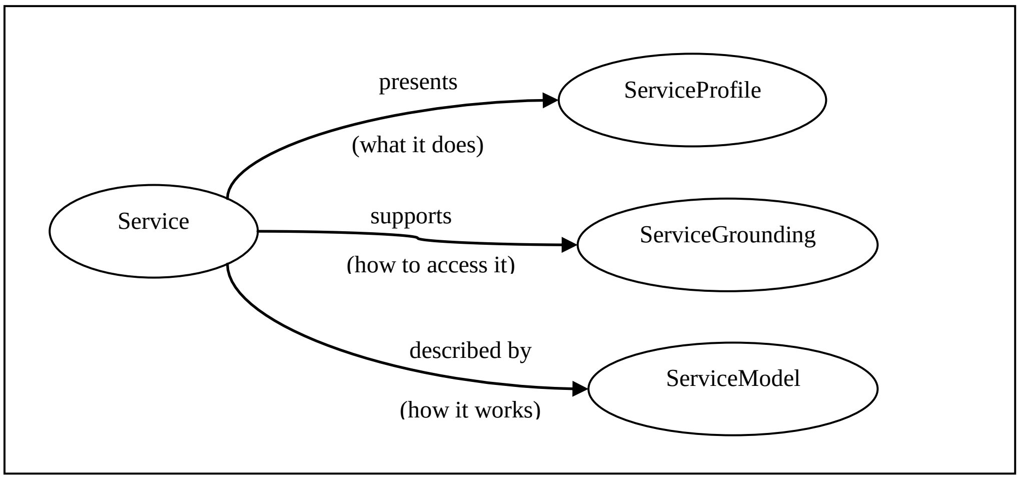


Figure 2.7: Top level of the OWL-S service ontology [13].

The *Service* class has to be present in every modelled Web Service instance. The *Service* class has three properties named *presents*, *describedBy* and *supports*, which point to classes *ServiceProfile*, *ServiceModel*, and *ServiceGrounding* respectively. The *ServiceProfile* property is used for service discovery, the *ServiceModel* and *ServiceGrounding* properties are used together for consumption of the service. OWL-S rules are expressed with the Semantic Web Rule Language (SWRL).

2.2.3.1.2 WSMO

Web Service Modelling Ontology WSMO, also a W3C submission, is another conceptual model for Semantic Web services. Some terms related with WSMO are:

- *WSML (Web Services Modelling Language):* WSML is the formal description language for all the constructs of WSMO environment.
- *WSMX (Web Service Execution Environment):* It is the execution environment for WSMO-based systems.

- *WSMF (Web Service Modelling Framework):* It is the modelling framework that WSMO is based on.

WSMO ontologies provide the terminology infrastructure that is used by other WSMO elements for addition of semantic knowledge. There are three levels of mediation in WSMO: Data Level (for mediation of heterogeneous Data Sources), Protocol Level (for mediation of heterogeneous Communication Patterns) and Process Level (for mediation of heterogeneous Business Processes). There are four top-level concepts in WSMO: Goals, Ontologies, Mediators and Web Services. WSMO top level concepts are shown in Figure 2.8:

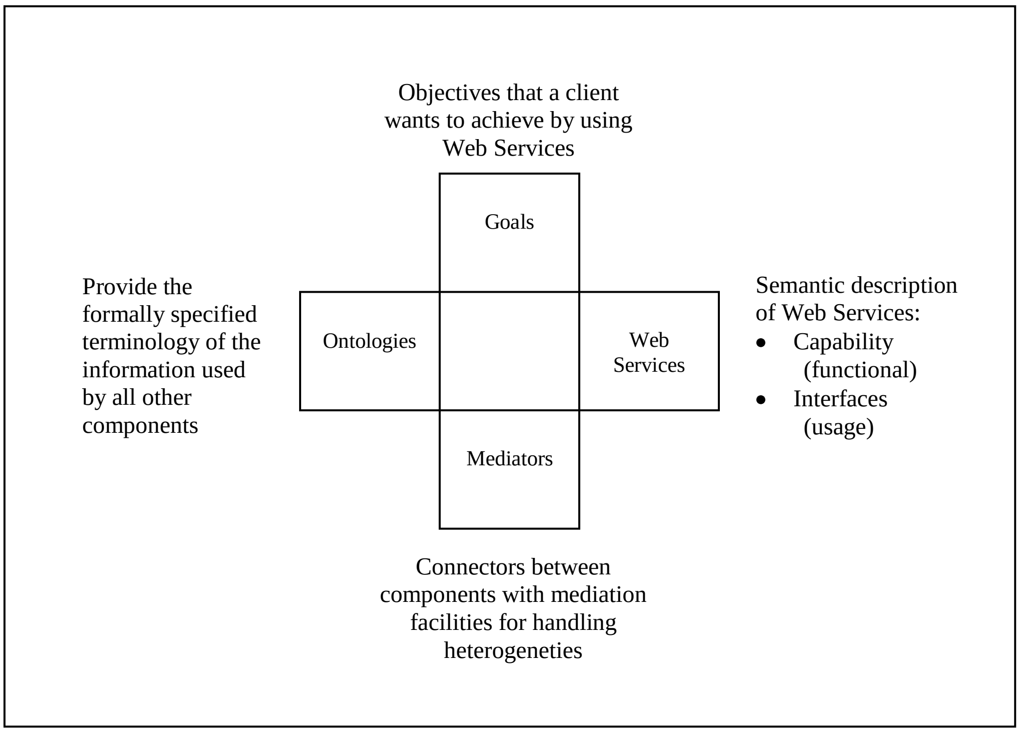

Objectives that a client wants to achieve by using Web Services
Goals
Provide the formally specified terminology of the information used by all other components
Ontologies
Web Services
Semantic description of Web Services:
• Capability (functional)
• Interfaces (usage)
Mediators
Connectors between components with mediation facilities for handling heterogeneties

Figure 2.8: WSMO top level concepts [16].

WSMO and OWL-S, both adopt top-down approach, they provide service ontologies to build semantic Web services.

2.2.3.1.3 SAWSDL

The Semantic Annotations for WSDL and XML Schema (SAWSDL) [4] specification is also a W3C Recommendation. It is built on existing Web service standards the community is already familiar with, and shows good promise of acceptance and quick realization [10]. SAWSDL specification does not specify a language for representing the semantic models in contrast with WSMO and OWL-S. Instead, it provides mechanisms for mapping of existing WSDL components with semantic concepts. The semantic concepts defined out of the WSDL document and referenced by extended attributes. SAWSDL specification is based on previous member submission WSDL-S [15].

The key design principles for SAWSDL specification are:

- Existing extensibility framework of WSDL is utilized as a base for semantic annotations for Web services.
- It allows building domain models in any preferred language (not necessarily in OWL as required by OWL-S) or reuse existing domain models.
- Semantic annotations are utilized for both Web Service discovery and Web Service invocation.

Based on these design principles, SAWSDL specification defines the following three new extensibility attributes to WSDL 2.0 elements to enable semantic annotation of WSDL components:

- *modelReference*: *modelReference* is an extension attribute specifying the association between a WSDL element and a concept in some semantic model. This attribute can provide only direct associations with the semantic model concepts, cannot point to mappings. It can be used especially to annotate XML Schema type definitions, element declarations, and attribute declarations as well as WSDL interfaces, operations, and faults.
- *liftingSchemaMapping and loweringSchemaMapping:* They are two extension attributes that are added to XML Schema element declarations and type

definitions for specifying mappings between semantic data and XML. *liftingSchemaMapping* lifts data from XML to a semantic model, whereas *loweringSchemaMapping* lowers data from a semantic model into an XML structure. [10].

Association of WSDL elements and semantic domain model is shown in Figure 2.9:

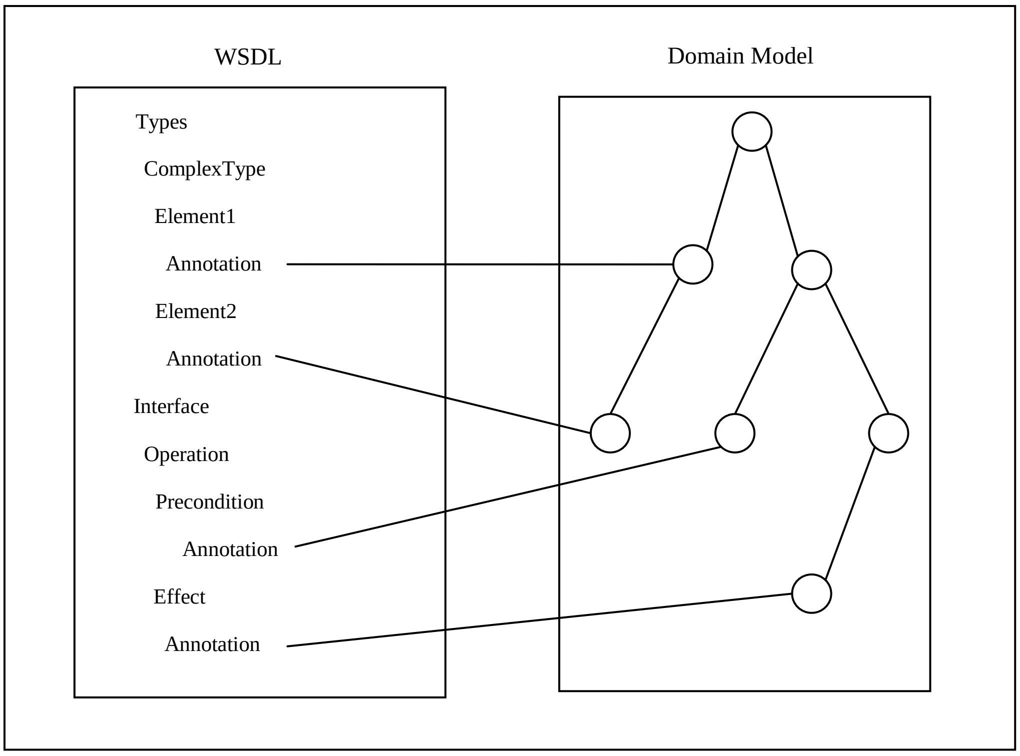


Figure 2.9: Association of WSDL elements and semantic domain model.[16]

2.2.4. RDF-Stores

An RDF-Store holds RDF data, similar to relational databases storing tabular data. As RDF triples are inserted, they are merged with the existing data in the storage. Semantic query languages, similar to relational query languages, are utilized to inquiry the data in the RDF-Store [39]. RDF-Stores also allow for inference on the existing RDF triples. Inferencing is done in response to queries only.

2.3. Integration of Customer Data

Enterprises are turning from product-centric sales strategies to customer-centric sales strategies in recent years. This change in attitude includes many business sectors like financial organizations, health institutions and telecommunications. It is realised that the number of *best customers* is finite, so the new sources of revenue do not have to come only from acquiring new customers, but also from understanding and improving the characteristics and relationships of the existing customers with the enterprise [46]. By knowing the customers better, enterprises may increase cross-sell/up-sell opportunities and effectiveness of marketing campaigns, reduce customer complaints and decrease customer service times. To achieve those business requirements, enterprise software systems should develop more reliable, adaptable, automated and intelligent ways for managing customer data.

In large enterprises, customer data that is required for the consuming applications can be scattered through many different applications and systems even this data represent information from the same domain. Customer data is frequently required to be collected from various systems before used by consumers like Customer Relationship Management (CRM) or Call Centre applications. The possible source systems can be grouped as below:

- Different departments of the same enterprise may hold customer-related information. For example in a bank, generally there is a central customer database which is the main storage for the customer data, however for some reasons some of the other departments may also hold similar customer information. For instance, credit card and loan departments may also store different addresses or phone numbers of the customers in their own systems. The reasons for this sort of disarrangements can be stated as providing usage and performance benefits for the departmental applications, using software packages that are not easily adaptable, utilizing different technologies from the central domain system which makes integration difficult.

- Group companies sometimes need to share data between themselves. For example an insurance company can supply customer information to a bank from the same group. The same individual possibly owns records in both companies so the information is needed to be combined before usage .It is highly possible that the web service offered by the other company provides data in a company-specific format so mappings and transformations are necessary.

- Mergers and acquisitions frequently happen between the companies in the same industry for many reasons. After a merger or acquisition, the resulting company has two different systems which may have information representing same individuals. For example, when two banks are merged, there can be many people that have accounts in both of them. The resulting company may prefer to keep the software systems separate, so the data for the same customers need to be conciliated before consumption [47].

In a Web Service enabled Service Oriented Architecture environment inside an enterprise, integration efforts are lead by sharing WSDL files between interacting parties. Necessary transformations arising from heterogeneous data structures are tried to be solved by manual efforts. Standard web service definitions are syntactic, and do not provide any information on the *meaning* of the exchanged attributes. Generally, web service providers prepare verbal definitions of the attributes and share with the consumers of the service. These definitions tend to be inadequate and subjective which increase the time required for integration. Software development becomes more error-prone due to misunderstandings between different software teams. Sample verbal attribute definition for a web service is shown in Table 2.1:

Table 2.1 Sample verbal attribute definition for a web service.

Source Attribute	**Verbal Definition**
Name	The name of the customer
AddressType	The type of customer address (i.e. home, work)
Profitability	The profitability of customer
PhoneNumber	The phone number of the customer

Many questions may arise after receiving such a definition document. For the previous sample, some of them can be:

- Does the *name* attribute contain name and surname together or just the last name?

- How the *AddressType* attribute encoded? What can be the possible values for this field?

- Does the *Profitability* attribute represent the gross profit or net profit after taxes?

- Does the *PhoneNumber* attribute contain the country code also? Are there spaces between country code and the actual number?

Certainly, those issues can be solved eventually by exchanging a series of enriched documents and definitions and integration occurs successfully occurs in current syntactic systems. Utilizing semantic technologies can reduce those efforts and provide a more organized and reliable way for information integration.

Customer data domain can be a good candidate for development of a semantic based integration system in an enterprise for several reasons:

- There can be many sources producing customer data and consumers need to consolidate it before use to provide effectiveness. As a result large scale integration efforts frequently occur in enterprises on this domain.

- Nature of customer domain allows performing inferences, which contributes to powerful analytic systems.

- Investment on software systems improving customer data has a potential for high return on investment.

CHAPTER 3

USING SEMANTIC TECHNOLOGIES FOR ENTERPRISE APPLICATION INTEGRATION

In order to overcome information integration problems, enterprise software systems should solve issues in three main subjects:

- the diverse formats of content
- the disparate nature of content
- the need to derive *intelligence* from this content [10]

Current software systems, applications and tools allow working with the syntactic metadata but it is not sufficient to handle above problems. Semantic metadata provides reliable and machine-interpretable definitions for information. By annotating existing documents with semantic model concepts, software agents can automatically understand the full meaning of information context.

Web Service enabled Service-Oriented Architectures (SOAs) used in EAI take advantage of Web Service Description Language (WSDL) which provides an abstraction layer for the involved interfaces. Data exchange structures in Web service technology is based on XML, which has no semantic model support currently [29].

3.1. Advantages

Semantics can provide advantages to the next generation of information integration and analysis systems in the following areas [26]:

- Main power of semantic models emerges for merging data coming from different source systems. Semantic technologies offer flexibility and simplicity for extraction, organization, standardization and unification of heterogeneous information. Source system information can be in different contents like structured, semi-structured and unstructured and in different formats like database tables and XML.

- For some information domains, metadata definitions and rules can be scattered through different software applications and systems. Each different system may interpret the attributes and rules in the domain differently, which increases complexity for interoperability. Building a semantic model for this kind of domains, composes all the metadata elements, their relations and rules running on them in a single and central place. All the other user applications for the domain refers to this semantic model, so it is easier to track and control the usage of the domain data throughout the enterprise. Enterprise-wide rules can be imposed more straightforwardly.

- Inferencing provides powerful ways for analyzing and correlating extracted information to discover previously unknown or non-obvious relationships between domain attributes and/or entities based on semantics which enriches the available information. Much better business decisions can be made by utilizing inferred data [10].

- Semantic models allow machine understandable metadata and content definitions which enables higher levels of automation in the process of data extraction, data transformation and interoperability.

- Semantic querying produces efficient, fast and high-quality (contextual) result sets which increases performance of data analysis efforts.

3.2. Semantic Approaches for EAI Layers

Various aspects of semantic technologies can be utilized to solve EAI problems. Semantic methods can differentiate for each layer of EAI.

3.2.1. Process Layer

The most important problem to be solved in the process layer is that different services may provide semantically same functionality although each has different message exchange patterns [27]. If you take an example of opening an account, one system might offer it with one single invocation whereas the other system requires first the creation of a user, followed by activation of the user and finally opening the account. In the first situation, one invocation is enough for the service success, whereas in the second situation several invocations are necessary to achieve the same functionality. Specific execution order is also important. Activation of the customer cannot be before the user creation. These differences can lead to heterogeneities after the discovery of the Web Service within a Service Oriented Architecture. For a successful invocation of the Web Service, the two interacting parties should be able to adapt their information exchange models or to use an external system for mediation to fulfil the process.

In a classical approach, participants readjust their exchange patterns before every invocation of Web services to solve heterogeneities. This approach hinders a dynamic invocation. [29].

In the semantic approach, process or behaviour mediation is used [28]. By applying Semantic Web Service principles, mediator systems analyze the runtime behaviour of web services by utilizing the semantic annotations that are created for them. Possible heterogeneities that may arise between client and the Web Service are compensated in order to acquire equivalent processes. Some examples can be generation of dummy acknowledgement messages, aggregating multiple messages in one single message, changing the order of messages or removing some of the messages to provide interoperability between the two interacting parties [29].

3.2.2. Transformation Layer

Although ontologies are used as shared conceptualizations of the same problem domain within an Enterprise software environment, there is always probability that different systems in the same enterprise, or different parties that the enterprise interact may use distinct semantic models for the same domain. In such a case where services use dissimilar ontologies the EAI infrastructure has to provide solutions to transform between them. This semantic transformation is also called mediation [28]. Mediation preserves the semantics of the involved systems, while allowing them for interoperability.

Mapping tools exist for automatic generation of transformation rules but auto-generation of the mapping rules with the sole decision of the tool is still impossible. The best results achieved in research projects are mapping tools that are able to suggest or validate possible mappings, however intervention by domain experts is still required at some stage in the mapping process [30]. In traditional transformation efforts, mediation takes place at the level of XML Schema. However in semantic transformation architectures, it is resolved at the level of ontologies. In traditional systems, two different XML Schema mapping rules have to be defined, in the semantic approach only one mediator definition between the participants is enough. The major difference is, the discovery and reuse of mappings between two dissimilar ontologies can be performed at runtime, while mappings between XML Schema definitions should be bound at design time.

It is possible that different ontology representations can be used in interacting parties (like WSMO and OWL-S). In this case, ontology mediation has to be applied to the conceptual model of the involved ontology representations as well [30]. However, these transformation problems are of restricted nature since the set of available ontology representation languages is limited [29].

CHAPTER 4

RELATED WORK

In this chapter, overview of the previous work that has been produced in semantic web services usage for Enterprise Application Integration area is presented. In [29], the authors have proposed to extend the concept of Service-Oriented Architectures by using Semantic Web Services. They call this new type of architecture as Semantic Service Oriented Architecture (SSOA). They apply the Web Service Modeling Ontology (WSMO) framework to this architecture and show how EAI benefits by it. This allows dynamic discovery and invocation of services published in the architecture.

The authors have shown how the functionality improvements occur in the transformation and process layers. In the proposed architecture, mediation takes place at a higher level of ontologies, instead of mapping between XML Schemas as it is done in traditional SOA. As a result, the number of required mappings greatly reduces. SWS principles are applied in the process layer in order to obtain equivalent public processes by dynamically compensating clients or web service communication patterns.

Only a conceptual model is not sufficient for achieving dynamic environments. The authors exposed some challenges and standardization efforts on Semantic Web Service frameworks to establish this goal. Their findings include definition of standards for business document ontologies, ontologies for description of Service Level Agreements, negotiation protocols and policy declarations as logical conditions (reliability, security etc.).

The authors make research on the possibilities of dynamic discovery and invocation in intra-EAI scenarios under definite assumptions. They applied WSMX framework in an integration use case.

In [31], the authors proposed a dynamic data mediation approach based on semantics among participating Web Services in a Collaborative Business Processes framework. Moreover, they presented a tool for semantic annotation of web services for the end users which allows for graphical definition of the needed up and down casting XSLT [7] transformations.

In addition, they presented a real-life B2B integration scenario to demonstrate the applicability of the proposed approach within a franchisor-franchisees network. Within this scenario, data heterogeneities are dynamically resolved at execution time.

In [32] the authors described a methodology to overcome problems arising from syntax-based standards in Enterprise Applications Integration (EAI) by using a Semantic Web Services based approach. The methodology involves multiple validation and integration steps that are needed to be executed both at design time and run time. Generalized and normalized ontologies are developed at design time which allow similarity analysis of the ontological models based on models. During run time, previously developed semantic models and automated inferencing tools are used for semantic translation of business document instances.

As a result of their experimental work, they conclude that the Semantic Web technologies are mature enough for solving reasonably realistic Enterprise Application Integration problems.

The distinctive attribute of this work is, it is targeting the operations inside one organization. Other approaches usually target inter-enterprise operations and they depend on service discovery. Our approach on the other hand can utilize the inference capability based on the ontology defined for the enterprise, while the other's inference capabilities depend on the matching performance of the semantic search. Comparison of related work mentioned above and our work is shown in Table 4.1:

Table 4.1 Comparison of related work and our work

	[29]	**[31]**	**[32]**	**Our Work**
Approach	Top-down	Bottom-up	Bottom-up	Bottom-up
Target	Inter-EAI	Inter-EAI	Inter-EAI	Intra-EAI
Discovery	Yes	Yes	Yes	No

CHAPTER 5

SEMANTIC WEB SERVICES BASED DATA INTEGRATION

5.1. Enterprise Integration Scenario

A software system is designed and implemented by using semantic web services for integration of customer data inside an enterprise. Web Service enabled Service Oriented Architecture is assumed to be used widely in the enterprise before semantic technologies are applied. The software systems constituting the enterprise network like the departmental software systems and partner companies are assumed to interact with web service technologies.

Since it is an intra-enterprise environment, the web services that will be involved in the integration phase are known at the beginning. Web service discovery is not required to be performed.

In the integration scenario, it is assumed that all the source systems relate their customer data with a globally unique key like *Turkish citizenship number*. Consumer application supplies the citizenship number of the customer to the semantically enriched integrator process. Semantic integrator process collects, transforms and standardizes all the data related with this customer. After data is mediated, the final response is delivered back to the consumer application.

Main goals in this study are:

- Developing an enterprise integration framework that requires no modification in the source web services,

- Encapsulation of all the mapping, transformation and standardization operations in a central unit which can smoothly integrate with the existing Web Service enabled Service Oriented Architecture,
- Providing an infrastructure to reduce integration problems that can occur as a result of misunderstandings on the meaning of exchanged data patterns,
- Extracting implicit knowledge from the domain data which is scattered through different source systems

High-level view of the proposed system is represented in Figure 5.1:

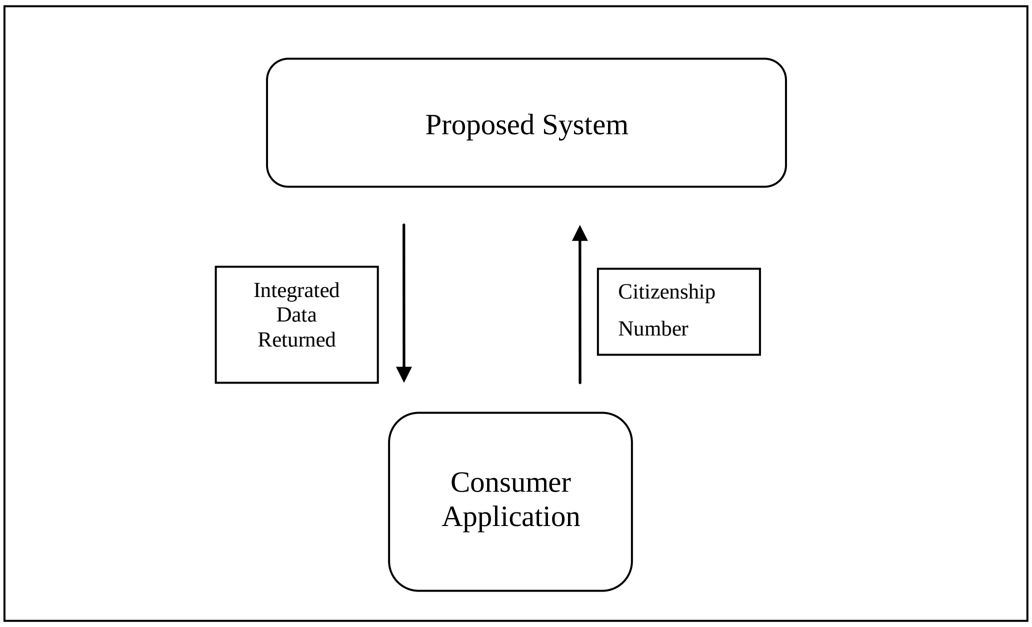


Figure 5.1 High-level view of the proposed system.

5.2. Semantic Integration Methodology

The central building block of the semantic integration system is the domain ontology. A new ontology is built from scratch instead of using one of the existing ontologies like ENIO [38]. Even similar concepts exists, ontologies modelling the same domain may have dissimilarities very often. Since the main aim in this work is intra-enterprise systems, it is preferred to build a dedicated ontology.

The initial architecture consists of standard web services and they are required to be converted to semantically enabled correspondents. Two main approaches exist for constructing semantic web services in the literature. These are top-down approach and bottom-up approach. Top-down approach requires rebuilding of a semantic version of each web service in order to achieve annotation. It also introduces new execution frameworks and modelling languages which will increase the development and testing costs. In contrast, bottom-up approach provides mechanisms for mapping of existing WSDL components with semantic concepts [10]. It is based on technologies already known by the Web Service community. So it can support relatively easy integration with existing technologies. Domain models can be externalized which allow decomposition of WSDL definitions and semantic annotation definitions. Domain models can be built on any ontology language, which will give flexibility to ontology development. Enterprises tend to possess more integrated, low cost, easily adaptable software systems, so a bottom-up approach is chosen to build semantic web services. SAWSDL [4] specification is preferred for the web service annotation standard because it is the newest W3C recommendation amongst bottom-up approaches.

A central transformation repository and a transformation web service are proposed as part of the semantic integration framework. There is no standard representation for schema mapping transformations in the SAWSDL specification. *liftingSchemaMapping* or *loweringSchemaMapping* attributes may point to any string. Uniform Resource Identifiers (URIs) are frequently used for this purpose. Client processor of the SAWSDL document reads the file pointed to by the URI and performs the transformation itself [31]. Storing transformations in a transformation repository increases reusability, security and modularity. This repository can supply web services so transformations can be performed in a central place. This usage is more suitable in an Enterprise SOA Architecture where interoperability is achieved by web services.

An RDF-Store is implemented as part of the proposed system. Information collected from service providers are first transformed into RDF triples and then stored in this RDF-Store. When all the necessary data is collected for the given request, the merged data is pulled out by executing semantic query statements. The RDF-Store data is erased after the execution of a request. RDF-Store implementations can be categorized into two with regard of data storage technique: Persistent RDF-Stores and in-memory RDF Stores. Since the data is erased and the RDF-Store only holds the data which is collected for the current request, an in-memory RDF-Store implementation is preferred for the framework. The intention is to speed up insertion and querying processes.

5.3. System Design and Implementation

The whole integration system implementation can be divided in two phases: setup phase and execution phase.

In the setup phase, enterprise software systems are modified so that semantic technologies based integration can occur. First, WSDL documents of the source system web services are identified. Domain ontology is created to provide the basis for semantic operations. Necessary transformations for the integration are defined using semantic model elements and response XML structure of WSDL documents. Source system web services are semantically annotated with transformations so that SAWSDL documents are created. A high-level representation of the setup phase is shown in Figure 5.2. Numbers in circles show the creation order of system elements:

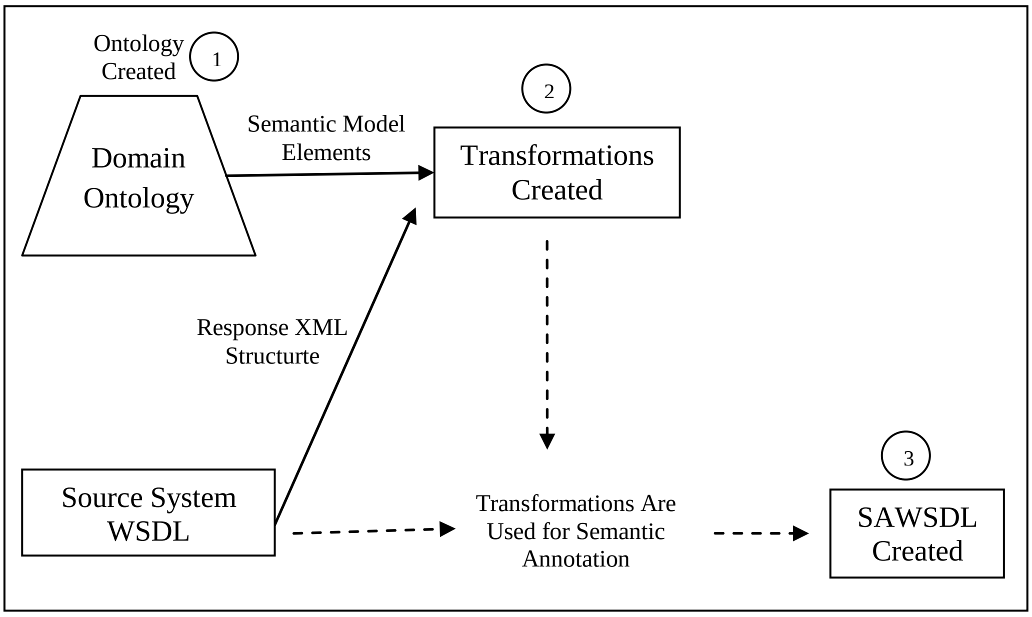


Figure 5.2 A high-level representation of the setup phase.

After the completion of the setup phase, semantic data integrator process is ready to execute. As the initial step, the domain ontology is loaded to the in-memory RDF-Store and SAWSDL document of each service is parsed to extract transformations. During the execution phase, data is gathered from the source system web services and transformed according to the needs of the consumer application, made to persist in the in-memory RDF-Store and semantically queried to capture the collected data and possible inferences. The result of the query is served to the consumer. A high-level representation of the execution phase is shown in Figure 5.3. Numbers in circles show the execution order. Curved arrows represent automatically executing operations:

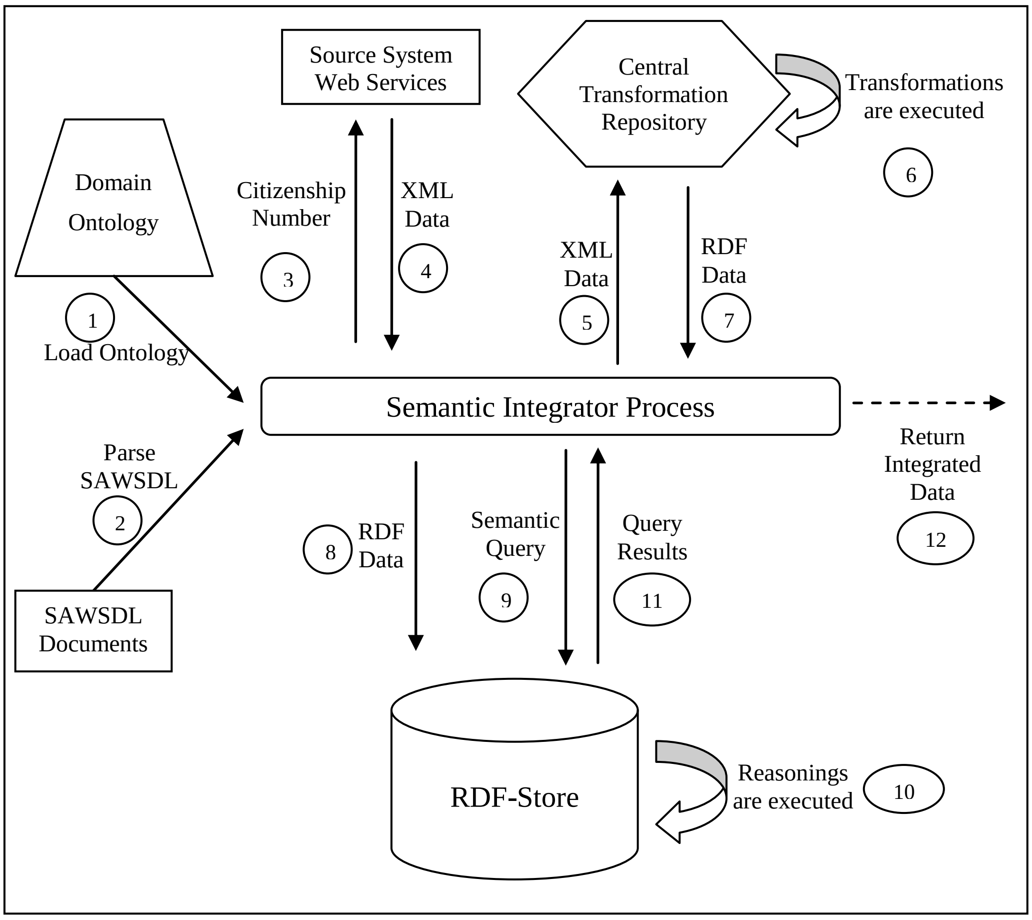


Figure 5.3 A high-level representation of the execution phase.

5.3.1 Setup Phase

In this section, detailed information about the setup phase is given.

5.3.1.1 Identification of the Source Systems

As a first step, the source systems which produce the required data should be identified. In the scenario, the enterprise integration infrastructure is web service-enabled Service Oriented Architecture, so all the systems are assumed to supply a web service for their data. The WSDL files are gathered from four different sources. Definitions of source system web services are shown in Table 5.1:

Table 5.1 Definitions of source system web services

Web Service Name	**Web Service Definition**
GetCustomerNames	provides the first name, middle name and last name of a customer.
GetCustomerAddress	provides the address information of a customer.
GetCustomerValue	provides the total of balance for the deposit accounts of the customer and the profitability of the customer for the enterprise.
GetCustomerStatus	provides information on the relationship between the customer and the enterprise.

A section of WSDL file defining the *GetCustomerNames* web service is shown in Figure 5.4:

Figure 5.4 continued

```
<wsdl:types>
<xsd:schema targetNamespace="http://127.0.0.1:8080/GetCustName/">
  <xsd:element name="NamesRequest">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="citizenshipNo" type="xsd:string" minOccurs="1"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="NamesResponse">
    <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="firstname" type="xsd:string" minOccurs="1" />
         <xsd:element name="middlename" type="xsd:string" minOccurs="0">
         </xsd:element>
```

```
              <xsd:element name="lastname" type="xsd:string" minOccurs="1"></xsd:element>

            </xsd:sequence>

          </xsd:complexType>

        </xsd:element>

      </xsd:schema>

    </wsdl:types>
```

Figure 5.4 WSDL file defining the GetCustomerNames web service.

5.3.1.2 Construction of the Domain Ontology

Basic construct of the semantic integration system is the domain ontology. The ontology contains necessary classes and class hierarchies, properties and property hierarchies, the relations between classes and properties, data validation rules, inference rules to produce asserted statements. All the other semantic elements of the system, like RDF-Store and web service annotations refer to the ontology.

First the attributes that will be a part of the model are identified. The needs of the consuming application and the information that are present in the source systems are analysed for this purpose. The attributes are grouped into reasonable classes. Generally the attributes of the same class is supplied by the same source system but the system allows different sources to provide information for the same class. A high-level representation of domain ontology classes is shown in Figure 5.5:

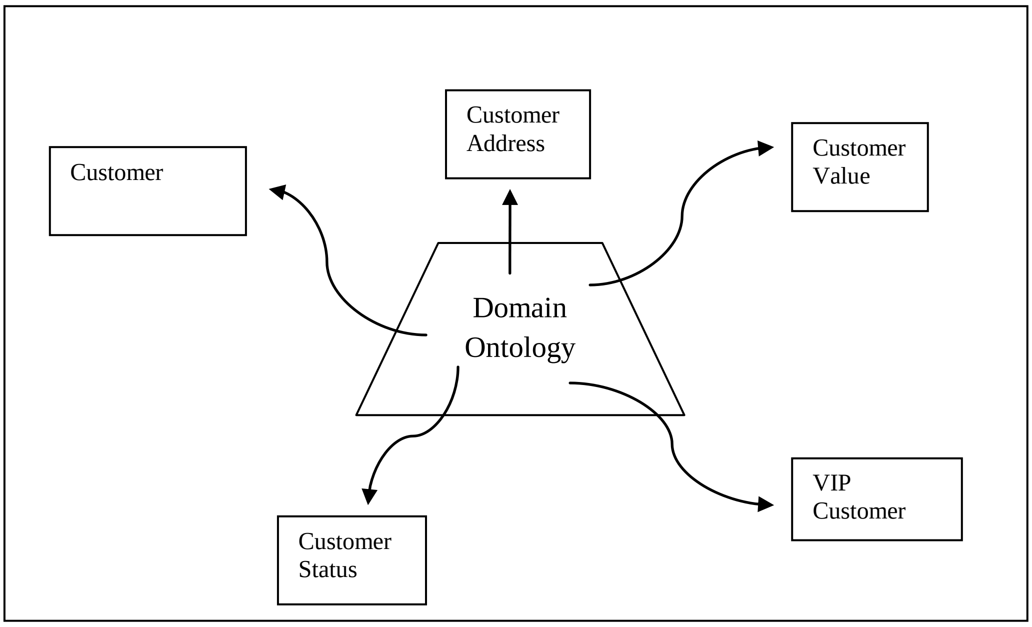


Figure 5.5 A high-level representation of domain ontology classes.

Is-a relationship graph of the semantic model is shown in Figure 5.6:

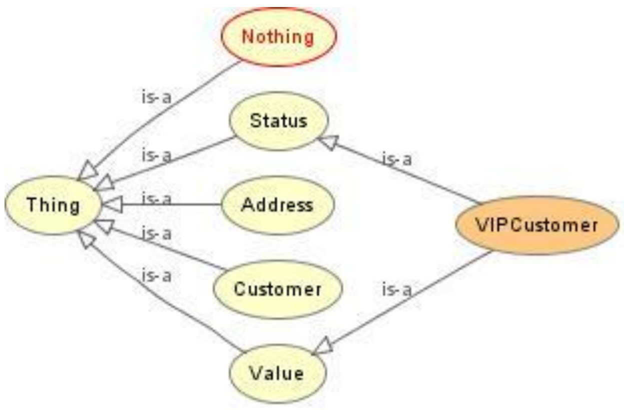


Figure 5.6 Is-a relationship graph of the semantic model.

Customer class represents the fundamental demographics information for the customer, *Address* class represents the address of the customer, *Value* class represents the value of the customer assets for the enterprise, *Status* class represents the formal relations of the enterprise and customer, *VIPCustomer* class represents if the customer is eligible to be a VIP customer or not. There are two predefined class identifiers in the model: the classes *owl:Nothing* and *owl:Thing*. *owl:Thing* class extension represents the set of all individuals. *owl:Nothing* class extension represents the empty set. Consequently, every class in the model is a subclass of *owl:Thing* class and *owl:Nothing* class is a subclass of every class [19].

For modelling of attributes, *owl:DatatypeProperty* is used. Domain value of the properties point to the classes they belong to and range values are simple literal types like *string*, *int*. Domain and range values of model properties are shown in Table 5.2:

Table 5.2 Domain and range values of model properties.

Data Property	**Domain**	**Range**
hasName	Customer	String
hasLastName	Customer	String
hasAddressType	Address	Int
hasCountryCode	Address	String
hasCityCode	Address	Int
hasCountyName	Address	String
hasDistrictName	Address	String
hasStreetName	Address	String
hasApartmentName	Address	String
hasFlatNo	Address	Int
hasBalance	Value	String
hasProfitability	Value	String
isOnBlackList	Status	String
hasAgreementNo	Status	String
hasBranchName	Status	String

Namespace declarations used in Notation 3 (N3) representations of RDF triples of the domain model are shown in Table 5.3:

Table 5.3 Namespace declarations

Namespace declarations
@prefix cust: <http://www.semanticweb.org/ontologies/2010/0/customer.owl#>
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
@prefix owl: <http://www.w3.org/2002/07/owl#>
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>

Notation 3 (N3) representations of RDF triples for *hasAddressType* property of the *Address* class are shown in Table 5.4:

Table 5.4 Notation 3 (N3) representation for *hasAddressType* property

Subject	**Predicate**	**Object**
cust:hasAddressType	rdf:type	owl:DatatypeProperty
cust:hasAddressType	rdfs:domain	cust:Address
cust:hasAddressType	rdfs:range	xsd:int

OWL models can also be expressed as XML documents in RDF-XML format. The RDF/XML representation of the RDF triples for *hasAddressType* property of the *Address* class is shown in Figure 5.7:

```
<owl:DatatypeProperty rdf:about="#hasAddressType">

    <rdfs:domain rdf:resource="#Address"/>

    <rdfs:range rdf:resource="&xsd;int"/>

 </owl:DatatypeProperty>
```

Figure 5.7 RDF/XML representation of the RDF triples for *hasAddressType* property.

Property Restriction feature of OWL is used to describe rules which infer new relations by utilizing asserted triples. *VIPCustomer* class is constructed by the intersection of a restriction class containing individuals whose *hasBalance* property has value *"High"* and another restriction class containing individuals whose *isOnBlackList* property has value *"False"*. RDF-Graph representation of *VIPCustomer* class is shown in Figure 5.8:

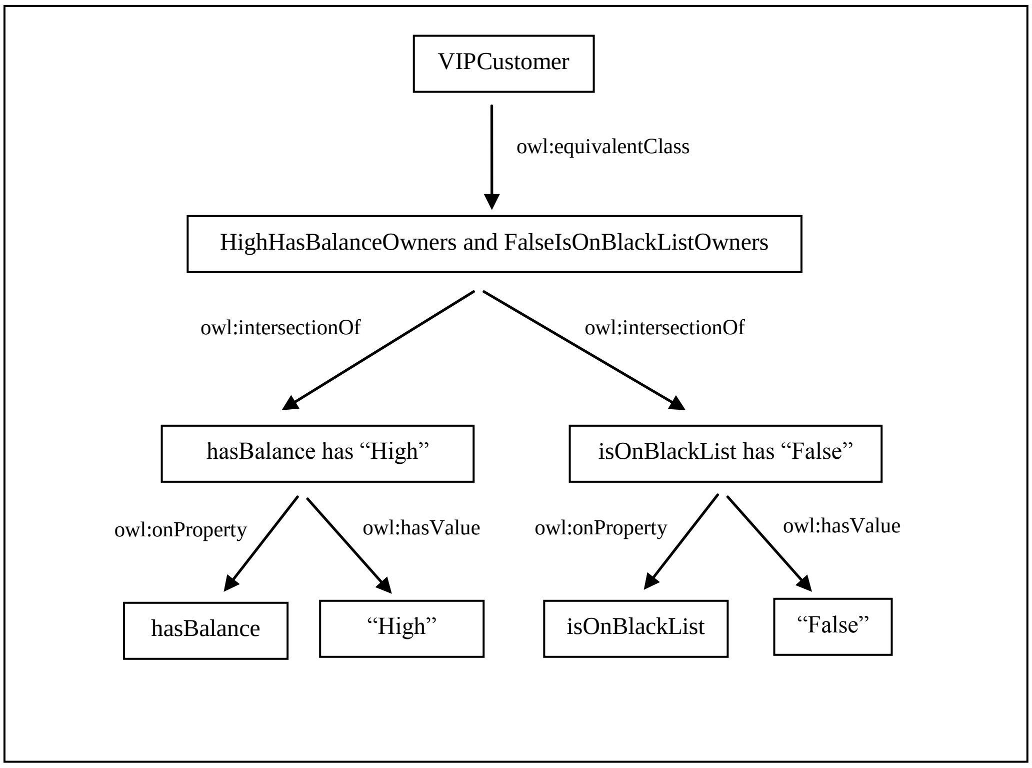
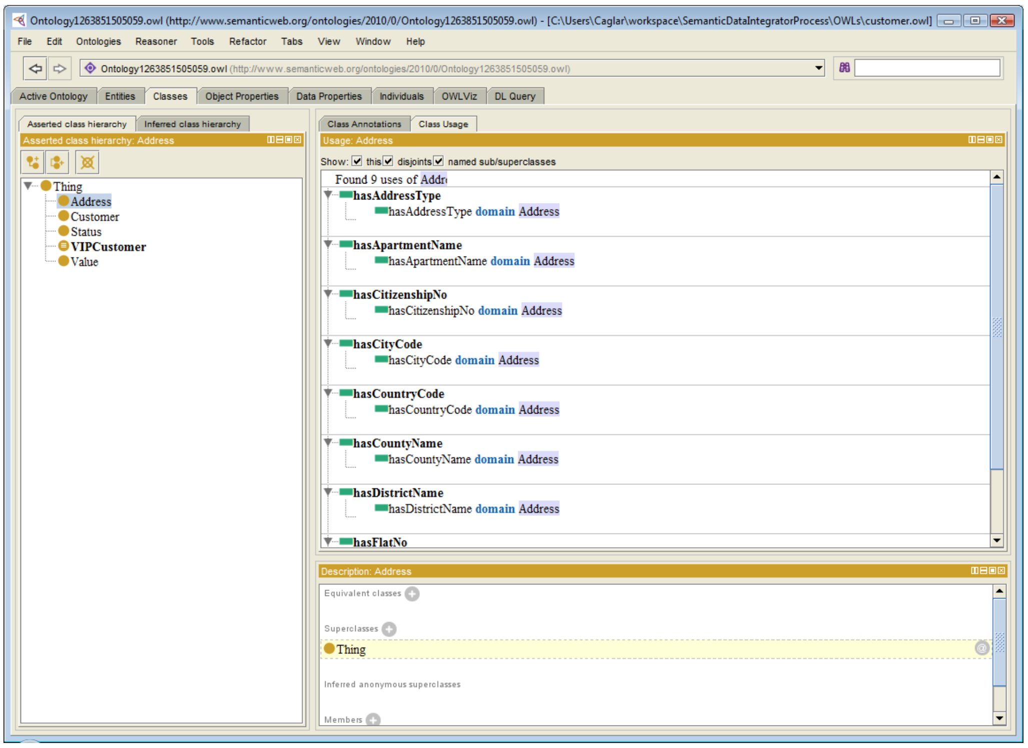


Figure 5.8 RDF-Graph representation of *VIPCustomer* class.

Ontology construction is done with the Protégé Ontology Editor. Protégé is a tool which implements an extensive set of knowledge-modelling features and actions that can be used in the creation, modification and visualization of semantic models in different formats [3]. A snapshot of constructed customer ontology in Protégé Ontology Editor is shown in Figure 5.9:



Ontology1263851505059.owl (http://www.semanticweb.org/ontologies/2010/0/Ontology1263851505059.owl) - [C:\Users\Caglar\workspace\SemanticDataIntegratorProcess\OWLs\customer.owl]
File Edit Ontologies Reasoner Tools Refactor Tabs View Window Help
Active Ontology Entities Classes Object Properties Data Properties Individuals OWLViz DL Query
Asserted class hierarchy Inferred class hierarchy
Asserted class hierarchy: Address
Class Annotations Class Usage
Usage: Address
Show: this disjoints named sub/superclasses
Description: Address
Equivalent classes
Superclasses
Inferred anonymous superclasses
Members

Thing
- Address
- Customer
- Status
- VIPCustomer
- Value

Found 9 uses of Addr

- hasAddressType
  - hasAddressType domain Address
- hasApartmentName
  - hasApartmentName domain Address
- hasCitizenshipNo
  - hasCitizenshipNo domain Address
- hasCityCode
  - hasCityCode domain Address
- hasCountryCode
  - hasCountryCode domain Address
- hasCountyName
  - hasCountyName domain Address
- hasDistrictName
  - hasDistrictName domain Address
- hasFlatNo

Superclasses: Thing

Figure 5.9 Customer ontology construction using Protégé

The benefits of the domain ontology is not only limited to the integration environment. It can be used as a central referral resource for this domain by all the other software departments and business reporting units in the enterprise.

Web Ontology Language (OWL) is used as the ontology language because it has sufficient expressive power and tool support required by an enterprise ontology.

RDF/XML representation of whole semantic model of the customer ontology is represented in Appendix A.

5.3.1.2 Semantic Annotation of Source System Web Services

Semantic annotation of the web services is the initial step for the conversion of the legacy data models into the semantic data model. Annotation constructs the connection between syntax-based xml representation of data and the semantically enriched representation of data.

In this thesis work, a bottom-up approach is applied to build semantic web services. The basic motivation of the bottom-up approaches is using the existing WSDL files to create semantic annotations.

Semantic Annotations for WSDL (SAWSDL) [4] specification is used as the annotation standard because it is the newest and most widely used bottom-up paradigm and has the strongest tool/API support.

SAWSDL specification allows annotating any part of the WSDL file. For a semantic data integration goal, “wsdl:types” can be thought as the best component to add semantics because request/response data definitions are made here. In this thesis work, since the request objects do not need to enter a transformation operation, only the response objects are annotated. In SAWSDL specification, when an XML tag is annotated, all the elements under that top-most element can be reached from the transformation file used in the annotation. So as to establish a standard throughout the integration system, it is preferred to annotate only the top level of the response object.

SAWSDL specification allows for three different types of annotation extension attribute: *liftingSchemaMapping*, *modelReference* and *loweringSchemaMapping*. Among them only *liftingSchemaMapping* is required to be used for this work. This attribute points to a transformation file which contains the mappings to convert the syntactic XML data to semantic RDF data. *modelReference* is not used because its functionality can be covered by a *liftingSchemaMapping* transformation. *loweringSchemaMapping* is not needed because RDF-to-XML data conversion is not performed by web service executions in this system. Semantic annotation for a source system is represented in Figure 5.10:

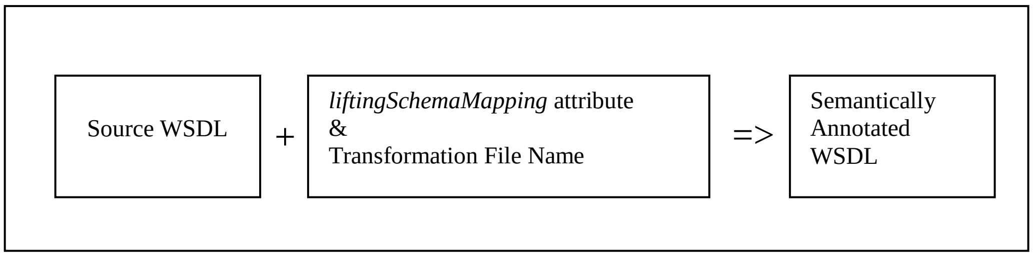


Figure 5.10 Semantic annotation for a source system.

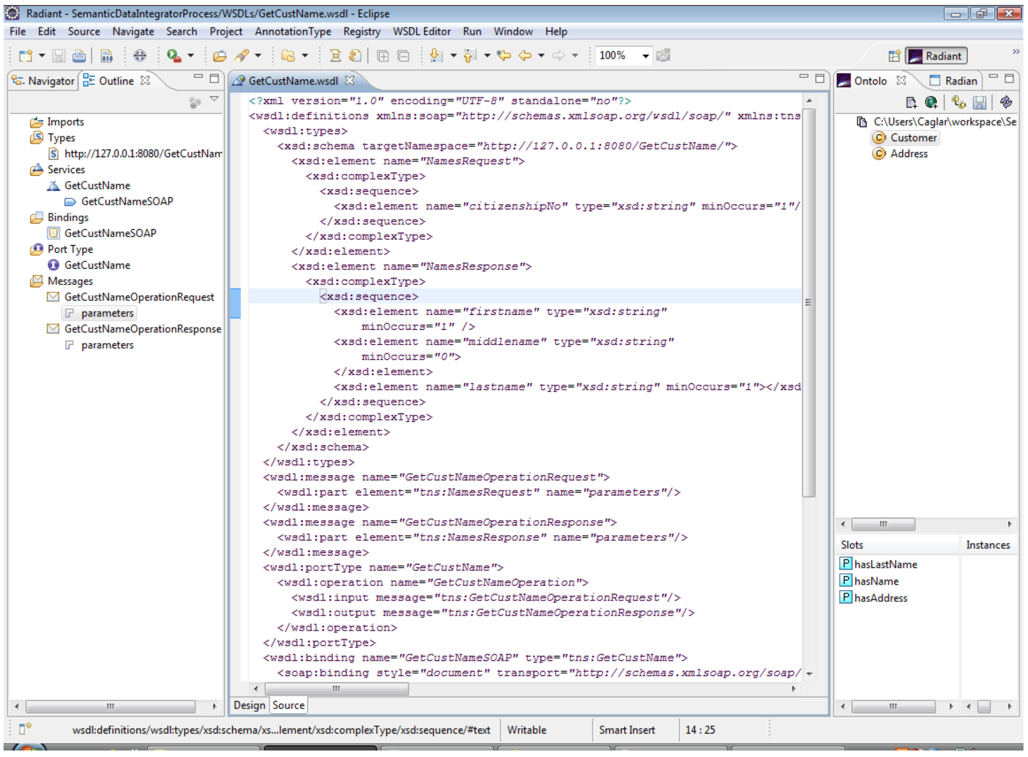
SAWSDL document for *GetCustomerNames* web service is shown in Figure 5.11:

```
          <!--   response object    !-->

        <xs:element name="NamesResponse"
sawsdl:liftingSchemaMapping="NamesResponseTransformation.xslt">

           <xs:complexType>

             <xs:sequence>

               <xs:element minoccurs="1" name="firstname" />

                <xs:element minoccurs="0" name="middlename" type="xs:string"/>

                <xs:element minoccurs="1" name="lastname" type="xs:string"/>

             </xs:sequence>

           </xs:complexType>

         </xs:element>
```

Figure 5.11 SAWSDL document for *GetCustomerNames* web service

There is some number of graphical tools for creation of SAWSDL documents from standard WSDL documents. Radiant tool [5], developed by Large Scale Distributed Information Systems Lab (LSDIS) in University of Georgia, is used in this implementation. This tool is a plug-in for the Eclipse IDE [6] and provides a useful

GUI for annotation of WSDL files using ontologies. A screenshot of the Radiant tool is shown in Figure 5.12:


Radiant - SemanticDataIntegratorProcess/WSDLs/GetCustName.wsdl - Eclipse
File Edit Source Navigate Search Project AnnotationType Registry WSDL Editor Run Window Help
Navigator
Outline
Imports
Types
http://127.0.0.1:8080/GetCustNam
Services
GetCustName
GetCustNameSOAP
Bindings
GetCustNameSOAP
Port Type
GetCustName
Messages
GetCustNameOperationRequest
parameters
GetCustNameOperationResponse
parameters
GetCustName.wsdl
Ontolo
Radian
C:\Users\Caglar\workspace\Se
Customer
Address
Slots
Instances
hasLastName
hasName
hasAddress
Design
Source

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns
  <wsdl:types>
    <xsd:schema targetNamespace="http://127.0.0.1:8080/GetCustName/">
      <xsd:element name="NamesRequest">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="citizenshipNo" type="xsd:string" minOccurs="1"/
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="NamesResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="firstname" type="xsd:string"
                minOccurs="1" />
            <xsd:element name="middlename" type="xsd:string"
                minOccurs="0">
            </xsd:element>
            <xsd:element name="lastname" type="xsd:string" minOccurs="1"></xsd
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:schema>
  </wsdl:types>
  <wsdl:message name="GetCustNameOperationRequest">
    <wsdl:part element="tns:NamesRequest" name="parameters"/>
  </wsdl:message>
  <wsdl:message name="GetCustNameOperationResponse">
    <wsdl:part element="tns:NamesResponse" name="parameters"/>
  </wsdl:message>
  <wsdl:portType name="GetCustName">
    <wsdl:operation name="GetCustNameOperation">
      <wsdl:input message="tns:GetCustNameOperationRequest"/>
      <wsdl:output message="tns:GetCustNameOperationResponse"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="GetCustNameSOAP" type="tns:GetCustName">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/
```

wsdl:definitions/wsdl:types/xsd:schema/xs...lement/xsd:complexType/xsd:sequence/#text | Writable | Smart Insert | 14 : 25

Figure 5.12 Semantic annotation of WSDL documents using Radiant

5.3.1.3 Defining Data Transformation Mappings

Semantically annotated web service definitions point to transformation files that contain the necessary mappings. The main purpose of the transformations is to convert the syntactic XML data produced by source systems into the semantic model based RDF data.

Source system attributes can be processed in three ways under a transformation operation:

- The source attribute can be totally ignored. Source systems do not have to return data dedicated to the consumer application, so unrelated attributes are not transformed into RDF. This is one of the advantages of the bottom-up approach against top-down approach, where the web service should be completely translated. This advantage reduces the needed time and cost for adaptation of semantic technology into an enterprise software system.

- The source attribute can be directly mapped to an ontology attribute. This can happen when the meaning and format of the source attribute is exactly the same with the target attribute. For example the source attribute *lastname* can be directly mapped to the ontology attribute *hasLastName*, if they both mean the family name of a customer that is of type string.

- The source attribute goes under some transformations or concatenates with another source attribute before mapping a target attribute. This is the most frequently encountered situation because different source systems generally have different attributes for the concepts of the same domain. For example the source system may have two different attributes for the names (other than last name) of a customer: first name and middle name. But the enterprise-wide usage may be having only one attribute for the names of the customer, where the first name and the middle name are hold concatenated with a space between them. As the domain ontology relies on the enterprise-wide most accepted usage, a transformation should occur between the source and semantic model.

SAWSDL specification does not enforce a strict transformation/mapping standard. XSL Transformations (XSLT) [7] is preferred as the transformation language. XSLT is used for transformation of one XML document into another XML document with a different schema structure. Since the RDF data and semantic model can be serialized as RDF/XML format, XSLT is a convenient language for semantic data transformation operations. A transformation expressed in XSLT describes rules for transforming source XML structure into target XML structure. XSLT makes use of the functions and expression language defined by XPath [33] for processing.

5.3.1.3.1 GetCustomerNames Web Service Transformation

GetCustomerNames service returns the first name, middle name and last name of a customer. However, the domain ontology holds the names in just two attributes, name and last name, instead of three. As a result, a transformation is necessary to format the source data. Transformation/mapping definitions for the *GetCustomerNames* web service is represented in Table 5.5:

Table 5.5 Transformations/mappings for the *GetCustomerNames* web service

Source Attribute	**Ontology Attribute**	**Transformation/Mapping**
firstname	hasName	Concatenate (firstname,’ ‘, middlename) => hasName
middlename		
lastname	hasLastName	lastname => hasLastName

Part of the XSLT transformation file for *GetCustomerNames* web service is shown in Figure 5.13:

```
<owl:Thing rdf:about= "#{citizenshipNo}"   >

 <rdf:type rdf:resource="#Customer"/>

 <hasName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">

 <xsl:value-of select="concat(firstname , ' ', middlename)"/>

 </hasName>

   <hasLastName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">

 <xsl:value-of select="lastname"/>

 </hasLastName>

 <hasCitizenshipNo rdf:datatype="http://www.w3.org/2001/XMLSchema#string">

 <xsl:value-of select="citizenshipNo"/>

 </hasCitizenshipNo>

</owl:Thing>
```

Figure 5.13 XSLT transformation file for *GetCustomerNames* web service.

5.3.1.3.2 GetCustomerAddress Web Service Transformation

GetCustomerAddress service returns the address information of a customer. Response data coming from this service should also go under transformation for some reasons. The Address Type attribute, which demonstrates the type of the address as home, work or factory, is enumerated differently in the source system and central ontology. The other transformation need is because of enterprise-wide standardization. The consuming applications need to add some keywords to some attributes of the address information for increased comprehensibility. For example *"str."* keyword is added at the end of street name, "apt." keyword is added at the end of apartment name. Transformation/mapping definitions for the *GetCustomerAddress* web service is represented in Table 5.6:

Table 5.6 Transformations/mappings for the *GetCustomerAddress* web service

Source Attribute	**Ontology Attribute**	**Transformation/Mapping**
addresstype	hasAddressType	If (addresstype = H) => 1 If (addresstype = W) => 2 If (addresstype = F) => 3
countrycode	hasCountryCode	countrycode => hasCountryCode
citycode	hasCityCode	citycode => hasCityCode
countyname	hasCountyName	countyname => hasCountyName
districtname	hasDistrictName	districtname => hasDistrictName
streetname	hasStreetName	Concatenate(streetname,' ','str.') => hasStreetName
apartmentname	hasApartmentName	Concatenate(apartmentname,' ','apt.') => hasApartmentName
flatno	hasFlatNo	flatno => hasFlatNo

Part of the XSLT transformation file for *GetCustomerAddress* web service is shown in Figure 5.14:

Figure 5.14 continued

```
<owl:Thing rdf:about= "#{citizenshipNo}"   >

 <rdf:type rdf:resource="#Address"/>

 <hasAddressType rdf:datatype="http://www.w3.org/2001/XMLSchema#int">

      <xsl:choose>

            <xsl:when test="addresstype = 'H'">  1 </xsl:when>

            <xsl:when test="addresstype = 'W'">  2 </xsl:when>

            <xsl:when test="addresstype = 'F'">  3 </xsl:when>

            <xsl:otherwise> 1 </xsl:otherwise>

      </xsl:choose>

 </hasAddressType>

   <hasStreetName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">

      <xsl:value-of select="concat( streetname , ' str.')"/>

 </hasStreetName>

 <hasApartmentName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">

      <xsl:value-of select="concat( apartmentname , ' apt.')"/>

 </hasApartmentName>

 <hasCountryCode rdf:datatype="http://www.w3.org/2001/XMLSchema#string">

      <xsl:value-of select="countrycode"/>

 </hasCountryCode>

 <hasCityCode rdf:datatype="http://www.w3.org/2001/XMLSchema#int">

      <xsl:value-of select="citycode"/>

 </hasCityCode>
```

```
          <hasCountyName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
             <xsl:value-of select="countyname"/>
          </hasCountyName>
          <hasDistrictName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
             <xsl:value-of select="districtname"/>
          </hasDistrictName>
          <hasFlatNo rdf:datatype="http://www.w3.org/2001/XMLSchema#int">
             <xsl:value-of select="flatno"/>
          </hasFlatNo>
          <hasCitizenshipNo rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
             <xsl:value-of select="citizenshipNo"/>
          </hasCitizenshipNo>
        </owl:Thing>
```

Figure 5.14 XSLT transformation file for *GetCustomerAddress* web service.

5.3.1.3.3 GetCustomerValue Web Service Transformation

GetCustomerValue service returns the total of balance for the deposit accounts of the customer and the profitability of the customer for the enterprise. Balance and profitability data is returned as decimal values like 75.000, however consumer application expects aggregate information like *"Medium"*. As a result a transformation is required. Transformation/mapping definitions for the *GetCustomerValue* web service is represented in Table 5.7:

Table 5.7 Transformations/mappings for the *GetCustomerValue* web service

Source Attribute	**Ontology Attribute**	**Transformation/Mapping**
balance	hasBalance	If (balance > 100.000) => “High” If (balance > 10.000) => “Medium” If (balance > 1.000) => “Low”
profitability	hasProfitability	If (profitability > 100.000) => “High” If (profitability > 10.000) => “Medium” If (profitability > 1.000) => “Low”

Part of the XSLT transformation file for *GetCustomerValue* web service is shown in Figure 5.15:

Figure 5.15 continued

```
<owl:Thing rdf:about= "#{citizenshipNo}"   >

<rdf:type rdf:resource="#Value"/>

<hasBalance rdf:datatype="http://www.w3.org/2001/XMLSchema#string">

    <xsl:choose>

        <xsl:when test="balance &gt; 100000">High</xsl:when>

        <xsl:when test="balance &gt; 10000">Middle</xsl:when>

        <xsl:when test="balance &gt; 1000">Low</xsl:when>

        <xsl:otherwise> None </xsl:otherwise>

    </xsl:choose>

</hasBalance>

<hasProfitability rdf:datatype="http://www.w3.org/2001/XMLSchema#string">

    <xsl:choose>
```

```
                    <xsl:when test="profitability &gt; 100000">High</xsl:when>

                    <xsl:when test="profitability &gt; 10000">Middle</xsl:when>

                    <xsl:when test="profitability &gt; 1000">Low</xsl:when>

                    <xsl:otherwise> None </xsl:otherwise>

                </xsl:choose>

            </hasProfitability>

          </owl:Thing>
```

Figure 5.15 XSLT transformation file for *GetCustomerValue* web service

5.3.1.3.4 GetCustomerStatus Web Service Transformation

GetCustomerStatus service returns information on the relationship between the customer and the enterprise. Response data includes the name of the branch that customer belongs, service agreement number and if the customer is on black list or not. Block list information sometimes cannot be supplied in the source system, however consumer application expects a value all the time. Transformation always assigns the "False" value if the source system does not return a value. Transformation/mapping definitions for the *GetCustomerStatus* web service is represented in Table 5.8:

Table 5.8 Transformations/mappings for the *GetCustomerStatus* web service

Source Attribute	**Ontology Attribute**	**Transformation/Mapping**
isonblacklist	isOnBlackList	If (isonblacklist = " ") => "False" Else isonblacklist => isOnBlackList
branchname	hasBranchName	branchname => hasBranchName
agreementno	hasAgreementNo	agreementno => hasAgreementNo

Part of the XSLT transformation file for *GetCustomerStatus* web service is shown in Figure 5.16:

```
    <owl:Thing rdf:about= "#{citizenshipNo}"   >

    <rdf:type rdf:resource="#Status"/>

    <isOnBlackList rdf:datatype="http://www.w3.org/2001/XMLSchema#string">

        <xsl:choose>

            <xsl:when test="isonblacklist = ' '">  False </xsl:when>

            <xsl:otherwise> <xsl:value-of select="isonblacklist"/> </xsl:otherwise>

        </xsl:choose>

    </isOnBlackList>

    <hasBranchName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">

        <xsl:value-of select="branchname"/>

    </hasBranchName>

    <hasAgreementNo rdf:datatype="http://www.w3.org/2001/XMLSchema#string">

        <xsl:value-of select="agreementno"/>

    </hasAgreementNo>

  </owl:Thing>
```

Figure 5.16 XSLT transformation file for *GetCustomerStatus* web service

Full transformation files for four web services can be found in Appendix B.

5.3.2 Execution Phase

In this section, detailed information about the execution phase is given.

5.3.2.1 Semantic Data Integrator Process

Semantic Data Integrator Process is the main unit in the system. This process orchestrates source system service calls, transformation service calls, in-memory RDF-Store persistence, execution of semantic query statements and constructing the response to the consumer application.

The process communicates with the consumer application by a web service. This *Integrator Service* takes the citizenship number of the customer in request and returns the consolidated, transformed, standardized and possibly enriched customer data in response.

In the start-up, integrator process loads the domain ontology to the in-memory RDF-Store and parses SAWSDL documents of each service to extract transformations. As soon as the source XML data is transformed for each service, resulting semantic data is united with the domain ontology statements to allow semantic querying and reasoning before construction of the response. Process flow diagram of the *Semantic Data Integrator Process* is represented in Figure 5.17:

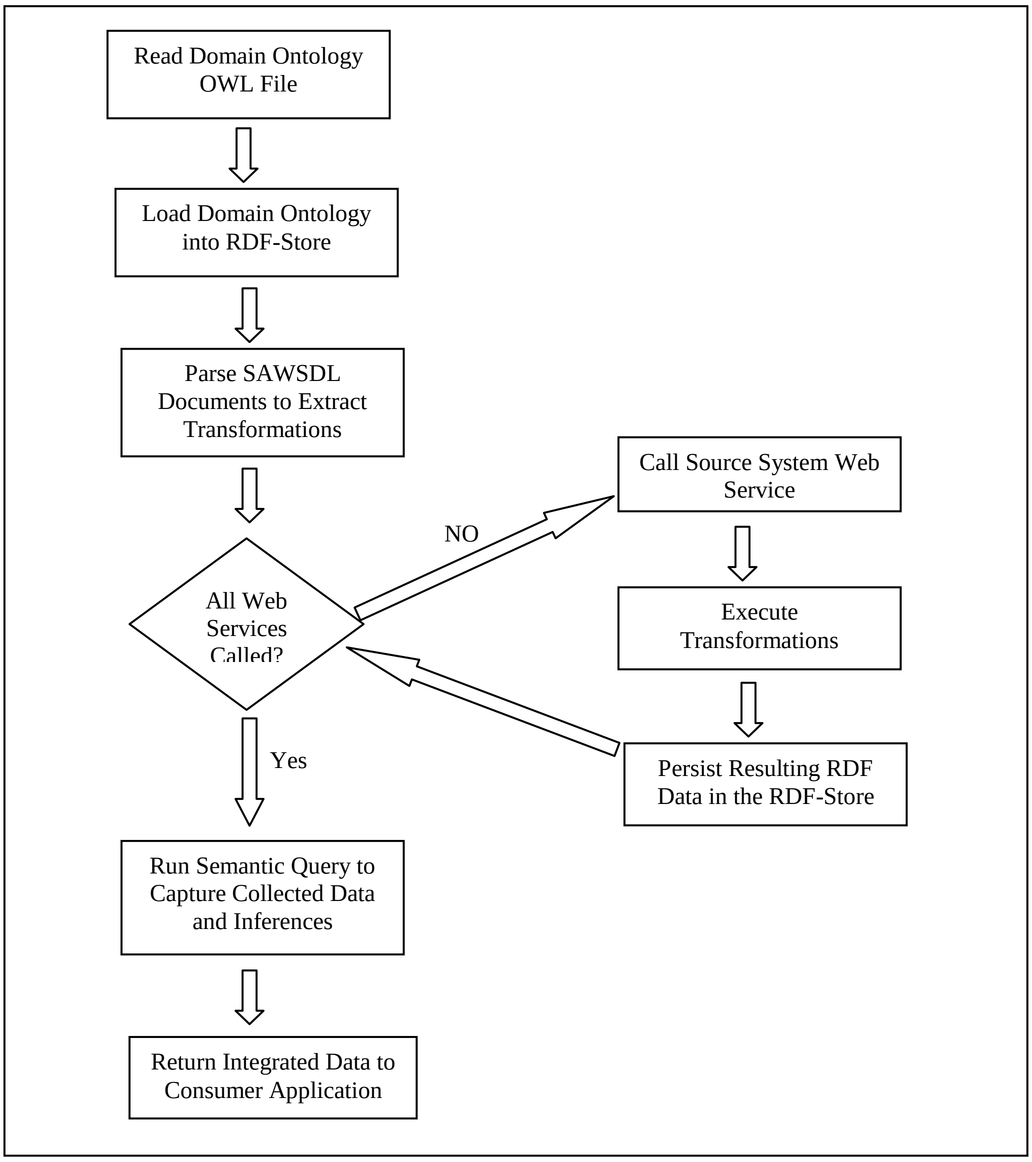

Read Domain Ontology OWL File
Load Domain Ontology into RDF-Store
Parse SAWSDL Documents to Extract Transformations
All Web Services Called?
NO
Call Source System Web Service
Execute Transformations
Persist Resulting RDF Data in the RDF-Store
Yes
Run Semantic Query to Capture Collected Data and Inferences
Return Integrated Data to Consumer Application

Figure 5.17 Process flow diagram of the *Semantic Data Integrator Process.*

In the development of *semantic data integrator process*, *Eclipse IDE for Java EE Developers v1.2.0* [6] is used as the IDE platform, *J2EE v1.6* [34] as the programming language and *Apache Tomcat v6.0* [35] as the web server.

5.3.2.2 Extraction of Transformations from SAWSDL

When a new request is made against the integrator process from the consumer application, the first operation is to extract the transformation information from the SAWSDL documents of the source web services. Transformations file names are located in the *liftingSchemaMapping* attribute of the top-level element of response object. To get the value of the attribute, SAWSDL document should be parsed.

EasySAWSDL API [36] is used to parse the SAWSDL documents. It is an extension for EasyWSDL library which is a powerful WSDL parsing library.

After *liftingSchemaMapping attribute* is found, the value of the attribute is stored in the process to use as the transformation file name.

5.3.2.3 Data Conciliation and Transformation

The *integrator process* initiates collecting data from source system web services after transformation information is extracted from SAWSDL files. Response of each web service is gathered by the integrator process and an XML block is prepared by using the response data and the response object XML schema. The *citizenship number* attribute, which is the input parameter for the *integrator service*, is also added to the XML block. This attribute is the key field for all the information belonging to a specified customer and information is integrated with the help of this key field. In each transformation, *citizenship number* is placed as URI name of the individual so that all RDF triples belonging to the same customer is aggregated under the same individual. Constructed XML blocks are sent to the *Central Transformation Repository* for XML-to-RDF conversion.

All the transformations are performed by the *Central Transformation Repository* which is a separate unit independent of the *integrator process*. *Central Transformation Repository* stores all the transformation definition files and provides a *transformation service* to communicate with the other systems. *Transformation service* takes the name of the transformation file and the XML block to be transformed as input. Corresponding XSLT file is located in the *transformation repository* and the XML block is converted into RDF/XML block by executing the transformation file. Java API for XML Processing (JAXP) [9] library is used for the execution of transformations. The architecture of *Central Transformation Repository* is shown in Figure 5.18:

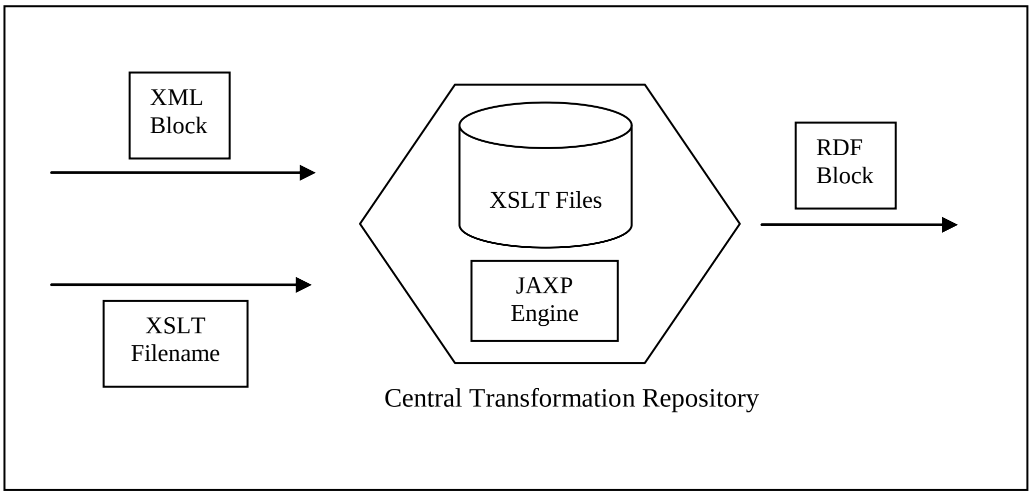


Figure 5.18 Architecture of *Central Transformation Repository*.

5.3.2.3.1 Transformation Execution for GetCustomerNames

A sample XML block which will be transformed for *GetCustomerNames* web service is shown in Figure 5.19:

```
<NamesResponse>

     <firstname>    Mehmet </firstname>

    <middlename>    Ali </middlename>

    <lastname>    Kaya  </lastname>

    <citizenshipNo>   16534878800  </citizenshipNo>

</NamesResponse>
```

Figure 5.19 A sample XML block for *GetCustomerNames* service.

A section of resulting RDF block after execution of the XSLT transformation to the XML block for *GetCustomerNames* service is shown in Figure 5.20:

```
<owl:Thing rdf:about="#16534878800">

  <rdf:type rdf:resource="#Customer"/>

 <hasName rdf:datatype="string">Mehmet Ali</hasName>

 <hasLastName rdf:datatype="string">Kaya</hasLastName>

 <hasCitizenshipNo rdf:datatype="string">16534878800</hasCitizenshipNo>

</owl:Thing>
```

Figure 5.20 Transformed RDF block for *GetCustomerNames* service.

Notation 3 (N3) representation of resulting RDF data triples for *GetCustomerNames* service is shown in Table 5.9:

Table 5.9 *N3* representation of RDF triples for *GetCustomerNames* service

Subject	**Predicate**	**Object**
cust:16534878800	rdf:type	cust:Customer
cust:16534878800	cust:hasName	“Mehmet Ali”
cust:16534878800	cust:hasLastName	“Kaya”
cust:16534878800	cust:hasCitizenshipNo	“16534878800”

5.3.2.3.2 Transformation Execution for GetCustomerAddress

A sample XML block will be transformed for *GetCustomerAddress* web service is shown in Figure 5.21:

Figure 5.21 continued

```
<AddressResponse>

    <addresstype> H    </addresstype>

    <countrycode>  TR   </countrycode>
```

```
        <citycode>  34    </citycode>

        <countyname>  Pendik   </countyname>

        <districtname> Barbaros    </districtname>

        <streetname>  Sedef   </streetname>

        <apartmentname>  Ocak   </apartmentname>

        <flatno>  5   </flatno>

        <citizenshipNo>  16534878800   </citizenshipNo>

    </AddressResponse>
```

Figure 5.21 A sample XML block will be transformed for *GetCustomerAddress* service.

A section of resulting RDF block after execution of the XSLT transformation to the XML block for *GetCustomerAddress* service is shown in Figure 5.22:

Figure 5.22 continued

```
  <owl:Thing rdf:about="#16534878800">

      <rdf:type rdf:resource="#Address"/>

   <hasAddressType rdf:datatype="int">  1 </hasAddressType>

   <hasStreetName rdf:datatype="string">Sedef str.</hasStreetName>

   <hasApartmentName rdf:datatype="string">Ocak apt.</hasApartmentName>

   <hasCountryCode rdf:datatype="string">TR</hasCountryCode>

   <hasCityCode rdf:datatype="int">34</hasCityCode>

   <hasCountyName rdf:datatype="string">Pendik</hasCountyName>

   <hasDistrictName rdf:datatype="string">Barbaros</hasDistrictName>

   <hasFlatNo rdf:datatype="int">5</hasFlatNo>

   <hasCitizenshipNo rdf:datatype="string">16534878800</hasCitizenshipNo>
```

```
</owl:Thing>
```

Figure 5.22 Transformed RDF block for *GetCustomerAddress* service.

Notation 3 (N3) representation of resulting RDF data triples for *GetCustomerAddress* service is shown in Table 5.10:

Table 5.10 *N3* representation of RDF triples for *GetCustomerAddress* service.

Subject	**Predicate**	**Object**
cust:16534878800	rdf:type	cust:Address
cust:16534878800	cust:hasAddressType	1
cust:16534878800	cust:hasStreetName	“Ocak apt.”
cust:16534878800	cust:hasApartmentName	“16534878800”
cust:16534878800	cust:hasCountryCode	TR
cust:16534878800	cust:hasCityCode	34
cust:16534878800	cust:hasCountyName	“Pendik”
cust:16534878800	cust:hasDistrictName	“Barbaros”
cust:16534878800	cust:hasFlatNo	5
cust:16534878800	cust:hasCitizenshipNo	“16534878800”

5.3.2.3.3 Transformation Execution for GetCustomerValue

A sample XML block will be transformed for *GetCustomerValue* web service is shown in Figure 5.23:

```
<ValueResponse>

    <balance>250000</balance>

    <profitability>90000</profitability>

    <citizenshipNo>16534878800</citizenshipNo>

</ValueResponse>
```

Figure 5.23 A sample XML block for *GetCustomerValue* service.

A section of resulting RDF block after execution of the XSLT transformation to the XML block for *GetCustomerValue* service is shown in Figure 5.24:

```
<owl:Thing rdf:about="#16534878800">

    <rdf:type rdf:resource="#Value"/>

    <hasBalance rdf:datatype="string">High</hasBalance>

    <hasProfitability rdf:datatype=" string">Middle</hasProfitability>

    <hasCitizenshipNo rdf:datatype="string">16534878800</hasCitizenshipNo>

</owl:Thing>
```

Figure 5.24 Transformed RDF block for *GetCustomerValue* service.

Notation 3 (N3) representation of resulting RDF data triples for *GetCustomerValue* service is shown in Table 5.11:

Table 5.11 *N3* representation of RDF triples for *GetCustomerValue* service.

Subject	**Predicate**	**Object**
cust:16534878800	rdf:type	cust:Value
cust:16534878800	cust:hasBalance	“High”
cust:16534878800	cust:hasProfitability	“Middle”
cust:16534878800	cust:hasCitizenshipNo	“16534878800”

5.3.2.3.4 Transformation Execution for GetCustomerStatus

A sample XML block will be transformed for *GetCustomerStatus* web service is shown in Figure 5.25:

```
<StatusResponse>

    <isonblacklist>False</isonblacklist>

    <agreementno>A45745</agreementno>

    <branchname>ISTANBUL</branchname>

    <citizenshipNo>16534878800</citizenshipNo>

</StatusResponse>
```

Figure 5.25 A sample XML block for *GetCustomerStatus* service.

A section of resulting RDF block after execution of the XSLT transformation to the XML block for *GetCustomerStatus* service is shown in Figure 5.26:

Figure 5.26 continued

```
<owl:Thing rdf:about="#16534878800">

<rdf:type rdf:resource="#Status"/>

<isOnBlackList rdf:datatype="string">False</isOnBlackList>

<hasAgreementNo rdf:datatype="string">A45745</hasAgreementNo>
```

```
<hasBranchName rdf:datatype="string">ISTANBUL</hasBranchName>

<hasCitizenshipNo rdf:datatype="string">16534878800</hasCitizenshipNo>

</owl:Thing>
```

Figure 5.26 Transformed RDF block for *GetCustomerStatus* service.

Notation 3 (N3) representation of resulting RDF data triples for *GetCustomerStatus* service is shown in Table 5.12:

Table 5.12 *N3* representation of RDF triples for *GetCustomerStatus* service.

Subject	**Predicate**	**Object**
cust:16534878800	rdf:type	cust:Status
cust:16534878800	cust:isOnBlackList	“False”
cust:16534878800	cust:hasAgreementNo	“A45745”
cust:16534878800	cust:hasBranchName	“ISTANBUL”
cust:16534878800	cust:hasCitizenshipNo	“16534878800”

5.3.2.4 Persistence in RDF-Store

Transformations produce semantic data which are compatible with the domain ontology. Before the execution of any service, the RDF-Store contains only the RDF triples belonging to domain ontology. After the execution of each web service, RDF data triples generated by the service are persisted into the RDF-Store and merged with existing data. At the end of source system calls, RDF-Store holds all the information collected for the specified customer. A section of merged RDF triples in RDF-Store is shown in Figure 5.27:

```
<rdf:Description rdf:about="#16534878800">

  <rdf:type rdf:resource="#Customer"/>

  <rdf:type rdf:resource="#Address"/>

  <rdf:type rdf:resource="#Value"/>

   <rdf:type rdf:resource="#Status"/>

  <hasCountryCode rdf:datatype="string">TR</hasCountryCode>

  <hasCitizenshipNo rdf:datatype="string">16534878800</hasCitizenshipNo>

  <hasCountyName rdf:datatype="string">Pendik</hasCountyName>

  <hasName rdf:datatype="string">Mehmet Ali</hasName>

  <hasCityCode rdf:datatype="int">34</hasCityCode>

  <hasAddressType rdf:datatype="int">  1 </hasAddressType>

  <hasApartmentName rdf:datatype="string">Ocak apt.</hasApartmentName>

  <hasLastName rdf:datatype="string">Kaya</hasLastName>

  <hasFlatNo rdf:datatype="int">5</hasFlatNo>

  <hasDistrictName rdf:datatype="string">Barbaros</hasDistrictName>

  <hasStreetName rdf:datatype="string">Sedef str.</hasStreetName>

  <hasBalance rdf:datatype="string">High</hasBalance>

  <hasProfitability rdf:datatype=" string">Middle</hasProfitability>

  <isOnBlackList rdf:datatype="string">False</isOnBlackList>

  <hasAgreementNo rdf:datatype="string">A45745</hasAgreementNo>

  <hasBranchName rdf:datatype="string">ISTANBUL</hasBranchName>

</rdf:Description>
```

Figure 5.27 Merged RDF triples in RDF-Store.

High-level architecture of RDF-Store is represented in Figure 5.28:

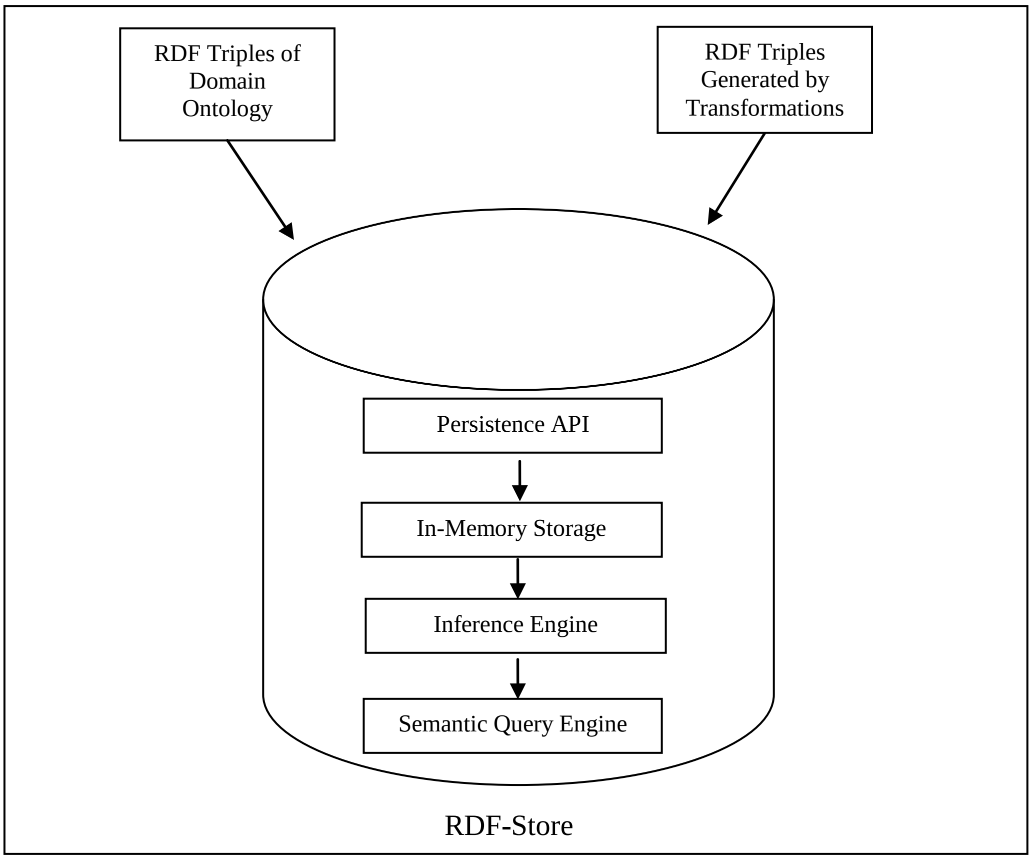


Figure 5.28 High-level architecture of RDF-Store.

For the implementation of RDF-Store, Jena Framework [37] is used. Jena is a semantic web framework for Java which includes an RDF/OWL API, in-memory/persistent storage and semantic query engine.

5.3.2.5 Semantic Query

Before construction of the response, the semantic information in the RDF-Store should be queried to get the results. Semantic query takes *citizenship number* as input parameter and returns whole the data related with this customer.

SPARQL [48] is used as the semantic query language since it has support for the Jena-based RDF–Store. A section of the semantic query statement in SPARQL is shown in Figure 5.29:

```
PREFIX customer: < http://www.semanticweb.org/ontologies/2010/0/customer.owl#>

SELECT ?Name ?LastName ?AddressType ?ApartmentName ?CityCode

 ?CountryCode ?CountyName ?DistrictName ?FlatNo ?StreetName

 ?Balance ?Profitability ?IsOnBlackList ?AgreementNo ?BranchName

WHERE {

     ?cust customer:hasCitizenshipNo  16534878800

     ?cust customer:hasName ?Name .

     ?cust customer:hasLastName ?LastName .

     ?cust customer:hasAddressType ?AddressType .

     ?cust customer:hasApartmentName ?ApartmentName .

     ?cust customer:hasCityCode ?CityCode .

     ?cust customer:hasCountryCode ?CountryCode .

     ?cust customer:hasCountyName ?CountyName .

     ?cust customer:hasDistrictName ?DistrictName .

     ?cust customer:hasFlatNo ?FlatNo .

     ?cust customer:hasStreetName ?StreetName .

     ?cust customer:hasBalance ?Balance .

     ?cust customer:hasProfitability ?Profitability .

    ?cust customer:isOnBlackList ?IsOnBlackList .

    ?cust customer:hasAgreementNo ?AgreementNo .

   ?cust customer:hasBranchName ?BranchName .

     }
```

Figure 5.29 Semantic query statement in SPARQL

5.3.2.6 Inference

Property Restriction feature of OWL is used to describe rules which infer new relations by utilizing asserted triples. *VIPCustomer* class is constructed by the intersection of a restriction class containing individuals whose *hasBalance* property has value *"High"* and another restriction class containing individuals whose *isOnBlackList* property has value *"False"*.

Inference rules are modelled with *Property Restriction* classes. *VIPCustomer* class is described as the individuals whose *hasBalance* property has value *"High"* and whose *isOnBlackList* property has value *"False"*. If those conditions hold for an individual, an inferred triple is added to the data store automatically. For the sample individual *#16534878800*, two triples exists so a new triple is inferred indicating that this individual belongs to *VIPCustomer* class. Inference rule for *VIPCustomer* class is shown in Figure 5.30:

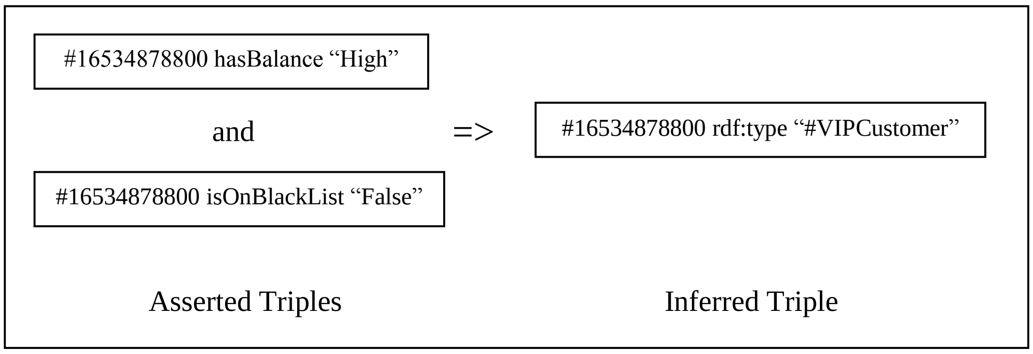

#16534878800 hasBalance "High"
and
#16534878800 isOnBlackList "False"
=>
#16534878800 rdf:type "#VIPCustomer"
Asserted Triples
Inferred Triple

Figure 5.30 Inference rule for *VIPCustomer* class.

SPARQL statement executed to test if the individual belongs to *VIPCustomer* class or not is represented in Figure 5.31:

```
PREFIX customer: < http://www.semanticweb.org/ontologies/2010/0/customer.owl#>

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT ?Name

WHERE {

 ?cust customer:hasCitizenshipNo "16534878800" .

?cust customer:hasName ?Name .

?cust rdf:type customer:VIPCustomer .

}
```

Figure 5.31 SPARQL statement executed for *VIPCustomer* class membership.

5.3.2.7 Construction of Response

The response object of the integrator process is filled by the fields in the result set of the semantic query. The response object is returned to the consumer application as a regular web service response.

5.3.2.8 Integrator Process Execution Time Analysis

Key operations in *Semantic Integrator Process* are analysed in terms of their execution times. The total response time for the *Semantic Integrator Process* against one consumer application request is divided into four main operations:

- *Source System Calls:* This is the time elapsed for calling four different source system web services and getting their response.

- *Transformation Executions:* This is the time elapsed for sending the XML data block to the *Central Transformation Repository,* executing the transformation inside it and getting the RDF data in response. The time is calculated as the aggregation of four different source system web services.

- *Persisting RDF Triples:* This is the time elapsed for persisting the RDF data inside the RDF-Store for four different source system web services.

- *Semantic Query and Inference:* This is the time elapsed for semantically querying the data, inferencing inside the RDF-Store and getting the result set from the RDF-Store.

Average execution times of each operation for one consumer application request are shown in Table 5.13. In the table, *Operation Name* column shows the name of the operation in *Semantic Integrator Process*, *Execution Time* column shows the execution time of the operation in milliseconds and *Percentage* column shows the percentage of the execution time for this operation in the total execution time.

Table 5.13 Average execution times for one consumer application request.

Operation Name	**Execution Time (in ms)**	**Percentage**
Source System Calls	189	%39
Transformation Executions	128	%27
Persisting RDF Triples	32	%7
Semantic Query and Inference	134	%27
Total of All Operations	483	%100

From the Table 5.13, it is seen that most of the processing time is used for the *Source System Calls*. However, the total cost of *Transformation Executions, Persisting RDF Triples* and *Semantic Query and Inference* is %61 of total time and this is more than the execution time of *Source System Calls.* This shows that operations that are performed for establishing data integration take more time than producing raw data in the source systems.

Apache Derby [49] relational database management system is used for source system data. Tests are performed on a personal computer with Intel Core 2 Duo processor and 2 GB RAM.

CHAPTER 6

CONCLUSION

In this work, semantic technologies are used to implement an Intra-Enterprise Data Integration scenario where a Web Service enabled Service Oriented Architecture is in use.

An important percentage of data integration problems occur as the result of syntactic definition of web services. Misunderstandings on the meaning of exchanged information are aimed to be reduced using semantic models. Data domain is modelled by a semantic ontology which allowed for a formal definition of classes, properties and rules constituting the domain. Transformations and mappings on the exchanged data patterns are defined based on the semantic model so that development efforts became more organized and systematic. Since a real-life project involving many number of development teams working together cannot be implemented during this study, it is not possible to give precise information about the magnitude of improvement on development time.

Domain-specific standardization, transformation and mapping rules were embedded inside the integrator processes in the syntax –based system which resulted in the change of source code each time a new rule is required. In addition, generating a comprehensive list of executing rules, which is frequently asked by the business units, were difficult because review of source codes is needed. Rules performing same operations were coded multiple times for different source systems. A main contribution in this thesis work is building a *Central Transformation Repository* for data transformations so that enterprise-wide standardization, transformation and mapping rules are managed at a single point. Transformation definitions became available to be shared and reused by different source systems. Enterprise-level standardization rules can be controlled and listed by just analyzing the transformation files. This repository communicates with integrator process with a web service which provides a smooth integration with the existing Service Oriented Architecture. The transformation information in the repository is stored in separate XSLT files and filename based retrieval is performed. Execution performance of repository can be improved by storing the transformation files in a light-weight database and retrieving transformation files using indexing mechanisms. Schema mapping files (XSLT files) are created manually in the thesis work. Utilizing a visual tool for mappings can speed up creation process.

Another purpose of the thesis work is applying semantic technologies with minimum modification on source systems. To achieve this, a bottom-up approach is applied for semantic annotation of web services. Only the required parts of the web service definitions are involved in the annotation process.

Generally, main purpose on semantic annotation of web services in the literature is to facilitate web service discovery and composition. Inferencing capabilities of semantic models are used to improve the service matching algorithms. However for an intra-enterprise data integration scenario as presented, web service discovery is not required since all the web services which will be involved in the process are already known at the beginning. A main contribution of this work is to expose whole inference power of underlying semantic model to capture the implicit data in an intra-enterprise environment, rather than relying on the quality of a service matching algorithm to find the possible inferences which will limit the reasoning power eventually. An RDF-Store based on Jena semantic web framework is developed to provide inferencing on integrated data. By inferencing, implicit relationships on integrated data are revealed so that additional information can be provided to consumer application. Only a core subset of customer domain can be modelled during this study. The semantic model can be more realistic by adding more classes and property restrictions.

As a future work, the proposed framework can be used in the development process of a real-life integration project to calculate the improvement on development efforts. *Central Transformation Repository* development can be integrated with semantic annotation tools to make transformation repository creation easier for developers. Other XML transformation languages like XQuery [50] can be utilized in place of XSLT to compare execution times.

REFERENCES

[1] Bernard Manouvrier, Laurent Menard. “Application Integration: EAI, B2B, BPM and SOA”. Wiley, 2008.

[2] RosettaNet. URL: http://www.rosettanet.org/, last access date 7 April 2010.

[3] Protégé Ontology Editor. URL: http://protege.stanford.edu/, last access date 8 April 2010.

[4] Semantic Annotations for WSDL. URL: http://www.w3.org/2002/ws/sawsdl/, last access date 3 April 2010.

[5] Radiant: WSDL-S/SAWSDL Annotation Tool.

URL: http://lsdis.cs.uga.edu/projects/meteor-s/downloads/index.php?page=1, last access date 13 April 2010.

[6] Eclipse IDE. URL: http://www.eclipse.org/, last access date 5 April 2010.

[7] XSL Transformations (XSLT). URL: http://www.w3.org/TR/xslt, last access date 1 April 2010.

[8] SAWSDL4J Object Model.

URL: http://knoesis.wright.edu/opensource/sawsdl4j/project-info.html, last access date 9 April 2010.

[9] Java API for XML Processing (JAXP). URL: https://jaxp.dev.java.net/, last access date 21 April 2010.

[10] Jorge Cardoso and Amit P. Sheth. “Semantic Web Services, Processes and Applications”. Springer, 2006.

[11] Curbera, F., W. Nagy, et al. “Web Services: Why and How”. Workshop on Object-Oriented Web Services, OOPSLA, Tampa, Florida, USA. 2001

[12] Chinnici, R., M. Gudgin, et al. Web Services Description Language (WSDL) Version 1.2, W3C Working Draft 24, http://www.w3 .org/TR/2003AVD-wsdl 12-20030124/. 2003.

[13] OWL-S. URL:http://www.w3.org/Submission/OWL-S/, last access date 8 April 2010.

[14] WSMO. URL: http://www.wsmo.org/, last access date 15 April 2010.

[15] WSDL-S. URL: http://www.w3.org/Submission/WSDL-S/, last access date 26 April 2010.

[16] Dieter Fensel, Holger Lausen, Axel Polleres, Jos de Bruijn, Michael Stollberg, Dumitru Roman, John Domingue. “Enabling Semantic Web Services: The Web Service Modeling Ontology”. Springer, 2007.

[17] Resource Description Framework (RDF). URL: http://www.w3.org/RDF/, last access date 28 April 2010.

[18] RDF Schema. URL: http://www.w3.org/TR/rdf-schema/, last access date 8 April 2010.

[19] Web Ontology Language (OWL). URL: http://www.w3.org/2004/OWL/, last access date 13 April 2010.

[20] I. Horrocks, P. F. Patel-Schneider and F. van Harmelen. “From SHIQ and RDF to OWL: The making of a web ontology language”. Journal of Web Semantics, 1(1):7–26, 2003.

[21] P. F. Patel-Schneider, P. Hayes and I. Horrocks. “OWL Web Ontology Language Semantics and Abstract Syntax”. W3C Recommendation, 10 February 2004.

[22] Verma, K., K. Sivashanmugam, et al. "METEOR-S WSDI: A Scalable Infrastructure of Registries for Semantic Publication and Discovery of Web Services." Journal of Information Technology and Management. 2004.

[23] Cardoso, J. and A. Sheth. "Semantic e-Workflow Composition." Journal of Intelligent Information Systems (JUS). ZIO): 191-225. 2003.

[24] Universal Description Discovery and Integration (UDDI). URL:http://uddi.xml.org/, last access date 22 April 2010.

[25] Sheth, A. and R. Meersman. "Amicaloia Report: Database and Information Systems Research Challenges and Opportunities in Semantic Web and Enterprises." SIGMOD Record 31(4): pp. 98-106. 2002.

[26] Sheth, A. . “Semantic Meta Data For Enterprise Information Integration”. DM Review Magazine. July 2003.

[27] C. Bussler. “The Role of Semantic Web Technology in Enterprise Application Integration.”. IEEE Data Engineering Bulletin, 26(4):62–68, 2003.

[28] D. Fensel and C. Bussler. “The Web Service Modelling Framework WSMF”. Electronic Commerce Research and Applications, 1(2), 2002.

[29] Armin Haller, Juan Miguel Gomez, Christoph Bussler. “Exposing Semantic Web Service principles in SOA to solve EAI scenarios”. In the Proceedings of the Workshop on Web Service Semantics: Towards Dynamic Business Integration, in conjunction with WWW2005. Chiba, Japan. 2005

[30] A. Mocan and E. Cimpian. “Mediation”. WSMX Working Draft D13.3 v0.1, URL: http://www.wsmo.org/2004/d13/d13.3/v0.1/20040906/, 2004, last access date 24 April 2010.

[31] Thanassis Bouras, Panagiotis Gouvas, Gregoris Mentzas. “Dynamic Data Mediation in Enterprise Application Integration”. Conference in O. Cunnigham and M. Cunnigham (eds) Collaboration and the Knowledge Economy: Issues, Applications and Case Studies, pp. 917-924, eChallenges e-2008 Conference, 22 - 24 October 2008, Stockholm, Sweden. 2008

[32] Nenad Anicic, Nenad Ivezic, Albert Jones. “An Architecture for Semantic Enterprise Application Integration Standards”. IJMTM 10(2/3): 205-226 2007

[33] XML Path Language (XPath). URL: http://www.w3.org/TR/xpath/, last access date 2 April 2010.

[34] Java Platform, Enterprise Edition (J2EE). URL: http://java.sun.com/javaee/, last access date 8 April 2010.

[35] Apache Tomcat. URL: http://tomcat.apache.org/, last access date 30 April 2010.

[36] Easy SAWSDL API. URL: http://easywsdl.ow2.org/extensions-sawsdl.html, last access date 29 April 2010.

[37] Jena Semantic Web Framework for Java. URL: http://jena.sourceforge.net/, last access date 28 April 2010.

[38] Bouras, A., Gouvas, P., & Mentzas, G. (2007). ENIO: An Enterprise Application Integration Ontology. In the Proceedings of the 1st International Workshop on Semantic Web Architectures for Enterprises (SWAE), DEXA’07, 3-7 September, 2007, Regensburg, Germany.

[39] Dean Allemang and James Hendler. “Semantic Web for the Working Ontologist”. Morgan Kaufmann Publications. 2008.

[40] World Wide Web Consortium (W3C). URL: http://www.w3.org/, last access date 24 April 2010.

[41] Hypertext Markup Language (HTML). URL: http://www.w3.org/html/, last access date 15 April 2010.

[42] WordNet Lexical Database. URL: http://wordnet.princeton.edu/, last access date 16 April 2010.

[43] CYC Ontology. URL: http://www.cyc.com/, last access date 12 April 2010.

[44] The United Nations Standard Products and Services Code (UNSPSC). URL: http://www.unspsc.org/, last access date 11 April 2010.

[45] A. Mocan and E. Cimpian. Mediation. WSMX Working Draft D13.3 v0.1, http://www.wsmo.org/2004/d13/d13.3/v0.1/20040906/, 2004.

[46] Alex Berson and Lawrence Dubov. “Master Data Management and Customer Data Integration for Global Enterprise”. McGraw-Hill, 2007

[47] Jill Dyche and Evan Levy. “Customer Data Integration”. Wiley, 2006

[48] SPARQL Query Language. URL: http://www.w3.org/TR/rdf-sparql-query/, last access date 10 April 2010.

[49] Apache Derby. URL: http://db.apache.org/derby/, last access date 21 April 2010.

[50] XQuery. URL: http://www.w3.org/TR/xquery/, last access date 23 April 2010.

[51] RDF/XML syntax. URL: http://www.w3.org/TR/rdf-syntax-grammar/, last access date 26 April 2010.

APPENDIX A

CUSTOMER ONTOLOGY

A.1. Semantic Model in OWL

```
<?xml version="1.0"?>

<rdf:RDF xmlns="http://www.semanticweb.org/ontologies/2010/0/customer.owl#"

     xml:base="http://www.semanticweb.org/ontologies/2010/0/customer.owl"

     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"

     xmlns:owl2xml="http://www.w3.org/2006/12/owl2-xml#"

     xmlns:owl="http://www.w3.org/2002/07/owl#"

     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"

     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

     xmlns:customer="http://www.semanticweb.org/ontologies/2010/0/customer.owl#">

    <owl:Ontology rdf:about=""/>

    <!--  ///////////////////////////////////////////////////////////////////////////////////////

    // Data properties

    ///////////////////////////////////////////////////////////////////////////////////////

     -->

    <!-- http://www.semanticweb.org/ontologies/2010/0/customer.owl#hasAddressType -->

    <owl:DatatypeProperty rdf:about="#hasAddressType">

        <rdfs:domain rdf:resource="#Address"/>
        <rdfs:range rdf:resource="&xsd;int"/>

    </owl:DatatypeProperty>

    <!-- http://www.semanticweb.org/ontologies/2010/0/customer.owl#hasAgreementNo -->

    <owl:DatatypeProperty rdf:about="#hasAgreementNo">

        <rdfs:domain rdf:resource="#Status"/>

        <rdfs:range rdf:resource="&xsd;string"/>

    </owl:DatatypeProperty>

    <!-- http://www.semanticweb.org/ontologies/2010/0/customer.owl#hasApartmentName -->

    <owl:DatatypeProperty rdf:about="#hasApartmentName">

        <rdfs:domain rdf:resource="#Address"/>

        <rdfs:range rdf:resource="&xsd;string"/>

    </owl:DatatypeProperty>

    <!-- http://www.semanticweb.org/ontologies/2010/0/customer.owl#hasBalance -->

    <owl:DatatypeProperty rdf:about="#hasBalance">

        <rdfs:domain rdf:resource="#Value"/>

        <rdfs:range rdf:resource="&xsd;string"/>

    </owl:DatatypeProperty>

    <!-- http://www.semanticweb.org/ontologies/2010/0/customer.owl#hasBranchName -->

    <owl:DatatypeProperty rdf:about="#hasBranchName">

        <rdfs:domain rdf:resource="#Status"/>

        <rdfs:range rdf:resource="&xsd;string"/>

    </owl:DatatypeProperty>

    <!-- http://www.semanticweb.org/ontologies/2010/0/customer.owl#hasCitizenshipNo -->

    <owl:DatatypeProperty rdf:about="#hasCitizenshipNo">

        <rdfs:domain rdf:resource="#Address"/>
        <rdfs:domain rdf:resource="#Customer"/>

        <rdfs:domain rdf:resource="#Status"/>

        <rdfs:domain rdf:resource="#Value"/>

        <rdfs:range rdf:resource="&xsd;string"/>

</owl:DatatypeProperty>

<!-- http://www.semanticweb.org/ontologies/2010/0/customer.owl#hasCityCode -->

<owl:DatatypeProperty rdf:about="#hasCityCode">

    <rdfs:domain rdf:resource="#Address"/>

    <rdfs:range rdf:resource="&xsd;int"/>

</owl:DatatypeProperty>

<!-- http://www.semanticweb.org/ontologies/2010/0/customer.owl#hasCountryCode -->

<owl:DatatypeProperty rdf:about="#hasCountryCode">

    <rdfs:domain rdf:resource="#Address"/>

    <rdfs:range rdf:resource="&xsd;string"/>

</owl:DatatypeProperty>

<!-- http://www.semanticweb.org/ontologies/2010/0/customer.owl#hasCountyName -->

<owl:DatatypeProperty rdf:about="#hasCountyName">

    <rdfs:domain rdf:resource="#Address"/>

    <rdfs:range rdf:resource="&xsd;string"/>

</owl:DatatypeProperty>

<!-- http://www.semanticweb.org/ontologies/2010/0/customer.owl#hasDistrictName -->

<owl:DatatypeProperty rdf:about="#hasDistrictName">

    <rdfs:domain rdf:resource="#Address"/>

    <rdfs:range rdf:resource="&xsd;string"/>

</owl:DatatypeProperty>
```

```
<!-- http://www.semanticweb.org/ontologies/2010/0/customer.owl#hasFlatNo -->

<owl:DatatypeProperty rdf:about="#hasFlatNo">

   <rdfs:domain rdf:resource="#Address"/>

   <rdfs:range rdf:resource="&xsd;int"/>

</owl:DatatypeProperty>

<!-- http://www.semanticweb.org/ontologies/2010/0/customer.owl#hasLastName -->

<owl:DatatypeProperty rdf:about="#hasLastName">

   <rdfs:domain rdf:resource="#Customer"/>

   <rdfs:range rdf:resource="&xsd;string"/>

</owl:DatatypeProperty>

<!-- http://www.semanticweb.org/ontologies/2010/0/customer.owl#hasName -->

<owl:DatatypeProperty rdf:about="#hasName">

   <rdfs:domain rdf:resource="#Customer"/>

   <rdfs:range rdf:resource="&xsd;string"/>

</owl:DatatypeProperty>

<!-- http://www.semanticweb.org/ontologies/2010/0/customer.owl#hasProfitability -->

<owl:DatatypeProperty rdf:about="#hasProfitability">

   <rdfs:domain rdf:resource="#Value"/>

   <rdfs:range rdf:resource="&xsd;string"/>

</owl:DatatypeProperty>

<!-- http://www.semanticweb.org/ontologies/2010/0/customer.owl#hasStreetName -->

<owl:DatatypeProperty rdf:about="#hasStreetName">

   <rdfs:domain rdf:resource="#Address"/>

   <rdfs:range rdf:resource="&xsd;string"/>

</owl:DatatypeProperty>
```

```
<!-- http://www.semanticweb.org/ontologies/2010/0/customer.owl#isOnBlackList -->

<owl:DatatypeProperty rdf:about="#isOnBlackList">

    <rdfs:domain rdf:resource="#Status"/>

    <rdfs:range rdf:resource="&xsd;string"/>

</owl:DatatypeProperty>

<!-- ///////////////////////////////////////////////////////////////////////////////////////

// Classes

///////////////////////////////////////////////////////////////////////////////////////

 -->

<!-- http://www.semanticweb.org/ontologies/2010/0/customer.owl#Address -->

<owl:Class rdf:about="#Address"/>

<!-- http://www.semanticweb.org/ontologies/2010/0/customer.owl#Customer -->

<owl:Class rdf:about="#Customer"/>

<!-- http://www.semanticweb.org/ontologies/2010/0/customer.owl#Status -->

<owl:Class rdf:about="#Status"/>

<!-- http://www.semanticweb.org/ontologies/2010/0/customer.owl#VIPCustomer -->

<owl:Class rdf:about="#VIPCustomer">

    <owl:equivalentClass>

        <owl:Class>

            <owl:intersectionOf rdf:parseType="Collection">

                <owl:Restriction>

                    <owl:onProperty rdf:resource="#hasBalance"/>

                    <owl:hasValue>High</owl:hasValue>

                </owl:Restriction>

                <owl:Restriction>
```

```
                    <owl:onProperty rdf:resource="#isOnBlackList"/>

                    <owl:hasValue>False</owl:hasValue>

                </owl:Restriction>

            </owl:intersectionOf>

        </owl:Class>

    </owl:equivalentClass>

  </owl:Class>

  <!-- http://www.semanticweb.org/ontologies/2010/0/customer.owl#Value -->

  <owl:Class rdf:about="#Value"/>

</rdf:RDF>

<!-- Generated by the OWL API (version 2.2.1.1138) http://owlapi.sourceforge.net -->
```

APPENDIX B

XSLT TRANSFORMATION FILES

B.1. XSLT Transformation File for GetCustomerNames

```
<?xml version="1.0"?>

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

<xsl:template match="NamesResponse">

     <rdf:RDF xmlns="http://www.semanticweb.org/ontologies/2010/0/customer.owl#"

    xml:base="http://www.semanticweb.org/ontologies/2010/0/customer.owl"

    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"

    xmlns:owl2xml="http://www.w3.org/2006/12/owl2-xml#"

    xmlns:owl="http://www.w3.org/2002/07/owl#"

    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"

    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

    xmlns:customer="http://www.semanticweb.org/ontologies/2010/0/customer.owl#">

     <owl:Ontology rdf:about=""/>

      <owl:Thing rdf:about= "#{citizenshipNo}"   >

          <rdf:type rdf:resource="#Customer"/>

          <hasName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">

          <xsl:value-of select="concat(

                firstname , ' ',
                middlename)"/>

          </hasName>

      <hasLastName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">

          <xsl:value-of select="lastname"/>

          </hasLastName>

          <hasCitizenshipNo rdf:datatype="http://www.w3.org/2001/XMLSchema#string">

          <xsl:value-of select="citizenshipNo"/>

          </hasCitizenshipNo>

  </owl:Thing>

  </rdf:RDF>

</xsl:template>

</xsl:stylesheet>
```

B.2. XSLT Transformation File for GetCustomerAddress

```
            <?xml version="1.0"?>

            <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

            <xsl:template match="AddressResponse">

      <rdf:RDF xmlns="http://www.semanticweb.org/ontologies/2010/0/customer.owl#"

                xml:base="http://www.semanticweb.org/ontologies/2010/0/customer.owl"

                xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"

                xmlns:owl2xml="http://www.w3.org/2006/12/owl2-xml#"

                xmlns:owl="http://www.w3.org/2002/07/owl#"

                xmlns:xsd="http://www.w3.org/2001/XMLSchema#"

                xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

                xmlns:customer="http://www.semanticweb.org/ontologies/2010/0/customer.owl#">
```

```
        <owl:Ontology rdf:about=""/>

         <owl:Thing rdf:about= "#{citizenshipNo}"   >

            <rdf:type rdf:resource="#Address"/>

            <hasAddressType rdf:datatype="http://www.w3.org/2001/XMLSchema#int">

                <xsl:choose>

                    <xsl:when test="addresstype = 'H'">  1 </xsl:when>

                    <xsl:when test="addresstype = 'W'">  2 </xsl:when>

                    <xsl:when test="addresstype = 'F'">  3 </xsl:when>

                    <xsl:otherwise> 1 </xsl:otherwise>

                </xsl:choose>

            </hasAddressType>

             <hasStreetName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">

                <xsl:value-of select="concat( streetname , ' str.')"/>

            </hasStreetName>

            <hasApartmentName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">

                <xsl:value-of select="concat( apartmentname , ' apt.')"/>

            </hasApartmentName>

            <hasCountryCode rdf:datatype="http://www.w3.org/2001/XMLSchema#string">

                <xsl:value-of select="countrycode"/>

            </hasCountryCode>

            <hasCityCode rdf:datatype="http://www.w3.org/2001/XMLSchema#int">

                <xsl:value-of select="citycode"/>

            </hasCityCode>

            <hasCountyName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">

                <xsl:value-of select="countyname"/>
        </hasCountyName>

        <hasDistrictName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">

            <xsl:value-of select="districtname"/>

        </hasDistrictName>

        <hasFlatNo rdf:datatype="http://www.w3.org/2001/XMLSchema#int">

            <xsl:value-of select="flatno"/>

        </hasFlatNo>

        <hasCitizenshipNo rdf:datatype="http://www.w3.org/2001/XMLSchema#string">

            <xsl:value-of select="citizenshipNo"/>

        </hasCitizenshipNo>

      </owl:Thing>

      </rdf:RDF>

    </xsl:template>

    </xsl:stylesheet>
```

B.3. XSLT Transformation File for GetCustomerValue

```
<?xml version="1.0"?>

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

<xsl:template match="ValueResponse">

<rdf:RDF xmlns="http://www.semanticweb.org/ontologies/2010/0/customer.owl#"

    xml:base="http://www.semanticweb.org/ontologies/2010/0/customer.owl"

    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"

    xmlns:owl2xml="http://www.w3.org/2006/12/owl2-xml#"

    xmlns:owl="http://www.w3.org/2002/07/owl#"

    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

    xmlns:customer="http://www.semanticweb.org/ontologies/2010/0/customer.owl#">

<owl:Ontology rdf:about=""/>

 <owl:Thing rdf:about= "#{citizenshipNo}"   >

     <rdf:type rdf:resource="#Value"/>

     <hasBalance rdf:datatype="http://www.w3.org/2001/XMLSchema#string">

         <xsl:choose>

             <xsl:when test="balance &gt; 100000">High</xsl:when>

             <xsl:when test="balance &gt; 10000">Middle</xsl:when>

             <xsl:when test="balance &gt; 1000">Low</xsl:when>

             <xsl:otherwise> None </xsl:otherwise>

         </xsl:choose>

     </hasBalance>

     <hasProfitability rdf:datatype="http://www.w3.org/2001/XMLSchema#string">

         <xsl:choose>

             <xsl:when test="profitability &gt; 100000">High</xsl:when>

             <xsl:when test="profitability &gt; 10000">Middle</xsl:when>

             <xsl:when test="profitability &gt; 1000">Low</xsl:when>

             <xsl:otherwise> None </xsl:otherwise>

         </xsl:choose>

     </hasProfitability>

   </owl:Thing>

   </rdf:RDF>

</xsl:template>

</xsl:stylesheet>
```

B.4. XSLT Transformation File for GetCustomerStatus

```
<?xml version="1.0"?>

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

<xsl:template match="StatusResponse">

<rdf:RDF xmlns="http://www.semanticweb.org/ontologies/2010/0/customer.owl#"

    xml:base="http://www.semanticweb.org/ontologies/2010/0/customer.owl"

    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"

    xmlns:owl2xml="http://www.w3.org/2006/12/owl2-xml#"

    xmlns:owl="http://www.w3.org/2002/07/owl#"

    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"

    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

    xmlns:customer="http://www.semanticweb.org/ontologies/2010/0/customer.owl#">

<owl:Ontology rdf:about=""/>

 <owl:Thing rdf:about= "#{citizenshipNo}"   >

     <rdf:type rdf:resource="#Status"/>

     <isOnBlackList rdf:datatype="http://www.w3.org/2001/XMLSchema#string">

          <xsl:choose>

               <xsl:when test="isonblacklist = ' '">  False </xsl:when>

               <xsl:otherwise> <xsl:value-of select="isonblacklist"/> </xsl:otherwise>

          </xsl:choose>

     </isOnBlackList>

     <hasBranchName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">

          <xsl:value-of select="branchname"/>
```

```
            </hasBranchName>

            <hasAgreementNo rdf:datatype="http://www.w3.org/2001/XMLSchema#string">

                <xsl:value-of select="agreementno"/>

            </hasAgreementNo>

        </owl:Thing>

        </rdf:RDF>

    </xsl:template>

    </xsl:stylesheet>
```