

122 8~

AN ebXML REGISTRY / REPOSITORY IMPLEMENTATION

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

119 195 BY

LEVENT ÇARKACIOĞLU

119195

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

**T.C. YÜKSEKÖĞRETİM KURULU  
DOKÜMANTASYON MERKEZİ**

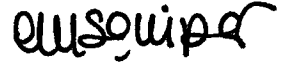
JULY 2002

Approval of the Graduate School of Natural And Applied Sciences.



Prof. Dr. Tayfur ÖZTÜRK  
Director

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



Prof. Dr. Ayşe KİPER  
Head of Department

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



Prof. Dr. Asuman DOĞAÇ  
Supervisor

Examining Committee Members :

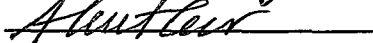
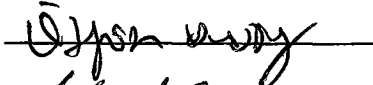
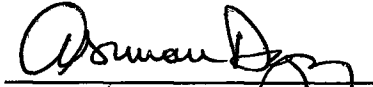
Prof. Dr. Asuman DOĞAÇ

Prof. Dr. Adnan YAZICI

Assoc. Prof. Dr. Volkan ATALAY

Assoc. Prof. Dr. Özgür ULUSOY

Assist. Prof. Dr. Ahmet COŞAR



# ABSTRACT

## AN EBXML REGISTRY/REPOSITORY IMPLEMENTATION

Çarkacıoğlu, Levent

M.Sc., Department of Computer Engineering

Supervisor: Prof. Dr. Asuman Doğaç

July 2002, 115 pages

Companies need to exchange data with their trading partners to conduct business. Exchange of data electronically enables the automation of business processes in a secure and reliable manner. However, electronic data exchange necessitates standards.

ebXML is one of the electronic business standards which aims to provide an electronic business framework to enable small or medium enterprises to do business easier via Internet, in order to increase the global trade. The vision of ebXML is to create an infrastructure for electronic business transactions.

One of the most important functional concepts in the ebXML framework is the Registry/Repository Service that enables the storage, access and discovery of business sources. In this thesis, an ebXML Registry/Repository Service is designed and implemented with two extensions.

In the ebXML framework, in order for a business to search for technically compatible partners, it has to download all the partners technical documents from the Repository. In this thesis, the ebXML Registry/Repository Service is extended for automatic compatible partner search.

The use of ontologies makes the search of sources easier and faster. ebXML Registry/Repository provides mechanisms to store ontologies in the form of classification trees. However, DAML+OIL is a widely accepted, RDF based semantic markup language for building richer ontology structures. In this thesis, ebXML Registry/Repository Service is extended to support DAML+OIL ontologies.

Keywords: ebXML Registry/Repository, ebXML Registry, Automatic CPA Construction, DAML+OIL ontology support on ebXML Registry.



# ÖZ

## EBXML KAYIT/DEPOLAMA GELİŞTİRİLMESİ

Çarkacıoğlu, Levent

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi: Prof. Dr. Asuman Doğaç

Temmuz 2002, 115 sayfa

Şirketler ticaret yapabilmek için ticari ortakları ile veri değişimine ihtiyaç duyarlar. Veri değişiminin elektronik olarak yapılması, ticari işlevlerin güvenilir ve emin bir şekilde otomasyonunu sağlar. Bununla birlikte, veri değişimi standartlar gereksindirir.

ebXML, evrensel ticareti artırmak için orta ve küçük ölçekli girişimlere İnternet üzerinden kolayca ticaret yapabilmeleri için bir elektronik ticaret iskeleti sağlamayı amaçlayan, elektronik ticaret standartlarından birisidir.

ebXML içindeki en önemli işlevsel kavramlardan birisi de, ticari kaynakları depolama, erişim ve keşfetmeyi sağlayan Kayıt/Depolama Servisidir. Bu tezde ebXML Kayıt/Depolama Servisi iki ek ile tasarlandı ve gerçekleştirildi.

ebXML iskeletinde, ticari şirketlerin teknik olarak uyuşabileceği ortakları araştırmaları için tüm ortakların teknik dokümanlarını depodan yüklemeleri gerekmektedir. Bu tezde, ebXML Kayıt/Depolama Servisi, otomatik uyuşabilir ortak taraması için genişletildi.

Ontoloji kullanımı kaynak taramasını kolaylaştırır ve hızlandırır. ebXML Kayıt/Depolama Servisi, ontolojileri desteklemek için sınıflandırma ağaçları biçimi-

minde bir yapı sağlamaktadır. Bununla birlikte, DAML+OIL zengin ontoloji yapıları kurmak için, büyük çapta kabul görmüş, RDF tabanlı anlamsal bir markup dilidir. Bu tezde ebXML Kayıt/Depolama Servisi, DAML+OIL ile oluşturulmuş ontolojileri desteklemek üzere genişletildi.

**Anahtar Kelimeler:** ebXML Kayıt/Depolama Servisi, Otomatik CPA oluşturulması, ebXML Kayıt/Depolama Servisi'nde DAML+OIL ontoloji desteği.





To my family

## ACKNOWLEDGMENTS

I would like to thank to Asuman Doğaç for her supervision, guidance and invaluable suggestions throughout the development of this thesis. I would also like to thank to METU Software Research and Development Center staff for their friendship and sharing their knowledge with me.



# TABLE OF CONTENTS

|                                                                       |      |
|-----------------------------------------------------------------------|------|
| ABSTRACT . . . . .                                                    | iii  |
| ÖZ . . . . .                                                          | v    |
| DEDICATON . . . . .                                                   | vii  |
| ACKNOWLEDGMENTS . . . . .                                             | viii |
| TABLE OF CONTENTS . . . . .                                           | ix   |
| LIST OF FIGURES . . . . .                                             | xii  |
| CHAPTER                                                               |      |
| 1 INTRODUCTION . . . . .                                              | 1    |
| 1.1 Motivation . . . . .                                              | 1    |
| 1.2 Scope and Objectives . . . . .                                    | 3    |
| 1.3 Organization of the Thesis . . . . .                              | 3    |
| 2 RELATED WORK . . . . .                                              | 5    |
| 2.1 Introduction . . . . .                                            | 5    |
| 2.2 EDI (Electronic Data Interchange) . . . . .                       | 6    |
| 2.3 UDDI (Universal Description, Discovery and Integration) . . . . . | 7    |
| 2.4 XML (Extensible Markup Language) . . . . .                        | 8    |
| 2.5 ebXML (Electronic Business Extensible Markup Language) . . . . .  | 9    |
| 2.5.1 Need and Vision . . . . .                                       | 9    |
| 2.5.2 General Architecture Overview . . . . .                         | 9    |
| 2.5.2.1 Trading Partner Information . . . . .                         | 10   |
| 2.5.2.2 Registry/Repository . . . . .                                 | 10   |
| 2.5.2.3 Business Process Specification Schema . . . . .               | 11   |
| 2.5.2.4 Messaging Service . . . . .                                   | 11   |

|   |         |                                                       |    |
|---|---------|-------------------------------------------------------|----|
|   | 2.5.3   | ebXML Registry/Repository Service . . . . .           | 13 |
|   | 2.5.3.1 | ebRIM (ebXML Registry Information Model) . . . . .    | 15 |
|   | 2.5.3.2 | ebRS (ebXML Registry Service Specification) . . . . . | 19 |
|   | 2.6     | DAML+OIL . . . . .                                    | 19 |
| 3 |         | DESIGN OF THE SYSTEM . . . . .                        | 22 |
|   | 3.1     | Introduction . . . . .                                | 22 |
|   | 3.2     | General Architecture Design . . . . .                 | 22 |
|   | 3.3     | Design of Registry Internal Structure . . . . .       | 24 |
|   | 3.3.1   | Automatic CPA Construction . . . . .                  | 24 |
|   | 3.3.1.1 | Role Matching . . . . .                               | 27 |
|   | 3.3.1.2 | Transport Matching . . . . .                          | 28 |
|   | 3.3.1.3 | Package Matching . . . . .                            | 29 |
|   | 3.3.2   | DAML+OIL Ontology Support . . . . .                   | 30 |
|   | 3.4     | Design of Registry Service Interfaces . . . . .       | 35 |
|   | 3.4.1   | SubmitObjectsRequest . . . . .                        | 38 |
|   | 3.4.2   | RemoveObjectsRequest . . . . .                        | 40 |
|   | 3.4.3   | FilterQuery . . . . .                                 | 41 |
|   | 3.4.4   | ReturnRegistryEntry . . . . .                         | 43 |
|   | 3.4.5   | ReturnRepositoryItem . . . . .                        | 46 |
|   | 3.4.6   | CPPQuery . . . . .                                    | 48 |
|   | 3.4.7   | ReturnCompatibleCPP . . . . .                         | 51 |
|   | 3.4.8   | DAMLOILQueryRequest . . . . .                         | 54 |
|   | 3.4.9   | GetRootClassificationNodeRequest . . . . .            | 57 |
|   | 3.4.10  | GetClassificationTreeRequest . . . . .                | 59 |
|   | 3.4.11  | GetClassifiedObjectsRequest . . . . .                 | 60 |
|   | 3.4.12  | SQLQueryRequest . . . . .                             | 61 |
|   | 3.4.13  | ErrorCase . . . . .                                   | 61 |
| 4 |         | IMPLEMENTATION DETAILS . . . . .                      | 63 |
|   | 4.1     | Introduction . . . . .                                | 63 |
|   | 4.2     | Development Tools . . . . .                           | 63 |
|   | 4.3     | ebXML Registry Service Execution Flow . . . . .       | 64 |

|       |                                                     |     |
|-------|-----------------------------------------------------|-----|
| 4.4   | Implementaion Details . . . . .                     | 66  |
| 4.4.1 | ebRIM Implementation . . . . .                      | 66  |
| 4.4.2 | ebRS Implementation . . . . .                       | 67  |
| 5     | CONCLUSION . . . . .                                | 68  |
|       | REFERENCES . . . . .                                | 70  |
|       | APPENDICES . . . . .                                | 74  |
| A     | RELATIONAL SCHEMA FOR SQL QUERIES . . . . .         | 74  |
| B     | ebXML REGISTRY DTD DEFINITION . . . . .             | 90  |
| C     | AUTOMATIC COMPATIBLE CPP SEARCH SQL QUERY . . . . . | 104 |
| D     | DAML+OIL BUILT SERVICE TAXONOMY ONTOLOGY . . . . .  | 107 |
| E     | CPA CONSTRUCTION ALGORITHM . . . . .                | 110 |
| F     | ebXML REGISTRY CPA . . . . .                        | 113 |

## LIST OF FIGURES

|     |                                                                   |    |
|-----|-------------------------------------------------------------------|----|
| 2.1 | A high level use case of ebXML architecture [1]. . . . .          | 10 |
| 2.2 | ebXML messaging conceptual view [2],[3]. . . . .                  | 12 |
| 2.3 | Registry architecture supports flexible topologies [1]. . . . .   | 14 |
| 2.4 | Information model inheritance view [4]. . . . .                   | 17 |
| 2.5 | Classification example. . . . .                                   | 18 |
| 2.6 | Association example. . . . .                                      | 19 |
| 3.1 | Architectural design and workflow of Registry/Repository Service. | 23 |
| 3.2 | Service taxonomy example[5]. . . . .                              | 31 |
| 3.3 | Registry inheritance hierarch . . . . .                           | 33 |
| 3.4 | Other registry internal objects. . . . .                          | 34 |
| 3.5 | Registry Server execution Flow. . . . .                           | 37 |
| 4.1 | ebXML Registry/Repository Service implementation work flow.       | 65 |

# CHAPTER 1

## INTRODUCTION

### 1.1 Motivation

Conducting business involves exchanging documents such as purchase order request, invoice, etc. among trading partners. Data exchange can be done electronically or manually. Electronic data exchange has many advantages over manual transmission such as fax or phone. The ability to automate the business process through application programs running at both end of exchange is one of the most important advantages. Timeliness, traceability, reliability, scalability and security are other important advantages. However, in order for applications at both ends to understand each other, exchanging data electronically necessitates standards.

ebXML is one of the electronic business standards which aims to increase the global trade by enabling small or medium enterprises to join electronic business easier.

ebXML is a modular suite of specifications that enables enterprises of any size and in any geographic location to conduct business over Internet [6]. It enables companies to have a standard method to exchange business messages, conduct trading relationships, communicate data in common terms and define and register business processes. Business can discover and negotiate agreements

with trading partners automatically and then exchange documents in a secure, reliable manner. The fundamental goal of ebXML is to create a single global electronic market [7].

ebXML architecture has four main functional concepts, Trading Partner Information, Business Process Specification Schema, Messaging Service and Registry/Repository Service.

Critical information such as, communication between applications and business processes and recording of specific technical parameters for electronic business conduction, are captured by Trading Partner Information [8].

Business Process Specification Schema models how business processes. It enables an organization to express its business process in a specific scenario so that they are understandable by other organizations. This enables the integration of business processes within a company or between companies' [8].

Messaging Service supplies a set of services and protocols that enables electronic business applications to exchange data.

Registry/Repository Service provides a set of services that enable sharing of information between interested parties for the purpose of enabling business process integration and let business to discover each other. It also acts as a container for process models, vocabularies and profiles.

The work within this thesis is motivated by two parts. One is the implementation of ebXML Registry/Repository Service, the other is the extension of registry for automatic compatible CPP search and DAML+OIL ontology support. At the time of development of this thesis, there were no publicly available ebXML compliant registries. In ebXML framework, companies find their technically compatible trading partners via downloading Collaboration Protocol Profile (CPP) documents one by one from repository. ebXML Registry is extended for automation of compatible CPP search. Also, although ebXML registry specifications make it possible to define ontologies through classification nodes, this specifications need to be extended to be able to store DAML+OIL ontologies to ebXML registries.

## 1.2 Scope and Objectives

Within the scope of this thesis an ebXML Registry/Repository Service is implemented with the following contributions:

1. In ebXML architecture, technical documents stating how a trading partner conduct in business, Collaboration Protocol Profile (CPP) documents for each business partner reside in the repository as a file (i.e., a file in a common file server). The meta-data related with these documents are stored in the registry, such as classification or association of the documents. In order for a business to search technically compatible partners, it has to download the CPP documents one by one from the repository to check compatibility.

If the registry is extended to store the necessary CPP internal data used in compatibility checking, the need for accessing and downloading CPP documents from the repository is avoided. This issue is not addressed in the ebXML architecture specifications. For this purpose, the Registry/Repository Service is re-designed to handle this extension. This is one of the contributions of the work within the scope of this thesis.

2. Building ontologies and then associating objects with the ontology instances makes the search of web sources and objects easier. ebXML Registry allows to build classification trees (i.e., ontologies) and associate registry objects with the nodes in the tree. DAML+OIL, a widely accepted RDF based semantic markup language, enables building richer ontology structures [9]. ebXML Registry is extended to support DAML+OIL ontologies and associate registry objects with these ontology instances. This is another contribution of the work within the scope of this thesis.

## 1.3 Organization of the Thesis

The organization of the thesis is as follows: a brief overview of ebXML architecture and the Registry/Repository Service is given in Chapter 2. Chapter 3

is reserved for the design of Registry/Repository Service. The implementation details are described in Chapter 4. The thesis concludes with Chapter 5 in which the presented study is discussed and the possible improvements for future work are stated.



## CHAPTER 2

### RELATED WORK

#### 2.1 Introduction

There are many business to business electronic commerce standards based on XML addressing diverse aspects of these issues. The general characteristics of these standards are, the description of the message formats exchanged (e.g. purchase orders), bindings to transport protocols, (e.g. HTTP), the sequencing (e.g. after sending a purchase order send an acknowledgement) and many more properties.

Some of these standards focus only on a subset of these properties, which are called, supporting standards; others provide recommendation for the complete set, called frameworks. ebXML is one of the most important frameworks. The other two important frameworks, Electronic Data Interchange (EDI) and Universal Description, Discovery and Integration (UDDI) are discussed in Sections 2.2 and 2.3. Then, a brief overview of Extensible Markup Language (XML) is given in Section 2.4. ebXML framework and the ebXML Registry/Repository Service are overviewed in Section 2.5.

The use of ontologies in registry services make the search of web sources and objects easier. ebXML Registry enables building ontologies with classification trees and associating registry objects with the nodes in the tree. DAML+OIL

is a widely accepted RDF based semantic markup language for building richer ontology structures. ebXML Registry can be extended to support DAML+OIL ontology. Section 2.6 gives an overview of DAML+OIL.

## 2.2 EDI (Electronic Data Interchange)

One of the first electronic data exchange standards is EDI, which runs on private networks. EDI was developed by the United Nations Electronic Data Interchange for Administration, Commerce and Transport (UN/EDIFACT) working group about twenty-five years ago. This was before the invention of Internet.

EDI can be described as an information sharing system utilizing an intermediate message store-and-forward entity and premeditated message syntax and interaction processes, which are modeled by trading partners around standard templates [10]. Partners have their own internal software systems. Special EDI adapters have to be implemented which will be able to interface with the internal system as well as communicate with the value added network [10].

The particulars of the message syntax and interaction process are negotiated between partners in advance. Sometimes a dominant partner will impose its standards on smaller partners [11].

EDI is a well-developed system with a lot of organizations using it extensively. However, it has some drawbacks, as listed below:

- EDI adapter software is too expensive for most organizations.
- The software has to be rewritten for different combinations of VANs, internal hardware and trading scenarios [10].
- VAN subscription costs and dedicated line costs are prohibitive for most small or medium enterprises [11].
- There is no common registry or discovery mechanism, partners have to retain information on institution codes, product codes, up-to-date catalogs etc. associated with every other partner they do business with [11].

## 2.3 UDDI (Universal Description, Discovery and Integration)

Another important initiative is the Universal Description, Discovery and Integration (UDDI) by Microsoft and IBM. UDDI is a service registry architecture that represents a standard way for business to build registry, discover each other and describe how to interact over the Internet [12].

UDDI is the name of a group of web-based registries that expose information about a business and its technical interfaces. These registries are run by multiple operator sites and can be used by anyone who wants to make information available about one or more business or entities, as well as anyone that wants to find that information [13]. Conceptually the information provided by UDDI registries consists of three components: "white pages" of company contact information; "yellow pages" that categorize business by standard taxonomies; and "green pages" that document the technical information about services [2].

The UDDI information model, defined through an XML schema, identifies five core types of information: business, service, binding, service specification information and relationship information between two parties. These structures use category bags for categorization progress. An item in a category bag contains a tModel key [14]. tModels provide the ability to describe compliance with specification, a concept or a shared design. When a particular specification is registered with the UDDI as a tModel, it is assigned a unique key, which is then used in the description of service instances to indicate compliance with the specification [14].

The categories referenced in the category bags can be used to find business or services of a particular category [2]. For example: a user looking for a particular product type can first obtain the product code from one of the defined taxonomies, like NAICS [15], then search the UDDI registry by using that APIs provided to find the business and their services that contain this code. However if the business fails to provide this exact code in its category bag, it becomes impossible to locate it in this way.

UDDI uses, Simple Object Access Protocol (SOAP) Messaging. SOAP is a lightweight protocol for exchange of information in a decentralized, distributed environment. SOAP provides an XML based messaging and remote procedure call mechanism for invoking methods. SOAP is composed of three parts: an envelope that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined data types and a convention for representing remote procedure calls and responses [16].

Drawbacks of UDDI framework can be listed as:

- A repository is missed in UDDI.
- UDDI does not provide a business process support.[2]
- Security is indirectly addressed by sending SOAP messages over HTTPS, instead of HTTP. [2]

## 2.4 XML (Extensible Markup Language)

XML stands for the Extensible Markup Language. It is designed to improve the functionality of the Web by providing more flexible and adaptable information identification [17].

Hyper Text Markup Language (HTML) provides universal methods for viewing data (i.e., bold, italic etc.), XML provides an advance in how data is described and exchanged by Web based applications. XML resembles and complements HTML.

XML is a notation for syntax; tagging the data with a meaning associated with it. For example: <name> Ian </name> means that Ian is a name. XML does not have a fixed format like HTML. Instead, XML is actually a "meta-language" (a language for describing other languages) which lets to design own customized markup language limitless different type of documents [18].

XML is desirable to transport data around networks, including Internet.

## **2.5 ebXML (Electronic Business Extensible Markup Language)**

### **2.5.1 Need and Vision**

ebXML stands for the electronic business extensible markup language. ebXML is an initiative from OASIS and United Nations Centre of Trade Facilitation and Electronic Business. (UN/CEFACT) [19].

The aim of ebXML is to provide the exchange of electronic business data in Business to Business (B2B) and Business to Consumer (B2C), environments. The initiative leverages from the success of EDI in large business and intends reaching to small and medium enterprises. In other words, the vision of ebXML is to create a single set of internationally agreed upon technical specification that consists of common XML semantics and related document structures to facilitate global trade [7]. It is important to note that ebXML is not about creating standard schemes for common business documents but instead is about creating an infrastructure.

### **2.5.2 General Architecture Overview**

The general architecture of ebXML, as shown in Figure 2.1 works as follows:

1. Company A submits its own business profile information to the ebXML Registry. The business profile submitted to the registry describes the company's ebXML capabilities and constraints.
2. Company B discovers the business scenarios supported by Company A. (Company B also has registered its business profile information to the registry).
3. Company B sends a request to company A, stating that they would like to engage in business scenario using ebXML.

The ebXML architecture provides a set of essential functional components [20]. These components are briefly described in the following sections:

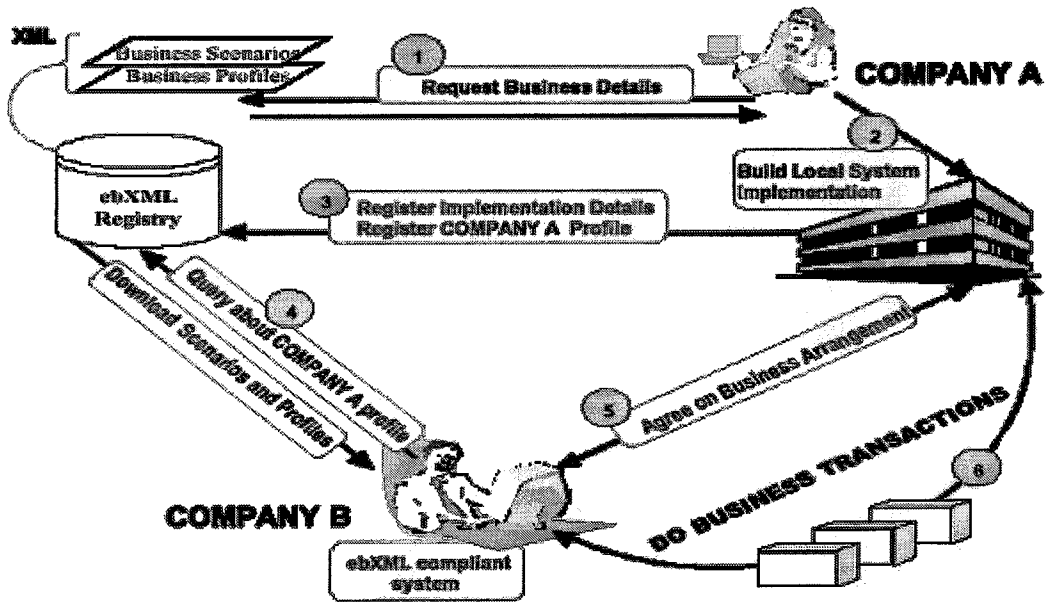


Figure 2.1: A high level use case of ebXML architecture [1].

### 2.5.2.1 Trading Partner Information

The details of how an organization is able to conduct business electronically are in the Collaboration Protocol Profile (CPP) document. CPP is an XML document, that specifies such items as how to locate contact and other information about the organization, the types of network and file transport protocol issues, implementation details (how the company does the business).

The details of two organizations that had been agreed to conduct business electronically are in the Collaboration Protocol Agreement (CPA) document. CPA is also an XML document that is formed by the combination of CPP documents of the two collaborating organizations.

### 2.5.2.2 Registry/Repository

Registry is the mechanism where relevant meta-data of business documents are stored. Registry stores relevant meta-data with a pointer to documents location, where these can be retrieved as a result of a query.

Repository is the location (i.e., file server) where documents are published

at by the registry resides and can be accessed by conventional means (i.e. http, ftp) [21].

### **2.5.2.3 Business Process Specification Schema**

The Specification Schema supports the specification of business transactions and the choreography of business transactions in to business collaborations. A set of standard patterns and a set of modeling elements common to those patterns accomplish it. Each business transaction can be implemented using one of many available standard patterns. These patterns describe the actual exchange of messages and signals between trading partners to achieve the required electronic transaction [22].

As a summary, the information of how an organization conducts its business is in the Business Process Specification Schema document. CPP and CPA documents deal with technical details where as the Business Process Specification Schema deals with the actual process.

### **2.5.2.4 Messaging Service**

ebXML Messaging Service provides a standard way to exchange messages between organizations. ebXML Messaging Service is structured to allow messaging reliability, persistence, security and extensibility without relying on proprietary technologies and solutions [23].

The ebXML Messaging Service defines the message enveloping and header document schema used to transfer ebXML messages over a communication protocol such as HTTP, SMTP, etc. It extends the base Simple Object Access Protocol with Attachments [16],[3],[24]. Figure 2.2 shows the logical structure of an ebXML message.

The ebXML Messaging Service may be conceptually broken down in to three parts: (i) an abstract service interface, (ii) functions provided by the Message Service Handler (MSH) and (iii) the mapping to underlying transport service(s).

The functional modules of the ebXML Messaging Service are [2],[23],[3],[25]:

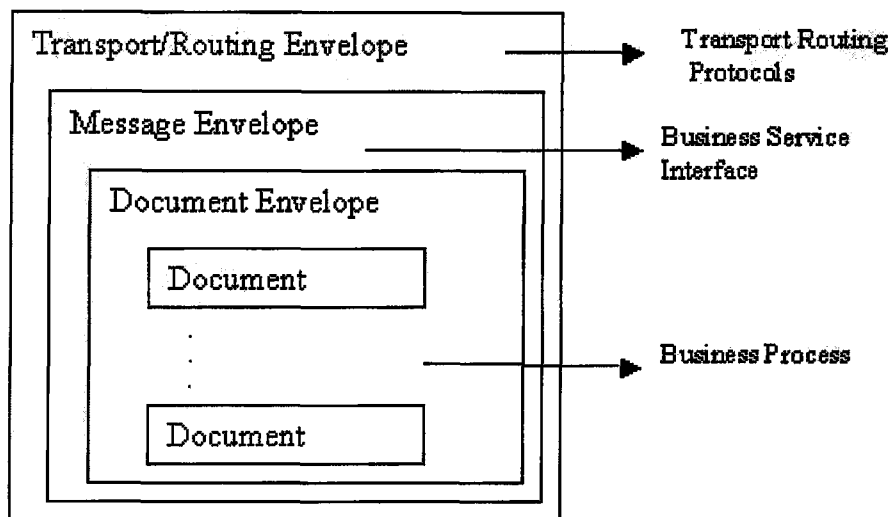


Figure 2.2: ebXML messaging conceptual view [2],[3].

1. Header Processing: The creation of the SOAP header elements for the ebXML message uses the input from the application, passed through the Message Service Interface, information from the CPA that governs the message and generated information such as digital signature, timestamps and unique identifiers.
2. Header Parsing: Extracting or transforming information from a received SOAP header or body element into a form that is suitable for processing by the MSH implementation.
3. Security Services: Digital signature creation and verification, authentication and authorization.
4. Reliable Messaging Services: Handles the delivery and acknowledgement of ebXML messages sent with delivery semantics of "Once And Only Once". The service includes handling for persistence, error notification and acknowledgement of messages requiring delivery.
5. Message Packaging: The final enveloping of an ebXML message in to SOAP messages with attachments container.

6. **Error Handling:** This component handles the reporting of errors encountered during MSH or application processing of a message.
7. **Message Service Interface:** An abstract service interface that applications use to interact with the MSH to send and receive messages which the MSH uses to interface with applications that handle received messages.

All of the interaction in between ebXML components must be done over the Messaging Service [8],[2],[20],[23],[3].

### **2.5.3 ebXML Registry/Repository Service**

Industry defined business processes and scenarios that are applicable to most of the business transactions and core components are stored in the ebXML Repository. As well as core components, profiles for businesses that have already registered themselves for performing ebXML transactions with other trading partners are also stored in ebXML Repository [2],[20],[23].

The set of services that manage the repository and enable the sharing of information between trading partners is provided by the ebXML Registry [1]. Communication into and out of the ebXML Registry has to be done over the ebXML Messaging Service [2],[23],[3],[1],[4].

The general ebXML Registry architecture consists of two components, ebXML Registry and ebXML Registry Clients. Registry client interfaces may be local to registry or to user [1]. Topologies supported by the registry with respect to clients and registry are shown in Figure 2.3.

The scenario where the registry provides a web based thin client application for accessing the registry that is available to the user, using a common web browser, is shown in the architecture on the left side of Figure 2.3. In this scenario, from the user's view, the registry client interfaces reside across the Internet and are local to the registry.

The scenario where the user is using a fat client registry browser application to access the registry is shown in the architecture on the right side of Figure 2.3.

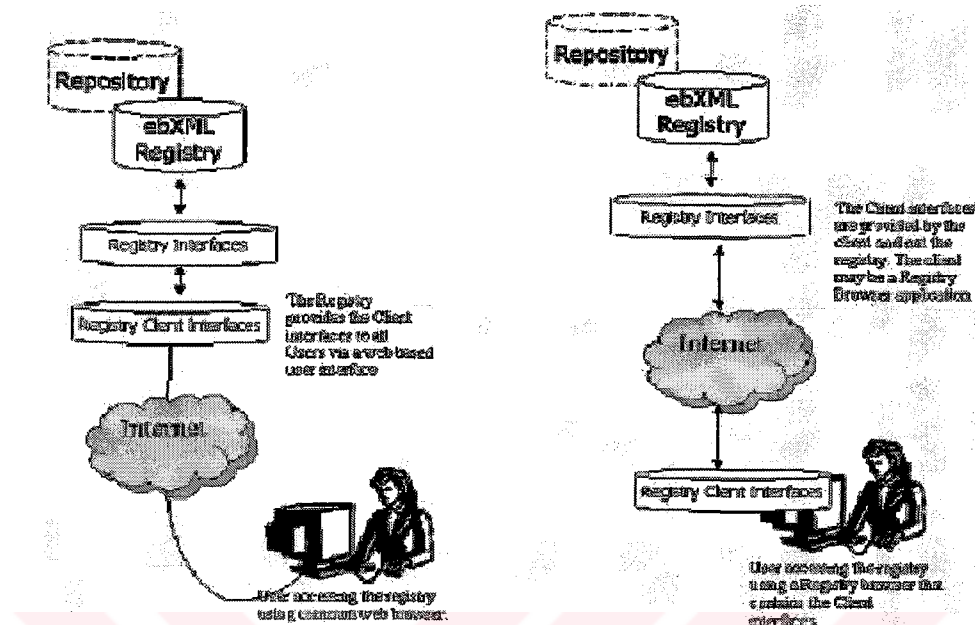


Figure 2.3: Registry architecture supports flexible topologies [1].

In this scenario, from the user's view, the registry client interfaces reside within the registry browser tool and are local to registry.

In both scenarios, the clients communicate with the registry using the ebXML Messaging Service, in the same manner as any ebXML applications communicate with each other.

In the ebXML framework, a particular implementation of the Registry/Repository Service is not specified. Only how other applications interact with the registry, that is, registry interface service mechanism and the minimum information model that the registry has to support, are specified. These specifications are documented in ebXML Registry Services Specification V1.0 and ebXML Registry Information Model V1.0 respectively, published by the ebXML Organization Committee.

There does not exist a specification for the repository development. Repository is desired as a file server, which can be accessed individually by the clients.

The registry can be examined in two parts, internal structure as Registry Information Model (ebRIM) and the registry services as Registry Service Spec-

ifications (ebRS).

### 2.5.3.1 ebRIM (ebXML Registry Information Model)

ebRIM defines the minimal internal structure of the ebXML Registry and outlines an object oriented design of this structure. The registry information should be kept in a relational database management system for permanent storage. ebRIM also gives the relational database design (i.e. table structure) of the ebXML Registry.

The aim of the ebRIM is to provide a blueprint or high-level schema for ebXML Registry. Information on the type of meta-data that is stored in registry as well as the relationship among meta-data classes is also provided to ebXML implementers.

According to ebRIM, the objects that the internal structure of ebXML Registry should contain are [4]:

**RegistryEntry Object:** For each content instance submitted to the registry, there exists an instance of RegistryEntry object. Meta-data about a repository item is provided by instances of RegistryEntry objects. The actual repository item is not contained in an instance of RegistryEntry class.

**Slot Object:** A dynamic way to add arbitrary attributes to RegistryEntry is provided by Slot objects.

**Association Object:** RegistryEntry objects that are used to define many-to-many associations between objects in the information model are Association objects.

**ExternalIdentifier Object:** Additional identifier information to RegistryEntry such as DUNS number, Social Security Number, or an alias name of the organization are provided by ExternalIdentifier objects.

**ExternalLink Object:** RegistryEntry objects that model a named URI to the content that is managed by the registry are ExternalLink objects.

**ClassificationNode Object:** ClassificationNode instances are RegistryEntry objects that are used to define tree structures where each node in the tree is a ClassificationNode. Classification trees constructed with ClassificationNodes are used to define classification schemes or ontologies.

**Classification Object:** RegistryEntry objects that are used to classify repository items by associating their RegistryEntry instances with a ClassificationNode within a classification scheme are Classification objects.

**Package Object:** RegistryEntry objects that group logically related RegistryEntry objects together are Package objects.

**AuditableEvent Object:** Objects that are used to provide an audit trail for RegistryEntry objects are Auditable event objects.

**User Object:** User objects provide information about registered users within the registry object instances.

**PostalAddress Object:** A simple reusable entity class that defines attributes of a postal address.

**Organization Object:** RegistryEntry objects that provide information on organizations, such as submitting organization are Organization objects.

The inheritance view of the registry objects is shown in Figure 2.4 [4].

ebXML Registry allow to build classification trees and associate the objects in the registry with these classification schemes via Classification objects. In Figure 2.5, there is a simple example showing the relation between ClassificationNode, Classification and RegistryEntry objects.

A Classification Scheme supports the followings:

- A RegistryEntry to be classified with a ClassificationNode object in a classification tree, by defining a Classification object that associates it.
- A RegistryEntry to be classified along multiple facets with multiple ClassificationNodes, by having multiple classifications that associate it.

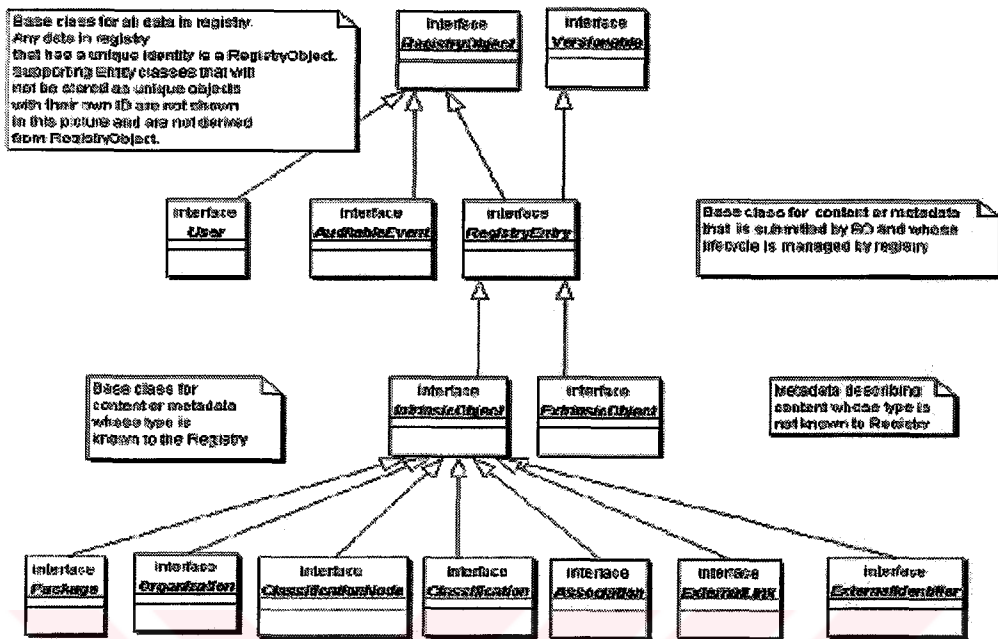


Figure 2.4: Information model inheritance view [4].

In order to provide a structured coded vocabulary, in various industries standardized taxonomies also referred as ontologies or coding schemes exists. A support for specific taxonomies is not defined by the ebXML Registry. But instead, a general capability to link registry entries to codes defined by various taxonomies are provided. Two alternatives in order to use standardized taxonomies for classification of registry entries are supplied by ebRIM, Full-Featured Taxonomy Based Classification and Light Weight Taxonomy Based Classification [4].

**Full-Featured Taxonomy Based Classification:** The information model provides a full-featured taxonomy based classification alternative based on Classification and ClassificationNode object instances. A standard taxonomy consisting of ClassificationNode instances as a classification tree, has to be imported in to the ebXML Registry in this alternative. A specification for converting standard taxonomies into ebXML Registry classification trees does not exist.

**Light Weight Taxonomy Based Classification:** A light weight alternative for classifying RegistryEntry instances by codes defined by standard taxonomies, where the submitter does not wish to import the entire taxonomy

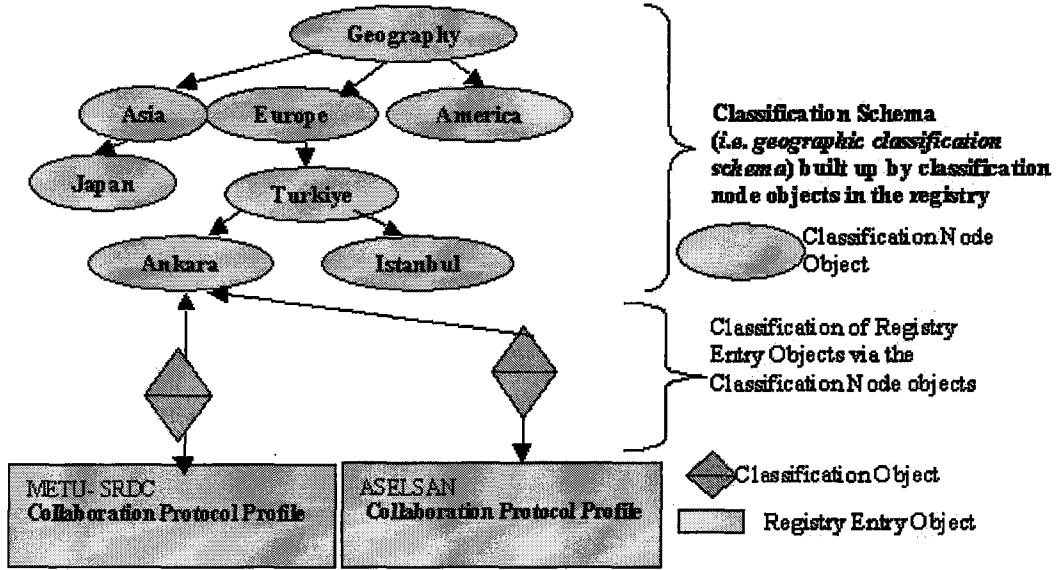


Figure 2.5: Classification example.

as a native classification scheme. The submitter adds one or more taxonomy related slots to the RegistryEntry instances for a submitted repository item in this alternative. A standardized taxonomy is identified by each slot's name, where the slot's value is the code within the specified taxonomy. For example: The book publishers in NAICS taxonomy is classified by a RegistryEntry instance having a slot name "NAICS", a slot type of "Classification" and a value "51113" [15]. There is no need to import the entire NAICS taxonomy nor to create Classification or ClassificationNode instances, in this alternative. [4]

Association of RegistryEntry objects, a part from classification schemes, is also supported by the ebXML Registry. A registry entry may be associated with zero or more registry objects. The information model defines an Association class for this purpose. An Association instance represents an association between one RegistryEntry instance to another. There are many types of associations between RegistryEntry objects. For example: An association between ExtrinsicObjects that catalogue a new CPP and an older CPP where the newer CPP supersedes the older CPP, is shown in Figure 2.6.

The information model also specifies a Security View System where each

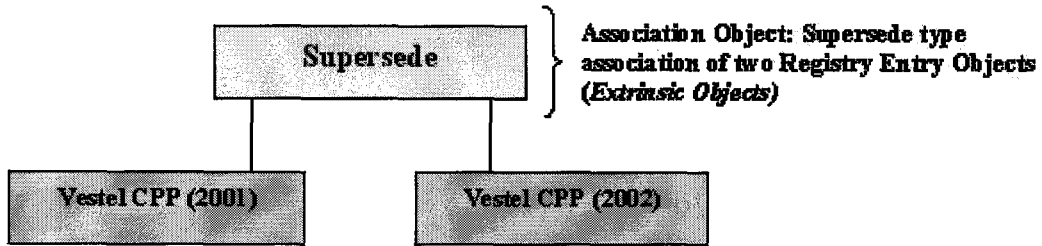


Figure 2.6: Association example.

RegistryEntry object is related with an AccessControlPolicy object for security.

### 2.5.3.2 ebRS (ebXML Registry Service Specification)

The services that have to be supplied by the ebXML Registry are defined by ebRS. The aim of ebRS is to provide the methods that are used by the clients to discover service-specific interfaces implemented by the registry. ebRS composes of two sub-services, namely: Object Manager and Object Query Manager Service [1].

**Object Manager Service:** This is the interface exposed by the registry where the object life cycle management functionality of the registry is implemented [1]. The Registry Client invokes this service's methods. Object Manager Service provides the functionality to manage the objects in the registry such as creation, deletion, etc.

**Object Query Manager Service:** This is the interface exposed by the registry that the Object Query Management of the registry is implemented [1]. Object Query Manager Service provides the functionality to query the objects in the registry.

ebRS gives the syntax specifications for the registry interface, called ebXML Registry Document Tag Definition (DTD), presented in Appendix B.

## 2.6 DAML+OIL

DAML+OIL is a widely accepted RDF based semantic markup language for building richer ontology structures [9].

DAML+OIL divides the world up into objects, which are elements of DAML classes and datatype values [5]. Datatype values are used to help define classes. DAML+OIL also allows definition of properties. Property definitions relate objects to other objects or objects to datatype values. Through DAML+OIL, it is also possible to restrict range on a property [5]. Relations such as: Sub-Class, Cardinality, SubProperty, Inverse Of, Transitive Property, Complement Of, Intersection, Union, etc can be defined among classes [9].

Individual objects represent the ontology instances that are used for association of objects within the ontology. A DAML+OIL ontology sample and an instance from the sample is given below: [9]

```
<daml:Class rdf:ID="Female">
    <rdf:subClassOf rdf:resource="#Animal"/>
</daml:Class>
<daml:Class rdf:ID="Male">
    <rdf:subClassOf rdf:resource="#Animal"/>
</daml:Class>
<daml:Class rdf:ID="Person">
    <rdfs:comment> Every person is a man or woman </rdfs:comment>
    <daml:disjointUnionOf rdf:parseType="daml:collection">
        <damlClass rdf:about="#Man"/>
        <damlClass rdf:about="#Woman"/>
    </daml:disJointUnionOf>
    <rdfs:subClassOf>
        <daml:Restriction>
            <daml:onProperty rdf:resource="#age"/>
        </daml:Restriction>
    </rdfs:subClassOf>
</daml:Class>
</daml:DatatypeProperty rdf:ID="age">
    <rdfs:comment> Age is a datatype Property </rdfs:comment/>
```

```

    <rdf:type rdf:resource="http://www.daml.org/2001/03/daml+oil#Integer"/>
</daml:DatatypeProperty>
<Person rdf:Id="Levent">
    <rdfs:comment> A Person of age 25 as DAML+OIL built ontology, with Id Levent
    <rdfs:comment/>
    <age> <xsd:integer rdf:value="25"/> </age>
</Person>

```

A part of a DAML+OIL built ontology is represented and a Person instance with id "Levent" is created in the example above. The Person class is defined as a disjoint union of two other classes, Man and Woman. Man and Woman classes are subclasses of Animal class, which is not shown in the example for the sake of simplicity. Also a property called age, is defined over the Person class. "Levent" is an instance of Person class having 25 as age property.

## CHAPTER 3

# DESIGN OF THE SYSTEM

### 3.1 Introduction

This chapter presents the design issues of the ebXML Registry/Repository Service. After giving a general flow of work of the ebXML Registry/Repository Service, design details, extensions made on registry internal structure and registry service Interfaces are presented.

### 3.2 General Architecture Design

The general architecture design and workflow of the ebXML Registry/Repository Service is shown in Figure 3.1.

As it can be seen from Figure 3.1, all communication in the Registry Service architecture is done over the Messaging Service. In the architecture, the Registry Server and Registry Client are applications that are always alive, waiting for ebXML messages to process.

The Registry Client and Registry Server act as two independent businesses, with their own Collaboration Protocol Profiles (CPP) describing their processes. Since Registry Client and Server are viewed as two independent businesses, a Collaboration Protocol Agreement (CPA) is formed out from their CPPs. This CPA has the details of how Registry Server and Registry Client contact with

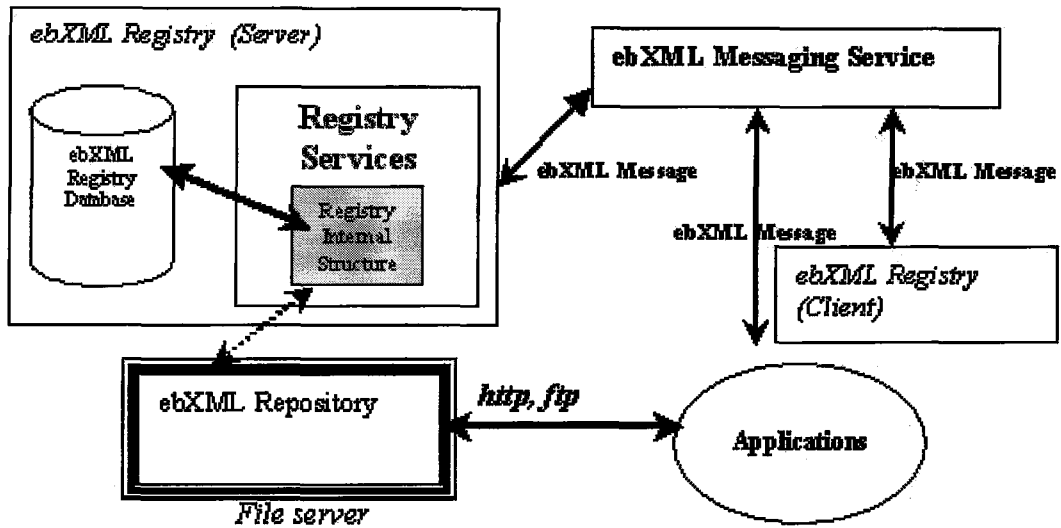


Figure 3.1: Architectural design and workflow of Registry/Repository Service.

each other, as of any other actual business. The generated CPA is given in APPENDIX F.

When an application needs to query the registry, it sends a registry query attached ebXML message to the Registry Client. As the client receives the incoming message, it starts the business process with the Registry Server, by using the registry CPA. The Registry Client sends a query attached ebXML message to the server. After the server gets the ebXML message, it detaches the query from the message, executes the query and prepares the results. The result of the query is attached to a new ebXML message and sent back to the Registry Client by the Registry Server. Finally the Registry Client passes the result message to the requester application.

The ebXML Repository is designed as a file server, accessed via conventional means, such as http or ftp. The repository can be accessed via applications and it is the responsibility of the applications to manage the repository.

The thin client model is used for the Registry Client, where the server supplies the services supported and client just invokes these service methods. For ebXML the ebXML Messaging Service developed at METU-SRDC is used.

The design of ebXML Registry Server is mainly divided in to two sections,

the Registry Internal Structure design and Registry Service design, as suggested in ebRIM and ebRS documents by the ebXML Organization Committee.

### **3.3 Design of Registry Internal Structure**

Since the registry data has to be kept in a relational database, all registry objects and members are mapped to relational database tables. When an instance of a registry object has to be created, the data read from the related database tables and the newly created object is filled with that data.

The Registry Internal Structure is similar to the specifications in ebRIM, with some additions and modifications. The composite object handling of the internal architecture is modified. Two other important additions are done on the internal structure; one to handle automatic CPA construction, the other to support DAML+OIL ontologies.

The composite object return of registry objects in ebRIM specifications are not very clear and efficient. Therefore, a modification for composite object function return values is done. Instead of returning array of objects as the ebRIM states, it is better to return array of the object ids. The caller can retrieve the details of each object if it is necessary. This modification improves computational time and memory usage.

#### **3.3.1 Automatic CPA Construction**

One of the most important points in ebXML framework is the engaging of business partners in business transactions. In order to handle this, each partner's capabilities (both commercial/business and technical) to engage in electronic message exchanges with other partners are described in the Trading Partner Profile (TPP) document. The agreed interactions between two partners are documented in Trading Partner Agreement (TPA). Thus, computing the intersection of two partners' TPP documents creates a TPA [23].

An enterprise may choose to present itself as multiple parties [23]. For example: it might represent a central office supply procurement organization and

manufacturing procurement organizations as separate parties.

The message exchange capabilities of a party are described in a CPP within the TPP documents. The message-exchange agreements between two parties are described by a CPA within the TPA documents. The CPP and CPA include the details of transport, messaging, document exchange protocols, security constraints and bindings to a Business Process Specification (or Process Specification) document that contains the definition of interactions between two parties while engaging in a specific business collaboration. The CPP document describes the capabilities of an individual party. A CPA document describes the capabilities that two parties have agreed to perform particular business collaboration. CPA documents define the "information technology terms and conditions" that enable business documents to be electronically interchanged between parties [26].

A party describes it self in a CPP document. To enable parties to find other parties that are suitable business partners to do business with, CPPs are stored in the repository. This means that, CPPs are stored in a file server and only some meta-data about the CPP are stored in the registry as an Extrinsic object, that just gives the name, id and location attributes or further classification, package attributes of it, if there exists any.

CPA is constructed from two CPPs'. The CPA formation algorithm works as follows [23],[26]:

1. Party A has already registered it self to the ebXML Registry.
2. Party B discovers the trading partner A by searching through the registry and downloads it's CPP from the repository. (Party B first finds the repository location of Party A's CPP from the registry and then downloads it from the repository).
3. Party B, from two CPPs, creates a CPA and sends to Party A.
4. Parties A and B negotiate and store the identical copies of CPAs. Then, they do the business.

In order to find compatible CPPs of Party B, all CPP documents have to be downloaded one by one and checked for compatibility. It is obvious that the CPA construction will be more feasible if this process can be automated.

If the ebXML Registry/Repository is modified so that, the CPP data of party's are also stored in the registry, the CPA construction can be automated. However, storing all CPP data in the registry is not feasible. It is better to store only that data that is used in CPA construction or compatible checking of CPPs. The registry objects are also stored in a relational database which enables a faster search on a given party's compatible party candidates, via SQL.

The ebXML Collaboration Protocol Profile and Agreement Specification document give the basic tasks in forming a CPA from two CPPs. It is stated that, in order to perform a successful CPA from two CPPs, Role, Packaging and Transport attributes of the two CPPs have to match [26].

CPA algorithm can work within the registry (without referring to repository to download CPP files), if Collaboration Role, Delivery Channel, Process Spec, Party Info, Role, Service Binding, Composite, Constituent, Simple Package Part, Transport, Transport Security attributes of a CPP (that represent the minimal necessary data for Role, Packaging and Transport matching) are stored in the registry database. In order to work on these CPP attributes, there also has to be related registry entry objects in the registry internal structure.

These additional registry entry objects for automatic CPA construction can be categorized as registry related information and can be inherited from Intrinsic objects in the registry inheritance hierarchy.

As a summary, in order two CPPs to be compatible; Role, Transport, Transport Security and Document Packaging attributes of these two CPPs have to match with each other. These attributes are registry internal objects and stored in the registry database. Therefore, for a given CPP a SQL query can be executed in order to retrieve the candidate CPPs having matching attributes.

The matching process for compatible CPP search, derived from the CPA Construction Algorithm, implemented for UDDI registries in [27] is presented in

APPENDIX E. This matching process is used in ebXML registry as follows:

The matching CPP search algorithm can be examined in three parts, Role matching, Transport matching and Package matching, as described in the following sections.

### 3.3.1.1 Role Matching

---

**Algorithm 1** Matching Roles Algorithm (Binary-matching case) [27].

---

```
for each <Collaboration Role> attribute data in CPP A
  for each <Collaboration Role> attribute data in CPP B
    if (CPP A Process Spec Name equal CPP B Process Spec Name)
      and (Role Name of CPP A NOT equal Role Name of CPP B)
        then Mark CPP A and CPP B as Compatible Candidates
```

---

Matching Roles algorithm can be converted to the following SQL query, (using the relational database tables given in APPENDIX A) as follows.

#### SQL Query 3.1 *Role Matching Query*

```
SELECT C1.id, C2.id FROM CollaborationRole C1, CollaborationRole C2
WHERE C2.ProcessSpecId IN (
  -- CPPA.<ProcessSpec>.Name = CPPB.<ProcessSpec>.Name
  SELECT P2.id From ProcessSpec P1 , ProcessSpec P2 Where
    P1.name = P2.name AND P1.id = C1.processSpecId AND
    P2.id = C2.processSpecId
)
AND C2.roleId IN (
  -- CPPA.<Role>.Name <> CPPB.<Role>.Name
  SELECT R2.id FROM Role R1, Role R2 Where
    R1.name <> R2.name AND R1.id = C1.roleId
    AND R2.id = C2.roleId
)
AND C1.id IN (
  -- For each CollaboratrionRole in CPPA
  SELECT id FROM CollaborationRole Where partyInfoId in (
    SELECT id FROM PartyInfo Where CPPId ='urn:uuid:CPP_1'
  )
)
)
```

Call the result set of the SQL Query 3.1 as Set **ROLE**.

### 3.3.1.2 Transport Matching

---

**Algorithm 2** Transport matching algorithm [27].

---

```
for each <Service Binding> element marked in Role Matching (of CPP A)
  1. Get ChannelId of CPP 1, ChannelId CPP A
  2. Find <Delivery Channel> having ChannelId = ChannelId of CPP A
  3. Get TransportId of CPP 1, Transport id CPP A
  4. Find <Transport> having TransportId = TransportId of CPP A
for each <Service Binding> element marked in Role Matching (of CPP B)
  1. Get ChannelId of CPP 2, ChannelId CPPB
  2. Find <Delivery Channel> having ChannelId = Channel id of CPP A
  3. Get TransportId of CPP 2, TransportId CPP B
  4. Find <Transport> having TransportId = TransportId of CPP B
for each <Transport> element marked in Role Matching (of CPP A)
  if CPPA.<Transport>.SendingProtocol =
    CPPA.<Transport>.ReceivingProtocol
    for each <Transport> element marked in Role Matching of CPP B
      if CPPB.<Transport>.SendingProtocol =
        CPPA.<Transport>.ReceivingProtocol
      then Mark these <Service Binding>,<Transport> Elements
        Within Marked <Transport> elements of CPPA and CPPB,
        Get <Transport Security> elements
if CPPA.<TransportSecurity>.Protocol
  = CPPB.<TransportSecurity>.Protocol
  and CPPA.<TransportSecurity>.Version
  = CPPB.<TransportSecurity>.Version
  then mark CPP A and CPP B as compatible candidates.
```

---

Transport Matching algorithm can be converted to the following SQL query, (using the relational database tables given in APPENDIX A) as follows.

#### SQL Query 3.2 *Transport matching query*

```
SELECT S1.packageId, S2.packageId FROM ServiceBinding S1,
                                     ServiceBinding S2
Where S2.channelId In (
  SELECT D2.channelId FROM DeliveryChannel D1DeliveryChannel D2
  Where S1.channelId = D1.ChannelId AND D2.transportId in (
    -- Transport Match
    SELECT T2.id FROM Transport T1, Transport T2 Where
      T1.senderProtocol = T2.receiverProtocol AND
```

```

        T1.receiverProtocol = T2.senderProtocol AND
        T1.senderProtocolVer = T2.receiverProtocolVer AND
        T1.receiverProtocolVer = T2.senderProtocolVer AND
        T1.id = D1.transportId )
    -- Transport Security Match
    AND D2.transportId in (
        SELECT TS2.transportId FROM TransportSecurityTS1,
                                TransportSecurity TS2
        Where
            TS1.protocolVersion = TS2.protocolVersion AND
            TS1.transportId      = D1.transportId )
    )
    AND (S1.channelId,S2.ChannelId) IN SET ROLE

```

Call the result set of the SQL Query 3.2 as Set **SERVIEBINDING**.

### 3.3.1.3 Package Matching

---

**Algorithm 3** Package matching algorithm.[27]

---

```

for each <Service Binding> element marked in Transport Matching
    1. Get PackageId 's of, CPPA and CPPB
    2. Find <Packaging> having packageId = packageId of CPPA and CPP B
for each<SimplePart> element marked in Packaging element (of CPP A)
    if there Exists <SimplePart> element within marked Packaging of CPPB
    with CPPA.<SimplePackagePart>.mimeType =
        CPPB.<SimplePackagePart>.mimeType
    if structurally <CompoisteList> element subtrees are Equal
        mark CPP A and CPP B as compatible CPPs.

```

---

Package Matching algorithm can be converted to the following SQL query, (using the relational database tables given in APPENDIX A) as follows.

#### SQL Query 3.3 *Package matching query*

```

SELECT C1.packageId,C2.packageId FROM Composite C1, Composite C2
Where C1.mimeType = C2.mimetype
AND NOT EXISTS (
    -- Structurally compare
    SELECT order_no FROM Constituent Where compositeId = C2.id
    MINUS

```

```

        SELECT order_no FROM Constituent Where compositeId = C1.id
    )
    AND NOT EXISTS (
        SELECT order_no FROM Constituent Where compositeId = C1.id
        MINUS
        SELECT order_no FROM Constituent Where compositeId = C2.id
    )
    AND (C1.packageId,C2.packageId) In (
        SELECT SPP1.packageId,SPP2.packageId FROM SimplePackagePart SPP1,
                                                SimplePackagePart SPP2
        Where SPP1.mimeType in (
            SELECT mimeType FROM SimplePackagePart Where id=SPP2.id)
    )
    AND (SPP1.packageId,SPP2.packageId) In Set SERVICEBINDING.

```

The complete SQL query statement for the automatic compatible CPP search is given in APPENDIX C. This final query can be embedded to the database via a stored procedure and if the necessary indexes are created on related tables, the compatible CPP search or automatic CPA construction can be efficiently used.

This automation process also necessitates both of the compatible CPP candidates have to be in the ebXML Registry.

Query support for these additional CPP internal data objects are also handled in the Registry Service Specifications.

### 3.3.2 DAML+OIL Ontology Support

Building ontologies and associating objects with the ontology instances makes the search of objects easier. ebXML Registry allows building classification trees and associating objects with the nodes of the tree.

DAML+OIL allows building richer ontology structures than ebXML classification trees. DAML+OIL allows defining properties and instances for ontology classes. ebXML can be extended to support DAML+OIL ontology structures. However, to implement a full support for DAML+OIL ontology can be a research area. In the scope of this thesis, only a small part of this support is designed and implemented. Service Taxonomy ontology, implemented by METU-SRDC,

shown in Figure 3.2 is chosen as the example ontology to study on. The definition of the Service Taxonomy ontology in DAML+OIL syntax written by METU-SRDC, is given in APPENDIX D.

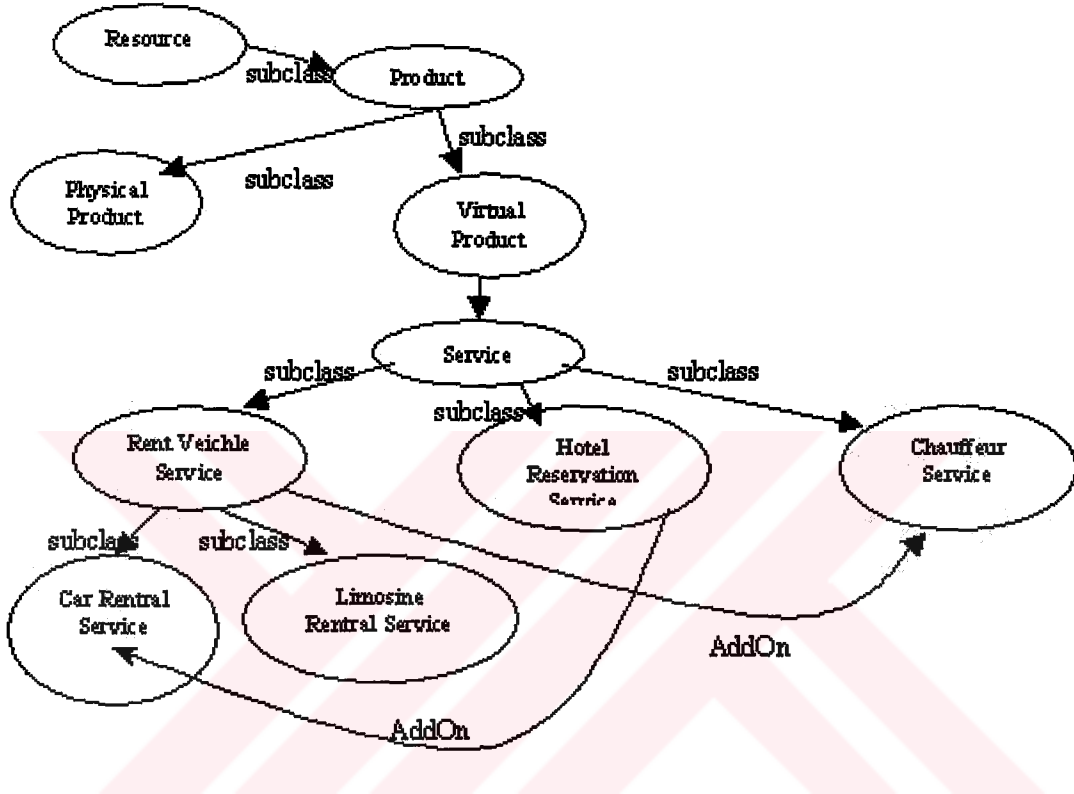


Figure 3.2: Service taxonomy example[5].

The Service Taxonomy example in Figure 3.2 gives a sample ontology on services. SubClass relations show the subservice relation between services. Car Cheuffeur Service is a subService of Chauffeur Service. Add On relations show that a service is an Add On of another service. Rent Veichle Service has an Add On relation with Chauffeur Service, because a driver is also needed after renting a car. A car rental is probably needed after a hotel reservation that is because Hotel Reservation Service has an Add On relation with Car Rental Service.

In order to store the ontology in the Registry, first a relational database schema has to be designed to map the ontology and the classes that store the instances. Then the associated registry internal objects have to be determined. SubClass, SubProperty and Restriction properties of DAML+OIL ontology are

selected to be supported within this thesis. Three relational database tables are created to store these three properties (i.e., Relational database tables to store SubClass, SubProperty and Restriction relations among classes). In order to store the ontology instances, a relational database table is created for each class in the Service Taxonomy (i.e., a relational database table for, Hotel Reservation Service, Car Rental Service...etc.). The registry internal objects corresponding to both instance and property tables are also designed.

The association of DAML+OIL ontology instances and other registry objects are handled in the same way as in the classification tree ontology in the Registry Service. A Classification object is used to represent each association of a registry object with an ontology instance.

Query support for these additional objects are also designed in the Registry Service Specifications.

After the addition of automatic compatible CPP search and DAML+OIL ontology support to registry internal structure, the inheritance hierarchy is re-designed considering all these additions, as in Figure 3.3.

Common characteristics of registry internal objects are:

1. There is no direct access to a member variable of an object. Each member variable in the object can be accessed by the appropriate `getMemberVariable()` or `setMemberVariable(arg...)` function.
2. Each object's constructor allocates its own member variables memory space.
3. Each object has a `objectnameRetrieve(arg objectId)` member function to retrieve the contents of the member variables by the appropriate data in registry database. This function executes a SQL query and sets the members variables according to the result set of the query.
4. The use of composite objects as function arguments is replaced by the use of composite object ids. Instead of array of object usage in function arguments, array of object ids are used. The ebRIM member functions are

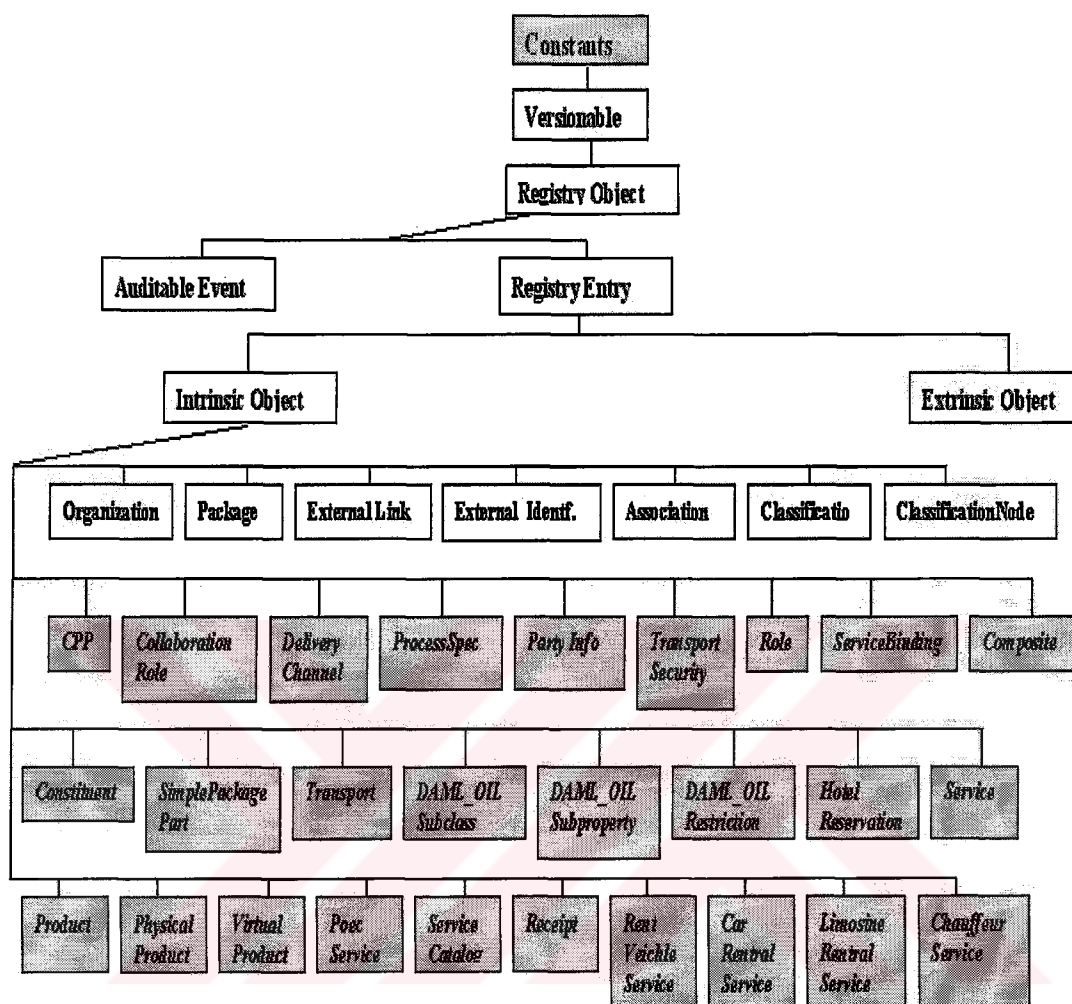


Figure 3.3: Registry inheritance hierarchy

re-designed for this usage. UUID is an object to deal with the object ids in the registry.

The newly added objects to the registry internal structure for compatible CPP search and DAML+OIL ontology support also obey the same general characteristics as the other objects given in eBRIM specifications.

Other than the registry objects in the registry inheritance diagram in Figure 3.3, there are also other objects that has to be in the internal structure which are shown in Figure 3.4. These objects also have the same general characteristics described above.

The other important registry internal objects are:

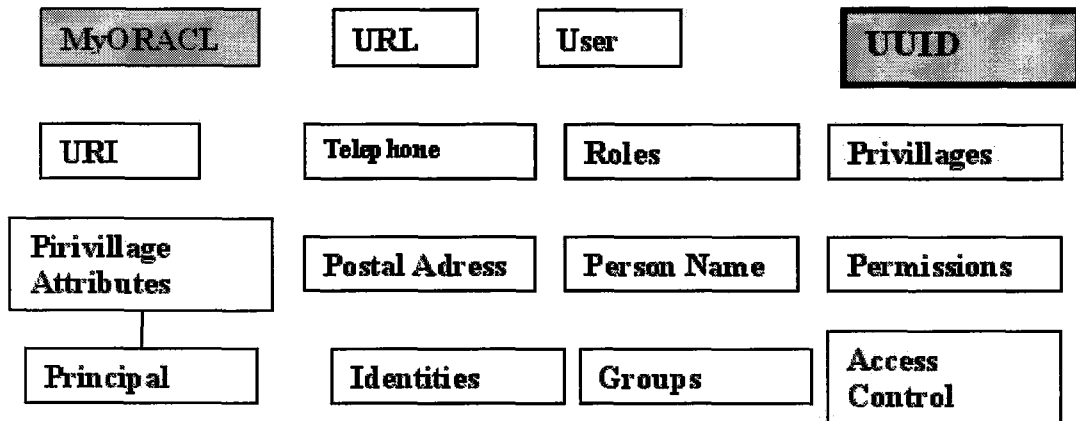


Figure 3.4: Other registry internal objects.

**MyORACLE Object:** Since the registry data is stored in a relational database, the MyORACLE object handles the communication via the registry objects with the database management system. The connection, querying and error handling with database are handled through this object.

**UUID Object:** A simple reusable entity class where object ids are managed.

**URL Object:** A simple reusable entity class where the URL attributes of the registry entry objects are stored.

**User Object:** User object provides the information about registered users within the registry object instances

**URI Object:** A simple reusable entity class where the URI attributes the registry entry objects are stored.

**Telephone Object:** A simple reusable entity class that defines attributes of a telephone number.

**Postal Address Object:** A simple reusable entity class that defines attributes of a postal address.

**PersonName Object:** A simple reusable entity class that defines attributes of user identifications.

**Roles, Privillages, PrivillageAttributes, Principal, Identities, Groups  
AccessControlPolicy and Principal Objects:** Simple reusable entity  
classes that define authentication attributes in the registry.

### 3.4 Design of Registry Service Interfaces

Instead of directly accessing the members and member functions of the registry internal objects, there is an interface mechanism on both Registry Client and Server. The predefined services and interface syntax are given as an XML Document Type Definition (DTD) of ebRS in APPENDIX B.

The interface between Registry Server and Client is done via XML queries over the Messaging Service. The Client sends an XML query document (compatible with the Registry Service DTD) attached ebXML message to the Registry Server. The Server parses the XML document, executes the query and returns the query result as another new ebXML Message.

The execution flow of Registry Server among its' internal objects, shown in Figure 3.5, is as follows:

1. Registry Server extracts the attached request document from the incoming ebXML message and gets the registry query. Then, passes the bare query document to its RegQuery internal object.
2. RegQuery object parses the registry request.
3. If the request is a SubmitObject or RemoveObject request (an Object Management request), then
  - (a) The objects to be submitted or removed are created and filled with the data in the request document. Then, each object's insert or delete member functions are called in order to operate with registry database.
4. Else

- (a) A SQL statement is formed out from the request document.
  - (b) The SQL statement is passed to a related Registry Server internal object.
  - (c) The related internal object prepares the result string of the query and passes the string back to RegQuery object.
5. The RegQuery object passes the result string back to the Registry Server.
6. Finally the Registry Server object sends the result string, as an XML document attached ebXML message to the Client, via ebXML Messaging Service.

According to the Registry Service Specifications there can be eight possible registry query types. By the addition of new objects to the registry internal structure for automatic CPA construction and DAML+OIL ontology support, three new query types are introduced. The optional SQL query support is also implemented.

The possible registry service query types are as follows:

**SubmitObjectsRequest:** A registry request denoting a new object submission.

**RemoveObjectsRequest :** A registry request denoting an existing object deletion.

**FilterQuery:** A registry request for getting primitive object attributes for a quick view of the object.

**ReturnRegistryEntry:** A registry request to get all meta-data of an object in registry.

**ReturnRepositoryItem:** Similar to ReturnRegistryEntry request, but if the item is an Extrinsic, Package or ExternalLink object, a reference to it's DTD document is also retrieved.

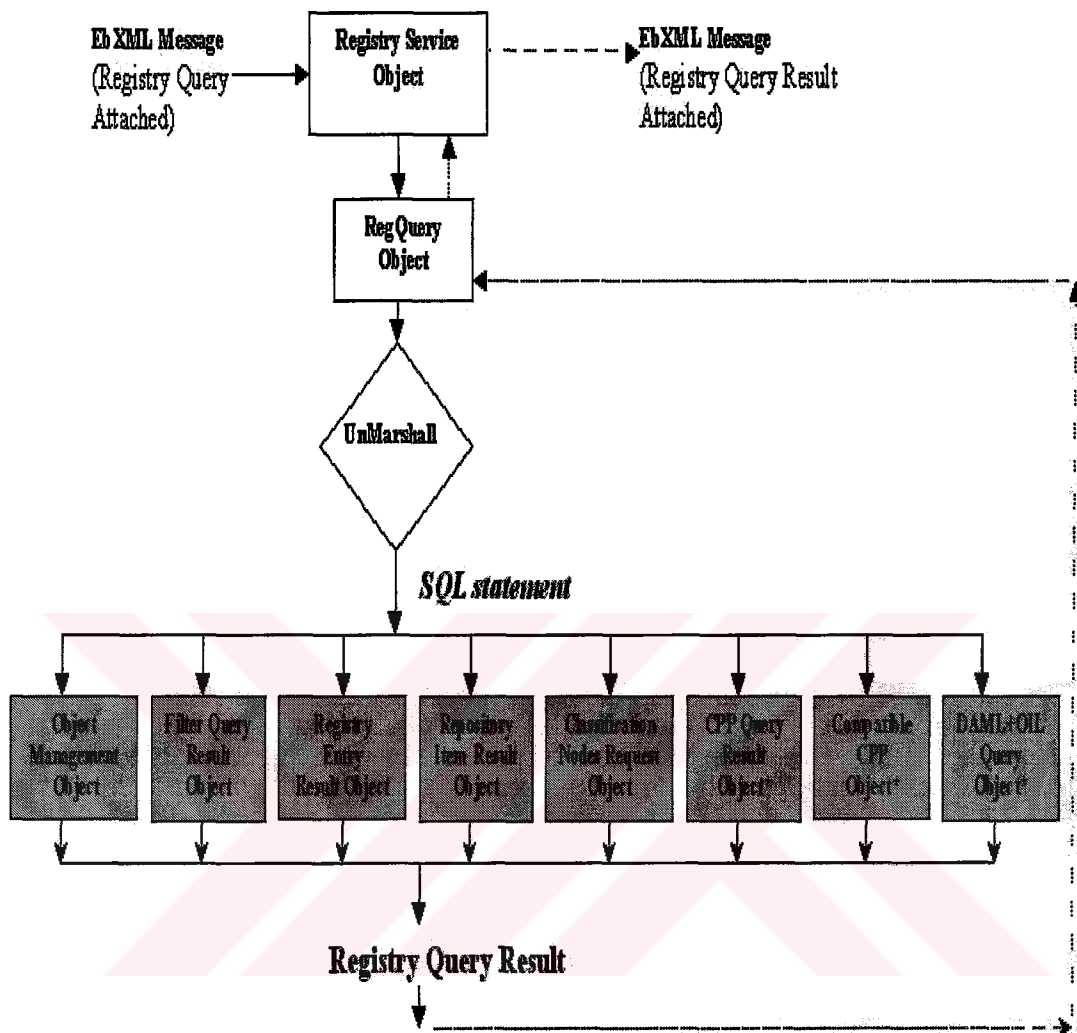


Figure 3.5: Registry Server execution Flow.

**CPPQuery:** In order to automate CPA construction, the CPP data is stored in registry. CPPQuery request is to get the data for CPPs in the registry. This query type is a modification made on ebRS.

**ReturnCompatibleCPP:** Since compatible CPP search mechanism is embedded in the internal structure of registry, ReturnCompatibleCPP is to query the compatible CPP's for a given CPP in the registry. This query type is also a modification made on ebRS.

**DAMLOILQueryRequest:** A query request to query the DAML+OIL ontology structures and instances. This query type is another modification

made on ebRS.

**GetRootClassificationNode:** A registry query to query the objects in classification trees that are root nodes and stored as ClassificationNodes in the registry.

**GetClassificationTree:** A registry query for getting a classification tree below a given node in a classification tree.

**GetClassifiedObjectsRequest:** A registry query to get objects that are classified by a given classification tree node.

**SQLQuery:** An optional query type for advanced users familiar with registry database, to build up their own SQL queries.

According to the root element in the registry query document, one of the query types in above is specified and executed.

Design details and the flow of execution of each query type with examples are given in Sections 3.4.1 till 3.4.12 below. Examples for each query type use the sample data in the registry database. Error case is given in Section 3.4.13.

### 3.4.1 SubmitObjectsRequest

SubmitObjectsRequest is an object submission request to the registry. The query request document contains the data of the new registry object to be submitted.

A related registry internal object for each (new) object in submit request is created. This newly created internal objects' member variables are filled with the data given in the query document. Then, for each created internal object, the insert member function is called to submit the object to the registry database.

Each object in the registry has to have a universally unique identifier. These unique ids are in the form "urn:uuid:#####-####-####-#####", where each # represent a hexa-decimal number. The submitter can explicitly give the id for the object being submitted or he/she can let the registry to assign

a universally unique id to the object. If the object id is explicitly given, the registry will assume the id is universally unique and will not check whether it is true or not. In the case of registry generated object id's, the Universally Unique Id generation algorithm [28],[29] is used. The algorithm roughly generates a 128-bit hexa-decimal number by using current date, time and machine internal codes, which is proven to be universally unique. In the case of registry generated object ids', an object id can be referenced within the same document, where all referenced objects will have the same id when submitted to the registry.

The object creation and submission takes place in the ObjectManagement internal object of the Registry Server. An example for object submission request and it's execution is as follows:

**ebXML Registry Query 3.1** *SubmitObjectRequest query example.*

```
<SubmitObjectRequest>
  <RegistryEntryList>
    <ClassificationNode id='istanbul' name='istanbul'
      parent='urn:uuid:Turkey.1'
    >
      <Association id='urn:uuid:Association.1' associationType='Package'>
        sourceObject='istanbul' targetObject='urn:uuid:CPP.2'>
      </Association>
    </ClassificationNode>
  </RegistryEntryList>
</SubmitObjectRequest>
```

In the ebXML Registry Query 3.1 example, two objects are submitted to the registry, i.e., an Association and a ClassificationNode object. The ClassificationNode object's parent attribute and Association object's id and targetobject attributes are given explicitly to the registry. However, the ClassificationNode object's id attribute is not in universally unique id format, hence there has to be a unique id generation for this object by the registry. Further more, the ClassificationNode's id attribute, is referenced by another object, (i.e. Association object's sourceobject attribute) thus, there will not be another id generation for this referenced id.

A ClassificationNode and an Association object is created and filled with the data given in query document and each objects insert to database member function is called. A success or error message indicating the result is returned. The success message is presented in the ebXML Registry QueryResult 3.1.

### **ebXML Registry Query Result 3.1** *SubmitObjectRequest query example result.*

```
<!--ebXML REGISTRY/REPOSITORY MECHANISM Developed by Levent CARKACIOGLU-->
<RegistryResponse status='success'>
</RegistryResponse>
```

### **3.4.2 RemoveObjectsRequest**

RemoveObjectsRequest is a deletion request of an object from the registry. The query request contains the ids' of the objects to be deleted. The deletion has two alternatives, one from just the repository and the other from both registry and repository. Since, the repository is a file server and is under control of the clients', the first choice is left to clients' responsibility and is not implemented.

In order to delete a registry object, the unique id of the object to be deleted has to be known.

The object creation and deletion takes place in the ObjectManagement internal object of the Registry Server.

An example for object deletion request and it's execution is as follows:

### **ebXML Registry Query 3.2** *RemoveObjectsRequest query example.*

```
<RemoveObjectsRequest deletionScope='DeleteAll'>
  <ObjectRefList> <ObjectRef id='urn:uuid:Association1'> </ObjectRefList>
</RemoveObjectsRequest>
```

In the ebXML Registry Query 3.2 example, a RegistryEntry internal object is created for temporal usage. The id attribute of this temporal object is set to the id attribute value in the query. The object type specifying which registry internal object to be deleted, is retrieved from the registry database. According to the object type, the actual object which is requested for deletion is created and it's id attribute is set with the id attribute value in the document. Finally the objects "delete from database" member function is called and the object is deleted from the registry database. A success or error message indicating the result is returned. The success message is presented in the ebXML Registry QueryResult 3.1.

### 3.4.3 FilterQuery

FilterQuery is a request type for querying the registry objects. It gives a short view of the requested objects' object Id's, object URN's and content URI's.

From the FilterQuery query request document a SQL query is formed. A RegistryEntry internal object is created for temporal usage. By using this temporary object, the object ids in the result set of the formed SQL query are retrieved. Then, using each object's id, the id, urn and uri values of the objects are retrieved from the database. Finally, a result string is formed.

The SQL query construction from FilterQuery type queries is done in the RegQuery internal object of the Registry Server. The retrieval of objects of the prepared SQL query and result string preparation takes place in the FilterQuery object of the Registry Server.

Examples for FilterQuery requests and their execution are as follows:

#### ebXML Registry Query 3.3 *Filterquery query example-1.*

```
<FilterQuery>
  <ClassificationNodeQuery>  <HasParentNode>  <HasParentNode>
    <ClassificationNodeFilter>
      <Clause>                <SimpleClause leftargument='name'>
        <StringClause StringPredicate='contains'> Geo </StringClaues>
      </SimpleClause>        </Clause>
    </ClassificationNodeFilter>
  </HasParentNode>  </HasParentNode>  </ClassificationNodeQuery>
</FilterQuery>
```

The ebXML Registry Query 3.3 example, is to query a classification tree in registry. It is, to query the ClassificationNode objects whose parent of parent node's name attribute contains the 'Geo' word.

The SQL statement of the ebXML Registry Query 3.3 example, from the point of registry database view is:

```
SELECT ID FROM CLASSIFICATIONNODE WHERE PARENT IN (
  SELECT ID FROM CLASSIFICATIONNODE WHERE PARENT IN (
    SELECT ID FROM CLASSIFICATIONNODE WHERE
      NAME LIKE '%GEO%') )
```

From the query in the ebXML Registry Query 3.3 example, the SQL statement (as given above) is formed out. The RegistryEntry internal object is created for temporary access to registry database. The member function of the temporary object is called with this SQL statement and object ids are retrieved from the registry database. Then, for each object id in the result set, the temporary registry object is reused to retrieve the object Id, objectURN and contentURI attributes of the object.

Finally, the ebXML Registry Query Result 3.2 is formed from the ebXML Registry Query 3.3.

**ebXML Registry Query Result 3.2** *FilterQuery query example-1 result.*

```
<!--ebXML REGISTRY/REPOSITORY MECHANISM Developed by Levent CARKACIOGLU-->
<!--SELECT ID FROM CLASSIFICATIONNODE WHERE PARENT IN ( SELECT ID
FROM CLASSIFICATIONNODE WHERE PARENT IN (SELECT ID FROM
CLASSIFICATIONNODE WHERE(name LIKE'%Geo%')))-->
<RegistryResponse status='success'>
  <ClassificationNodeQueryResult>
    <ClassificationNodeView objectURN='Japan'
      contentURI='ClassificationEx' objectIf='urn:uuid:Japan.1'>
    <ClassificationNodeView objectURN='Turkey'
      contentURI='ClassificationEx' objectIf='urn:uuid:Turkey.1'>
  </ClassificationNodeQueryResult>
</RegistryResponse>
```

The ebXML Registry Query 3.4 example, is to query the objects which are classified by; the ClassificationNodes whose id attribute contains 'urn:uuid:Turkey' word.

**ebXML Registry Query 3.4** *FilterQuery query example-2.*

```
<FilterQuery>
  <RegistryEntryQuery>
    <HasClassificationBranch>
      <Clause>
        <SimpleClause leftArgument='id'>
          <StringClause StringPredicate='contains'> Turkey </StringClaues>
        </SimpleClause>
      </Clause>
    </ClassificationNodeFilter>
  </HasClassificationBranch>
</RegistryEntryQuery>
</FilterQuery>
```

The SQL statement of the ebXML Registry Query 3.4 example, from the point of registry database view is:

```
"SELECT TARGETOBJECT FROM CLASSIFICATION WHERE
SOURCEOBJECT IN ( SELECT ID FROM CLASSIFICATIONNODE WHERE
ID LIKE '%urn:uuid:Turkey1%' ) )"
```

From the query in the ebXML Registry Query 3.4 example, the SQL statement (as given above) is formed out. The RegistryEntry internal object is created for temporary access to registry database. A member function of the temporary object is called with this SQL statement and the object ids are retrieved from the registry database. Then, for each object id in the result set, the temporary registry object is reused to retrieve the object Id, objectURN and contentURI attributes of the object. And finally, the ebXML Registry Query Result 3.3 is formed from the ebXML Registry Query 3.4.

**ebXML Registry Query Result 3.3** *FilterQuery query example-2 result.*

```
<!--ebXML REGISTRY/REPOSITORY MECHANISM Developed by Levent CARKACIOGLU-->
<!--SELECT TARGETOBJECT FROM CLASSIFICATION WHERE SOURCEOBJECT IN(
SELECT ID FROM CLASSIFICATIONNODE WHERE ID LIKE'%urn:uuid:Turkey1%')-->
WHERE(name LIKE'%Geo%'))-->
<RegistryResponse status='success'>
  <RegistryEntryQueryResult>
    <RegistryEntryView objectURN='CPP_1
      contentURI='CPP Example1' objectIf='urn:uuid:Ccpp.1'>
    </RegistryEntryQueryResult>
  </RegistryResponse>
```

#### 3.4.4 ReturnRegistryEntry

ReturnRegistryEntry is a request type provides detailed querying in the registry. Despite the FilterQuery it gives all details of the requested objects. And some additional meta-data about the associated objects can also be requested by; withClassifications (get all the classification objects that the object acts as sourceobject), withSourceAssociation (get all the association objects that the object acts as sourceObject), withTargetAssociations (get all the association objects that the object acts as targetObject) and withExternalLinks (get all the externalLink objects that the object is externally linked to) keywords.

From the ReturnRegistryEntry query request, a SQL query if formed. Then a RegistryEntry object is created for temporal usage. By using this temporary

object, the object ids' and object types' of the formed SQL query result are retrieved. For each object id and corresponding object type, the related registry internal objects are created. The member variables of these internal objects are retrieved from the registry database. At this point, if any of the withClassification, withSourceAssociation, withTargetAssociation, WithExternalLinks keyword exists, the object's associated classification, association or externallink objects are also retrieved from the registry database.

The retrieval of SQL queries and result string preparation takes place in the RegistryEntryResult internal object of the Registry Server.

An example for ReturnRegistryEntry request and it's execution is as follows:

In the ebXML Registry Query 3.5 example, the objects whose id attribute contains the 'Turkey' word and the classification objects that they (objects whose id attribute contain the 'Turkey' word) act as sourceobject and the association objects that they act as sourceObject having HasMember associationType are queried.

The SQL statements for the ebXML Registry Query 3.5 example, from the point of registry database view are:

(From each id in the result set of this statement),

```
"SELECT ID FROM REGISTRYENTRY WHERE ID LIKE '%Turkey_1%'(1)
```

**ebXML Registry Query 3.5** *ReturnRegistryEntry query example.*

```
<ReturnRegistryEntry>
  <RegistryEntryQuery>
    <RegistryEntryFilter>      <Clause>
      <SimpleClause leftArgument='id'>
        <StringClause StringPredicate='contains'> Turkey </StringClaues>
      </SimpleClause>
    </Clause>      </RegistryEntryFilter>
  </RegistryEntryQuery>
  <WithClassifications/>
  <WithSourceAssociations/>
    <AssociationFilter>      <Clause>
      <SimpleClause leftArgument='associationType'>
        <StringClause StringPredicate='contains'> Turkey </StringClaues>
      </SimpleClause>
    </Clause>      </AssociationFilter>
  </WithSourceAssociations/>
</ReturnRegistryEntry>
```

(In order to retrieve all classification objects that the given idth object act as sourceobject)

```
"SELECT ID FROM CLASSIFICATION WHERE SOURCEOBJECT = id"(2)
```

(In order to retrieve all Association objects that the given idth object act as sourceobject with HasMember association type)

```
" SELECT ID FROM ASSOCIATION
```

```
WHERE ASSOCIATIONTYPE = 402 and SOURCEOBJECT = id"(3)
```

From the request in the ebXML Registry Query 3.5 example, the SQL statements (1),(2),(3) (given above) are formed out. A RegistryEntry internal object is created for temporary access to registry database. This temporary object is used to execute the SQL statement (1) and retrieve the object ids and types from the registry database.

Then, for each object id and object type in the result set,

- Each object's member variables are retrieved from registry database.
- The SQL statement (2) (given above) is dynamically used to find out the object ids' of the classification objects that are classified by this object. For each object id in the result of this query, a Classification object is created and it's contents are retrieved from database.
- The SQL statement (3) (given above) is dynamically used to find out the object ids' of the association objects that are associated with this object with an HasMember association type. For each object id in the result of this query, an Association object is created and contents are retrieved from database.
- The Result string is updated for each retrieved object.

The ebXML Registry Query Result 3.4 is formed from the ebXML Registry Query 3.5.

### ebXML Registry Query Result 3.4 *ReturnRegistryEntry* example result.

```
<!--ebXML REGISTRY/REPOSITORY MECHANISM Developed by Levent CARKACIOGLU-->
SELECT ID REGISTRYENTRY WHERE(ID LIKE'%Turkey%'))-->
<RegistryResponse status='success'>
  <RegistryEntryQueryResult> <RegistryEntryMetaData>
    <RegistryEntry id='urn:uuid:Turkey_1' name='Turkey'
      description='CPP ClassificationExample' majorVersion='0'
      minorVersion='1' status='0'
    <Classification id='urn:uuid:Classification_1' name='clsf1'
      description='classificationEx' majorVersion='0' minorVersion='1'
      status='0' userVersion='1.a' classifiedObject='urn:uuid:Ccpp1'
      classificationNode='urn:uuid:Istanbul_1'/>
    <Association id='urn:uuid:Association_1' name='assoc1'
      description='assoc.Ex' majorVersion='2' minorVersion='1'
      status='4' userVersion='1.2.a' sourceRole='sRole1'
      targetRole='tRole1' associationType='402'
      bidirectional='false' sourceObject='urn:uuid:Turkey_1'
      targetObject='urn:uuid:Test_1' />
  </RegistryEntry>
</RegistryEntryMetaData> </RegistryEntryQueryResult>
</RegistryResponse>
```

#### 3.4.5 ReturnRepositoryItem

ReturnRepositoryItem request type provides detailed query capabilities in the registry. ReturnRepositoryItem request support is similar to ReturnRegistryEntry request, but if the object is an Extrinsic, Package or ExternalLink object, a reference to it's DTD document is also retrieved.

From the ReturnRepositoryItem request, a SQL query is formed out. Then a RegistryEntry internal object is created for temporal usage. By the use of this temporary object, the object id's and object type's of the query result are retrieved. For each object id in the result set, according to object type, the related registry internal object is created. And its member variables are retrieved from the registry database. Further more, if the object is an Extrinsic, Package or ExternalLink object then, the related Extrinsic, Package or ExternalLink object's are created and contents are retrieved from the registry database.

The retrieval of objects of the prepared SQL query and result string preparation takes place in the RepositoryItemResult internal object of the Registry Server.

An example for ReturnRepositoryItem request and its execution is as fol-

lows:

**ebXML Registry Query 3.6** *ReturnRepositoryItem query example.*

```
<ReturnRepositoryItem>
  <RegistryEntryQuery>    <RegistryEntryFilter>
    <Clause>              <SimpleClause leftArgument='objectType'>
      <StringClause StringPredicate='equal'>
        ExtrinsicObject
      </StringClause>
    </SimpleClause>      </Clause>
  </RegistryEntryFilter> </RegistryEntryQuery>
</ReturnRepositoryItem>
```

The ebXML Registry Query 3.6 example is to query the ExtrinsicObject type objects.

The SQL statements for the ebXML Registry Query 3.6 example, from the point of registry database view is:

```
"SELECT ID FROM REGISTRYENTRY WHERE OBJECTTYPE = 128 "
```

From the request in the ebXML Registry Query 3.6 example, the SQL statement (given above) is formed. A RegistryEntry internal object is created for temporary access to registry database. This temporary object is used to execute the SQL statement and retrieve the object ids and types from the registry database. Since the object is an Extrinsic object, an Extrinsic object is created and contents are retrieved from the registry database.

The ebXML Registry Query Result 3.5 is formed from the ebXML Registry Query 3.6.

**ebXML Registry Query Result 3.5** *ReturnRepositoryItem example result.*

```
<!--ebXML REGISTRY/REPOSITORY MECHANISM Developed by Levent CARKACIOGLU-->
SELECT ID REGISTRYENTRY WHERE ( OBJECTTYPE = 128 ))!--
<RegistryResponse status='success'>
  <ReturnRepositoryItemResult>
    <RepositoryItem id='urn:uuid:test.1 name='Levent'
      description='Levent Extrinsic Test' majorVersion='3'
      minorVersion='0' status='1'>
      <ExtrinsicObject contentURI='http://www.ceng.metu.edu.tr/ leventc
        mimetype='HTTP' objectType='CPA' opaque='true'>
      </ExtrinsicObject>
    </RepositoryItem>
  </ReturnRepositoryItemResult>
</RegistryResponse>
```

### 3.4.6 CPPQuery

For the automation of CPP compatible search, meta-data of CPP documents are decided to be stored in registry internal structures. In order to query these newly added registry objects, an extension also has to be done to the Registry Service request types. A new query type for querying CPP meta-data called CPPQuery is added.

There are four optional keywords for accessing further detail of a CPP object. WithCollaborationRole, WithDeliveryChannel, WithTransport and WithPackage options. WithCollaborationRole option is used to get the CollaborationRole details of the result CPP. WithDeliveryChannel option is used to get the DeliveryChannel details, withTransport option is used to get the Transport details of the CPP. And finally, WithPackage option is used to get the packaging details of the CPP. Filtration can also be applied on these additional options to get rid of irrelevant data.

From the CPPQuery request, a SQL query is formed to get the object id's of the requested objects. Then a RegistryEntry internal object is created for temporal usage. By using this temporary object, the object ids' in the result set of the formed SQL query are retrieved. Then, for each object id, a CPP object is created and it's member variables are retrieved from the registry database. At this point, if any of the withCollaborationRole, withDeliveryChannel, withTransport or WithPackage options exists, the related CollaborationRole, DeliveryChannel, Transport or Package objects are created and contents are retrieved from registry database.

The retrieval of objects of the prepared SQL query result set and result string preparation takes place in the CPPQueryResult internal object of the Registry Server.

An example for CPPQuery request and it's execution is as follows:

The ebXML Registry Query 3.7 example, is to query the CPPs whose role name attribute contains 'buy' word and also retrieve all the CollaborationRole, Packaging and DeliveryChannel objects (CPPs whose role name attribute con-

tain the 'buy' word) and the transport objects having 'HTTP' Sender Protocol attribute.

### ebXML Registry Query 3.7 *CPPQuery query example.*

```
<CPPQuery>
  <RegistryEntryQuery>    <CPPHasId>    <PartyInfoHasCPP>
    <PartyInfoIdOfCollaborationRole> <CollaborationRoleHasRole>
      <RoleFilter> <Clause> <SimpleClause leftArgument='name'>
        <StringClause StringPredicate='contains'> buy </StringClause>
      </SimpleClause> </Clause> </RoleFilter>
    </CollaborationRoleHasRole> <PartyInfoIdOfCollaborationRole>
  </PartyInfoHasCPP>    </CPPHasId>    </RegistryEntryQuery>
</WithCollaborationRole/>
<WithDeliveryChannel/>
<WithTransport>
  <TransportFilter>
    <Clause> <SimpleClause leftArgument='SenderProtocol'>
      <StringClause StringPredicate='equals'> HTTP </StringClause>
    </SimpleClause> </Clause>
  </SimpleClause>
</WithTransport>
<WithPackaging/>
</CPPQuery>
```

The SQL statements for the ebXML Registry Query 3.7 example, from the point of registry database view are:

"SELECT ID FROM CPP WHERE CPPId In (

SELECT CPPID FROM PARTYINFO WHERE Id In (

SELECT PARTYINFOID FROM COLLABORATIONROLE WHERE Id In(

SELECT ID FROM COLLABORATIONROLE WHERE RoleId In (

SELECT ID FROM ROLE WHERE ( name LIKE '%buy%' ) )))) "(1)

(Get all CPP object ids that contain 'buy' word in their role's name attributes)

(From each id the result set of the SQL query in (1), Retrieve all the CollaborationRole objects of the given CPP)

"SELECT ID FROM COLLABORATIONROLE WHERE PARTYINFOID IN (

SELECT ID FROM PARTYINFO WHERE CPPID= id )" (2)

(Retrieve all the DeliveryChannel objects of the given CPP)

"SELECT ID FROM DELIVERYCHANNEL WHERE CHANNELID IN (

```

SELECT CHANNELID FROM SERVICEBINDING WHERE
COLLABORATIONROLEID IN ( SELECT ID FROM COLLABORATIONROLE
WHERE PARTYINFOID IN ( SELECT PARTYINFO
WHERE CPPID = id "))) (3)

```

(Retrieve all the Package objects of the given CPP)

```

" SELECT ID FROM PACKAGE WHERE PACKAGEID IN (
SELECT PACKAGEID FROM CPP WHERE ID= id )" (4)

```

(Retrieve all the Transport objects of the given CPP, with having 'HTTP' in Senderprotocol attributes)

```

" SELECT ID FROM TRANSPORT WHERE SENDERPROTOCOL = 'HTTP'
AND ID IN ( SELECT ID FROM TRANSPORT WHERE ID (
SELECT TRANSPORTID FROM DELIVERYCHANNEL WHERE
CHANNELID IN ( SELECT CHANNELID FROM SERVICEBINDING
WHERE COLLABORATIONROLEID IN (
SELECT ID FROM COLLABORATIONROLE WHERE
PARTYINFOID IN ( SELECT ID FROM PARTYINFO WHERE
CPPID IN ( SELECT CPPID FROM CPP WHERE ID = id " )
))))(5)

```

From the request in the ebXML Registry Query 3.7 example, SQL statements (1),(2),(3),(4),(5) (given above) are formed out. A RegistryEntry internal object is created for temporary access to registry database. Then a member function of the temporary object is called to execute the SQL statement (1) and object ids' in the result set are retrieved from the registry database. Then, a CPP object is created.

For each CPP object created:

- The CPP object's member variables are retrieved from registry database.
- The SQL statement (2) given above is dynamically used to find out the object ids' of the CollaborationRole objects that the CPP is related to.

Then, a CollaborationRole object is created for each object in the result of query and contents are retrieved from database.

- The SQL statement (3) given above is dynamically used to find out the object ids' of the ServiceBinding objects that the CPP is related to. Then, a ServiceBinding object is created for each object in the result of query and contents are retrieved from database.
- The SQL statement (4) given above is dynamically used to find out the object ids' of the Transport objects having senderProtocol attributes as 'HTTP', that CPP is related with. Then, a Transport object is created for each object in the result of query and contents are retrieved from database.
- The SQL statement (4) given above is dynamically used find out the object ids' of the Packaging objects in the registry that the CPP is related to. Then, a TransportSecurity object is created for each object in the result of query and contents are retrieved from database.
- The Result string is updated by the contents of the each object created from the SQL statements.

The ebXML Registry Query Result 3.6 is formed from the ebXML Registry Query 3.7.

### **3.4.7 ReturnCompatibleCPP**

The automation of CPP compatible search is embedded in to registry internal structure. ReturnCompatibleCPP query type provides the search for a compatible CPP for the given idth CPP in the Registry database. This is another extension made to Registry Service query types.

From ReturnCompatibleCPP request, a SQL statement querying the object id's of the requested objects is formed out. A CPP object is created and the internal method for finding compatible CPP in the registry is called. According to the result set of the compatible CPP's, a result string is formed.

### ebXML Registry Query Result 3.6 CPPQuery example result.

```
<!--ebXML REGISTRY/REPOSITORY MECHANISM Developed by Levent CARKACIOGLU-->
SELECT PARTYINFOID FROM COLLABORATIONROLE WHERE Id In ( SELECT ID
FROM COLLABORATIONROLE WHERE RoleId In ( SELECT ID FROM ROLE WHERE
( name LIKE '%buy%' ) ))))!-->
<RegistryResponse status='success'>
  <ReturnCPPQueryResult>  <CPPMetaData>
    <CPP id='urn:uuid:CPP_1' name='CPPtest1'
      description='CPPtest1' majorVersion='0'
      minorVersion='1' status='0'>
      <CollaborationRole id='N00'>
        <ProcessSpec version='1.0' name='buySell' xlinktype='simple'
          xlinkhref='http://www.ebxml.org/prs/buySell.xml'/>
        <Role name='buyer' xlinktype='simple'
          xlinkhref='http://www.ebxml.org/prs/buySell.xml#buyer'/>
        <ServiceBinding channelId='N04' packageId='N0402'/>
      </CollaborationRole>
      <DeliveryChannel channelId='N04' transportId='N05'
        docexchangeId='N06' />
      <Transport transportId='N04' Send.Protocolversion='1.1'
        Send.Protocol='HTTP' Recv.Protocolversion='1.1'/>
        Recv.Protocol='HTTP'
      <TransportSecurity protocolVer='3.0' Protocol='SSL' certId='N03'/>
      </Transport>
      <Packaging id='N0402'>
        <SimplePart id='N40' mimetype='text'>
        </SimplePart>
        <CompositeList>
          <Composite id='N42' mimetype='multi./rel.'>
            <Constituent idref='N40'>
            <Constituent idref='N41'>
          </Composite>
        </CompositeList>
      </Packaging>
    </RepositoryItem>
  </CPPMetaData>  </ReturnCPPQueryResult>
</RegistryResponse>
```

The retrieval of objects of the prepared SQL query result set and result string preparation takes place in the CompatibleCPP internal object of the Registry Server.

An example for ReturnCompatibleCPP request and it's execution is as follows:

The ebXML Registry Query 3.8 example, is to query the compatible CPP's in the registry whose role name attribute contains 'buy' word.

The SQL statements for the ebXML Registry Query 3.7 example, from the point of registry database view are:

**ebXML Registry Query 3.8 *ReturnCompatibleCPP* query example.**

```
<ReturnCompatibleCPP>
  <RegistryEntryQuery>    <CPPHasId>    <PartyInfoHasCPP>
    <PartyInfoIdOfCollaborationRole> <CollaborationRoleHasRole>
      <RoleFilter> <Clause>    <SimpleClause leftArgument='name'>
        <StringClause StringPredicate='contains'> consume </StringClause>
      </SimpleClause> </Clause> </RoleFilter>
    </CollaborationRoleHasRole> <PartyInfoIdOfCollaborationRole>
  </PartyInfoHasCPP>    </CPPHasId>    </RegistryEntryQuery>
</ReturnCompatibleCPP>
```

(Get all CPP object ids that contain 'consume' word in their role name attributes)

```
"SELECT ID FROM CPP WHERE CPPId In (
  SELECT CPPID FROM PARTYINFO WHERE Id In (
    SELECT PARTYINFOID FROM COLLABORATIONROLE WHERE Id In (
      SELECT ID FROM COLLABORATIONROLE WHERE RoleId In (
        SELECT ID FROM ROLE WHERE( name LIKE '%consume%'))))
```

From the request in the ebXML Registry Query 3.8 example, the SQL statement above is formed out. A RegistryEntry internal object is created for temporary access to registry database. Then a CPP object is created.

For each CPP object created

- The CPP object's member method for finding compatible CPPs in the registry with the given CPP id is called.
- The result string is formed out for each result.

**ebXML Registry Query Result 3.7 *ReturnCompatibleCPP* example result.**

```
<!--ebXML REGISTRY/REPOSITORY MECHANISM Developed by Levent CARKACIOGLU-->
SELECT ID FROM CPP WHERE CPPId In ( SELECT CPPID FROM PARTYINFO
WHERE Id In( SELECT PARTYINFOID FROM COLLABORATIONROLE WHERE Id In(
( SELECT ID FROM COLLABORATIONROLE WHERE RoleId In (SELECT ID
FROM ROLE WHERE ( name LIKE '%buy%' )))) )!-_
<RegistryResponse status='success'>
  <CPPMatchingQueryResult>
    <ComCPPs ID='urn:uuid:CPP4' matchCPPId='urn:uuid:CPP3'>
  </CPPMatchingQueryResult>
</RegistryResponse>
```

The ebXML Registry Query Result 3.6 is formed from the ebXML Registry Query 3.7.

#### **3.4.8 DAMLOILQueryRequest**

In this thesis, Service Taxonomy ontology implemented by METU-SRDC, given in APPENDIX D is supported. Service Taxonomy example ontology structure and instance classes with the corresponding relational database tables are imported to the ebXML registry internal structure.

In order to support DAML+OIL ontology structures, the DAML+OIL ontology class relations (i.e., subClass, subProperty and Restriction relations) and class instances (instances of Service Taxonomy classes) are added to registry internal structures. In order to query these newly added registry objects, a new query type for DAML+OIL ontology classes and instances, called DAMLOIL-QueryRequest is added to the registry query types.

The relation among ontology classes, instances of any class in the ontology and classified objects by a given ontology instance can be queried under DAMLOILQueryRequest query type.

For the sake of simplicity the three relation types between ontology classes: SubClass, subProperty and Restriction are supported in this thesis. A DAMLOIL-QueryRequest query about one of these relationships is converted to a SQL query which uses one of the subClass, subProperty and Restriction tables in the registry database. The SQL query is executed and the result set consisting of related data is retrieved.

If the query is on Restriction relations between ontology classes, then, as the DAML+OIL semantic states, all the parent classes of the requested classes' restrictions are taken in consideration too.

The SQL query construction from DAMLOILQueryRequest type queries is done in the RegQuery internal object of the Registry Server. The retrieval of objects of the prepared SQL query and result string preparation takes place in the DAMLOILQuery object of the Registry Server.

Some examples of DAMLOILQueryRequest that are related with ontology class relations are shown below. These examples use the Service Taxonomy ontology given in APPENDIX D.

**ebXML Registry Query 3.9** *DAMLOILQueryRequest query example-1.*

```
<DAMLOILQueryRequest>
  <DAMLOIL_SubClass_Get_Id>  <DAMLOIL_Has_SubclassId>
    <DAMLOIL_Restriction_Has_ClassId>  <DAMLOIL_Restriction_Filter>
      <Clause>    <SimpleClause leftArgument='property'>
        <StringClause StringPredicate='equal'> Added_Value </StringClause>
      </SimpleClause>  </Clause>
    </DAMLOIL_Restriction_Filter>  </DAMLOIL_Restriction_Has_ClassId>
  </DAMLOIL_Has_SubclassId>  </DAMLOIL_SubClass_Get_Id>
</DAMLOILQueryRequest>
```

The ebXML Registry Query 3.9 example, requests the DAML+OIL ontology classes parent classes that contain the restriction having the property 'Added\_Value'. In other words, it searches for the added value of parent ontology classes in the Service Taxonomy. It can be used to get the added value related ontology classes of all parent classes in the Service Taxonomy.

The SQL statement for the ebXML Registry Query 3.9 example, from the point of registry database view is:

```
" SELECT DAMLID FROM DAMLOIL.SUBCLASS WHERE SUBCLASSID IN(
      SELECT CLASSID FROM DAMLOIL.RESTRICTION WHERE
        Property = 'Added_Value' )"

```

From the query in the ebXML Registry Query 3.9 example, the above SQL statement is formed out. A RegistryEntry internal object is created for temporary access to registry database. A member function of the temporary object is called with this SQL statement. Then a DAMLOIL.SUBCLASS object is created and for each object id in the result set, the contents are retrieved from the registry database.

The ebXML Registry Query Result 3.8 is formed from the ebXML Registry Query 3.9.

**ebXML Registry Query Result 3.8 DAMLOILQueryRequest example-1 result.**

```
<!--ebXML REGISTRY/REPOSITORY MECHANISM Developed by Levent CARKACIOGLU-->
SELECT DAMLID FROM DAMLOIL.SUBCLASS WHERE SUBCLASSID IN(SELECT
CLASSID FROM DAMLOIL.RESTRICTION WHERE Property = 'Added_Value'
( SELECT ID FROM COLLABORATIONROLE WHERE RoleId In (SELECT ID
FROM ROLE WHERE ( name LIKE '%buy%' ) ) ) ) )!->
<RegistryResponse status='success'>
  <DAML_OIL_SUBCLASS id='urn:uuid:daml.oil.class1' name='Service'
    description='DAMLoil service Ex1' majorVersion='0' minorVersion='0'
    userVersion='1.1.a',damlId='Service' subClassId='Car_Rentral_Service'/>
</RegistryResponse>
```

**ebXML Registry Query 3.10 DAMLOILQueryRequest query example-2.**

```
<DAMLOIL_Get_All_Restrictions classId= 'Hotel_Reservation_Service'
  property='AddOn_To' depth='-1'>
</DAMLOIL_Get_All_Restrictions>
```

The ebXML Registry Query 3.10 example, requests the DAML+OIL built ontology classes having a restriction property of "AddOn\_To" and consider the parent classes of "Hotel\_Reservation\_Service" class if they satisfy the "AddOn\_To" property. In other words, the query is to search for the Add On services of Hotel Reservation Service in the Service Taxonomy (parent classes of Hotel Reservation Service is also considered). This query can be used to get the services that can be needed after a hotel reservation.

A RegistryEntry object is created for temporal usage. The damlid's of all the parent classes of "Hotel\_Reservation\_Service" class are queried in the registry database. For each id in the result set, the Restriction property "AddOn\_To" is checked. A DAMLOIL.RESTRICION object is created and for each id that satisfies the restriction, the contents of the object is retrieved from database and the result string is formed.

The ebXML Registry Query Result 3.9 is formed from the ebXML Registry Query 3.10.

**ebXML Registry Query Result 3.9 DAMLOILQueryRequest example-2 result.**

```
<!--ebXML REGISTRY/REPOSITORY MECHANISM Developed by Levent CARKACIOGLU-->
<RegistryResponse status='success'>
  <DAML_OIL_RESTRICTION id='urn:uuid:daml.oil.Restriction1' name='Rest.AddOn'
    description='DamlOil RestrictionEx1' majorVersion='0' minorVersion='0'
    userVersion='1.1.a' property='AddOn_To' ClassId='Car_Rentral_Service'/>
</RegistryResponse>
```

The ebXML Registry Query 3.11 example, requests the DAML+OIL ontology Hotel\_Reservation\_Service instances, having starRating of '4'.

**ebXML Registry Query 3.11** *DAMLOILQueryRequest query example-3.*

```
<DAMLOILQueryRequest>
  <DAML_INSTANCE_Hotel_Reservation_Filter>
    <Clause>
      <SimpleClause leftArgument='starRating'>
        <StringClause StringPredicate='equal'> 4 </StringClause>
      </SimpleClause>
    </Clause>
  </DAML_INSTANCE_Hotel_Reservation_Filter>
</DAMLOILQueryRequest>
```

The SQL statement for the ebXML Registry Query 3.11 example, from the point of registry database view is:

" SELECT ID FROM HOTEL\_RESERVATION WHERE STARRATING ='4' "

From the ebXML Registry Query 3.11 request, the SQL statement, above is formed out. A RegistryEntry internal object is created for temporary access to registry database and object ids are retrieved from the registry database. Then, a DAMLOIL\_INSTANCE\_HOTEL\_RESERVATION object is created (the related ontology instance class) and for each object id in the result set, the contents are retrieved from registry database.

The ebXML Registry Query Result 3.10 is formed from the ebXML Registry Query 3.11.

**ebXML Registry Query Result 3.10** *DAMLOILQueryRequest example-3 result.*

```
<!--ebXML REGISTRY/REPOSITORY MECHANISM Developed by Levent CARKACIOGLU-->
<RegistryResponse status='success'>
  <DAML_OIL_INSTANCE_HOTEL_RESERVATION id='urn:uuid:Hilton1'
    name='Hilton Ankara'description='Hotel Resrv.Serv.' majorVersion='0'
    minorVersion='0' userVersion='1.1.a' creditcardNo='4245576544_To'
    creditCardtype='VISA' starRating='4'/>
</RegistryResponse>
```

### 3.4.9 GetRootClassificationNodeRequest

ebXML Registry supports classification trees. GetRootClassificationNode request type is used to query the root nodes of the classification trees.

From the request, a SQL statement querying the object ids of the request is formed out. A temporary RegistryEntry internal object is created. By using

this temporary object, the object id's of the formed SQL query are retrieved. Then, for each object id in the result set, a ClassificationNode object is created. And the member variables are retrieved from the registry database.

The retrieval of objects of the prepared SQL query and result string preparation takes place in the ClassificationNodesRequest object of the Registry Server.

An example for GetRootClassificationNode request and it's execution is as follows:

**ebXML Registry Query 3.12** *GetRootClassificationNodeRequest query example.*

```
<GetRootClassificationNodesRequest namePattern='Geo'>
</GetRootClassificationNodesRequest>
```

In the ebXML Registry Query 3.12 example, the root ClassificationNodes that contain the pattern 'Geo' in their name attribute are queried.

The SQL statements for the ebXML Registry Query 3.12 example, from the point of registry database view are:

```
" SELECT ID FROM CLASSIFICATIONNODE WHERE NAME LIKE '%Geo%' "
```

From the request in the ebXML Registry Query 3.12 example, the above SQL statement is formed out. A RegistryEntry internal object is created for temporary access to registry database. A member function of the temporary object is called with this SQL statement. Then, a ClassificationNode object is created and for each object id in the result set, the contents are retrieved from registry database, with updating the result string.

The ebXML Registry Query Result 3.11 is formed from the ebXML Registry Query 3.12.

**ebXML Registry Query Result 3.11** *GetRootClassificationNodeRequest example result.*

```
<!--ebXML REGISTRY/REPOSITORY MECHANISM Developed by Levent CARKACIOGLU-->
<RegistryResponse status='success'>
  <GetRootClassificationNodesResponse>
    <ClassificationNode id='urn:uuid:Geograph11' name='Geography'
      description='ClassificationEx11' majorVersion='0' minorVersion='0'
      userVersion='1.1.a' parent=' ' code=' ' />
  </GetRootClassificationNodesResponse>
</RegistryResponse>
```

### 3.4.10 GetClassificationTreeRequest

In order to operate on classification trees in the registry, GetClassificationTreeRequest request type is also used. A ClassificationNode internal object is created. Until the specified dept limit is reached or there does not exists any more sibling nodes, the contents of it and it's siblings are retrieved.

The retrieval of objects of the prepared SQL query and result string preparation takes place in the ClassificationNodesRequest object of the Registry Server.

An example for GetClassificationTreeRequest request and it's execution is as follows:

**ebXML Registry Query 3.13** *GetClassificationTreeRequest query example.*

```
<GetClassificationTreeRequest parent='urn:uuid:Geograph_1' depth='2'>
</GetClassificationTreeRequest>
```

The ebXML Registry Query 3.13 example queries the ClassificationNodes that are under the node having id 'urn:uuid:Geograph\_1' with depth 2.

A ClassificationNode internal object is created. The contents of the object are retrieved from registry database. Until the tree depth is reached or there is no more siblings, the ClassificationNode objects having this idth node as parent are retrieved from the registry database, using a member function of the ClassificationNode object. Then, for each object id in the result set, the object's contents are retrieved and result string is updated.

The ebXML Registry Query Result 3.12 is formed from the ebXML Registry Query 3.13.

**ebXML Registry Query Result 3.12** *GetClassificationTreeRequest example result.*

```
<!--ebXML REGISTRY/REPOSITORY MECHANISM Developed by Levent CARKACIOGLU-->
<RegistryResponse status='success'>
  <GetClassificationTreeResponse>
    <ClassificationNode id='urn:uuid:Asia_1' name='Asia'
      description='ClassificationEx_Asia1' majorVersion='0' minorVersion='0'
      userVersion='1.1.a' parent='urn:uuid:Geograph_1' code='"/>
    <ClassificationNode id='urn:uuid:Europe_1' name='Europe'
      description='ClassificationEx_Europe1' majorVersion='0' minorVersion='0'
      userVersion='1.1.a' parent='urn:uuid:Geograph_1' code='"/>
  </GetClassificationTreeResponse>
</RegistryResponse>
```

### 3.4.11 GetClassifiedObjectsRequest

In order to get the objects that are classified by a node in a classification tree, the GetClassifiedObjectsRequest query is used. The classified object ids and object types of given idth ClassificationNode object are retrieved. And for each id in the result set, the related registry internal object is created and contents are retrieved from the registry.

The retrieval of objects of the prepared SQL query and result string preparation takes place in the ClassificationNodesRequest object of the Registry Server.

An example for GetClassifiedObjectsRequest request and it's execution is as follows:

#### ebXML Registry Query 3.14 *GetClassifiedObjectsRequest query example.*

```
<GetClassifiedObjectsRequest>
  <ObjectRefList> <ObjectRef id='urn:uuid:Asia.1'> </ObjectRefList>
</GetClassifiedObjectsRequest>
```

The ebXML Registry Query 3.14 example queries the objects in the registry that are classified by the ClassificationNode having id, 'urn:uuid:Asia.1'.

A ClassificationNode is created with object id, 'urn:uuid:Asia.1'. The member function of ClassificationNode object is called to get the classified objects by the object. The result for this method is a set of object ids. For each object id in the result set, the related registry internal object is created and contents are retrieved from the registry database. The result string is updated for each retrieved object.

The ebXML Registry Query Result 3.13 is formed from the ebXML Registry Query 3.14.

#### ebXML Registry Query Result 3.13 *GetClassifiedObjectsRequest example result.*

```
<!--ebXML REGISTRY/REPOSITORY MECHANISM Developed by Levent CARKACIOGLU-->
<RegistryResponse status='success'>
  <GetClassifiedObjectsResponse>
    <CPP id='urn:uuid:CPP.1' name='CPPtest1' description='CPPtest1'
      majorVersion='0' minorVersion='1' status='0' packageId='N402' />
    <CPP id='urn:uuid:CPP.2' name='CPPtest1' description='CPPtest2'
      majorVersion='0' minorVersion='1' status='0' packageId='N403' />
  </GetClassifiedObjectsResponse>
</RegistryResponse>
```

### 3.4.12 SQLQueryRequest

SQLQueryRequest request type is an optional type for querying the registry database for advanced users, that are familiar with the registry database. The responsibility for the correctness of the query is left to the user. The SQL queries only have to query the object's id columns in the registry database. The SQL query is taken and then directly sent to the registry database. According to the result set (the result set should be the set of object ids), the related registry internal object is created and contents are retrieved from the registry database. The result string is updated for each object.

The retrieval of objects of the prepared SQL query result and result string preparation takes place in the SQLQuery internal object of the Registry Server.

An example for SQLQuery request and it's execution is as follows:

**ebXML Registry Query 3.15** *SQLQuery query example.*

```
<SQLQuery>
    SELECT ID FROM CPP WHERE ID = 'urn:uuid:CPP_Id.1'
</SQLQuery>
```

A temporary RegistryEntry object is created and the SQL query is executed. Then, for each object id in the SQL query result set, the related registry internal object is created and contents are retrieved. The result string is updated for each object.

The ebXML Registry Query Result 3.13 is formed from the ebXML Registry Query 3.14.

**ebXML Registry Query Result 3.14** *SQLQuery query result.*

```
<!--ebXML REGISTRY/REPOSITORY MECHANISM Developed by Levent CARKACIOGLU-->
<RegistryResponse status='success'>
  <SQLQuery>
    <CPP id='urn:uuid:CPP.1' name='CPPtest1' description='CPPtest1'
      majorVersion='0' minorVersion='1' status='0' packageId='N402' />
  </SQLQuery>
</RegistryResponse>
```

### 3.4.13 ErrorCase

In the case of an error in the registry database, in any of the query types explained above, the registry returns a failure result string and states the error

and codes.

An example for the error case is as follows:

**ebXML Registry Query Result 3.15 Error case.**

```
<!--ebXML REGISTRY/REPOSITORY MECHANISM Developed by Levent CARKACIOGLU-->
<!--Database Error-->
<RegistryResponse status='failure'>
  <RegistryError highestSeverity='error'>
    <RegistryErrorList>
      <RegistryError errorcode='-100' codeContext='SQLException has occured'>
    </RegistryErrorList>
  </RegistryError>
</RegistryResponse>
```



## CHAPTER 4

# IMPLEMENTATION DETAILS

### 4.1 Introduction

This chapter presents the implementation details of the ebXML Registry/ Repository Service. The software tools and packages used in development are briefly discussed in the Section 4.2. Section 4.3 describes flow of execution in the ebXML Registry Service implemented. Section 4.4 provides the implementation highlights.

### 4.2 Development Tools

Java programming language with Java Development Kit (JDK) Version 1.3 is used for ebXML Registry/Repository Service development.

In development, the following tools and packages are used:

- **Jakarta-tomcat-3.2.2:** Jakarta-Tomcat is a servlet container and a JAVA server pages implementation. It is used in ebXML Registry Server application, where the Registry Server application runs in.
- **Soap.jar:** Since, the ebXML Messaging uses SOAP messaging with attachments, this package is to manage the attachments of ebXML Messages.
- **Xerces-1.4.0/xerces.jar:** This is an XML parser package.

Initially, Oracle 8.i was selected as the Registry database management system. But the JDK compiler get in to conflict with the Oracle access driver in the adaptation of Messaging Service to the system due to versioning problems. In order to solve this problem, instead of Oracle, the Microsoft SQL Server is used as the Registry database management system.

Since, ebXML Messaging costs is not acceptable for the communication between registry internal objects and the registry database due to high access demand, the database management system is accessed internally by the Java Database Connectivity (JDBC) bridge. The Java native driver for the Microsoft SQL Server is used which made the database access faster.

The ebXML Messaging Service developed by METU-SRDC is used in the implementation.

### **4.3 ebXML Registry Service Execution Flow**

Initially, the Tomcat server is started in order to execute the Registry Server and Registry Client applications. The Registry Server, runs on the Tomcat server listens the port 1919 for incoming ebXML messages from the Messaging Service, where as Registry Client runs on Tomcat server and listens the port 1881 for incoming messages from the Message Service.

Registry Client and Server applications have their own CPP documents, stating how they execute as any other business in the ebXML architecture. When Registry Server and Client applications start execution, at first they communicate with each other to make an agreement on the flow of work. For this purpose, a CPA document is performed between the server and client from their CPPs, by the Registry Client. From that point on, the Registry Client and Server applications communicate with each other in the way stated within this CPA, as given in APPENDIX F. After this initialization phase, both the Registry Client and Registry Server applications wait for incoming ebXML messages.

When an application sends a message to the ebXML Messaging Service, the Message Router application in the Messaging Service forwards the message to the

ebXML Registry Client. Since the thin client model is used in the development, the Registry Client just passes the incoming message to the Registry Server over Messaging Service. As the Registry Server application is invoked with the incoming message, it executes the registry request in the registry query document (attached within the ebXML message) by using its internal objects, as discussed in the Chapter 3.

The prepared ebXML Registry result message is passed back to the Registry Client via ebXML Messaging Service again. The Registry Client gets the incoming message and transmits the message back to the requester application using Messaging Service and waits for new incoming messages in a loop.

The workflow of the implemented ebXML Registry/Repository Service is simulated in Figure 4.1.

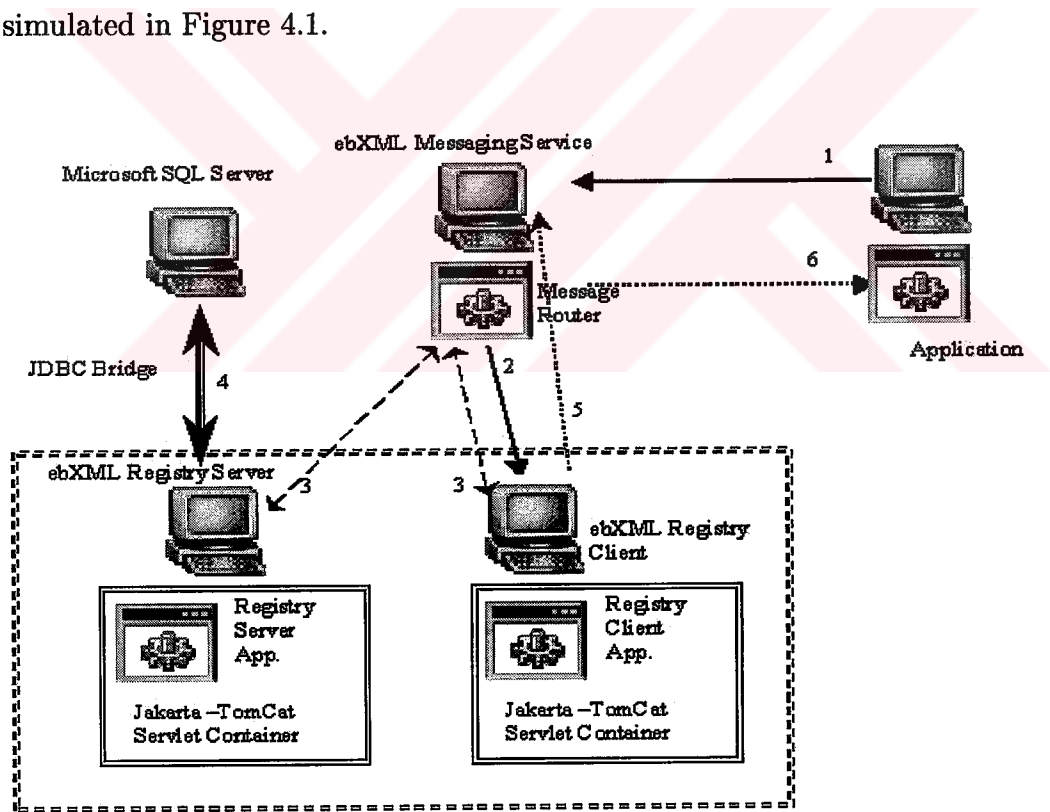


Figure 4.1: ebXML Registry/Repository Service implementation work flow.

Since the thin client model is used in the implementation, the Registry Client is an application for just passing registry queries to Registry Server, the Registry Client and Registry Server applications are decided to run on the same Jakarta-

Tomcat Servlet Container on the same computer. The Microsoft SQL Server and Messaging Service runs on different computers.

## 4.4 Implementaion Details

Since the thin client model is selected in Registry Client implementation, the actual registry implemantaion took place in the Registry Server application. The implementation of the Registry Server application is divided in to two parts:

- Registry internal structure (ebRIM) implementation
- Registry services (ebRS) implementation

### 4.4.1 ebRIM Implementation

In the ebRIM implementation, the objects that the registry has to support are designed and implemented. Since an object oriented design and implementation is realized, a java class for each internal object is created, therefore there are seventy internal objects (hence java classes) in the system.

The inheritance diagram shown in Figure 3.3 is conformed in class organizations. Each class has it's own uninherited member functions to access member variables. The inherited members management functions reside in the parent class. Each member variable has the two characteristic member functions:

- **setXXX(arg ...):** Set the member variable XXX with the value given in the argument
- **getXXX():** Get content of the member variable XXX.

Other than the functions for accessing member variables, each object in the ebRIM also has the following functions:

- **YYYObjectRetrieve():** Retrieve the member variables of the YYY object, by accessing the database, using it's object id as key.

- **YYYObjectInsertToORACLE:** Insert the contents of the YYY object to the related table in the database.
- **YYYObjectDeleteFromORACLE:** Delete the YYY object from the database using it's id as key.

Creating and destroying database connections is the most time consuming event. Therefore a database connection is created in the Registry Server application initialization and is passed in between objects need to access the database. All the database access is handled in the MyORACLE class.

#### 4.4.2 ebRS Implementation

The actual registry service is implemented in the ebRS part. Incoming registry requests are executed on the registry database and a result string is prepared to be returned. ebRS uses the internal objects implemented in ebRIM part of the system in order to operate on the registry.

In ebRIM, there is an object for each query type supported, i.e., there are nine classes for each query type in the system and two other classes namely RegQuery and CompoundParser. RegQuery class is to handle the execution flow in the ebRS, as shown in Figure 3.5. The CompoundParser class is used for all other classes to parse the compound clauses in the query documents.

In ebRS, the Xerces XML parser package is used for parsing XML queries.

## CHAPTER 5

### CONCLUSION

As the business move more on web, it becomes important to automate the business transactions and also important to include the small and medium enterprises easily, to increase the global trade. However, in order to build a global common schema, there has to be electronic business standards. ebXML is one of the electronic business standards. It is a framework to enable companies to have a standard method for conducting business over the Internet.

ebXML consists of four important functional concepts. The Trading Partner Information for dealing with the critical information such as recording of specific technical parameters; Business Process Schema for modeling business processes, Messaging Service to handle the communication in the infrastructure and the Registry/Repository Service is to act as a storage and discovery mechanism for business partners.

In this thesis, the Registry/Repository Service mechanism for the ebXML framework is designed and implemented. The ebXML Organization Committee submitted two technical specifications for ebXML Registry developers namely ebRIM and ebRS stating the minimal internal data and service query types to be supported. But, since the ebXML framework is newly evolving and there is not a working implementation for ebXML Registry/Repository Service, there are some missing points in the specifications. Within the scope of this thesis we

have outlined these points and designed solutions for them.

In the ebXML framework, the Collaboration Agreement Profile (CPA) document between two businesses has to be constructed manually by downloading the Collaboration Protocol Profiles (CPP) for two business parties. Therefore, a business party wishing to find all technically compatible partners in the registry has to download all the CPP documents. In this thesis, we partially extended the ebXML Registry to support the discovery of technically compatible partners automatically.

In order to search objects faster and easier, ebXML Registry enables building classification trees and associating the registry objects with those nodes in the tree. DAML+OIL, an RDF based semantic markup language is widely accepted for building richer ontologies. In this thesis, we demonstrated how to support DAML+OIL ontologies in ebXML Registry.

The specifications for the ebXML Repository are not clearly addressed by the ebXML Organization Committee. Therefore it is designed as a file server to be accessed by conventional means, i.e., http, ftp. Letting the management of the repository to business clients is not feasible and efficient. Therefore, another service for managing and versioning the repository to work in coordination with the registry can be designed and implemented as a future work.

The security issues in the registry specifications published by the organization committee is not very clear and complete. In this thesis, a security mechanism on registry objects is not implemented. But as a future work the security issue has to be taken in to consideration.

The compatible CPP search algorithm implemented as an extension to ebXML Registry considers the binary match cases. In order to decide two attributes to match, the equivalence of their values is considered (i.e., for business process elements of two CPPs to match, name attributes should be equal). However, looking for the attribute values may not be correct all the time. There can be different values having the same meaning. To handle such a case, a look-up table for each compared attribute values has to be implemented as a future work.

And the compatible search algorithm should be extended to check matching over these tables.

In this thesis, for the sake of simplicity, only a partial support for DAML+OIL built ontology support is implemented. SubClass, SubProperty and Restriction relations among ontology classes and the instances from Service Taxonomy ontology developed by METU-SRDC are supported. A dynamic way for building DAML+OIL ontologies in ebXML Registry necessities creating relational database tables and registry objects dynamically. In this implementation, the ontology has to be imported in the development phase and any other addition except from the relation definitions and instance creations have to be done by the registry development team. Enabling dynamic ontology class creation can be realized as a work. And a full support for ontology class relations can be another future work.

The Registry/Repository Service implemented in this thesis is to be used within application programs. Therefore a user interface for browsing the Registry/Repository is not considered. However such a user interface for the ebXML Registry/Repository Service is implemented in the Master Thesis of Pınar Pembecioğlu [27].

## REFERENCES

- [1] ebXML Registry Project Team. ebXML Registry Service Specification v.1.0, May 2001, <http://www.ebxml.org/specs/ebRS.pdf>.
- [2] A. Dogac. Business-to-business electronic commerce standards in electronic commerce technology, manuscript. (in pereperation).
- [3] OASIS ebXML Messaging services Technical Comittee. Message Service Specification Version 2.0 <http://www.oasis-opn.org/comittees/ebxml-msg/documents/ebMSV2.0.pdf>.
- [4] ebXML Registry Project Team. ebXML Registry Information Model v.1.0, May 2001, <http://www.ebxml.org/specs/ebRIM.pdf>.
- [5] A. Dogac, Ibrahim Cingil, Gokce Laleci, and Yildiray Kabak. Improving the functionality of uddi registries through web service semantics. August 23-24, 2002. 3rd VLDB Workshop on Technologies for E-Services (TES-02), Hong Kong, China.
- [6] ebXML. Frequently Asked Questions, <http://www.ebxml.org/faq.htm>.
- [7] ebXML. Electronic Business Extensible Markup Language, <http://www.ebxml.org>.
- [8] Stuart Campell. Conducting business via ebxml. In *Internationales Congress Centrum (ICC)*, Berlin, Germany 2001.
- [9] Dan Connoly, Frank van Harmelen, Ian Horrocks, Deborah L. Mc. Guinness, Peter F. Patel-Schneider, and Lean Andrea Stein. DAML+OIL (March 2001 Reference Description), W3C Note, 18 December 2001.
- [10] David Whiteley. *e-Commerce: Strategy, Technologies and Appications*. 2000. McGraw Hill, Maidenhead.
- [11] Bjorn-Andersenm O'Callaghan Eds, Krcmar. *EDI in Europe : How it works in practice*. 1995. Wiley.
- [12] UDDI. Universal Discription,Discovery and Integration, <http://www.uddi.org>.

- [13] Barbara McKee(IBM) and Dave Ehnebuske (IBM) and Dan Rogers (Microsoft). UDDI Version 2.0 API Specification, 18 June 2001.
- [14] UDDI.ORG. UDDI Technical White Paper, September 6, 2000.
- [15] North American Industrial Classification Schema (NAICS) codes, <http://www.naics.com>.
- [16] Simple Objects Access Protocol (SOAP), <http://www.w3.org/TR/SOAP>.
- [17] Peter Flynn. The XML Faq v2.1, 01.01.2002.
- [18] Extensible Markup Language (XML) 1.0, W3C Recommendation, <http://www.w3.org/XML>.
- [19] ebXML Requirements Team. ebXML Requirements Specifications v1.06, May 2001, <http://www.ebxml.org/specs/ebREQ.pdf>.
- [20] M. Rawlinks. Overview of the ebXML architectures, <http://www.rawlingsecconsulting.com>.
- [21] F. Casati and M-C. Shan. Models and languages for describing and discovering e-services. In *Proceedings of ACM SIGMOD Intl. Conf. On Management of Data, Santa Barbara*, 2001.
- [22] ebXML. Business Process Specification Schema Version 1.0, 16 February 2001, <http://www.ebxml.org/specs/ebBPSS.pdf>.
- [23] Anders Grangard (EDI France), Brian Einsenberg (Data Channel), and Duane Nickull (XML Global Technologies). ebXML Technical Architecture Specifications v.1.0.4, 16 February 2001, <http://www.ebxml.org/specs/ebXMLTA v.1.0.4.doc>.
- [24] John J. Barton (Hewlett Packard Labs), Satish Thatte, and Henrik Frystyk Nielsen (Microsoft). SOAP Messages with Attachments, 9 Oct. 2000, <http://www.w3.org/TR/2000/NOTE-SOAP-attachments-20001211>.
- [25] Dirk Reinshagen. XML Messaging Part 3, December 2001, <http://www.javaworld.com/javaworld/jw-09-2001/jw-0914-xmlmessage3.p.html>.
- [26] ebXML Trading Partners Team. Collaboration Protocol Profile and Agreement Specification Version 1.0, May 2001, <http://www.ebxml.org/specs/ebCCP.pdf>.
- [27] Pinar Pembecioğlu. Partner discovery and integration through service registries. Master's thesis, Middle East Technical University, 2002.
- [28] DCE 128 bit Universal Unique Identifier, <http://www.opengroups.org/onlinepubs/009629399/apdxa.htm>.

- [29] DCE 128 bit Universal Unique Identifier,  
<http://www.opengroups.org/publications/catalog/c706.htm>.



## APPENDIX A

### RELATIONAL SCHEMA FOR SQL QUERIES

```
CREATE TABLE ExtrinsicObject (  
  id          VARCHAR(64) PRIMARY KEY NOT NULL,  
  name        VARCHAR(128),  
  description  VARCHAR(256),  
  accessControlPolicy VARCHAR(64) NOT NULL,  
  majorVersion INT DEFAULT 0 NOT NULL,  
  minorVersion INT DEFAULT 1 NOT NULL,  
  status       INT DEFAULT 0 NOT NULL,  
  userVersion  VARCHAR(64),  
  stability    INT DEFAULT 0 NOT NULL,  
  expirationDate DATE,  
  contentURI   VARCHAR(128),  
  mimeType     VARCHAR(64),  
  objectType   INT DEFAULT 0 NOT NULL,  
  opaque       NUMBER(1) DEFAULT 0 NOT NULL  
);  
  
CREATE TABLE Package (  
  id          VARCHAR(64) PRIMARY KEY NOT NULL,  
  name        VARCHAR(128),  
  description  VARCHAR(256),  
  accessControlPolicy VARCHAR(64) NOT NULL,  
  majorVersion INT DEFAULT 0 NOT NULL,  
  minorVersion INT DEFAULT 1 NOT NULL,  
  status       INT DEFAULT 0 NOT NULL,  
  userVersion  VARCHAR(64),  
  stability    INT DEFAULT 0 NOT NULL,  
  expirationDate DATE,  
);  
  
CREATE TABLE ExternalLink (  
  id          VARCHAR(64) PRIMARY KEY NOT NULL,  
  name        VARCHAR(128),
```

```

description          VARCHAR(256),
accessControlPolicy  VARCHAR(64) NOT NULL,
majorVersion         INT DEFAULT 0 NOT NULL,
minorVersion         INT DEFAULT 1 NOT NULL,
status               INT DEFAULT 0 NOT NULL,
userVersion          VARCHAR(64),
stability            INT DEFAULT 0 NOT NULL,
expirationDate       DATE,
externalURI          VARCHAR(128) NOT NULL
);

CREATE TABLE ExternalIdentifier (
id                   VARCHAR(64) PRIMARY KEY NOT NULL,
name                VARCHAR(128),
description          VARCHAR(256),
accessControlPolicy VARCHAR(64) NOT NULL,
majorVersion        INT DEFAULT 0 NOT NULL,
minorVersion        INT DEFAULT 1 NOT NULL,
status              INT DEFAULT 0 NOT NULL,
userVersion          VARCHAR(64),
stability            INT DEFAULT 0 NOT NULL,
expirationDate       DATE,
value               VARCHAR(64) NOT NULL
);

CREATE TABLE SlotValue (
registryEntry        VARCHAR(64) NOT NULL,
name                 VARCHAR(128) NOT NULL,
value                VARCHAR(64) NOT NULL ,
PRIMARY KEY(registryEntry,name)
);

CREATE TABLE Association (
id                   VARCHAR(64) PRIMARY KEY NOT NULL,
name                VARCHAR(128),
description          VARCHAR(256),
accessControlPolicy VARCHAR(64) NOT NULL,
majorVersion        INT DEFAULT 0 NOT NULL,
minorVersion        INT DEFAULT 1 NOT NULL,
status              INT DEFAULT 0 NOT NULL,
userVersion          VARCHAR(64),
stability            INT DEFAULT 0 NOT NULL,
expirationDate       DATE,
associationType      INT NOT NULL,
bidirectional        NUMBER(1) DEFAULT 0 NOT NULL,
sourceObject         VARCHAR(64) NOT NULL,
sourceRole           VARCHAR(64),
label               VARCHAR(64),
targetObject         VARCHAR(64) NOT NULL,
targetRole           VARCHAR(64)
);

```

```

CREATE TABLE Classification (
  id                VARCHAR(64) PRIMARY KEY NOT NULL,
  name              VARCHAR(128),
  description       VARCHAR(256),
  accessControlPolicy VARCHAR(64) NOT NULL,
  majorVersion      INT DEFAULT 0 NOT NULL,
  minorVersion      INT DEFAULT 1 NOT NULL,
  status            INT DEFAULT 0 NOT NULL,
  userVersion       VARCHAR(64),
  stability          INT DEFAULT 0 NOT NULL,
  expirationDate    DATE,
  sourceObject      VARCHAR(64) NOT NULL,
  targetObject      VARCHAR(64) NOT NULL
);

CREATE TABLE ClassificationNode (
  id                VARCHAR(64) PRIMARY KEY NOT NULL,
  name              VARCHAR(128),
  description       VARCHAR(256),
  accessControlPolicy VARCHAR(64) NOT NULL,
  majorVersion      INT DEFAULT 0 NOT NULL,
  minorVersion      INT DEFAULT 1 NOT NULL,
  status            INT DEFAULT 0 NOT NULL,
  userVersion       VARCHAR(64),
  stability          INT DEFAULT 0 NOT NULL,
  expirationDate    DATE,
  parent            VARCHAR(64),
  path              VARCHAR(512) NOT NULL,
  code              VARCHAR(64)
);

CREATE TABLE AuditableEvent (
  id                VARCHAR(64) PRIMARY KEY NOT NULL,
  name              VARCHAR(128),
  description       VARCHAR(256),
  accessControlPolicy VARCHAR(64) NOT NULL,
  user_name         VARCHAR(64),
  eventType         INT DEFAULT 0 NOT NULL,
  registryEntry     VARCHAR(64) NOT NULL,
  timestamp         DATE NOT NULL
);

CREATE TABLE TUSER(
  id                VARCHAR(64) PRIMARY KEY NOT NULL,
  name              VARCHAR(128),
  description       VARCHAR(256),
  accessControlPolicy VARCHAR(64) NOT NULL,
  organization       VARCHAR(64) NOT NULL ,
  address_city      VARCHAR(64),
  address_country   VARCHAR(64),
  address_postalCode VARCHAR(64),

```

```

address_state          VARCHAR(64),
address_street         VARCHAR(64),
email                 VARCHAR(64),
fax_areaCode          VARCHAR(4) NOT NULL,
fax_countryCode       VARCHAR(4),
fax_extension         VARCHAR(8),
fax_umber             VARCHAR(8) NOT NULL,
fax_url              VARCHAR(64),
mobilePhone_areaCode  VARCHAR(4) NOT NULL,
mobilePhone_countryCode VARCHAR(4),
mobilePhone_extension VARCHAR(8),
mobilePhone_number    VARCHAR(8) NOT NULL,
mobilePhone_url       VARCHAR(64),
name_firstName        VARCHAR(64),
name_middleName       VARCHAR(64),
name_lastName         VARCHAR(64),
pager_areaCode        VARCHAR(4) NOT NULL,
pager_countryCode     VARCHAR(4),
pager_extension       VARCHAR(8),
pager_number          VARCHAR(8) NOT NULL,
pager_url             VARCHAR(64),
telephone_areaCode    VARCHAR(4) NOT NULL,
telephone_countryCode VARCHAR(4),
telephone_extension   VARCHAR(8),
telephone_number      VARCHAR(8) NOT NULL,
telephone_url         VARCHAR(64),
url                  VARCHAR(64)
);

CREATE TABLE Organization (
id          VARCHAR(64) PRIMARY KEY NOT NULL,
name        VARCHAR(128),
description VARCHAR(256),
accessControlPolicy VARCHAR(64) NOT NULL,
majorVersion INT DEFAULT 0 NOT NULL,
minorVersion INT DEFAULT 1 NOT NULL,
status      INT DEFAULT 0 NOT NULL,
userVersion VARCHAR(64),
stability   INT DEFAULT 0 NOT NULL,
expirationDate DATE,
address_city VARCHAR(64),
address_country VARCHAR(64),
address_postalCode VARCHAR(64),
address_state VARCHAR(64),
address_street VARCHAR(64),
contact      VARCHAR(64) NOT NULL,
fax_areaCode VARCHAR(4) NOT NULL,
fax_countryCode VARCHAR(4),
fax_extension VARCHAR(8),

```

```

fax_umber          VARCHAR(8) NOT NULL,
fax_url            VARCHAR(64),
parent VARCHAR(64),
telephone_areaCode VARCHAR(4) NOT NULL,
telephone_countryCode VARCHAR(4),
telephone_extension VARCHAR(8),
telephone_umber    VARCHAR(8) NOT NULL,
telephone_url      VARCHAR(64)
);
CREATE TABLE CPP (
id          VARCHAR(64) PRIMARY KEY NOT NULL,
name        VARCHAR(128),
description  VARCHAR(256),
accessControlPolicy VARCHAR(64) NOT NULL,
majorVersion INT DEFAULT 0 NOT NULL,
minorVersion INT DEFAULT 1 NOT NULL,
status      INT DEFAULT 0 NOT NULL,
userVersion VARCHAR(64),
stability   INT DEFAULT 0 NOT NULL,
expirationDate DATE,
CPPIid      VARCHAR(64),
packageId   VARCHAR(64)
);
CREATE TABLE PartyInfo (
id          VARCHAR(64) PRIMARY KEY NOT NULL,
name        VARCHAR(128),
description  VARCHAR(256),
accessControlPolicy VARCHAR(64) NOT NULL,
majorVersion INT DEFAULT 0 NOT NULL,
minorVersion INT DEFAULT 1 NOT NULL,
status      INT DEFAULT 0 NOT NULL,
userVersion VARCHAR(64),
stability   INT DEFAULT 0 NOT NULL,
expirationDate DATE,
CPPIid      VARCHAR(64),
partyId     VARCHAR(64)
);
CREATE TABLE CollaborationRole (
id          VARCHAR(64) PRIMARY KEY NOT NULL,
name        VARCHAR(128),
description  VARCHAR(256),
accessControlPolicy VARCHAR(64) NOT NULL,
majorVersion INT DEFAULT 0 NOT NULL,
minorVersion INT DEFAULT 1 NOT NULL,
status      INT DEFAULT 0 NOT NULL,
userVersion VARCHAR(64),
stability   INT DEFAULT 0 NOT NULL,
expirationDate DATE,

```

```

partyInfoId          VARCHAR(64),
roleId               VARCHAR(64),
processSpecId        VARCHAR(64)
);
CREATE TABLE ProcessSpec (
id                   VARCHAR(64) PRIMARY KEY NOT NULL,
name                VARCHAR(128),
description          VARCHAR(256),
accessControlPolicy VARCHAR(64) NOT NULL,
majorVersion        INT DEFAULT 0 NOT NULL,
minorVersion         INT DEFAULT 1 NOT NULL,
status               INT DEFAULT 0 NOT NULL,
userVersion          VARCHAR(64),
stability            INT DEFAULT 0 NOT NULL,
expirationDate       DATE,
version              VARCHAR(128),
xLink                VARCHAR(256),
xLinkType            VARCHAR(128)
);
CREATE TABLE Role (
id                   VARCHAR(64) PRIMARY KEY NOT NULL,
name                VARCHAR(128),
description          VARCHAR(256),
accessControlPolicy VARCHAR(64) NOT NULL,
majorVersion        INT DEFAULT 0 NOT NULL,
minorVersion         INT DEFAULT 1 NOT NULL,
status               INT DEFAULT 0 NOT NULL,
userVersion          VARCHAR(64),
stability            INT DEFAULT 0 NOT NULL,
expirationDate       DATE,
xLink                VARCHAR(256),
xLinkType            VARCHAR(128)
);
CREATE TABLE ServiceBinding (
id                   VARCHAR(64) PRIMARY KEY NOT NULL,
name                VARCHAR(128),
description          VARCHAR(256),
accessControlPolicy VARCHAR(64) NOT NULL,
majorVersion        INT DEFAULT 0 NOT NULL,
minorVersion         INT DEFAULT 1 NOT NULL,
status               INT DEFAULT 0 NOT NULL,
userVersion          VARCHAR(64),
stability            INT DEFAULT 0 NOT NULL,
expirationDate       DATE,
channelId            VARCHAR(64),
packageId            VARCHAR(64),
collaborationRoleId VARCHAR(64)
);

```

```

CREATE TABLE DeliveryChannel (
  id          VARCHAR(64) PRIMARY KEY NOT NULL,
  name        VARCHAR(128),
  description  VARCHAR(256),
  accessControlPolicy VARCHAR(64) NOT NULL,
  majorVersion INT DEFAULT 0 NOT NULL,
  minorVersion INT DEFAULT 1 NOT NULL,
  status      INT DEFAULT 0 NOT NULL,
  userVersion  VARCHAR(64),
  stability    INT DEFAULT 0 NOT NULL,
  expirationDate DATE,
  channelId    VARCHAR(64),
  transportId  VARCHAR(64),
  docExchangeId VARCHAR(64)
);
CREATE TABLE Transport(
  id          VARCHAR(64) PRIMARY KEY NOT NULL,
  name        VARCHAR(128),
  description  VARCHAR(256),
  accessControlPolicy VARCHAR(64) NOT NULL,
  majorVersion INT DEFAULT 0 NOT NULL,
  minorVersion INT DEFAULT 1 NOT NULL,
  status      INT DEFAULT 0 NOT NULL,
  userVersion  VARCHAR(64),
  stability    INT DEFAULT 0 NOT NULL,
  expirationDate DATE,
  senderProtocol VARCHAR(128),
  receiverProtocol VARCHAR(128),
  senderProtocolVer VARCHAR(128),
  receiverProtocolVer VARCHAR(128)
);
CREATE TABLE TransportSecurity(
  id          VARCHAR(64) PRIMARY KEY NOT NULL,
  name        VARCHAR(128),
  description  VARCHAR(256),
  accessControlPolicy VARCHAR(64) NOT NULL,
  majorVersion INT DEFAULT 0 NOT NULL,
  minorVersion INT DEFAULT 1 NOT NULL,
  status      INT DEFAULT 0 NOT NULL,
  userVersion  VARCHAR(64),
  stability    INT DEFAULT 0 NOT NULL,
  expirationDate DATE,
  protocolVersion VARCHAR(128),
  certId        VARCHAR(64),
  transportId    VARCHAR(64)
);
CREATE TABLE SimplePackagePart(
  id          VARCHAR(64) PRIMARY KEY NOT NULL,

```

```

name                VARCHAR(128),
description          VARCHAR(256),
accessControlPolicy VARCHAR(64) NOT NULL,
majorVersion        INT DEFAULT 0 NOT NULL,
minorVersion         INT DEFAULT 1 NOT NULL,
status              INT DEFAULT 0 NOT NULL,
userVersion          VARCHAR(64),
stability            INT DEFAULT 0 NOT NULL,
expirationDate       DATE,
mimeType             VARCHAR(64),
packageId            VARCHAR(64)
);
CREATE TABLE Composite(
id                  VARCHAR(64) PRIMARY KEY NOT NULL,
name               VARCHAR(128),
description         VARCHAR(256),
accessControlPolicy VARCHAR(64) NOT NULL,
majorVersion        INT DEFAULT 0 NOT NULL,
minorVersion         INT DEFAULT 1 NOT NULL,
status              INT DEFAULT 0 NOT NULL,
userVersion          VARCHAR(64),
stability            INT DEFAULT 0 NOT NULL,
expirationDate       DATE,
mimeType             VARCHAR(64),
packageId            VARCHAR(64)
);
CREATE TABLE Constituent(
id                  VARCHAR(64) PRIMARY KEY NOT NULL,
name               VARCHAR(128),
description         VARCHAR(256),
accessControlPolicy VARCHAR(64) NOT NULL,
majorVersion        INT DEFAULT 0 NOT NULL,
minorVersion         INT DEFAULT 1 NOT NULL,
status              INT DEFAULT 0 NOT NULL,
userVersion          VARCHAR(64),
stability            INT DEFAULT 0 NOT NULL,
expirationDate       DATE,
idref               VARCHAR(64),
compositeId          VARCHAR(64),
order_no            VARCHAR(64)
);
CREATE TABLE DAMLOIL_SUBCLASS (
id                  VARCHAR(64) PRIMARY KEY NOT NULL,
name               VARCHAR(128),
description         VARCHAR(256),
accessControlPolicy VARCHAR(64) NOT NULL,
majorVersion        INT DEFAULT 0 NOT NULL,
minorVersion         INT DEFAULT 1 NOT NULL,

```

```

status                                INT DEFAULT 0 NOT NULL,
userVersion                          VARCHAR(64),
stability                            INT DEFAULT 0 NOT NULL,
expirationDate                       DATE,
damlId                               VARCHAR(128),
subClassId                           VARCHAR(128)
);
CREATE TABLE DAMLOIL_RESTRICTION(
id                                    VARCHAR(64) PRIMARY KEY NOT NULL,
name                                  VARCHAR(128),
description                           VARCHAR(256),
accessControlPolicy                   VARCHAR(64) NOT NULL,
majorVersion                          INT DEFAULT 0 NOT NULL,
minorVersion                          INT DEFAULT 1 NOT NULL,
status                                INT DEFAULT 0 NOT NULL,
userVersion                          VARCHAR(64),
stability                            INT DEFAULT 0 NOT NULL,
expirationDate                       DATE,
Property                              VARCHAR(64),
ClassId                              VARCHAR(64)
);
CREATE TABLE DAMLOIL_SUBPROPERTY (
id                                    VARCHAR(64) PRIMARY KEY NOT NULL,
name                                  VARCHAR(128),
description                           VARCHAR(256),
accessControlPolicy                   VARCHAR(64) NOT NULL,
majorVersion                          INT DEFAULT 0 NOT NULL,
minorVersion                          INT DEFAULT 1 NOT NULL,
status                                INT DEFAULT 0 NOT NULL,
userVersion                          VARCHAR(64),
stability                            INT DEFAULT 0 NOT NULL,
expirationDate                       DATE,
damlId                               VARCHAR(128),
subPropId                             VARCHAR(128)
);
CREATE TABLE HOTEL_RESERVATION (
id                                    VARCHAR(64) PRIMARY KEY NOT NULL,
name                                  VARCHAR(128),
description                           VARCHAR(256),
accessControlPolicy                   VARCHAR(64) NOT NULL,
majorVersion                          INT DEFAULT 0 NOT NULL,
minorVersion                          INT DEFAULT 1 NOT NULL,
status                                INT DEFAULT 0 NOT NULL,
userVersion                          VARCHAR(64),
stability                            INT DEFAULT 0 NOT NULL,
expirationDate                       DATE,
CreditCardNo                         VARCHAR(64),
CreditCardType                       VARCHAR(64),

```

```

StarRating          VARCHAR(64)
);
CREATE TABLE PRODUCT (
id                  VARCHAR(64) PRIMARY KEY NOT NULL,
name               VARCHAR(128),
description        VARCHAR(256),
accessControlPolicy VARCHAR(64) NOT NULL,
majorVersion      INT DEFAULT 0 NOT NULL,
minorVersion      INT DEFAULT 1 NOT NULL,
status            INT DEFAULT 0 NOT NULL,
userVersion       VARCHAR(64),
stability         INT DEFAULT 0 NOT NULL,
expirationDate    DATE,
ProductId         VARCHAR(64)
);

```

```

CREATE TABLE PHYSICAL_PRODUCT (
id                  VARCHAR(64) PRIMARY KEY NOT NULL,
name               VARCHAR(128),
description        VARCHAR(256),
accessControlPolicy VARCHAR(64) NOT NULL,
majorVersion      INT DEFAULT 0 NOT NULL,
minorVersion      INT DEFAULT 1 NOT NULL,
status            INT DEFAULT 0 NOT NULL,
userVersion       VARCHAR(64),
stability         INT DEFAULT 0 NOT NULL,
expirationDate    DATE,
type              VARCHAR(64)
);

```

```

CREATE TABLE VIRTUAL_PRODUCT (
id                  VARCHAR(64) PRIMARY KEY NOT NULL,
name               VARCHAR(128),
description        VARCHAR(256),
accessControlPolicy VARCHAR(64) NOT NULL,
majorVersion      INT DEFAULT 0 NOT NULL,
minorVersion      INT DEFAULT 1 NOT NULL,
status            INT DEFAULT 0 NOT NULL,
userVersion       VARCHAR(64),
stability         INT DEFAULT 0 NOT NULL,
expirationDate    DATE,
location          VARCHAR(64)
);

```

```

CREATE TABLE POEC_SERVICE (
id                  VARCHAR(64) PRIMARY KEY NOT NULL,
name               VARCHAR(128),
description        VARCHAR(256),
accessControlPolicy VARCHAR(64) NOT NULL,

```

```

majorVersion          INT DEFAULT 0 NOT NULL,
minorVersion          INT DEFAULT 1 NOT NULL,
status               INT DEFAULT 0 NOT NULL,
userVersion          VARCHAR(64),
stability            INT DEFAULT 0 NOT NULL,
expirationDate       DATE,
availableDay         VARCHAR(64)
);
CREATE TABLE SERVICE_CATALOG (
id                   VARCHAR(64) PRIMARY KEY NOT NULL,
name                VARCHAR(128),
description          VARCHAR(256),
accessControlPolicy VARCHAR(64) NOT NULL,
majorVersion        INT DEFAULT 0 NOT NULL,
minorVersion        INT DEFAULT 1 NOT NULL,
status              INT DEFAULT 0 NOT NULL,
userVersion         VARCHAR(64),
stability           INT DEFAULT 0 NOT NULL,
expirationDate      DATE,
location            VARCHAR(128)
);
CREATE TABLE RECEIPT (
id                   VARCHAR(64) PRIMARY KEY NOT NULL,
name                VARCHAR(128),
description          VARCHAR(256),
accessControlPolicy VARCHAR(64) NOT NULL,
majorVersion        INT DEFAULT 0 NOT NULL,
minorVersion        INT DEFAULT 1 NOT NULL,
status              INT DEFAULT 0 NOT NULL,
userVersion         VARCHAR(64),
stability           INT DEFAULT 0 NOT NULL,
expirationDate      DATE,
payType             VARCHAR(64)
);
CREATE TABLE RENT_VEICHLE_SERVICE (
id                   VARCHAR(64) PRIMARY KEY NOT NULL,
name                VARCHAR(128),
description          VARCHAR(256),
accessControlPolicy VARCHAR(64) NOT NULL,
majorVersion        INT DEFAULT 0 NOT NULL,
minorVersion        INT DEFAULT 1 NOT NULL,
status              INT DEFAULT 0 NOT NULL,
userVersion         VARCHAR(64),
stability           INT DEFAULT 0 NOT NULL,
expirationDate      DATE,
country             VARCHAR(64),
city                VARCHAR(64)
);

```

```

CREATE TABLE CAR_RENTRAL_SERVICE (
  id                VARCHAR(64) PRIMARY KEY NOT NULL,
  name              VARCHAR(128),
  description       VARCHAR(256),
  accessControlPolicy VARCHAR(64) NOT NULL,
  majorVersion      INT DEFAULT 0 NOT NULL,
  minorVersion      INT DEFAULT 1 NOT NULL,
  status            INT DEFAULT 0 NOT NULL,
  userVersion       VARCHAR(64),
  stability         INT DEFAULT 0 NOT NULL,
  expirationDate    DATE,
  carname           VARCHAR(64),
  model             VARCHAR(64),
  color             VARCHAR(64)
);
CREATE TABLE LIMOSINE_RENTRAL_SERVICE(
  id                VARCHAR(64) PRIMARY KEY NOT NULL,
  name              VARCHAR(128),
  description       VARCHAR(256),
  accessControlPolicy VARCHAR(64) NOT NULL,
  majorVersion      INT DEFAULT 0 NOT NULL,
  minorVersion      INT DEFAULT 1 NOT NULL,
  status            INT DEFAULT 0 NOT NULL,
  userVersion       VARCHAR(64),
  stability         INT DEFAULT 0 NOT NULL,
  expirationDate    DATE,
  model             VARCHAR(64),
  color             VARCHAR(64)
);
CREATE TABLE CHAUFFEUR_SERVICE(
  id                VARCHAR(64) PRIMARY KEY NOT NULL,
  name              VARCHAR(128),
  description       VARCHAR(256),
  accessControlPolicy VARCHAR(64) NOT NULL,
  majorVersion      INT DEFAULT 0 NOT NULL,
  minorVersion      INT DEFAULT 1 NOT NULL,
  status            INT DEFAULT 0 NOT NULL,
  userVersion       VARCHAR(64),
  stability         INT DEFAULT 0 NOT NULL,
  expirationDate    DATE,
  chauffeurage      VARCHAR(64),
  chauffeurexperience VARCHAR(64)
);
CREATE VIEW RegistryEntry (
  id, name, description, accessControlPolicy, majorVersion, minorVersion, status,
  userVersion, stability, expirationDate, objectType
) AS
SELECT

```

```

id, name, description, accessControlPolicy, majorVersion, minorVersion, status,
        userVersion, stability, expirationDate, 128
FROM ExtrinsicObject
UNION
SELECT
id, name, description, accessControlPolicy, majorVersion, minorVersion, status,
        userVersion, stability, expirationDate, 111
FROM ExternalLink
UNION
SELECT
id, name, description, accessControlPolicy, majorVersion, minorVersion, status,
        userVersion, stability, expirationDate, 112
FROM ExternalIdentifier
UNION
SELECT
id, name, description, accessControlPolicy, majorVersion, minorVersion, status,
        userVersion, stability, expirationDate, 114
FROM Classification
UNION
SELECT
id, name, description, accessControlPolicy, majorVersion, minorVersion, status,
        userVersion, stability, expirationDate, 113
FROM Association
UNION
SELECT
id, name, description, accessControlPolicy, majorVersion, minorVersion, status,
        userVersion, stability, expirationDate, 117
FROM Organization
UNION
SELECT
id, name, description, accessControlPolicy, majorVersion, minorVersion, status,
        userVersion, stability, expirationDate, 110
FROM Package
UNION
SELECT
id, name, description, accessControlPolicy, majorVersion, minorVersion, status,
        userVersion, stability, expirationDate, 102
FROM CPP
UNION
SELECT
id, name, description, accessControlPolicy, majorVersion, minorVersion, status,
        userVersion, stability, expirationDate, 118
FROM PartyInfo
UNION
SELECT
id, name, description, accessControlPolicy, majorVersion, minorVersion, status,
        userVersion, stability, expirationDate, 119
FROM CollaborationRole

```

```

UNION
SELECT
id, name, description, accessControlPolicy, majorVersion, minorVersion, status,
        userVersion, stability, expirationDate, 120
FROM ProcessSpec
UNION
SELECT
id, name, description, accessControlPolicy, majorVersion, minorVersion, status,
        userVersion, stability, expirationDate, 104
FROM Role
UNION
SELECT
id, name, description, accessControlPolicy, majorVersion, minorVersion, status,
        userVersion, stability, expirationDate, 121
FROM ServiceBinding
UNION
SELECT
id, name, description, accessControlPolicy, majorVersion, minorVersion, status,
        userVersion, stability, expirationDate, 122
FROM DeliveryChannel
UNION
SELECT
id, name, description, accessControlPolicy, majorVersion, minorVersion, status,
        userVersion, stability, expirationDate, 107
FROM Transport
UNION
SELECT
id, name, description, accessControlPolicy, majorVersion, minorVersion, status,
        userVersion, stability, expirationDate, 123
FROM TransportSecurity
UNION
SELECT
id, name, description, accessControlPolicy, majorVersion, minorVersion, status,
        userVersion, stability, expirationDate, 124
FROM SimplePackagePart
UNION
SELECT
id, name, description, accessControlPolicy, majorVersion, minorVersion, status,
        userVersion, stability, expirationDate, 125
FROM Composite
UNION
SELECT
id, name, description, accessControlPolicy, majorVersion, minorVersion, status,
        userVersion, stability, expirationDate, 126
FROM Constituent
UNION
SELECT
id, name, description, accessControlPolicy, majorVersion, minorVersion, status,

```

```

        userVersion,stability, expirationDate,127
FROM ClassificationNode
UNION
SELECT
id, name, description, accessControlPolicy,majorVersion, minorVersion, status,
        userVersion,stability, expirationDate,900
FROM DAMLOIL_SUBCLASS
UNION
SELECT
id, name, description, accessControlPolicy,majorVersion, minorVersion, status,
        userVersion,stability, expirationDate,901
FROM DAMLOIL_RESTRICTION
UNION
SELECT
id, name, description, accessControlPolicy,majorVersion, minorVersion, status,
        userVersion,stability, expirationDate,902
FROM DAMLOIL_SUBPROPERTY
UNION
SELECT
id, name, description, accessControlPolicy, majorVersion, minorVersion, status,
        userVersion,stability, expirationDate,903
FROM HOTEL_RESERVATION
UNION
SELECT
id, name, description, accessControlPolicy,majorVersion, minorVersion, status,
        userVersion,stability, expirationDate,904
FROM PRODUCT
UNION
SELECT
id, name, description, accessControlPolicy, majorVersion, minorVersion, status,
        userVersion,stability, expirationDate,905
FROM PHYSICAL_PRODUCT
UNION
SELECT
id, name, description, accessControlPolicy,majorVersion, minorVersion, status,
        userVersion,stability, expirationDate,906
FROM VIRTUAL_PRODUCT
UNION
SELECT
id, name, description, accessControlPolicy, majorVersion, minorVersion, status,
        userVersion,stability, expirationDate,907
FROM POEC_SERVICE
UNION
SELECT
id, name, description, accessControlPolicy, majorVersion, minorVersion, status,
        userVersion,stability, expirationDate,908
FROM SERVICE_CATALOG
UNION

```

```

SELECT
id, name, description, accessControlPolicy, majorVersion, minorVersion, status,
        userVersion,stability, expirationDate,909
FROM RECEIPT
UNION
SELECT
id, name, description, accessControlPolicy, majorVersion, minorVersion, status,
        userVersion,stability, expirationDate,910
FROM RENT_VEICHLE_SERVICE
UNION
SELECT
id, name, description, accessControlPolicy,majorVersion, minorVersion, status,
        userVersion,stability, expirationDate,911
FROM CAR_RENTRAL_SERVICE
UNION
SELECT
id, name, description, accessControlPolicy, majorVersion, minorVersion, status,
        userVersion,stability, expirationDate,912
FROM LIMOSINE_RENTRAL_SERVICE
UNION
SELECT
id, name, description, accessControlPolicy, majorVersion, minorVersion, status,
        userVersion,stability, expirationDate,913
FROM CHAUFFEUR_SERVICE;

```

## ebXML REGISTRY DTD DEFINITION

```
<!ENTITY % RegistryEntryAttributes " %ObjectAttributes;
majorVersion      CDATA  '1'
minorVersion      CDATA  '0'
```

```

status          CDATA #IMPLIED
userVersion     CDATA #IMPLIED
stability       CDATA 'Dynamic'
expirationDate  CDATA #IMPLIED">
<!ELEMENT RegistryEntry (SlotList?)>
<!ATTLIST RegistryEntry      %RegistryEntryAttributes; >
<!ELEMENT Value (#PCDATA)>
<!ELEMENT ValueList (Value*)>
<!ELEMENT Slot (ValueList?)>
<!ATTLIST Slot
name CDATA #REQUIRED
slotType CDATA #IMPLIED
<!ELEMENT SlotList (Slot*) >
<!-- ExtrinsicObject are attributes from the ExtrinsicObject interface in ebRIM.
It inherits RegistryEntryAttributes -->

<!ELEMENT ExtrinsicObject EMPTY >
<!ATTLIST ExtrinsicObject
  %RegistryEntryAttributes;
contentURI CDATA #REQUIRED
mimeType CDATA #IMPLIED
objectType CDATA #REQUIRED
opaque (true | false) "false">
<!ENTITY % IntrinsicObjectAttributes " %RegistryEntryAttributes;">
<!-- Leaf classes that reflect the concrete classes in ebRIM -->
<!ELEMENT RegistryEntryList
(Association | Classification | ClassificationNode | Package | ExternalLink |
ExternalIdentifier | Organization | ExtrinsicObject | ObjectRef | CPP |
CollaborationRole | Role | ProcessSpec | DeliveryChannel | ServiceBinding | Transport |
TransportSecurity | SimplePackagePart | Constituent | Composite | PartyInfo |
DAML_OIL_SUBCLASS | DAML_OIL_RESTRICTION | DAMLOIL_SUBPROPERTY |
DAML_OIL_INSTANCE_HOTEL_RESERVATION | DAML_OIL_INSTANCE_PRODUCT |
DAML_OIL_INSTANCE_PHYSICAL_PRODUCT | DAML_OIL_INSTANCE_VIRTUAL_PRODUCT |
DAML_OIL_INSTANCE_POEC_SERVICE | DAML_OIL_INSTANCE_SERVICE_CATALOG |
DAML_OIL_INSTANCE_RECEIPT | DAML_OIL_INSTANCE_CAR_RENTAL_SERVICE |
DAML_OIL_INSTANCE_LIMOSINE_RENTAL_SERVICE | DAML_OIL_INSTANCE_CHAUFFEUR_SERVICE)* >

<!ELEMENT CPP EMPTY>
<!ATTLIST CPP
%IntrinsicObjectAttributes;
packageId CDATA #IMPLIED >
<!ELEMENT CollaborationRole EMPTY>
<!ATTLIST CollaborationRole
%IntrinsicObjectAttributes;
partyInfo CDATA #IMPLIED
processSpecId CDATA #IMPLIED >
<!ELEMENT Role EMPTY>
<!ATTLIST Role

```

```

%IntrinsicObjectAttributes;
xLink CDATA #IMPLIED
xLinkType CDATA #IMPLIED >
<!--ELEMENT ProcessSpec EMPTY-->
<!--ATTLIST ProcessSpec
%IntrinsicObjectAttributes;
xLink CDATA #IMPLIED
xLinkType CDATA #IMPLIED
version CDATA #IMPLIED >
<!--ELEMENT DeliveryChannel EMPTY-->
<!--ATTLIST DeliveryChannel
%IntrinsicObjectAttributes;
channelId CDATA #IMPLIED
transportId CDATA #IMPLIED
docExchangeId CDATA #IMPLIED >
<!--ELEMENT ServiceBinding EMPTY-->
<!--ATTLIST ServiceBinding
%IntrinsicObjectAttributes;
channelId CDATA #IMPLIED
packageId CDATA #IMPLIED
collaborationRoleId CDATA #IMPLIED >
<!--ELEMENT Transport EMPTY-->
<!--ATTLIST Transport
%IntrinsicObjectAttributes;
senderProtocol CDATA #IMPLIED
senderProtocolVersion CDATA #IMPLIED
receiverProtocol CDATA #IMPLIED
receiverProtocolVersion CDATA #IMPLIED >
<!--ELEMENT TransportSecurity EMPTY-->
<!--ATTLIST TransportSecurity
%IntrinsicObjectAttributes;
protocolVersion CDATA #IMPLIED
certId CDATA #IMPLIED
transportId CDATA #IMPLIED >
<!--ELEMENT SimplePackagePart EMPTY-->
<!--ATTLIST SimplePackagePart
%IntrinsicObjectAttributes;
mimeType CDATA #IMPLIED
packageId CDATA #IMPLIED-->
<!--ELEMENT Constituent EMPTY-->
<!--ATTLIST Constituent
%IntrinsicObjectAttributes;
idRef CDATA #IMPLIED
compositeId CDATA #IMPLIED-->
<!--ELEMENT Composite EMPTY-->
<!--ATTLIST Composite
%IntrinsicObjectAttributes;
packageId CDATA #IMPLIED

```

```

mimeType CDATA #IMPLIED>
<!ELEMENT PartyInfo EMPTY>
<!ATTLIST PartyInfo
%IntrinsicObjectAttributes;
cppId CDATA #IMPLIED
partyId CDATA #IMPLIED>
<!ELEMENT DAML_OIL_SUBCLASS EMPTY>
<!ATTLIST DAML_OIL_SUBCLASS
%IntrinsicObjectAttributes;
damlId CDATA #IMPLIED
subClassId CDATA #IMPLIED>
<!ELEMENT DAML_OIL_RESTRICTION EMPTY>
<!ATTLIST DAML_OIL_ RESTRICTION
%IntrinsicObjectAttributes;
property CDATA #IMPLIED
classId CDATA #IMPLIED>
<!ELEMENT DAML_OIL_SUBPROPERTY EMPTY>
<!ATTLIST DAML_OIL_ SUBPROPERTY
%IntrinsicObjectAttributes;
damlId CDATA #IMPLIED
subPropId CDATA #IMPLIED>
<!ELEMENT DAML_OIL_INSTANCE_HOTEL_RESERVATION EMPTY>
<!ATTLIST DAML_OIL_ INSTANCE_HOTEL_RESERVATION
%IntrinsicObjectAttributes;
creditCardNo CDATA #IMPLIED
creditCardType CDATA #IMPLIED
starRating CDATA #IMPLIED>
<!ELEMENT DAML_OIL_INSTANCE_PRODUCT EMPTY>
<!ATTLIST DAML_OIL_ INSTANCE _PRODUCT
%IntrinsicObjectAttributes;
productId CDATA #IMPLIED>
<!ELEMENT DAML_OIL_INSTANCE_PHYSICAL_PRODUCT EMPTY>
<!ATTLIST DAML_OIL_ INSTANCE _ PHYSICAL _PRODUCT
%IntrinsicObjectAttributes;
pType CDATA #IMPLIED>
<!ELEMENT DAML_OIL_INSTANCE_VIRTUAL_PRODUCT EMPTY>
<!ATTLIST DAML_OIL_ INSTANCE _ VIRTUAL _PRODUCT
%IntrinsicObjectAttributes;
location CDATA #IMPLIED>
<!ELEMENT DAML_OIL_INSTANCE_POEC_SERVICE EMPTY>
<!ATTLIST DAML_OIL_ INSTANCE _ POEC_SERVICE
%IntrinsicObjectAttributes;
availableDay CDATA #IMPLIED>
<!ELEMENT DAML_OIL_INSTANCE_SERVICE_CATALOG EMPTY>
<!ATTLIST DAML_OIL_ INSTANCE _ SERVICE _CATALOG
%IntrinsicObjectAttributes;
location CDATA #IMPLIED>
<!ELEMENT DAML_OIL_INSTANCE_RECEIPT EMPTY>

```

```

<!--ATTLIST DAML_OIL_INSTANCE_RECEIPT
%IntrinsicObjectAttributes;
payType CDATA #IMPLIED>
<!--ELEMENT DAML_OIL_INSTANCE_RENT_VEICHEL_SERVICE EMPTY>
<!--ATTLIST DAML_OIL_INSTANCE_RENT_VEICHEL_SERVICE
%IntrinsicObjectAttributes;
country CDATA #IMPLIED
city CDATA #IMPLIED>
<!--ELEMENT DAML_OIL_INSTANCE_CAR_RENTAL_SERVICE EMPTY>
<!--ATTLIST DAML_OIL_INSTANCE_CAR_RENTAL_SERVICE
%IntrinsicObjectAttributes;
carname CDATA #IMPLIED
model CDATA #IMPLIED
color CDATA #IMPLIED>
<!--ELEMENT DAML_OIL_INSTANCE_LIMOSINE_RENTAL_SERVICE EMPTY>
<!--ATTLIST DAML_OIL_INSTANCE_LIMOSINE_RENTAL_SERVICE
%IntrinsicObjectAttributes;
model CDATA #IMPLIED
color CDATA #IMPLIED>
<!--ELEMENT DAML_OIL_INSTANCE_CHAUFFEUR_SERVICE EMPTY>
<!--ATTLIST DAML_OIL_INSTANCE_CHAUFFEUR_SERVICE
%IntrinsicObjectAttributes;
chauffeurage CDATA #IMPLIED
chauffeursexperience CDATA #IMPLIED>
<!--An ExternalLink specifies a link from a RegistryEntry and an external URI
<!--ELEMENT ExternalLink EMPTY>
<!--ATTLIST ExternalLink
%IntrinsicObjectAttributes;
externalURI CDATA #IMPLIED>
<!-- An ExternalIdentifier provides an identifier for a RegistryEntry
The value is the value of the identifier (e.g. the social security number) -->
<!--ELEMENT ExternalIdentifier EMPTY>
<!--ATTLIST ExternalIdentifier
%IntrinsicObjectAttributes;
value CDATA #REQUIRED >
<!-- An Association specifies references to two previously submitted registry
entrys. The sourceObject is id of the sourceObject in association The
targetObject is id of the targetObject
in association -->
<!--ELEMENT Association EMPTY>
<!--ATTLIST Association
%IntrinsicObjectAttributes;
sourceRole CDATA #IMPLIED
targetRole CDATA #IMPLIED
associationType CDATA #REQUIRED
bidirection (true | false) "false"
sourceObject IDREF #REQUIRED
targetObject IDREF #REQUIRED>

```

```

<!-- A Classification specifies references to two registry entrys. The
classifiedObject is id of the Object being classified. The classificationNode
is id of the ClassificationNode classifying the
object -->
<!ELEMENT Classification EMPTY>
  <!-- ATTLIST Classification
%IntrinsicObjectAttributes;
classifiedObject IDREF #REQUIRED
classificationNode IDREF #REQUIRED>
<!-- A Package is a named collection of objects. -->
<!ELEMENT Package EMPTY>
<!-- ATTLIST Package
%IntrinsicObjectAttributes;>
<!-- Attributes inherited by various types of telephone number elements -->
<!-- ENTITY % TelephoneNumberAttributes "
areaCode CDATA #REQUIRED
contryCode CDATA #REQUIRED
extension CDATA #IMPLIED
number CDATA #REQUIRED
url CDATA #IMPLIED">
<!ELEMENT TelephoneNumber EMPTY>
<!-- ATTLIST TelephoneNumber
%TelephoneNumberAttributes;>
<!-- ELEMENT FaxNumber EMPTY>
<!-- ATTLIST FaxNumber
%TelephoneNumberAttributes;>
<!-- ELEMENT PagerNumber EMPTY>
<!-- ATTLIST PagerNumber
%TelephoneNumberAttributes;>
<!-- ELEMENT MobileTelephoneNumber EMPTY>
<!-- ATTLIST MobileTelephoneNumber
%TelephoneNumberAttributes;>
<!-- PostalAddress -->
<!-- ELEMENT PostalAddress EMPTY>
<!-- ATTLIST PostalAddress
city CDATA #REQUIRED
country CDATA #REQUIRED
postalCode CDATA #REQUIRED
state CDATA #IMPLIED
street CDATA #REQUIRED>
<!-- PersonName -->
<!-- ELEMENT PersonName EMPTY>
<!-- ATTLIST PersonName
firstName CDATA #REQUIRED
middleName CDATA #IMPLIED
lastName CDATA #REQUIRED>
<!-- Organization -->
<!-- ELEMENT Organization (PostalAddress, FaxNumber?, TelephoneNumber)>

```

```

<!ATTLIST Organization
%IntrinsicObjectAttributes;
parent IDREF #IMPLIED
primaryContact IDREF #REQUIRED>

<!ELEMENT User (PersonName, PostalAddress, TelephoneNumber, MobileTelephoneNumber?,
FaxNumber?, PagerNumber?)>
<!ATTLIST User
%ObjectAttributes;
organization IDREF #IMPLIED
email DATA #IMPLIED
url CDATA #IMPLIED>
<!ELEMENT AuditableEvent EMPTY>
<!ATTLIST AuditableEvent
%ObjectAttributes;
eventType CDATA #REQUIRED
registryEntry IDREF #REQUIRED
timestamp CDATA #REQUIRED
user IDREF #REQUIRED>
<!--ClassificationNode is used to submit a Classification tree to the Registry.
parent is the idto the parent node. code is an optional code value for a
ClassificationNode often defined by an external taxonomy (e.g. NAICS)-->
<!ELEMENT ClassificationNode EMPTY>
<!ATTLIST ClassificationNode
%IntrinsicObjectAttributes;
parent IDREF #IMPLIED
code CDATA #IMPLIED>
<!--End information model mapping. Begin Registry Services Interface-->
<!ELEMENT RequestAcceptedResponse EMPTY>
<!ATTLIST RequestAcceptedResponse
xml:lang NMTOKEN #REQUIRED>

<!-- The SubmitObjectsRequest allows one to submit a list of RegistryEntry elements.
Each RegistryEntryelement provides metadata for a single submitted object.
Note that the repository item being submittedis in a separate document that is not
in this DTD. The ebXML Messaging Services Specfication defines packaging, for
submission, of the metadata of a repository item with the repository item itself.
The valueof the contentURI attribute of the ExtrinsicObject element must be the
same as the xlink:href attribute within the Reference element within the Manifest
element of the MessageHeader.-->

<!ELEMENT SubmitObjectsRequest (RegistryEntryList)>
<!ELEMENT AddSlotsRequest (ObjectRef, SlotList)+>
<!-- Only need name in Slot within SlotList -->
<!ELEMENT RemoveSlotsRequest (ObjectRef, SlotList)+>
<!--The ObjectRefList is the list ofrefs to the registry entrys being
approved.-->
<!ELEMENT ApproveObjectsRequest (ObjectRefList)>

```

```

<!--The ObjectRefList is the list of refs to the registry entrys being
deprecated.-->
<!ELEMENT DeprecateObjectsRequest (ObjectRefList)>
<!--The ObjectRefList is the list of refs to the registry entrys being
removed-->
<!ELEMENT RemoveObjectsRequest (ObjectRefList)>
<!--ATTLIST RemoveObjectsRequest deletionScope (DeleteAll |
DeleteRepositoryItemOnly) "DeleteAll">
<!ELEMENT GetRootClassificationNodesRequest EMPTY>
<!--The namePattern follows SQL-92 syntax for the pattern specified in
LIKE clause. It allows for selecting only those root nodes that match the
namePattern. The default value of '*' matches all root nodes.-->

<!--ATTLIST GetRootClassificationNodesRequest namePattern CDATA "*"
-->
<!--The response includes one or more ClassificationNodes-->

<!ELEMENT GetRootClassificationNodesResponse ( ClassificationNode+ )>
<!--Get the classification tree under the ClassificationNode specified parentRef.
If depth is 1 just fetch immediate child nodes, otherwise fetch the descendant
tree upto the specified depth level. If depth is 0 that implies fetch entire
sub-tree-->
<!--ATTLIST GetClassificationTreeRequest EMPTY>
<!--ATTLIST GetClassificationTreeRequest
parent CDATA #REQUIRED
depth CDATA "1">
<!--The response includes one or more ClassificationNodes which includes only
immediate ClassificationNode children nodes if depth attribute in
GetClassificationTreeRequest was 1,otherwise the decendent nodes upto specified
depth level are returned.-->

<!--ELEMENT GetClassificationTreeResponse ( ClassificationNode+ )>
<!--Get refs to all registry entrys that are classified by all the
ClassificationNodes specified by ObjectRefList. Note this is an implicit
logical AND operation -->

<!--ELEMENT GetClassifiedObjectsRequest (ObjectRefList)>
<!--objectType attribute can specify the type of objects that the
registry client is interested in, that is mclassified by this ClassificationNode.
It is a String that matches a choice in the type attribute of ExtrinsicObject.
The default value of '*' implies that client is interested in all types of
registry entrys that are classified by the specified ClassificationNode. -->
<!--The response includes a RegistryEntryList which has zero or more
RegistryEntrys that are classified by the ClassificationNodes specified
in the ObjectRefList in GetClassifiedObjectsRequest.-->

<!--ELEMENT GetClassifiedObjectsResponse ( RegistryEntryList )>
<!--An Ad hoc query request specifies a query string as defined by [RS]
in the queryString attribute-->
<!--ELEMENT AdhocQueryRequest (FilterQuery | ReturnRegistryEntry |

```

```

ReturnRepositoryItem| SQLQuery| CPPQuery | ReturnCompatibleCPP |
DAMLOILQueryRequest)>

<!ELEMENT SQLQuery (#PCDATA)>
<!--The response includes a RegistryEntryList which has zero or more
RegistryEntrys that match the query specified in AdhocQueryRequest.-->
<!ELEMENT AdhocQueryResponse ( RegistryEntryList | FilterQueryResult |
ReturnRegistryEntryResult |ReturnRepositoryItemResult |
CPPMatchingQueryResult | ReturnCPPQueryResult )>

<!--Gets the actual content (not metadata) specified by the ObjectRefList-->
<!ELEMENT GetContentRequest (ObjectRefList)>
<!--The GetObjectsResponse will have no sub-elements if there were no
errors.The actual contents will bein the other payloads of the message.-->
<!ELEMENT GetContentResponse EMPTY >
<!--Describes the capability profile for the registry and what optional
features are supported-->
<!ELEMENT RegistryProfile (OptionalFeaturesSupported)>
<!ATTLIST RegistryProfile version CDATA #REQUIRED>
<!ELEMENT OptionalFeaturesSupported EMPTY>
<!ATTLIST OptionalFeaturesSupported sqlQuery (true | false) "false"
xQuery (true | false) "false">
<!-- Begin FilterQuery DTD -->
<!ELEMENT FilterQuery (RegistryEntryQuery | AuditableEventQuery |
ClassificationNodeQuery | RegistryPackageQuery| OrganizationQuery)>
<!ELEMENT FilterQueryResult (RegistryEntryQueryResult |
AuditableEventQueryResult| ClassificationNodeQueryResult|
RegistryPackageQueryResult | OrganizationQueryResult)>

<!ELEMENT RegistryEntryQueryResult (RegistryEntryView*)>
<!ELEMENT RegistryEntryView EMPTY>
<!ATTLIST RegistryEntryView
objectURN CDATA #REQUIRED
contentURI CDATA #IMPLIED
objectId CDATA #IMPLIED>
<!ELEMENT AuditableEventQueryResult (AuditableEventView*)>
<!ELEMENT AuditableEventView EMPTY>
<!ATTLIST AuditableEventView
objectId CDATA #REQUIRED
timestamp CDATA #REQUIRED>
<!ELEMENT ClassificationNodeQueryResult (ClassificationNodeView*)>
<!ELEMENT ClassificationNodeView EMPTY>
<!ATTLIST ClassificationNodeView
objectURN CDATA #REQUIRED
contentURI CDATA #IMPLIED
objectId CDATA #IMPLIED>
<!ELEMENT RegistryPackageQueryResult (RegistryPackageView*)>
<!ELEMENT RegistryPackageView EMPTY>

```

```

<!--ATTLIST RegistryPackageView
objectURN CDATA #REQUIRED
contentURI CDATA #IMPLIED
objectId CDATA #IMPLIED>
<!--ELEMENT OrganizationQueryResult (OrganizationView*)>
<!--ELEMENT OrganizationView EMPTY>
<!--ATTLIST OrganizationView
orgURN CDATA #REQUIRED
objectId CDATA #IMPLIED>
<!--ELEMENT RegistryEntryQuery(RegistryEntryFilter?,
SourceAssociationBranch*, TargetAssociationBranch*,
HasClassificationBranch*, SubmittingOrganizationBranch?,
ResponsibleOrganizationBranch?, ExternalIdentifierFilter*,
ExternalLinkFilter*, SlotFilter*, HasAuditableEventBranch*, CPPHasId*,
CPPHasPackageId*, PartyInfoHasCPP*, PartyInfoHasCollaborationRole*,
CollaborationRoleHasRole*, CollaborationRoleHasProcessSpec*,
DeliveryChannelHasChannelId*, DeliveryChannelHasTransportId*,
DeliveryChannelHasDocExchangeId*, ServiceBindingHasChannelId*,
ServiceBindingHasPackageId*, TransportSecurityHasTransportId*,
SimplePackagePartHasPackageId*, ConstituentHasCompositeId*,
CompositeHasPackageId*, DAMLOIL_HasSubClassId*,
DAMLOIL_Restriction_Has_ClassId*, DAMLOIL_Get_SubclassId*,
DAMLOIL_SubClass_GetId*, DAMLOIL_SubProperty_GetId* , CPPFilter?,
PartyInfoFilter?, CollaborationRoleFilter?, RoleFilter?,
ProcessSpecFilter?, DeliveryChannelFilter?, ServiceBindingFilter?,
TransportFilter?, TransportSecurityFilter?, SimplePackagePartFilter?,
ConstituentFilter?, CompositeFilter?, DAMLOIL_Subclass_Filter?,
DAMLOIL_Restriction_Filter?, DAMLOIL_SubProperty_Filter?,
DAML_INSTANCE_Product_Filter ?, DAML_INSTANCE_Hotel_Reservation_Filter?,
DAML_INSTANCE_Physical_Product_Filter?,
DAML_INSTANCE_Physical_Product_Filter?, DAML_INSTANCE_PoecService_Filter?,
DAML_INSTANCE_Service_Category_Filter?,
DAML_INSTANCE_Receipt_Filter?,
DAML_INSTANCE_Rent_Veichle_Filter?,
DAML_INSTANCE_Car_Rentral_Service_Filter?,
DAML_INSTANCE_Limosine_Rentral_Service_Filter?,
DAML_INSTANCE_Chauffeur_Service_Filter?)>

<!--ELEMENT SourceAssociationBranch (AssociationFilter?,
RegistryEntryFilter?)>
<!--ELEMENT TargetAssociationBranch (AssociationFilter?,
RegistryEntryFilter?)>
<!--ELEMENT HasClassificationBranch (ClassificationFilter?,
ClassificationNodeFilter?)>
<!--ELEMENT SubmittingOrganizationBranch (OrganizationFilter?,
ContactFilter?)>
<!--ELEMENT ResponsibleOrganizationBranch (OrganizationFilter?,
ContactFilter?)>

```

```

<!ELEMENT HasAuditableEventBranch (AuditableEventFilter?, UserFilter?,
OrganizationFilter?)>
<!ELEMENT AuditableEventQuery (AuditableEventFilter?,
RegistryEntryQuery*,
InvokedByBranch? )>
<!ELEMENT InvokedByBranch ( UserFilter?, OrganizationQuery? )>
<!ELEMENT ClassificationNodeQuery (ClassificationNodeFilter?,
PermitsClassificationBranch*,HasParentNode?, HasSubnode* )>

<!ELEMENT PermitsClassificationBranch (ClassificationFilter?,
RegistryEntryQuery?)>
<!ELEMENT HasParentNode (ClassificationNodeFilter?, HasParentNode?)>
<!ELEMENT HasSubnode (ClassificationNodeFilter?, HasSubnode*)>
<!ELEMENT RegistryPackageQuery (PackageFilter?, HasMemberBranch*)>
<!ELEMENT HasMemberBranch (RegistryEntryQuery?)>
<!ELEMENT CPPHasId(RegistryEntryQuery?)>
<!ELEMENT CPPHasPackageId(RegistryEntryQuery?)>
<!ELEMENT PartyInfoHasCPP(RegistryEntryQuery?)>
<!ELEMENT PartyInfoHasCollaborationRole(RegistryEntryQuery?)>
<!ELEMENT CollaborationRoleHasRole(RegistryEntryQuery?)>
<!ELEMENT CollaborationRoleHasProcessSpec(RegistryEntryQuery?)>
<!ELEMENT DeliveryChannelHasChannelId(RegistryEntryQuery?)>
<!ELEMENT DeliveryChannelHasTransportId(RegistryEntryQuery?)>
<!ELEMENT DeliveryChannelHasDocExchangeId(RegistryEntryQuery?)>
<!ELEMENT ServiceBindingHasChannelId(RegistryEntryQuery?)>
<!ELEMENT ServiceBindingHasPackageId(RegistryEntryQuery?)>
<!ELEMENT TransportSecurityHasTransportId(RegistryEntryQuery?)>
<!ELEMENT SimplePackagePartHasPackageId(RegistryEntryQuery?)>
<!ELEMENT ConstituentHasCompositeId(RegistryEntryQuery?)>
<!ELEMENT CompositeHasPackageId(RegistryEntryQuery?)>
<!ELEMENT DAMLOIL_HasSubClassId(RegistryEntryQuery?)>
<!ELEMENT DAMLOIL_Restriction_Has_ClassId(RegistryEntryQuery?)>
<!ELEMENT DAMLOIL_Get_SubclassId(RegistryEntryQuery?)>
<!ELEMENT DAMLOIL_SubClass_GetId(RegistryEntryQuery?)>
<!ELEMENT DAMLOIL_SubProperty_GetId(RegistryEntryQuery?)>
<!ELEMENT OrganizationQuery (OrganizationFilter?, SubmitsRegistryEntry*,
HasParentOrganization?,InvokesEventBranch*, ContactFilter*)>
<!ELEMENT SubmitsRegistryEntry (RegistryEntryQuery?)>
<!ELEMENT HasParentOrganization (OrganizationFilter?,
HasParentOrganization?)>
<!ELEMENT InvokesEventBranch (UserFilter?, AuditableEventFilter?,
RegistryEntryQuery?)>
<!ELEMENT ReturnRegistryEntry (RegistryEntryQuery, WithClassifications?,
WithSourceAssociations?,
WithTargetAssociations?, WithAuditableEvents?, WithExternalLinks?)>
<!ELEMENT WithClassifications (ClassificationFilter?)>
<!ELEMENT WithSourceAssociations (AssociationFilter?)>
<!ELEMENT WithTargetAssociations (AssociationFilter?)>

```

```

<!--ELEMENT WithAuditableEvents (AuditableEventFilter?)>
<!--ELEMENT WithExternalLinks (ExternalLinkFilter?)>
<!--ELEMENT ReturnRegistryEntryResult (RegistryEntryMetadata*)>
<!--ELEMENT RegistryEntryMetadata (RegistryEntry, Classification*,
SourceAssociations?,
TargetAssociations?, AuditableEvent*, ExternalLink*)>
<!--ELEMENT SourceAssociations (Association*)>
<!--ELEMENT TargetAssociations (Association*)>
<!--ELEMENT ReturnRepositoryItem (RegistryEntryQuery,
RecursiveAssociationOption?, WithDescription?)>
<!--ELEMENT CPPQuery (RegistryEntryQuery, WithCollaborationRole? ,
WithDeliveryChannel?, WithTransport?, WithPackaging?)>
<!--ELEMENT ReturnCompatibleCPP(RegistryEntryQuery)>
<!--ELEMENT DAMLOILQueryRequest(RegistryEntryQuery)>
<!--ELEMENT RecursiveAssociationOption (AssociationType+)>
<!--ATTLIST RecursiveAssociationOption
depthLimit CDATA #IMPLIED>
<!--ELEMENT AssociationType EMPTY>
<!--ATTLIST AssociationType
role CDATA #REQUIRED>
<!--ELEMENT WithDescription EMPTY>
<!--ELEMENT ReturnRepositoryItemResult (RepositoryItem*)>
<!--ELEMENT CPPMatchingQueryResult>
<!--ATTLIST CPPMatchingQuery
OriginalCPPIId CDATA #REQUIRED
CompatibleCPPIId CDATA #REQUIRED      >
<!--ELEMENT ReturnCPPQueryResult(RepositoryItem*)>
<!--ELEMENT RepositoryItem (RegistryPackage | ExtrinsicObject |
WithdrawnObject | ExternalLink| CPP)>
<!--ELEMENT CPP (CollaborationRole | SimplePart | CompositeList |
Composite | Constituent |ProcessSpec | Role | ServiceBinding |
Transport | TransportSecurity)>
<!--ATTLIST RepositoryItem
identifier CDATA #REQUIRED
name CDATA #REQUIRED
contentURI CDATA #REQUIRED
objectType CDATA #REQUIRED
status CDATA #REQUIRED
stability CDATA #REQUIRED
description CDATA #IMPLIED>
<!--ELEMENT RegistryPackage EMPTY>
<!--ELEMENT WithdrawnObject EMPTY>
<!--ELEMENT ExternalLinkItem EMPTY>
<!--ELEMENT ObjectFilter (Clause)>
<!--ELEMENT RegistryEntryFilter (Clause)>
<!--ELEMENT IntrinsicObjectFilter (Clause)>
<!--ELEMENT ExtrinsicObjectFilter (Clause)>
<!--ELEMENT PackageFilter (Clause)>

```

```

<!--ELEMENT OrganizationFilter (Clause)>
<!--ELEMENT ContactFilter (Clause)>
<!--ELEMENT ClassificationNodeFilter (Clause)>
<!--ELEMENT AssociationFilter (Clause)>
<!--ELEMENT ClassificationFilter (Clause)>
<!--ELEMENT ExternalLinkFilter (Clause)>
<!--ELEMENT SlotFilter (Clause)>
<!--ELEMENT ExternalIdentifierFilter (Clause)>
<!--ELEMENT AuditableEventFilter (Clause)>
<!--ELEMENT UserFilter (Clause)>
<!--ELEMENT CPPFilter(Claue)>
<!--ELEMENT PartyInfoFilter(Claue)>
<!--ELEMENT CollaborationRoleFilter(Claue)>
<!--ELEMENT RoleFilter(Claue)>
<!--ELEMENT ProcessSpecFilter(Claue)>
<!--ELEMENT DeliveryChannelFilter(Claue)>
<!--ELEMENT ServiceBindingFilter(Claue)>
<!--ELEMENT TransportFilter(Claue)>
<!--ELEMENT TransportSecurityFilter(Claue)>
<!--ELEMENT SimplePackagePartFilter(Claue)>
<!--ELEMENT ConstituentFilter(Claue)>
<!--ELEMENT CompositeFilter(Claue)>
<!--ELEMENT DAMLOIL_Subclass_Filter(Claue)>
<!--ELEMENT DAMLOIL_Restriction_Filter(Claue)>
<!--ELEMENT DAMLOIL_SubProperty_Filter(Claue)>
<!--ELEMENT DAML_INSTANCE_Hotel_Reservation_Filter(Claue)>
<!--ELEMENT DAML_INSTANCE_Product_Filter(Claue)>
<!--ELEMENT DAML_INSTANCE_Physical_Product_Filter(Claue)>
<!--ELEMENT DAML_INSTANCE_PoecService_Filter(Claue)>
<!--ELEMENT DAML_INSTANCE_Service_Category_Filter(Claue)>
<!--ELEMENT DAML_INSTANCE_Receipt_Filter(Claue)>
<!--ELEMENT DAML_INSTANCE_Rent_Veichle_Filter(Claue)>
<!--ELEMENT DAML_INSTANCE_Car_Rentral_Service_Filter(Claue)>
<!--ELEMENT DAML_INSTANCE_Limosine_Rentral_Service_Filter(Claue)>
<!--ELEMENT DAML_INSTANCE_Chauffeur_Service_Filter(Claue)>
<!--The following lines define the XML syntax for Clause.-->
<!--ELEMENT Clause (SimpleClause | CompoundClause)>
<!--ELEMENT SimpleClause (BooleanClause | RationalClause |
StringClause)>
<!--ATTLIST SimpleClause leftArgument CDATA #REQUIRED>
<!--ELEMENT CompoundClause (Clause, Clause+)>
<!--ATTLIST CompoundClause connectivePredicate
(And | Or) #REQUIRED>
<!--ELEMENT BooleanClause EMPTY>
<!--ATTLIST BooleanClause booleanPredicate (true|false)
#REQUIRED>
<!--ELEMENT RationalClause (IntClause | FloatClause)>
<!--ATTLIST RationalClause logicalPredicate (LE | LT

```

```

| GE | GT | EQ | NE) #REQUIRED>
<!--ELEMENT IntClause (#PCDATA)>
<!--ATTLIST IntClause e-dtype NMTOKEN #FIXED "int">
<!--ELEMENT StringClause (#PCDATA)>
<!--ATTLIST StringClause
stringPredicate (contains | -contains | startswith |
-startswith | equal | -equal |
endswith | -endswith) #REQUIRED>

<!-- End FilterQuery DTD -->
<!-- Begin RegistryError definition -->
<!-- The RegistryErrorList is derived from the ErrorList element
from the ebXML Message Service Specification -->
<!--ELEMENT RegistryErrorList (RegistryError+ )>
<!--ATTLIST RegistryErrorList highestSeverity ( Warning |
Error ) 'Warning' >
<!--ELEMENT RegistryError (#PCDATA) >
<!--ATTLIST RegistryError
codeContext CDATA #REQUIRED
errorCode CDATA #REQUIRED
severity ( Warning | Error ) 'Warning'
location CDATA #IMPLIED
xml:lang NMTOKEN #IMPLIED>
<!--ELEMENT RegistryResponse (( AdhocQueryResponse |
GetContentResponse | GetClassificationTreeResponse |
GetClassifiedObjectsResponse | GetRootClassificationNodesResponse )?,
RegistryErrorList? )>
<!--ATTLIST RegistryResponse status (success | failure )
#REQUIRED >

<!-- The contrived root node -->
<!--ELEMENT RootElement ( SubmitObjectsRequest |
ApproveObjectsRequest |
DeprecateObjectsRequest | RemoveObjectsRequest |
GetRootClassificationNodesRequest |GetClassificationTreeRequest
| GetClassifiedObjectsRequest | AdhocQueryRequest |
GetContentRequest | AddSlotsRequest |
RemoveSlotsRequest | RegistryResponse | RegistryProfile) >

<!--ELEMENT Href (#PCDATA )>
<!--ELEMENT XMLDocumentErrorLocn (DocumentId , path )>
<!--ELEMENT DocumentId (#PCDATA )>
<!--ELEMENT Xpath (#PCDATA)

```

## APPENDIX C

### AUTOMATIC COMPATIBLE CPP SEARCH SQL QUERY

```
Select CPP1.CPPid, CPP2.CPPid from CPP CPP1, CPP CPP2 where
CPP1.packageId IN (
    SELECT C1.packageId from Composite C1, Composite C2
    Where
        C2.packageId=CPP2.packageId
    AND
        C1.mimeType = C2.mimetype
    AND
        NOT EXISTS
        (
            SELECT order_no from Constituent
            Where
                compositeId = C2.id
            AND order_no not in (
                SELECT order_no from Constituent
                Where
                    compositeId = C1.id
            )
        )
    AND
        NOT EXISTS
        (
            SELECT order_no from Constituent
            Where
                compositeId = C1.id
            AND order_no not in (
                SELECT order_no from Constituent
```

```

        Where
            compositeId = C2.id
        )
    )
    AND
    C1.packageId In
    (
        SELECT SPP1.packageId from SimplePackagePart SPP1,
            SimplePackagePart SPP2
        Where
            SPP2.packageId=C2.packageId
            AND
            SPP1.mimeType in(
                SELECT mimeType from SimplePackagePart
            Where
                id = SPP2.id
        )
    )
    AND
    SPP1.packageId In
    (
        SELECT S1.packageId from ServiceBinding S1, ServiceBinding S2
        Where
            S2.packageId=SPP2.packageId
        AND
        S2.channelId In (
            SELECT D2.channelId from DeliveryChannel D1, DeliveryChannel D2
            Where
                S1.channelId = D1.ChannelId AND
                D2.transportId in (
                    SELECT T2.id from Transport T1, Transport T2
                    Where
                        T1.senderProtocol = T2.receiverProtocol AND
                        T1.receiverProtocol = T2.senderProtocol AND
                        T1.senderProtocolVer = T2.receiverProtocolVer AND
                        T1.receiverProtocolVer = T2.senderProtocolVer AND
                        T1.id = D1.transportId
                )
            )
        AND
        D2.transportId in (
            SELECT TS2.transportId from TransportSecurity TS1,
                TransportSecurity TS2
            Where
                TS1.protocolVersion = TS2.protocolVersion AND
                TS1.transportId = D1.transportId
        )
    )
    AND
    S1.channelId in (

```

```

SELECT SB1.ChannelId from ServiceBinding SB1,ServiceBinding SB2
Where
    SB2.ChannelId =SB2.ChannelId
    AND
    SB1.collaborationRoleId in (
        SELECT C1.id from CollaborationRole C1,
            CollaborationRole C2
        Where
            C2.id=SB2.collaborationRoleId
            AND
            C2.processSpecId In (
                SELECT P2.id from ProcessSpec P1 , ProcessSpec P2
                Where
                    P1.name = P2.name AND
                    P1.id = C1.processSpecId AND
                    P2.id = C2.processSpecId
            )
            AND
            C2.roleId In (
                SELECT R2.id from Role R1, Role R2
                Where
                    R1.name <> R2.name AND
                    R1.id = C1.roleId AND
                    R2.id = C2.roleId
            )
            AND
            C1.id IN (
                SELECT id from CollaborationRole
                Where
                    partyInfoId in (
                        SELECT id from PartyInfo
                        Where CPPId = 'XXXX'
                    )
            )
        )
    )
    )
    )
    );

```

## APPENDIX D

# DAML+OIL BUILT SERVICE TAXONOMY ONTOLOGY

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<!DOCTYPE uridef[
  <!ENTITY rdf      "http://www.w3.org/1999/02/22-rdf-syntax-ns">
  <!ENTITY rdfs     "http://www.w3.org/2000/01/rdf-schema">
  <!ENTITY xsd      "http://www.w3.org/2000/10/XMLSchema">
  <!ENTITY daml     "http://www.daml.org/2001/03/daml+oil">
  <!ENTITY dex      "http://www.daml.org/2001/03/daml+oil-ex">
  <!ENTITY exd      "http://www.daml.org/2001/03/daml+oil-ex-dt">
  <!ENTITY service  "http://www.daml.org/services/daml-s/2001/05/Service.daml">
  <!ENTITY profile  "http://www.daml.org/services/daml-s/2001/05/Profile.daml">
  <!ENTITY POEC     "http://www.srdc.metu.edu.tr/poec/common.schema.rdf#">]]>
<rdf:RDF xmlns:rdf="&rdf;#" xmlns:rdfs="&rdfs;#" xmlns:xsd="&xsd;#" xmlns:daml
="&daml;#" xmlns:dex = &dex;#" xmlns:exd = "&exd;#" xmlns:service = "&service;#"
xmlns:profile = "&profile;#" xmlns:POEC = "&poec;#" >

  <daml:Ontology rdf:about="">
    <rdfs:comment> A Hotel Reservation Service Ontology example
    </rdfs:comment>
    <daml:imports rdf:resource="&daml;"/>
    <daml:imports rdf:resource="&service;"/>
    <daml:imports rdf:resource="&profile;"/>
    <daml:imports rdf:resource="&poec;"/> </daml:Ontology>
    <rdfs:Class rdf:ID="Hotel_Reservation_Service">
      <rdfs:subClassOf resource="#poecService"/>
      <rdfs:subClassOf> <daml:Restriction>
        <daml:onProperty rdf:resource="&profile;#serviceType"/>
        <daml:hasValue rdf:resource="&poec;#Generic"/>
      </daml:Restriction>
    </rdfs:Class>
  </rdf:RDF>

```

```

        </daml:Restriction> </rdfs:subClassOf>
    <rdfs:subClassOf> <daml:Restriction>
        <daml:onProperty rdf:resource="#poec;#Added_Value"/>
        <daml:toClass rdf:resource="#Car_Rental_Service"/>
    </daml:Restriction> </rdfs:subClassOf>
</rdfs:Class>
<rdf:Description ID="Rent_Vehicle_Service">
    <rdf:type resource="http://www.w3.org/TR/WD-rdf-schema#Class"/>
    <rdfs:subClassOf resource="#poecService"/>
    <rdfs:subClassOf> <daml:Restriction>
        <daml:onProperty rdf:resource="#profile;#serviceType"/>
        <daml:hasValue rdf:resource="#poec;#Generic"/>
    </daml:Restriction> </rdfs:subClassOf>
    <rdfs:subClassOf> <daml:Restriction>
        <daml:onProperty rdf:resource="#poec;#Added_Value"/>
        <daml:toClass rdf:resource="#Chauffeur_Service"/>
    </daml:Restriction> </rdfs:subClassOf>
    <rdfs:comment>All kinds of vehicle rental services</rdfs:comment>
    <rdfs:label>RentVehicleService</rdfs:label>
</rdf:Description>
<rdf:Description ID="Car_Rental_Service">
    <rdf:type resource="http://www.w3.org/TR/WD-rdf-schema#Class"/>
    <rdfs:subClassOf resource="#Rent_Vehicle_Service"/>
    <rdfs:subClassOf> <daml:Restriction>
        <daml:onProperty rdf:resource="#profile;#serviceType"/>
        <daml:hasValue rdf:resource="#poec;#Generic"/>
    </daml:Restriction> </rdfs:subClassOf>
    <rdfs:subClassOf> <daml:Restriction>
        <daml:onProperty rdf:resource="#poec;#Added_Value"/>
        <daml:toClass rdf:resource="#Car_Delivery_Service"/>
    </daml:Restriction> </rdfs:subClassOf>
    <rdfs:subClassOf> <daml:Restriction>
        <daml:onProperty rdf:resource="#poec;#AddOn_To"/>
        <daml:toClass rdf:resource="#Hotel_Reservation_Service"/>
    </daml:Restriction> </rdfs:subClassOf>
    <rdfs:subClassOf> <daml:Restriction>
        <daml:onProperty rdf:resource="#poec;#helperService"/>
        <daml:toClass rdf:resource="#Car_Selection_Service"/>
    </daml:Restriction> </rdfs:subClassOf>
    <rdfs:subClassOf> <daml:Restriction>
        <daml:onProperty rdf:resource="#poec;#helperService"/>
        <daml:toClass rdf:resource="#Package_Selection_Service"/>
    </daml:Restriction> </rdfs:subClassOf>
    <rdfs:comment>All kinds of car rental services</rdfs:comment>
    <rdfs:label>CarRentalService</rdfs:label>
</rdf:Description>
</rdf:RDF>

```

```

<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE uridef[
  <!ENTITY rdf      "http://www.w3.org/1999/02/22-rdf-syntax-ns">
  <!ENTITY rdfs     "http://www.w3.org/2000/01/rdf-schema">
  <!ENTITY xsd      "http://www.w3.org/2000/10/XMLSchema">
  <!ENTITY daml     "http://www.daml.org/2001/03/daml+oil">
  <!ENTITY dex      "http://www.daml.org/2001/03/daml+oil-ex">
  <!ENTITY exd      "http://www.daml.org/2001/03/daml+oil-ex-dt">
  <!ENTITY service  "http://www.daml.org/services/daml-s/2001/05/Service.daml">
  <!ENTITY profile  "http://www.daml.org/services/daml-s/2001/05/Profile.daml">
  <!ENTITY POEC     "http://www.srdc.metu.edu.tr/poec/common.schema.rdf#">]]>
<rdf:RDF
  xmlns:rdf =      "&rdf;#"
  xmlns:rdfs =     "&rdfs;#"
  xmlns:xsd =      "&xsd;#"
  xmlns:daml =     "&daml;#"
  xmlns:dex =      "&dex;#"
  xmlns:exd =      "&exd;#"
  xmlns:service =  "&service;#"
  xmlns:profile =  "&profile;#"
  xmlns:POEC =     "&poec;#" >
  <daml:Ontology rdf:about="">
  <rdfs:comment> An implementation of Car\_Rental\_Service:
    My carrentalervice
  </rdfs:comment>
  <daml:imports rdf:resource="&daml;"/>
  <daml:imports rdf:resource="&service;"/>
  <daml:imports rdf:resource="&profile;"/>
  <daml:imports rdf:resource="&poec;"/>
  </daml:Ontology>
  <rdf:Description ID="My_Car_Rental_Service">
    <poec:service_Type="Implementation"/>
    <rdf:type resource="http://www.srdc.metu.edu.tr/poec/common.schema.
      rdf#Car_Rental_Service"/>
    <poec:Added_Value resource="http://www.srdc.metu.edu.tr/poec/common.
      schema.rdf#Limosin_Chauffeurs_Service"/>
    <poec:Added_Value resource="http://www.srdc.metu.edu.tr/poec/hire.a.
      chauffeur.service.rdf#Hire_A_Chauffeur"/>
    <poec:service_Catalog resource="http://www.serviceprovider.com/cars.
      catalog.xml"/>
    <rdfs:label>MyCarRentalService</rdfs:label>
    <rdfs:comment>An implementation of the Car_Rental_Service</rdfs:comment>
  </rdf:Description>
</rdf:RDF>

```

## APPENDIX E

### CPA CONSTRUCTION ALGORITHM

Inputs to CPA formation:

- 1) User preference about intended Binary Collaboration(s) BussColl
- 2) Our CPP, CPP1
- 3) Our collaborators CPP, CPP2

Tasks in CPA Formation Process

- 1) Matching Roles
- 2) Matching Transport
  - a. Matching Transport Security
- 3) Matching Packaging
  - a. Matching Document Level Security

For each BinaryCollaboration BussColl that is given as input the following 3 tasks are performed.

Task 1 Matching Roles (Binary Collaboration Case Considered)

1.1 For each CollaborationRole element in CPP1

1.1.1 For each CollaborationRole element in CPP2

- 3.1. If value of name attribute of ProcessSpecification element in CPP1= BussColl and value of name attribute of ProcessSpecification element in CPP2= BussColl and value of name attribute of Role element in CPP1 <>value of name attribute of Role element in CPP2  
//We found a match in Roles, and the correct Collaboration Role elements in CPP1 and CPP2

3.2. Mark matching CollaborationRole elements

4. If no match in roles is found exit CPA creation.

## Task 2 Matching Transport

1. For each ServiceBinding within the marked CollaborationRole element in CPP1
  - 1.1 Retrieve value of channelID attribute say (channelxx)
  - 1.2 Find corresponding DeliveryChannel element with channelID attribute with value = channelxx, Retrieve value of transportID attribute say (transportxx), Retrieve value of docExchangeID attribute say (docExchangexx)
  - 1.3 Find corresponding Transport element with transportID attribute with value = transportxx,
  - 1.4 For each ServiceBinding within the marked CollaborationRole element in CPP2
    - 1.4.1 Retrieve value of channelID attribute say (channelyy)
    - 1.4.2 Find corresponding DeliveryChannel element with channelID attribute with value = channelxx, Retrieve value of transportID attribute say (transportyy), Retrieve value of docExchangeID attribute say (docExchangeyy)
    - 1.4.3 Find corresponding Transport element with transportID attribute with value = transportyy,
    - 1.4.4 For each SendingProtocol subelement of Transport element in CPP1
      - 1.4.4.1 If value of SendingProtocol subelement of Transport element in CPP1 equals ReceivingProtocol subelement of Transport element in CPP2  
//We found a match in our Sending Protocol and our Collaborators Receiving Protocol
    - 1.4.5 If no match between our Sending Protocol and our Collaborators Receiving Protocol exist then , exit CPA creation.  
else continue
    - 1.4.6 For each SendingProtocol subelement of Transport element in CPP2
      - 1.4.6.1 If value of SendingProtocol subelement of Transport element in CPP2 equals ReceivingProtocol subelement of Transport element in CPP1  
//We found a match in our Receiving Protocol and our Collaborators Sending Protocol
    - 1.4.7 If no match between our Receiving Protocol and our Collaborators Sending Protocol exist , then exit CPA creation.
  - else Mark the corresponding ServiceBinding elements, and Transport in both CPPs  
//We found a match in transport,
2. If no match in transport is found exit CPA creation.
3. Within marked Transport elements in CPP1 and CPP2 retrieve TransportSecurity subelements.
4. Within TransportSecurity elements retrieve values of Protocol subelements in both CPPs. Retrieve values of version attributes of Protocol subelements in both CPPs.
5. Lookup SecurityCompatibilityTable for the version and protocol values of both parties.
6. If parties are compatible then  
mark matching Transport, DocExchange, DeliveryChannel elements with id attributes channelxx, transportxx, docExchangexx, and channelyy,

transportyy, docExchangeyy, for CPP1 and CPP2 respectively,  
continue with TASK 3.  
else exit CPA creation.

#### Task 3 Matching Packaging

1. For the marked ServiceBinding elements in both CPPs, retrieve value of packageID attribute say (packagexx) in CPP1, retrieve value of packageID attribute say (packageyy) in CPP2
2. Find corresponding Packaging element with ID attribute with value= packagexx,
3. Find corresponding Packaging element with ID attribute with value = packageyy,
4. For each SimplePart subelement within the marked Packaging element in CPP1 retrieve value of mimetype attribute say (mimexx) in CPP1
  - 4.1. If there exist a SimplePart subelement within the marked Packaging element in CPP2 with attribute mimetype =mimexx then continue.  
else exit CPA creation
5. Structurally compare two CompositeList subtrees
6. If structurally equal then mark Packaging elements in both CPPs  
else exit CPA creation

#### Task 4 Output Generation

Output marked Collaboration Role, Transport, Delivery Channel, DocExchange elements, and common Packaging element into CPA document

## APPENDIX F

### ebXML REGISTRY CPA

```
<?xml version="1.0" encoding="UTF-8"?>
<CollaborationProtocolAgreement cpaid="54145" xmlns="http://www.ebxml.org/namespaces
/tradePartner" version="1.0">
  <Status value="proposed"/>
  <Start>2001-05-15T17:08:03.062</Start>
  <End>2032-09-27T09:16:06.124</End>
  <ConversationConstraints concurrentConversations="100" invocationLimit="10000"/>
  <PartyInfo>
    <PartyId type="uriReference">urn:www.srdc.com</PartyId>
    <PartyRef href="" type="simple"/>
    <CollaborationRole id="CR1">
      <ProcessSpecification href="Query"
        name="BinaryCollaborationOne" type="simple" version="1.0"/>
      <Role xlink:href="Query#client" name="client" xlink:type="simple"/>
      <ServiceBinding channelId="C1" packageId="P1">
        <Service type="b2bserver">localhost:1881:Query:BinaryCollaboration
          One#client</Service>
        <Override action=" " channelId="C1" packageId="P1" type="simple"/>
      </ServiceBinding>
    </CollaborationRole>
    <DeliveryChannel channelId="C1" docExchangeId="DE1" transportId="T1">
      <Characteristics authenticated="false" authorized="false"
        confidentiality="false" nonrepudiationOfOrigin="false"
        nonrepudiationOfReceipt="false" secureTransport="false"
        syncReplyMode="none"/>
    </DeliveryChannel>
    <Transport transportId="T1">
      <SendingProtocol version="1.0">HTTP</SendingProtocol>
      <ReceivingProtocol version="1.0">HTTP</ReceivingProtocol>
      <Endpoint type="allPurpose" uri="http://localhost:8080/registry/client
```

```

/servlet/srdc.ebxml.messaging.provider.servlets.EbXMLMessageRouterServlet"/>
</Transport>
<DocExchange docExchangeId="DE1">
  <ebXMLBinding version="0.99">
    <ReliableMessaging deliverySemantics="BestEffort"
      idempotency="false" messageOrderSemantics="NotGuaranteed">
      <Retries>0</Retries>
      <RetryInterval>0</RetryInterval>
      <PersistDuration>P</PersistDuration>
    </ReliableMessaging>
  </ebXMLBinding>
</DocExchange>
</PartyInfo>
<PartyInfo>
  <PartyId type="uriReference">urn:www.srdc2.com</PartyId>
  <PartyRef href="" type="simple"/>
  <CollaborationRole id="CR1">
    <ProcessSpecification href="Query"
      name="BinaryCollaborationOne" type="simple" version="1.0"/>
    <Role xlink:href="Query#registry" name="registry" xlink:type="simple"/>
    <ServiceBinding channelId="C1" packageId="P1">
      <Service type="b2bserver">localhost:1919:Query:BinaryCollaborationOne
#registry</Service>
      <Override action=" " channelId="C1" packageId="P1" type="simple"/>
    </ServiceBinding>
  </CollaborationRole>
  <DeliveryChannel channelId="C1" docExchangeId="DE1" transportId="T1">
    <Characteristics authenticated="false" authorized="false"
      confidentiality="false" nonrepudiationOfOrigin="false"
      nonrepudiationOfReceipt="false" secureTransport="false"
      syncReplyMode="none"/>
  </DeliveryChannel>
  <Transport transportId="T1">
    <SendingProtocol version="1.0">HTTP</SendingProtocol>
    <ReceivingProtocol version="1.0">HTTP</ReceivingProtocol>
    <Endpoint type="allPurpose" uri="http://localhost:8080/registry/registry/
servlet/srdc.ebxml.messaging.provider.servlets.EbXMLMessageRouterServlet"/>
  </Transport>
  <DocExchange docExchangeId="DE1">
    <ebXMLBinding version="0.99">
      <ReliableMessaging deliverySemantics="BestEffort"
        idempotency="false" messageOrderSemantics="NotGuaranteed">
        <Retries>0</Retries>
        <RetryInterval>0</RetryInterval>
        <PersistDuration>P</PersistDuration>
      </ReliableMessaging>
    </ebXMLBinding>
  </DocExchange>

```

```
</PartyInfo>
<Packaging id="P1">
  <ProcessingCapabilities generate="true" parse="true"/>
  <SimplePart id="SP1" mimetype="text/xml"/>
</Packaging>
</CollaborationProtocolAgreement>
```

