SEMANTIC INTEROPERABILITY OF THE UN/CEFACT CCTS BASED ELECTRONIC BUSINESS DOCUMENT STANDARDS

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

BY

YILDIRAY KABAK

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
IN
COMPUTER ENGINEERING

JULY 2009

Approval of the thesis:

SEMANTIC INTEROPERABILITY OF THE UN/CEFACT CCTS BASED ELECTRONIC BUSINESS DOCUMENT STANDARDS

submitted by **YILDIRAY KABAK** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy in Computer Engineering , Middle East Technical University** by,

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

Prof. Dr. Müslim Bozyiğit
Head of Department, **Computer Engineering** _____

Prof. Dr. Asuman Doğaç
Supervisor, **Department of Computer Engineering, METU** _____

Examining Committee Members:

Prof. Dr. İsmail Hakkı Toroslu
Department of Computer Engineering, METU _____

Prof. Dr. Asuman Doğaç
Department of Computer Engineering, METU _____

Prof. Dr. Semih Bilgen
Department of Electrical and Electronics Engineering, METU _____

Prof. Dr. Özgür Ulusoy
Department of Computer Engineering, Bilkent University _____

Assoc. Prof. Dr. Nihan Kesim Çiçekli
Department of Computer Engineering, METU _____

Date: _____

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

Name, Last Name: YILDIRAY KABAK

Signature :

ABSTRACT

SEMANTIC INTEROPERABILITY OF THE UN/CEFACT CCTS BASED ELECTRONIC BUSINESS DOCUMENT STANDARDS

Kabak, Yıldıray

M.S., Department of Computer Engineering

Supervisor : Prof. Dr. Asuman Doğaç

July 2009, 156 pages

The interoperability of the electronic documents exchanged in eBusiness applications is an important problem in industry. Currently, this problem is handled by the mapping experts who understand the meaning of every element in the involved document schemas and define the mappings among them which is a very costly and tedious process. In order to improve electronic document interoperability, the UN/CEFACT produced the Core Components Technical Specification (CCTS) which defines a common structure and semantic properties for document artifacts. However, at present, this document content information is available only through text-based search mechanisms and tools. In this thesis, the semantics of CCTS based business document standards is explicated through a formal, machine processable language as an ontology. In this way, it becomes possible to compute a harmonized ontology, which gives the similarities among document schema ontology classes of different document standards through both the semantic properties they share and the semantic equivalences established through reasoning. However, as expected, the harmonized ontology only helps discovering the similarities of structurally and semantically equivalent elements. In order to handle the structurally different but semantically similar document artifacts, heuristic rules are developed

describing the possible ways of organizing simple document artifacts into compound artifacts as defined in the CCTS methodology. Finally, the equivalences discovered among document schema ontologies are used for the semi-automated generation of XSLT definitions for the translation of real-life document instances.

Keywords: Electronic Business Documents, Semantic Interoperability, Ontology, Description Logics

ÖZ

UN/CEFACT CCTS TABANLI ELEKTRONİK İŞ DOKÜMANLARININ ANLAMSAL BİRLİKTE ÇALIŞABİLİRLİĞİ

Kabak, Yıldıray
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi : Prof. Dr. Asuman Doğaç

Temmuz 2009, 156 sayfa

E-İş uygulamalarında kullanılan elektronik dokümanların birlikte çalışbilirliği önemli bir problemdir. Şu anda bu problem, doküman şemalarındaki her bir elemanın anlamını iyi bilen eşleştirme uzmanları tarafından şemalar arasında manuel eşleştirmelerin yapılmasıyla çözülmektedir ve bu süreç oldukça yorucu ve pahalıdır. Elektronik dokümanların birlikte çalışabilirliğini artırmak amacıyla UN/CEFACT organizasyonu, doküman parçacıklarının ortak yapısını ve anlamını tanımlayan, Esas Parçacıklar Teknik Spesifikasyonu'nu yayınladı. Fakat günümüzde bu doküman içerik bilgisi sadece metin tabanlı sorgu mekanizmaları ve araçları ile kullanılabilmektedir. Bu tezde Esas Parçacıklar Teknik Spesifikasyonu tabanlı doküman standartları makine tarafından işlenebilir formal ontolojiler ile tanımlanmaktadır. Bu sayede sahip oldukları anlamsal özellikleri kullanan, muhakeme yardımı ile elde edilmiş, farklı standartların doküman şema ontoloji sınıfları arasındaki ilişkileri veren bağdaştırılmış ontolojiyi hesaplamak mümkündür. Fakat, beklenildiği üzere, bağdaştırılmış ontoloji sayesinde sadece yapısal ve anlamsal olarak eşit olan doküman elemanlarının benzerlikleri bulunabilmektedir. Bu yüzden bu tezde, anlamsal olarak benzer ama yapısal olarak farklı doküman parçacıklarının benzerliklerini bulmak için Esas Parçacıklar Teknik Spesifikasyonun'da tanımlanan ve basit

doküman parçacıklarından bileşik doküman parçacıklarının oluşturulmasının olası yollarını tanımlayan buluşal kurallar geliştirlimiştir. Son olarak, doküman şema ontolojileri arasındaki belirlenen eşitlikler gerçek hayatta kullanılan elektronik dokümanların çevrilmesinde işe yarayan XSLT tanımlarının yarı otomatik olarak üretilmesinde kullanılmaktadır.

Anahtar Kelimeler: Elektronik İş Dokümanları, Anlamsal Birlikte Çalışabilirlik, Ontoloji, Betimleme Mantığı

To my family, especially to our new member Tuna...

ACKNOWLEDGMENTS

First of all, I would like to thank my supervisor and my mentor Prof. Dr. Asuman Doğaç for all her guidance, encouragement and patience. Needless to say, without her, this work would never been possible.

I wish to express a lot of thanks to Prof. Dr. Semih Bilgen and Assoc. Prof. Dr. Nihan Kesim Çiçekli for their valuable suggestions and comments throughout the steering meetings of this study.

Special thanks to SRDC Team (especially to Gökçe) for their support.

I want to express my gratefulness to Hande for her support and friendship.

Finally, I would like to thank to my family for their infinite understanding and I would like to dedicate this thesis to my nephew Tuna.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ACRONYMS

ABIE Aggregate Business Information Entity

ACC Aggregate Core Component

AiAG Automotive Industry Action Group

ANSI American National Standards Institute

ASBIE Aggregate Business Information Entity

ASCC Association Core Component

ATG UN/CEFACT Applied Technology Group

B2B Business-to-Business

BBIE Basic Business Information Entity

BBC Basic Core Component

BIE Business Information Entity

BOD Business Object Document

CBL Common Business Library

CC Core Component

CCL Core Component Library

CCT Core Component Types

CCTS Core Components Technical Specification

CEFACT Centre for Trade Facilitation and Electronic Business

CIDX Chemical Industry Data Exchange

CLR TC OASIS Code List Representation Technical Committee

cXML Commerce XML

EAN European Article Number

ebXML Electronic Business eXtensible Markup Language

ebBP ebXML Business Process

EDI Electronic Data Interchange

EDIFACT Electronic Data Interchange For Administration, Commerce and Transport

EFT Electronic Funds Transfer

EPC Electronic Product Code

GDD Global Data Dictionary

GDSN Global Data Synchronization Network

GDT Global Data Types

GS1 Global Standards One

HL7 Health Level Seven

HTTP HyperText Transfer Protocol

HTTPS Secured HyperText Transfer Protocol

IATA International Air Transport Association

IEC International Electrotechnical Commission

IG Implementation Guides

ISO International Organization for Standardization

ISU Interoperability Service Utility

iSURF iSURF Project: An Interoperability Service Utility for Collaborative Supply Chain Planning across Multiple Domains Supported by RFID Devices

IT Information Technology

ITU International Telecommunication Union

MIME Multipurpose Internet Mail Extensions

NDR Naming and Design Rules

OAGI Open Applications Group, Inc.

OAGIS Open Applications Group Integration Specification

OASIS Organization for the Advancement of Structured Information Standards

OTA Open Travel Alliance

OWL Web Ontology Language

PIDX Petroleum Industry Data Exchange

QDT Qualified Data Types

RFID Radio Frequency Identification

SBDH Standard Business Document Header

SMTP Simple Mail Transfer Protocol

SPARQL RDF Query Language

STAR Standards in Automotive Retail

SWIFT Society for Worldwide Interbank Financial Telecommunication

UBL Universal Business Language

UBP Universal Business Process

UCC Uniform Commercial Code

UDT Unqualified Data Types

UMM UN/CEFACT Modelling Methodology

UN United Nations

UNECE United Nations Economic Commission for Europe

xCBL XML Common Business Library

XML eXtensible Markup Language

XSD XML Schema

XSL Extensible Stylesheet Language

XSLT Extensible Stylesheet Language Transformations

CHAPTER 1

INTRODUCTION

Interoperability is the ability of two or more systems or components to exchange information and to use the information that has been exchanged [31]. In other words, interoperability is said to exist between two applications when one application can accept data (including data in the form of a service request) from the other and perform the task in an appropriate and satisfactory manner (as judged by the user of the receiving system) without the need for extra operator intervention [6].

Business Document interoperability initiatives started in the 1970s before the invention of the Internet. The first standard developed was the Electronic Data Interchange (EDI) framework [16], where document exchange was realised through dialup connections using proprietary networks.

Starting with the late 1990s *eXtensible Markup Language* [96] became popular for describing data exchanged on the Internet. The relative human readability and the amount of XML tools available made XML a popular basis for a number of new document standards such as *Common Business Library* (CBL) [7] and *Commerce XML* [12]. This progress has been evolutionary because the later standards used the EDI experience. For example, CBL became *XML Common Business Library* [95] after including EDI experience in CBL.

EDI, CBL and xCBL are horizontal industry standards addressing several industry domains. There are also several vertical industry specific standard initiatives such as the ones from the *North American Automotive Industry Action Group* [2], *Health Level 7 (HL7) Standards Development Organization* [27], the *Petroleum Industry Data Exchange (PIDX) committee* [57], the *Chemical Industry Data Exchange (CIDX) organization* [8], *Open Travel Alliance* [53], and *RosettaNet Consortium* [61] to name but a few.

However, having more than one standard does not solve the interoperability problem, but carries it to an upper level and the problem becomes providing the interoperability of electronic business documents conforming to different standards. Furthermore, the earlier standards have focused on static message/document definitions which were inflexible to adapt to different requirements that arise according to a given context which could be a vertical industry, a country or a specific business process.

The leading effort for defining flexible and adaptable business documents came from the UN/-CEFACT *Core Components Technical Specification* [83] (also known as ISO 15000-5) in the early 2000s. UN/CEFACT CCTS provides a methodology to identify a set of reusable building blocks, called *Core Components* to create electronic documents. *Core Components* represent the common data elements of everyday business documents such as "Address", "Amount", or "Line Item". These reusable building blocks are then assembled into business documents such as "Order" or "Invoice" by using the CCTS methodology. *Core Components* are defined to be context-independent so that they can later be restricted to different contexts such as a specific industry or a country. Many *Core Components* defined by UN/CEFACT are available to users from UN/CEFACT *Core Component Library* [82].

After its publication, the CCTS has gained widespread adoption by the electronic document standard bodies. The *Universal Business Language* (UBL) [75] is the first implementation of the CCTS methodology in XML. Some earlier horizontal standards such as *Global Standard One* (GS1) XML [25] and *Open Applications Group Integration Specification 9.1* (OAGIS) [48], and many vertical industry standards such as CIDX [8] and RosettaNet [61] have also taken up CCTS. However, the existing document standards have well-established document schemas that are already in use and radical schema modifications to conform to CCTS will cause backward incompatibility problem. Therefore they apply the CCTS methodology selectively and more importantly do not always base their document artifacts on the core components defined in the UN/CEFACT *Core Component Library* [82]. As a result there are considerable differences among the CCTS based standards as detailed in [36].

As a natural consequence of these differences, these standards are not interoperable among themselves. When industries using different standards want to exchange data, the mapping experts and data consultants, who understand the semantic meaning of each entity in these standards' document schemas define the mappings among them manually and usually as XSLT

[98] rules. This is a very costly and tedious process. Furthermore, the current accepted practice of storing the document artifacts in spreadsheets does not facilitate to develop automated semantic interoperability support tools.

In order to help with the interoperability of the document artifacts, in this thesis, the CCTS based business document semantics is explicated. By "explicating", it is meant that their semantic properties are defined through a formal, machine processable language as an ontology and the Web Ontology Language (OWL) [54] is used for this purpose. The semantics is explicated at two levels: At the first level, an upper ontology describing the CCTS document content model is specified. The CCTS upper ontology consists of classes representing CCTS document artifacts such as the *Data Types*, the *Core Components* and the *Business Information Entities* and their properties. Furthermore, at this level, the upper ontologies for the prominent CCTS based standards, namely, GS1 XML, OAGIS 9.1 and UBL are also developed. The various equivalence relationships between the classes of the CCTS upper ontology and the CCTS based document standards' upper ontologies are defined. These relationships are later used to find the similarities among the document artifacts from different document schemas.

At the next level, the semantics of the document schemas in each standard are described through "document schema ontologies", which are based on their corresponding upper ontologies. The difference between the "document schema ontology" and the "upper ontology" is that the upper ontology describes the generic entities in a document content model, whereas document schema ontologies describe the actual document artifacts as the subclasses of the classes in the upper ontology.

When these ontologies are harmonized using a Description Logics (DL) reasoner, the automatically computed inferred ontology reveals the implicit equivalences and subsumption relationships between the document artifacts in different standards. In other words, the newly identified ontological relationships show the relations among the semantically similar document artifacts, which are in different standards. It should be noted that currently the common practice to find these relationships between the artifacts of different standards is manual. However, with the approach described in this thesis, these relationships are generated automatically.

The harmonized ontology is effective only to discover equivalence of both semantically and

structurally similar document artifacts. In other words, the use of description logics only allows finding relations between document artifacts, whose semantics and structure are similar. Therefore, for identifying the relations between semantically similar document artifacts, whose structure is different, further heuristics are provided in terms of predicate logic rules. Note that a DL reasoner by itself cannot process predicate logic rules and a well accepted practice of using a rule engine is resorted to execute the predicate logic rules and carry the results back to the DL reasoner. The results involve declaring further class equivalences in the harmonized ontology. Finally, the similarities discovered among the document artifacts are used to automate the mapping process by generating the XSLT rules. In other words, with the generation of the XSLT documents, the semantic equivalences at the conceptual level are converted to syntactical transformation rules, which can be applied directly to the XML business documents exchanged in real life business.

The apporach presented in this thesis is used in IST-213031 iSURF Project [34], where the Interoperability Service Utility (ISU) component of the iSURF Architecture implements my approach and provides translation between Universal Business Language's [75] and GS1 XML's [25] electronic business document schemas for Collaborative Planning, Forecasting and Replenishment.

The thesis is organized as follows: First, the related work and a state-of-the-art survey of electronic document standards are presented in Chapter 2. The aim of this survey is to identify the document components semantics, which is used to find the relations among the similar document components of different document standards. In this survey, some of the prominent horizontal business document standards, namely, EDI, UN/CEFACT CCL, UBL 2.0, OAGIS 9.1 and GS1 XML are inspected in detail to identify the semantics that they give to their components. This survey chapter also describes the differences among the business document standards in order to help with the development of the XSLT documents for cross-standard translation. In the thesis, the semantics is defined through description logics based ontologies and reasoning is used to identify the relations among the document components. Therefore, in this state-of-the-art chapter, brief information on description logics, ontologies and reasoning tools are also provided. In Chapters 3, 4 and 5, the proposed methodology to establish ontology-based semantic interoperability of electronic business document standards are described in detail based on the findings obtained in Chapter 2. Chapter 3 describes the development of the upper and document schema ontologies. In Chapter 4, the heuristics to discover

semantically similar but structurally different document artifacts are described and Chapter 5 addresses how to automate the XSLT generation process for the discovered equivalences among the document artifacts. Another benefit of representing document artifacts through ontologies is that it facilates querying document components. Spreadsheets only allow keyword based queries. However, the ontology representation allows the users to execute more enhanced queries. In this respect, the document component query support is also described in Chapter 5. The overall system architecture, the implementation results, the performance of the system and a use case to generate mappings between GS1 XML Planning documents [26] and UBL Collaborative Planning, Forecasting and Replenishment documents [76] are described in Chapter 6. Finally, Chapter 7 presents the future work and concludes the thesis.

CHAPTER 2

SURVEY OF THE STATE OF THE ART

The survey chapter is composed of three parts. First, Section 2.1 presents the related work in the literature. Then, in the second part, some of the prominent horizontal business document standards, namely, EDI, UN/CEFACT Core Components Library, UBL 2.0, OAGIS BOD 9.1 and GS1 XML are surveyed in order to identify the semantics that they provide for their document components. Section 2.2 summarizes the EDI initiative. Section 2.3 describes the UN/CEFACT Core Component Technical Specification. Section 2.4 introduces the Universal Business Language (UBL) 2.0 standard. In Section 2.5, Open Applications Group Integration Specification (OAGIS) 9.1 is presented. Global Standard One (GS1) XML standard is covered in Section 2.6 after briefly introducing the set of standards proposed by GS1.

In this part of the chapter, in order to identify the structure semantics of document artifacts, the surveyed standards are first analyzed based on their document design principles: the document design principles involve the document artifacts used in composing the documents, the code lists used to convey the meaning of the values in the elements and the use of XML namespaces. Furthermore, since all the document standards surveyed are based on UN/CEFACT CCTS, how this methodology is used in the design of the documents is also discussed.

Then how the standards handle extensibility and customization is discussed. It is an important aspect used in the methodology defined in Chapter 3, because the unmapped elements are inserted into the appropriate extension areas. The standards basically handle the customization and extensibility in two ways: either by introducing an “extension” element into the document schema or by allowing users to change the document schema. When an “extension” element is used, the document schema remains unchanged and the user can put any extra information into this element. When the document schemas are modified to accommodate extensions, the

document interoperability is reduced.

The industry relevance of these standards are also presented by providing some major usage examples in order to show the possible impact of this thesis. Most of the standards covered have very wide industry take up.

Another important issue is whether the standards address the other layers in the interoperability stack, namely the communication layer and the business process layer. The communication layer addresses the transport protocol and the message header. (e.g. after sending a Purchase Order message, an Purchase Order Acknowledgement message must be received), The business process layer involves the sequencing of the messages, and the business processes.

In the second part, Section 2.7 contains an analysis of the presented standards with respect to document design principles, customization and extensibility, coverage of other layers of interoperability and their industry relevance.

Having completed the business document standards survey, in the third part of the chapter, brief information on enabling technologies are presented. In Section 2.8, description logics is described. In Section 2.9, the description logics reasoners are surveyed. After that, in Section 2.10, brief information on ontology and the ontology languages are presented, and Section 2.11 summarizes SPARQL, which is used to discover document components.

Finally, this chapter is concluded by mentioning current harmonization efforts, which are manual.

2.1 RELATED WORK

Given the large number of electronic business document standards, conformance to one of these standards or implementing a combination of them will not solve the interoperability problem; there will always be some companies using a different, incompatible document standard.

Currently, transforming an electronic business document from one standard format into another is generally achieved by means of Extensible Stylesheet Language (XSL) [97] using schema matching techniques as described in [58], and this process is performed manually.

To the best of my knowledge, there is no existing work on automatic transformation among different UN/CEFACT CCTS based document standards. However, in the literature, there is work on the use of description logic techniques in electronic document interoperability. In [101], a Component Ontology specifically for UBL is developed by using the Web Ontology Language (OWL) to represent the semantics of individual components and their relationships within customized schemas. Then this ontology is processed through description logic reasoners for the discovery of similar components and the automation of the translation process among different UBL customizations. In this work, the authors focus on a single document standard, which is UBL. The different customizations, between which the translation is performed, are generated from the same information model. Therefore, the information models of different customizations do not differs much. However, in this work, the most difficult challenge is that the different document standards have different information models. The only commonality among them is the use of UN/CEFACT CCTS.

In [3], Semantic Web technologies are used to transform documents between two vertical industries standards both based on OAGIS: one conforming to *Standards in Automotive Retail* [68] schemas and the other conforming to *Automotive Industry Action Group* [2] schemas. First, the STAR and AiAG XML Schemas are converted to Web Ontology Language [54]. Then these independently developed ontologies are merged through description logic reasoners. By using the merged ontology, the STAR document instances are converted to the corresponding AiAG documents and vice versa. Like, in [101], the authors focus on a single document standard, OAGIS, and both of the document standards, STAR and AiAG, are generated from the same information model.

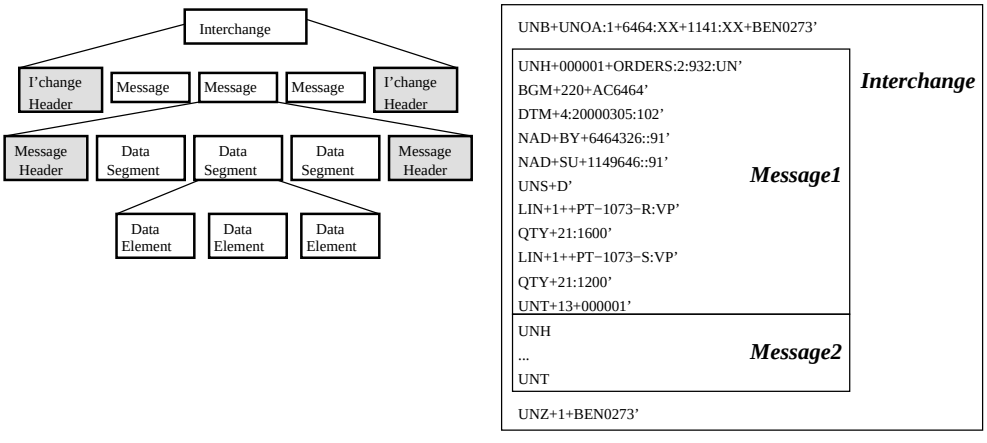
In [102], a supply chain management ontology, called Onto-SCM, is developed which represents a common semantic model of supply chain management. The Onto-SCM is defined using Ontolingua [47], which is an ontology representation language based on Knowledge Interchange Format (KIF) [38]. The authors then show how Onto-SCM can be used for converting document schemas of different standards. In this work, the authors construct both the Onto-SCM ontology itself and the mappings between the document standards to Onto-SCM manually. Considering the large numbers of elements in the document schemas and changes in the versions of the standards make the approach in [102] inapplicable. In the thesis, most of the equivalences are identified automatically.

2.2 ELECTRONIC DATA INTERCHANGE (EDI)

EDI is developed through two main branches: ANSI X12 and UN/EDIFACT. In the USA, the American National Standards Institute (ANSI) developed ANSI X12 [94] and internationally EDI is standardised as UN/EDIFACT (United Nations/Electronic Data Interchange For Administration, Commerce, and Transport) [85]. Through both of these initiatives, a large number of standard electronic documents in plain-text, quote-delimited formats have been specified for domains like procurement, logistics and finance. EDIFACT has also been standardised by the International Standards Organisation as ISO 9735 [85].

Figure 2.1: The Basic EDI Architecture

The basic EDI architecture is shown in Figure 2.1. The communications are through the Value Added Networks (VANs) which are responsible for routing, storing and delivering EDI messages. Special EDI adapters are implemented to interface the internal system of a partner to the value added network. The particulars of the message syntax and interaction process are negotiated between partners in advance. Sometimes a dominant partner imposes its standards on smaller partners.



(a) (b)

Figure 2.2: (a) The Basic EDI Message Structure, (b) An Example EDI Message

An EDI “interchange” document, as shown in Figure 2.2 (a) consists of “messages” which are in turn composed of “data segments”. The segments themselves consist of “data elements”. Figure 2.2 (b) shows an example EDI message.

When the Internet became an established networking environment starting with mid 1990s, there were several updates to the EDI architecture. First, the Internet protocol for email, Simple Mail Transfer Protocol (SMTP), and the File Transfer Protocol (FTP) came to be used to transfer EDI documents directly between parties connected to the Internet. Later, once the World Wide Web and its transfer protocol, the Hyper-Text Transfer Protocol (HTTP), was popularised, this became another mechanism for EDI document transfer.

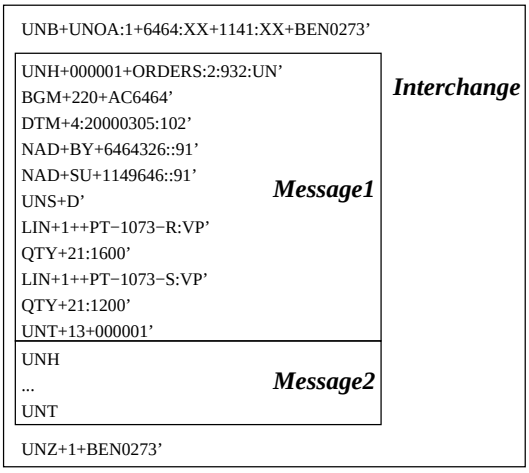


Figure 2.3: The EDI Message Example

2.3 UN/CEFACT CORE COMPONENT TECHNICAL SPECIFICATION (CCTS)

UN/CEFACT Core Components Technical Specification (CCTS) is defined as Part 8 of the ebXML (electronic business XML) Framework and is approved as ISO 15000-5 [83].

The essence of UN/CEFACT CCTS is to design documents from standard, re-usable building blocks, called *Core Components*. The aim is to provide interoperability among electronic business documents by requiring all *Business Information Entities* (BIEs) to be related back to the common *Core Components (CCs)*. A considerable number of *Core Components* are available from the *UN/CEFACT Core Component Library (CCL)* for discovery and reuse, and more will be available as the work progresses.

The first step to provide interoperability based on *Core Components* is to represent values in the components consistently. Hence the starting point for the design of *Core Components* is the *Core Component Types (CCT)* and *Data Types (DT)*, which are also termed as *Core Data Types (CDT)*.

2.3.1 CORE COMPONENT TYPES AND DATA TYPES

Core Component Types (CCT) constitute the leaf-level type space of UN/CEFACT *Core Components*. They specify the basic information types, such as amount, binary object, code and date time, and they are built from primitive data types (e.g. binary, decimal, integer and string). A CCT is composed of a *Content Component*, where the actual primitive content resides, and one or more *Supplementary Components*, which further describe the *Core Component Types*. In other words, *Supplementary Components* help to interpret a value in the *Content Component*.

For example, the "Code" CCT's *Content Component* is of type string and has a set of *Supplementary Components* such as *Code List Agency Identifier* which is the identifier of the Agency that maintains the code list and *Code List Agency Name* which is the name of the Agency that maintains the code list.

On the other hand, *Data Types* are based on one of the *Core Component Types* and further restrict them. In this respect, CCT's can be thought of as abstract types from which more specialized *Data Types* are produced. For example, in the current version of the UN/CEFACT *Data Types*, there is a *Data Type*, called the "CurrencyCode". This data type is based on the "Code" CCT and restricts it as follows:

- *Content Component*: The value in the *Content Component* should be a three-letter code.
- *Code List Identifier*: The identifier of the code list is ISO 4217.
- *Code List Version Identifier*: The version of the code list is 2006-11-21.

The relationship among *Core Component Types*, *Data Types* and other types of core components is shown in Figure 2.4 [83]. Up to now, UN/CEFACT has approved 14 *Core Component Types* and defined 35 permissible *Data Types*, and has undertaken their maintenance. Further-

more, the *Data Types* provided by UN/CEFACT can be used without restrictions (*Unqualified Data Types (UDT)*) or further restricted (*Qualified Data Types (QDT)*) to accommodate specific business needs. UN/CEFACT also provides the rules to restrict the *Data Types* to *Qualified Data Types*.

2.3.2 NAMING CONVENTION USED

Apart from the structure, CCTS provides a methodology to define properties for the document components to give them meaning. CCTS assigns *Object Class Term* property to every document components it defines. The aggregate components, created from the same document component by restricting them to different contexts, share the same *Object Class Term*. For example, the *Object Class Term* "Person" provides the meaning for all aggregate components obtained from the "Person" aggregate component by restricting it to a context. The *Object Class Term* for both the basic components and the association components is the *Object Class Term* of the component they are defined in. For example, the *Object Class Term* of the "name" basic component is "Person" if it is defined in the "Person" component or it is the "Product" if it is defined in the "Product" component.

Furthermore, CCTS defines the names of the basic document components and the association document components through *Property Term* property. The aim is that even when these components are qualified to create other components and hence are renamed, their *Property Term* remains the same. For example when "Identification" basic document component of a "Party" aggregate document component is qualified in "Trade" context, its name becomes "Tax_Identification", but its *Property Term* is still "Identification".

CCTS defines terms that represent the basic document components and the association document components through *Representation Term* property. The *Representation Terms* for the basic document components give their core data type. The aim is when the data type is later qualified and hence is renamed, it still has the same *Representation Term*. For example, the data type of the "Type" basic component of "Document" aggregate component is qualified to become "DocumentTypeCode", however its *Representation Term* is still "Code" stating that the new qualified data type is derived from "Code" core data type. The *Representation Term* for the association document components, on the other hand, give the *Object Class Term* of the component it refers to. As an example, the "EmailURI" association document component

of “Contact” aggregate component refers to “Internet_Communication” aggregate component and the *Representation Term* of this association component is “Communication” stating that the “Internet_Communication” component is qualified from the “Communication” component.

Another benefit of this way of giving semantics is to provide a naming convention that is necessary to consistently name the defined components to facilitate the comparison during the discovery and analysis process. Furthermore, ambiguities can be prevented such as developing multiple *Core Components* with different names that have the same meaning. This naming convention is derived from ISO 11179 Part 5 [33]. It has three major parts: *Object Class Term*, *Property Term* and *Representation Term*. For example, when the *Core Component* “Invoice. TaxAmount. Amount” basic component is expressed according to this naming convention, “Invoice” is the *Object Class Term*, “TaxAmount” is the *Property Term* and “Amount” is the *Representation Term*.

2.3.3 TYPES OF CORE COMPONENTS

A *Core Component* is a reusable building block for creating electronic business documents. There are three types of *Core Components*:

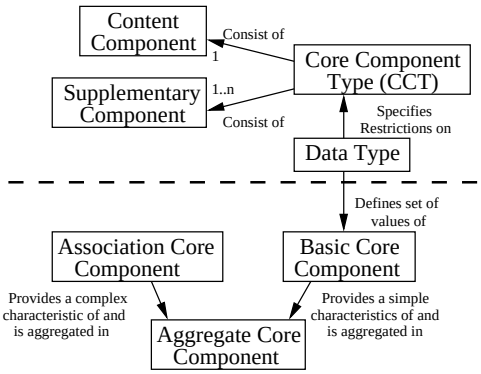


Figure 2.4: Core Component Overview [83]

- *Aggregate Core Component (ACC)*: A distinct real world object with a specific business meaning such as “Address” or “Purchase Order” is termed as an *Aggregate Core Component* (ACC). An *Aggregate Core Component* has at least one and possibly more *Basic Core Components* (BCCs). For example, as shown in Figure 2.5 “Address. Details”

is an *Aggregate Core Component (ACC)* containing several *Basic Core Components (BCCs)*.

- A *Basic Core Component* describes a property of an ACC by using a *Data Type*. For example, as shown in Figure 2.5, "Address. Details. Street" is a *Basic Core Component (BCC)* and is of "Text" Data Type. In other words, the *Data Types* are used as *Representation Terms* of *Basic Core Components*.

- Sometimes it is necessary to define an association between Aggregate Core Components. This is realized through *Association Core Components*. As shown in Figure 2.5, "Person. Details. Residence" is an *Association Core Component* (ASCC) referencing the "Address. Details" ACC.

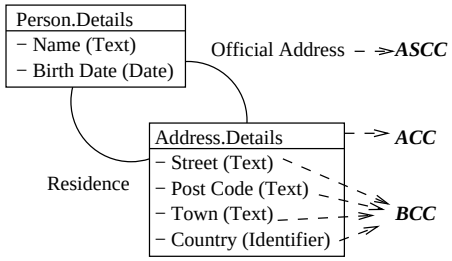


Figure 2.5: Examples of *Basic Core Component* (BCC), *Aggregate Core Component* (ACC) and *Association Core Components* (ASCC) [83]

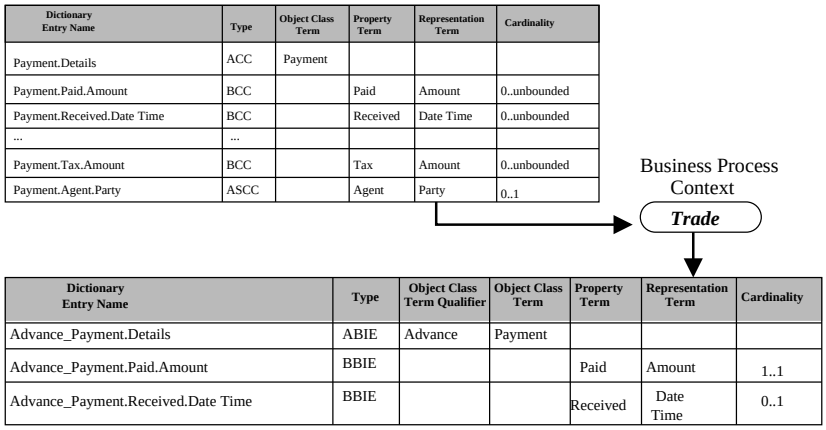
2.3.4 BUSINESS INFORMATION ENTITY (BIE)

A *Core Component* is designed to be context-independent so that it can later be adapted to different contexts and reused. When a *Core Component* is restricted to be used in a specific business context, it becomes a *Business Information Entity (BIE)* and given its own unique name.

The possible business contexts that can be used are defined to be: *Business Process Context; Product Classification Context; Industry Classification Context; Geopolitical Context; Business Process Role Context; Supporting Role Context; System Capabilities Context* and *Official Constraints Context*.

For example, when the *Business Process Context* is specialized to "Purchasing", and the

Geopolitical Context is set to be “EU”, the “Invoice. Tax. Amount” BCC becomes the “Invoice. VAT_ Tax. Amount” *Basic Business Information Entity (BBIE)*.



Dictionary Entry Name	Type	Object Class Term	Property Term	Representation Term	Cardinality
Payment.Details	ACC	Payment			
Payment.Paid.Amount	BCC		Paid	Amount	0..unbounded
Payment.Received.Date Time	BCC		Received	Date Time	0..unbounded
...	...				
Payment.Tax.Amount	BCC		Tax	Amount	0..unbounded
Payment.Agent.Party	ASCC		Agent	Party	0..1

Dictionary Entry Name	Type	Object Class Term Qualifier	Object Class Term	Property Term	Representation Term	Cardinality
Advance_Payment.Details	ABIE	Advance	Payment			
Advance_Payment.Paid.Amount	BBIE			Paid	Amount	1..1
Advance_Payment.Received.Date Time	BBIE			Received	Date Time	0..1

Figure 2.6: Customizing an *Aggregate Core Component* to the *Business Process Context* “Trade”

Similarly, when an *Association Core Component* is used in a context, it becomes *Association Business Information Entity (ASBIE)* and *Aggregate Core Component* becomes *Aggregate Business Information Entity (ABIE)*. For example, in Figure 2.6 an “Advance. Payment. Details” ABIE is created by customizing the “Payment. Details” ACC to the *Business Process Context* “Trade” as follows: An *Object Class Term Qualifier* is added as an additional property and the related BCCs are customized to create the BBIEs by restricting their cardinality.

Figure 2.7 [83] gives the relationship between the types of core components and the corresponding business information entities.

2.3.5 UN/CEFACT CORE COMPONENT LIBRARY

The Core Component Library [82] is the repository for UN/CEFACT CCTS artifacts. Currently there are quite a number of UN/CEFACT artifacts in the Core Component Library.

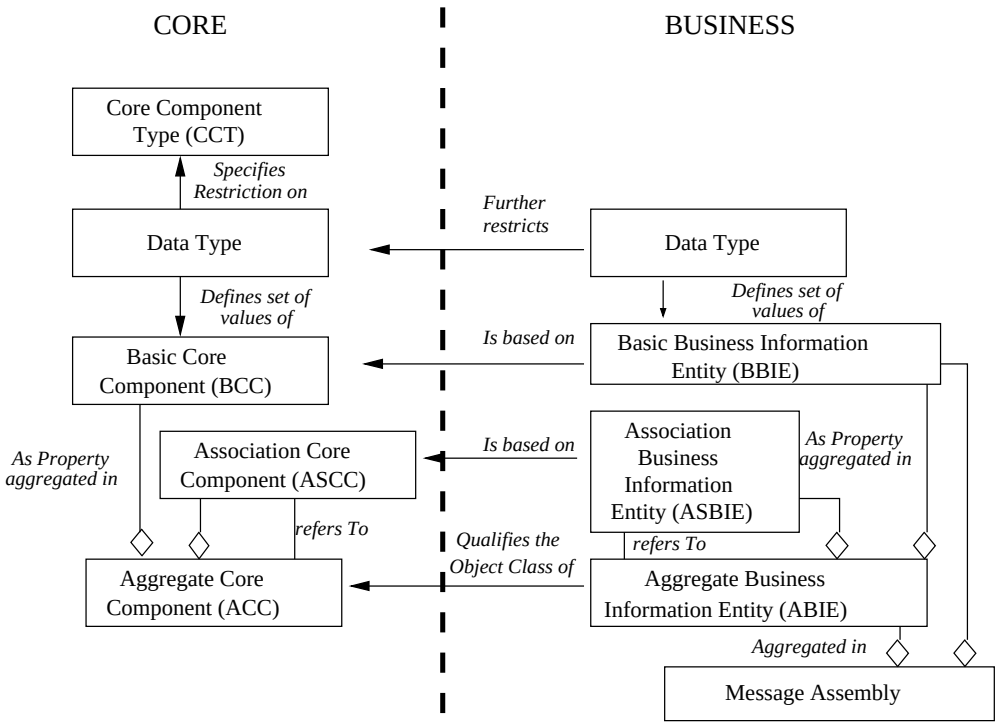


Figure 2.7: Relationship between Core Components and Business Information Entities [83]

2.4 UNIVERSAL BUSINESS LANGUAGE 2.0 (UBL)

The Universal Business Language [75] initiative from OASIS adopts the UN/CEFACT Core Component Technical Specification (CCTS) approach and develops a set of standard XML business document definitions.

Currently, the approved version of UBL is 2.0 [75] and there are thirty one XML schemas for common business documents such as “Order”, “Despatch Advice” and “Invoice”. In addition to the document definitions, UBL 2.0 provides a library of XML schemas (XSDs) [79] for reusable common data components like “Address”, “Item”, and “Payment” from which the documents are constructed. UBL 2.0 reuses *Core Component Type* and *Data Type* definitions

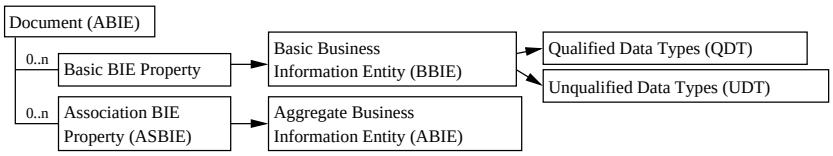


Figure 2.8: The UBL Components

from UN/CEFACT CCTS such as “AmountType”, “CodeType” and “DateTimeType”. When UN/CEFACT CCTS *Data Types* are imported to UBL type space, they are termed as the *Unqualified Data Types (UDT)*. Additionally, UBL defines *Qualified Data Types (QDT)* which are primarily for code lists such as *CurrencyCodeType* or *CountryIdentificationCodeType* defined for use within UBL.

At the time the UBL initiative had started, UN/CEFACT CCTS had not yet specified core components. Therefore UBL created its own BIEs based on CommerceOne’s xCBL (XML Common Business Library) 3.0 [95] and the UN/EDIFACT (EDI for Administration Commerce and Trade) dictionary [85]. Hence the UBL vocabulary consists primarily of *Aggregate Business Information Entities* (ABIEs).

Figure 2.8 shows the structure of the UBL Documents. It should be noted that in addition to identifying conceptual *Business Information Entities* (BIEs), UBL uses the CCTS artifacts such as ABIE, ASBIE and BBIE to compose its document schemas. This is in contrast to some other standards which use CCTS components in different document artifacts of their own and also name them differently.

In UBL, ABIEs are used in two different ways: (1) The document ABIEs which represent UBL Documents such as “Order” and “Invoice” and (2) More fine-grained reusable ABIEs such as “Address” and “Party”. As shown in Figure 2.7, an ABIE is composed of BBIEs and ASBIEs as in UN/CEFACT CCTS. In UBL 2.0, according to the UBL 2.0 Naming and Design Rules, this composition is realized through *BIE Properties*. A BBIE has a single content whose type is specified either with *Qualified Data Types* (QDT) or *Unqualified Data Types* (UDT). Figure 2.9 shows an example UBL 2.0 “Order” document.

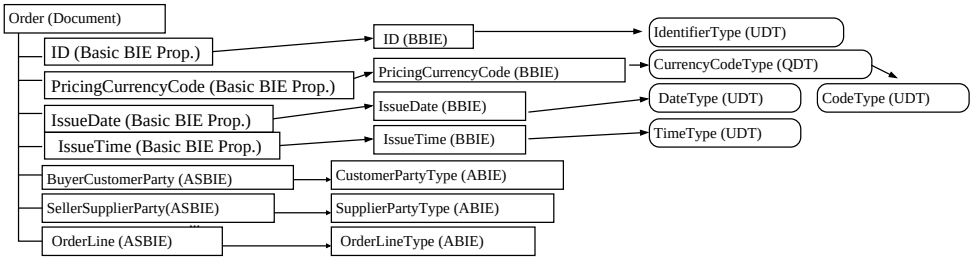


Figure 2.9: An Example UBL Document Schema Structure

2.4.1 UBL CUSTOMIZATION AND EXTENSIBILITY

There are two types of customizations specified in UBL 2.0: Conformant customization and Compatible customization.

Before going into details of customization, it is worth describing the validation of UBL documents. UBL 2.0 recommends a two-phase validation technique as shown in Figure 2.10. In the first phase, an incoming UBL document is validated against UBL 2.0 XSD schemas (or customized versions of them). If the instance passes the first phase, in the second phase it is checked against the rules, which specify additional constraints on the values of the elements in the instance. Generally, the rules are specified through XSL [97] or Schematron languages [64]. If the instance passes both of the phases successfully, it is delivered to the processing business application.

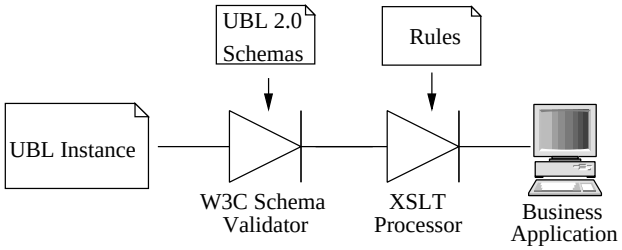


Figure 2.10: Two-phase validation of UBL Messages

2.4.1.1 CONFORMANT CUSTOMIZATION OF UBL 2.0

The key idea behind the conformant customization is that the XML instances in the customized implementation must also conform to the original standard UBL 2.0 schemas. There are four ways of performing conformant customizations:

1. *Inserting additional elements through the use of "UBLExtensions" element*: An optional *UBLExtensions* element appears as the first child of all UBL 2.0 documents and is used to include non-UBL data elements. For example, there could be elements containing data whose inclusion is mandated by law for certain business documents in certain regulatory environments. *UBLExtensions* element is composed of multiple *UBLExtension* elements, each containing a single element *ExtensionContent* of type "xsd:any" to accommodate the widest possible range of extensions. This means that

```
<Order
 xmlns="urn:oasis:names:specification:ubl:schema:xsd:Order-2"
 xmlns:cac="urn:oasis:names:specification:ubl:schema:xsd:CommonAggregateComponents-2"
 xmlns:cbc="urn:oasis:names:specification:ubl:schema:xsd:CommonBasicComponents-2"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

 <UBLExtensions xmlns="urn:oasis:names:specification:ubl:schema:xsd:
  CommonExtensionComponents-2">
  <UBLExtension>
   <ExtensionContent>
    <OrderExtensions>
     <productForm>Granule</productForm>
     <bonusPoint>100</bonusPoint>
    </OrderExtensions>
   </ExtensionContent>
  </UBLExtension>
 </UBLExtensions>
 ...
 <cac:BuyerCustomerParty> ...  </cac:BuyerCustomerParty>
 <cac:SellerSupplierParty> ...  </cac:SellerSupplierParty>
 <cac:AnticipatedMonetaryTotal> ...  </cac:AnticipatedMonetaryTotal>
 <cac:OrderLine> ...  </cac:OrderLine>
</Order>
```

Figure 2.11: UBL Extension Example

any well-formed XML element from any vocabulary can be inserted into *Extension-Content* element without modifying the schema.

An example UBL extension is given in Figure 2.11 where the “UBLExtensions” element is inserted into the beginning of the order document. It contains a “productForm” element, which shows the requested form of the ordered product, and a “bonusPoint” element, which is the bonus amount gained by the buyer upon purchasing the ordered products.

2. *Subsetting original UBL 2.0 schemas*: There are very many possible elements in a UBL document. For example, there are about 50,000 possible elements in a UBL Order Document. Most applications will not need all this data. Therefore, UBL 2.0 allows users to create subsets of its schemas. Subsets remove any optional information entities that are not necessary to the specific implementation. UBL 2.0 Small Business Subset [78] is an example of this subsetting mechanism.

3. *Placing constraints on the value space of information entities and/or putting constraints among these values*: In a specific implementation of UBL 2.0, there may be additional constraints on the value space of information entities. For example, “The Total Value of an Order cannot be more than 50,000 USD”. There may also be rules about dependencies between values of the elements, such as “The Shipping Address must be the same as the Billing Address” or ”The Start Date must be earlier than the End Date”. The former type of requirement can be reflected in the UBL schemas by type restriction; however, this requires schema modification. On the other hand, the latter type of requirement cannot be represented through XSD schemas. However, users can describe both of these constraints through Schematron [64] or XSL rules [97] and feed these rules into the second phase of validation as already described.

4. *Customizing the code lists*: Code list customization is described in Section 2.4.1.3.

2.4.1.2 COMPATIBLE CUSTOMIZATION OF UBL 2.0

Sometimes conformant customization may not be sufficient for a specific implementation. Users may need to perform more complex modifications such as extending an ABIE, creating a new ABIE or creating a new document. To handle these cases, the compatible customization approach can be used. In compatible customization, the users modify an existing UBL 2.0 schema or create a new one by re-using the “largest suitable” aggregation from the UBL library. When performing compatible customization, the users need to follow the UBL Naming and Design Rules [77].

2.4.1.3 THE USE OF CODE LISTS

In UBL 1.0, the standard and the default code list values are specified directly in the UBL schemas as XSD enumeration constraints. This allows all UBL 1.0 instances to be validated in a single pass using generic XSD processors. However, the specification of the default values directly in the schemas also makes it difficult to modify the code lists to meet customization requirements.

In UBL 2.0, only three code lists are enumerated in the schemas: (1) The *CurrencyCodeContentType* for internationally standardized currency codes, (2) The *BinaryObjectMimeCode-*

ContentType for MIME encoding identifiers and (3) The *UnitCodeContentType* for unit codes. In fact, these enumerations are specified in *Unqualified Data Types* from UN/CEFACT and UBL 2.0 includes them as they are for the attribute values.

The other code lists used in UBL are not enumerated in the schema expressions. Instead of enumerating the codes in the XSD schemas, UBL uses a common base type called *CodeType*, which is an extension of "xsd:normalizedString" for all elements expressing values from the code lists. The UBL 2.0 package includes files for every code list. These files are separate from the provided XSD schemas and they are in a standard format. Trading partners can modify or replace any of these files to meet their business requirements. After this step, they can convert these files in proprietary format to Schematron or XSL rules. OASIS Code List Representation Technical Committee [10] provides tools for this purpose. Later these rules can be fed into the second phase of validation as already described.

2.5 OPEN APPLICATIONS GROUP INTEGRATION SPECIFICATION (OAGIS) BUSINESS OBJECT DOCUMENTS (BOD) VERSION 9.1

The Open Applications Group, Inc. (OAGi) [44] is a not-for-profit open standards organization that defines electronic document standards called *Business Objects Documents* (BODs). Since its first release in 1995, several versions of Open Applications Group Integration Specification (OAGIS) BODs have been produced, the latest one being the OAGIS BOD version 9.1 [48]. This version is redesigned to be based on the UN/CEFACT Core Components Technical Specification.

The *Business Object Document* (BOD) is based on a pair of concepts called the *Noun* and the *Verb*. The *Verb* identifies the action to be applied to the *Noun*. *Noun* is the object or document such as "PurchaseOrder", "RequestForQuote", and "Invoice" that is being acted upon. Examples of *Verbs* include "Cancel", "Get", "Process", and "Synchronize". The *Verb* and *Noun* combination provides the name of the BOD. For example, when the *Verb* is "Process" and the *Noun* is "PurchaseOrder", the name of the BOD is "ProcessPurchaseOrder". There are 77 nouns and 12 verbs defined in OAGIS 9.1.

The separation of *Verb* and *Noun* components increases the reusability of data. For example, the *Noun* "PurchaseOrder" contains all of the information that might be present in a

"PurchaseOrder". The instantiation of each of the possible *Verb* and *Noun* combinations then further restricts the document to a context. For example, in a "ProcessPurchaseOrder" transaction, business partners and line item data must be provided, whereas in a "CancelPurchaseOrder" only the order identifier is enough to carry out the transaction. Note that these constraints do not change the schema of a document. Rather, they provide the constraint rules to be applied in the validation of a BOD. Like UBL, OAGIS recommends a two-phase validation. When an OAGIS document is received, it is first validated against the corresponding XML Schema and afterwards against the corresponding Schematron/XSL rules. Only after the OAGIS instance document passes this two-phase validation is it delivered to the business application that processes the document content.

OAGIS provides some recommendations on the usage of *Verbs*. *Verbs* may come in pairs meaning that the response to a *Verb* should be another specific *Verb*. For example, the response *Verb* of "Process" is "Acknowledge".

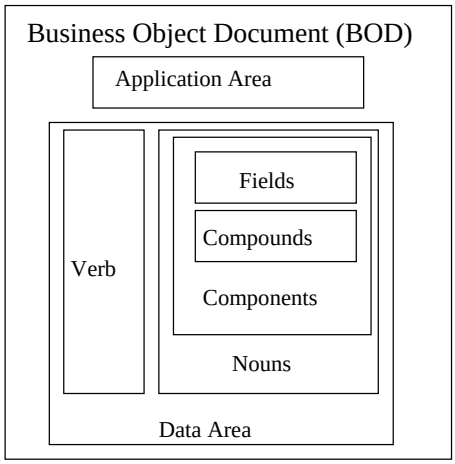


Figure 2.12: The Structure of OAGIS Business Object Document (BOD)

As shown in Figure 2.12, a BOD is a message structure composed of an *ApplicationArea* and a *DataArea*. The *ApplicationArea* carries necessary information for transport software to send the message to the destination such as the sender, the signature of the sender and the unique identifier of the BOD. The need for the *ApplicationArea* stems from the following: the application software that creates a BOD may be separate from the transport software that sends the BOD to the destination. Therefore the application software creating the BOD should provide the transport software with the necessary configuration information to send the BOD.

In other words, the *ApplicationArea* contains the configuration information created by the application software and conveyed to the transport software.

The *DataArea* contains a single *Verb* and multiple *Nouns*. A *Noun* may be assembled from *Component*, *Compound*, and *Field* document artifacts. *Components* are large-grained building blocks and may in turn consist of other *Components*, *Compounds*, and *Fields*. Examples of *Components* include: "PurchaseOrder Header", "Party", and "Address". *Compounds*, which are used across all BODs, are a logical grouping of *Fields* (low level elements). Examples include "Amount", "Quantity", "DateTime", and "Temperature". *Fields* are the lowest level elements used in OAGIS *Components* and *Compounds*. Figure 2.13 shows an example BOD assembly with OAGIS artifacts.

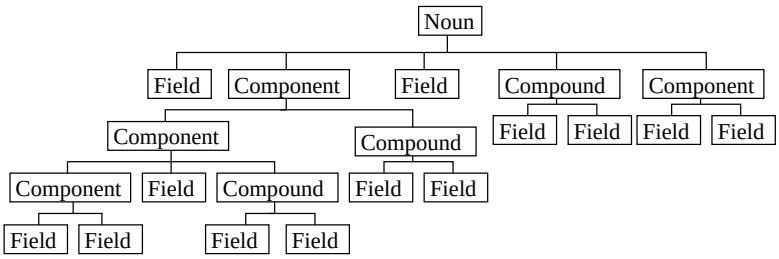


Figure 2.13: OAGIS Business Object Document (BOD) assembly example

OAGIS implementation of the Core Component Technical Specification (CCTS) is shown in Figure 2.14. In OAGIS 9.1, the *Core Component Types* and *Unqualified Data Types* are directly used in the OAGIS Schemas. In other words, all OAGIS *Field* types are based on UN/CEFACT *Core Component Types*. Furthermore, the code lists, such as ISO 54217 Currency Codes and ISO 5639 Language Codes, recommended by UN/CEFACT are also used as described in Section 2.5.1.3.

As shown in Figure 2.14, OAGIS incorporates the UN/CEFACT ABIEs into OAGIS *Components* rather than using them directly. When using these ABIEs in their *Components*, OAGIS appends "ABIEType" suffix to the name of the ABIE in order to identify that it is an ABIE from UN/CEFACT.

OAGIS Naming and Design Rules (NDR) are based on the version UN/CEFACT ATG2 Naming and Design Rules (NDR) [4].

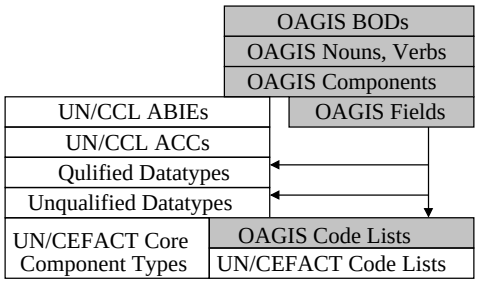


Figure 2.14: OAGIS usage of UN/CEFACT CCT

2.5.1 OAGIS EXTENSIBILITY

OAGIS provides two mechanisms to extend its specifications: *UserArea* Extensions and *Overlay* Extensions.

2.5.1.1 USERAREA EXTENSIONS

The *UserArea* extensibility provides a means of adding implementation specific content to an existing OAGIS *Component* in an existing OAGIS BOD. When a few simple fields are needed to complete the information for the exchange, *UserArea* extensions are used. There is a *UserArea* element of type "xsd:any" at the end of each OAGIS *Component* where the users can insert any valid XML instance without changing the original OAGIS schema.

For example, in Turkey, the addresses contain "Mahalle" information, which basically specify a district in a city. In OAGIS, "Address" *Component* does not have such a *Field* to carry "Mahalle" information. This "Mahalle" information can be inserted in the *UserArea* part of "Address" *Component* in a BOD instance when it is used in Turkey, as shown in Figure 2.15.

```
<Address>
....
        <UserArea xmlns:myTrImpl="http://www.myTrImpl.org">
                <myTrImpl:Mahalle>EsatOglu<myTrImpl:Mahalle>
        </UserArea>
</Address>
```

Figure 2.15: UserArea Example

2.5.1.2 OVERLAY EXTENSIONS

When the users need more complex changes such as creation of a new BOD or creation of new a *Component*, *Overlay* extension mechanism is used. The *Overlay* extensions result in the creation of new XML Schemas for the BOD in their own separate namespaces. It should be noted that only *Nouns* and *Components* are overlay extensible.

The *Overlay* extension mechanism adopts a layering approach. New layers, called overlays, are defined in their own respective namespaces on top of core OAGIS Schemas. Specialized BODs and *Components* are defined by extending BODs from lower layers and/or by composing new BODs from a combination of existing components, extended components, and new components. In Figure 2.16, an example for overlays is shown where "Automotive" overlay is created from core OAGIS schemas, whereas "Auto Parts" that is a subdomain of "Automotive", is built on "Automotive" and OAGIS core.

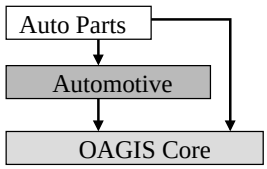


Figure 2.16: OAGIS Overlay Layering Example

```
<xs:complexType name="MyInvoiceType">
 <xs:complexContent>
  <xs:extension base="oa:Invoice">
   <xs:sequence>
    <xs:element ref="ia:TotalDiscounts" minOccurs="0"/>
    <xs:element name="GrandTotal" type="oa:Amount" minOccurs="0"/>
    <xs:element name="MyInfo" type="xs:string" minOccurs="0"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:element name="MyInvoice" type="my:MyInvoiceType"
  substitutionGroup="oa:Invoice"/>
```

Figure 2.17: Overlay Extension Example

With *Overlay* extensions, the users are allowed to create a new BOD, a *Noun*, a *Component*,

a *Compound* or a *Field*, or extend any of the previously defined OAGIS artifacts. For example, a user may extend the "Invoice" *Noun* of OAGIS by adding the following: a new *Component* for representing total discounts; an existing *Compound* for grand total and a new *Field* for a special purpose. Figure 2.17 shows how these extensions are realized. The user first creates a new *Noun* called "MyInvoiceType" by extending the "Invoice" provided by OAGIS. Afterwards, the user inserts the elements mentioned. Finally, the user defines the "MyInvoice" element of type "MyInvoiceType". Note that "MyInvoice" element is in the same "xsd:substitutionGroup" as OAGIS "Invoice", which means that anywhere the OAGIS "Invoice" element is included in a model, the "MyInvoice" element can be inserted as well. In order to preserve interoperability among different *Overlay Extensions*, XSLT transformations are defined to convert an instance document conforming to an overlay into another.

```
In the CodeLists.xsd:

        <xsd:simpleType name="PaymentMethodCodeEnumerationType">
                <xsd:restriction base="xsd:normalizedString">
                        <xsd:enumeration value="Cash"/>
                        <xsd:enumeration value="Cheque"/>
                        <xsd:enumeration value="CreditCard"/>
                        <xsd:enumeration value="DebitCard"/>
                        <xsd:enumeration value="ElectronicFundsTransfer"/>
                        <xsd:enumeration value="ProcurementCard"/>
                        <xsd:enumeration value="BankDraft"/>
                        <xsd:enumeration value="PurchaseOrder"/>
                        <xsd:enumeration value="CreditTransfer"/>
                </xsd:restriction>
        </xsd:simpleType>

        <xsd:simpleType name="PaymentMethodCodeContentType">
                <xsd:union memberTypes="PaymentMethodCodeEnumerationType xsd:normalizedString"/>
        </xsd:simpleType>

In the Fields.xsd:

        <xsd:simpleType name="PaymentMethodCodeContentType">
                <xsd:restriction base="oacl:PaymentMethodCodeContentType"/>
        </xsd:simpleType>
```

Figure 2.18: Code List Example

UserArea extensions are faster to apply than *Overlay* extensions. However, they do not pro-

vide the same level of control on the schemas as the *Overlay* extensions do. This is because the *UserArea* extensions are applied to the OAGIS BOD XML instance documents and not to the OAGIS BOD schema itself.

2.5.1.3 CODE LIST EXTENSIONS

OAGIS uses and recommends the code lists from UN/CEFACT, and allows additional values to be present. This is accomplished as follows: OAGIS defines two "xsd:simpleType" for each coded *Field*: (1) an enumeration type, which lists the codes to be used and (2) a "xsd:simpleType" which is a union of that enumeration type and the "xsd:normalizedString". In other words, with the specification of "xsd:normalizedString" any code can be inserted to a BOD XML instance without affecting the validity against the BOD Schema. For example, as presented in Figure 2.18, the "PaymentMethodCodeContentType" *Field* is associated with "oacl:PaymentMethodCodeContentType" which is the union of "PaymentMethodCodeEnumerationType" and "xsd:normalizedString". The use of "xsd:normalizedString" allows the users to send codes that are not listed in "PaymentMethodCodeEnumerationType".

2.6 GLOBAL STANDARDS ONE (GS1)

Global Standards One (GS1) [23] is a family of standards focusing on different aspects of supply chain integration such as electronic products codes, product information synchronization and the electronic document standards. GS1 was formed in early 2005 by the European Article Number [13] and the Uniform Commercial Code [81] organizations when they joined together. EAN and UCC were two organizations that heavily contributed to the adoption and proliferation of barcodes.

The part addressing the electronic document interoperability in this family of standards is GS1 eCom. In GS1 eCom, there are two distinct categories: the earlier eCom standards that are based on Electronic Document Interchange (EDI), called EANcom [14] and the newer generation GS1 XML [25] which is defined using XML Schema.

The other standards in GS1 family include the Global Data Synchronization Network [22] and EPCglobal [20]. The Global Data Synchronization Network (GDSN) enables product

data and location information synchronization so that trading partners have consistent item data in their respective systems.

EPCglobal drives the development of the Electronic Product Code (EPC) related with RFID standards. The specifications are based on the Radio Frequency Identification (RFID) research performed at the MIT AutoID Labs [41].

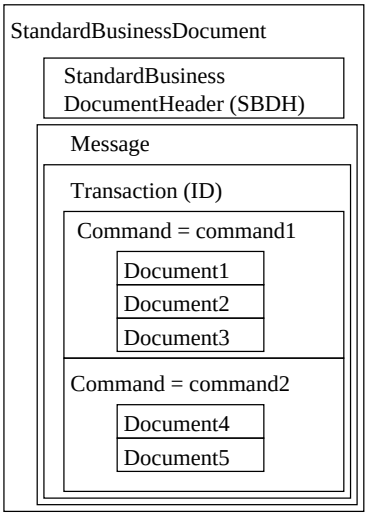


Figure 2.19: The Structure of GS1 XML Structure

2.6.1 GS1 XML

As shown in Figure 2.19, a GS1 XML document is represented with a *StandardBusinessDocument*, which contains a *StandardBusinessDocumentHeader* (SBDH) and a *Message*. *StandardBusinessDocumentHeader* is based on the SBDH defined by UN/CEFACT [88] and provides information about the routing and processing of the XML instance document contained in the GS1 XML *Message*. The SBDH is used for the same purpose as OAGIS's *ApplicationArea* element; that is, it contains the configuration information for the transport software to send the message to its destination.

A GS1 XML document includes either a set of *Commands* or a set of *Transactions* which in turn contain *Commands*:

- *Command*: A *Command* instructs the recipient to perform a particular action, such as "Add", "Delete" and "Refresh", related to the documents within the command. The use

of these commands decreases the number of documents needed. The same document can be used with different commands. Hence, no separate documents like “Add Order”, “Change Order” or “Delete Order” are needed; the same “Order” document can be sent with a relevant command. In a similar way, several documents can reuse the same command.

- *Transaction*: A *Transaction* provides the functionality of executing multiple commands atomically as in relational databases. If one command in a transaction fails, the transaction fails causing all other commands in the transaction to be discarded applying the principle of “all or nothing”.

As an example, assume that a sender needs to send a message about two products and the first product is related to the second one. Instead of sending two distinct transmissions, the sender can transmit them together in one *Transaction* that contains one *Command*, which holds two *Documents* each of which is for a product. If the products are not related then, the sender can send them without using the *Transaction* element. In other words, the user sends only one *Command* containing two *Documents*.

GS1 XML is compliant with UN/CEFACT CCTS methodology in that GS1 XML uses the same modelling, design and technical principles. However, unlike UBL or OAGIS, which use UN/CEFACT artifacts (such as *Core Component Types*, *Data Types* and *Business Information Entities*), GS1 XML does not use UN/CEFACT CCTS artifacts in their XML Schemas. Yet, the GS1 core components are submitted as an input to UN/CEFACT CCTS development.

While developing their e-business standards, GS1 uses its Global Data Dictionary [24] to store, reuse and share common components and business definitions, and their corresponding representations in XML. In other words, the GDD is the repository of:

- Data components, used to create the GS1 XML standards, developed according to the UN/CEFACT Core Components Technical Specification (CCTS).
- Business terms and their representation in GS1 XML.

Through GDD, the search of previously defined components is facilitated.

In the GS1 XML documents, some of the components such as *Measurement*, *DocumentStatus* and *MontetaryAmount* are common to more than one business document and more than one

context. Therefore, these components are included in a common library as a part of the GDD. This approach allows reusing the same information constructs in all business messages.

2.6.1.1 CUSTOMIZATION AND EXTENSIBILITY

In GS1 XML, the following context categories are defined for customization:

- *Business Process* context in which collaboration takes place such as ordering or delivery.
- *Industry Sector* context in which the business partners are involved such as automotive.
- *Geopolitical* context reflecting the geographical factors that influence the business semantics. This can be either country-specific, for example, only for France or Sweden, or limited to certain economic regions, for example, NAFTA or European Union, and finally, it can be applicable everywhere in the world, in which case the context is defined as “Global”.

The context information is reflected to the documents through their namespaces. In other words, the GS1 information components are assigned to a namespace that reflects the context they are defined in. For example, the namespace for the documents that are used in the Global Data Synchronization Network (GDSN) is “gdsn=urn:ean.ucc:gdsn:2”. As another example, the documents for alignment of trade items in Sweden use “sw=urn:ean.ucc:align:sweden:2” as their namespace. On the other hand, the schemas in the common library have “eanucc= urn:ean.ucc:2” as their namespace, because they do not belong to any specific context.

GS1 XML supports extensibility of its document schemas. Starting from release 2.0, there is an element called “extension” at the end of each business document XML schema where additional context-specific information that are not defined by GS1 XML can be inserted. This element is of type “xsd:any”, which allows the users to insert any XML data to the exchanged instance documents without changing the standard GS1 XML schema.

Before starting to exchange GS1 XML instances with other parties, each organization that requires additional elements in their documents publishes their extensions to the “Extended Attributes” section of the Global Data Dictionary Web site. When a sender wishes to send

a message to a receiver, the sender first checks whether the receiver has an extension by consulting the GDD Web site. If there is an extension, the sender sends the message using Attribute/Value Pair mechanism. Attribute/Value Pair mechanism is a way to populate the "extension" area of a document. As an example, assuming that the receiver requires two additional elements: "packagingWeightValue" and "packagingWeightUnitOfMeasure", the sender populates the "extension" area as shown in Figure 2.20.

```
<extension>
<gdsn:attributeValuePairExtension
  xsi:schemaLocation="urn:ean.ucc:2
  ../Schemas/AttributeValuePairExtensionProxy.xsd">
    <value name="packagingWeightValue">15</value>
    <value name="packagingWeightUnitOfMeasure">kg</value>
</gdsn:attributeValuePairExtension>
</extension>
```

Figure 2.20: Attribute/Value Pair Mechanism to populate extension area

```
<xsd:complexType name="ISO3166_1CodeType">
  <xsd:sequence>
    <xsd:element name="countryISOCode">
      <xsd:simpleType>
        <xsd:restriction base="xsd:string">
          <xsd:maxLength value="3"/>
          <xsd:minLength value="1"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>
```

Figure 2.21: Example Country Code Element

2.6.2 THE USE OF CODE LISTS

In GS1 XML, there are two types of code lists, external and internal. External code lists are defined and maintained by other standard bodies outside GS1 XML. The example external code lists include the following:

- Country Codes - ISO 3166-1:1997

- Country Subdivision Codes - ISO 3166-2:1998

- Currency Codes - ISO 4217:2001

The external code lists are defined as “xsd:string” and restricted to an appropriate number of characters. Figure 2.21 shows an example for “countryISOCode” element which is defined as type “xsd:string” whose length is three characters. However, GS1 XML does not import the code list values to the GS1 XML Schemas because of copyright and maintenance issues. In other words, they are not enumerated in the GS1 XML Schemas.

The internal code lists are those developed and maintained within the GS1 System. They are defined as “xsd:enumeration” and imported into the business document schema that uses them. Figure 2.22 provides an example internal coding list for payment method types used in GS1 XML. It should be noted that all of the possible values are enumerated in the provided XML Schemas.

```
<xsd:simpleType name="PaymentMethodListType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="BANK_CHEQUE">
    </xsd:enumeration>
    <xsd:enumeration value="CASH">
    </xsd:enumeration>
    <xsd:enumeration value="CERTIFIED_CHEQUE">
    </xsd:enumeration>
    <xsd:enumeration value="CHEQUE">
    </xsd:enumeration>
    <xsd:enumeration value="CREDIT_CARD">
    </xsd:enumeration>
    <xsd:enumeration value="LETTER_OF_CREDIT">
    </xsd:enumeration>
    ...
  </xsd:restriction>
</xsd:simpleType>
```

Figure 2.22: Example Payment Method List Element

2.7 ANALYSIS OF THE ELECTRONIC BUSINESS DOCUMENT STANDARDS

In this section, the surveyed electronic document standards are analyzed with respect to their document design principles, how they handle customization and extensibility, their coverage of the other layers of interoperability and their industry relevance.

2.7.1 THE DOCUMENT DESIGN PRINCIPLES

The document design principles involve the document artifacts used in composing the documents, the code lists used to convey the meaning of the values in the elements and the use of XML namespaces. Furthermore, since all the document standards surveyed are based on UN/CEFACT CCTS, how this methodology is used in the design of the document schemas is also discussed. Table 2.1 summarizes the document design principles.

2.7.1.1 DOCUMENT ARTIFACTS AND THE USE OF UN/CEFACT CCTS METHODOLOGY

The document artifacts used in EDI are “Interchange”, “Message”, “Segment” and “Element” (Section 2.2). Note that EDI is not based on UN/CEFACT CCTS Methodology. UBL 2.0 uses the CCTS methodology to generate the document artifacts. UBL 2.0 currently considers only the “Business Process” context and identifies the *Business Information Entities* (BIEs) and bases the type of their artifacts on UN/CEFACT *Unqualified Datatypes* and *Core Component Types*. UN/CEFACT develops its own BIEs, *Core Components* and *Datatypes* and stores them at the UN Core Component Library (UN/CCL). OAGIS 9.0 uses some of the UN/CEFACT ABIEs in their *Components* and bases the types of its *Fields* on UN/CEFACT *Unqualified Datatypes* (UDT) and *Core Component Types* (CCTs). GS1 XML uses the UN/CEFACT CCTS methodology to generate its own artifacts by using its Global Data Dictionary.

2.7.1.2 THE USE OF CODE LISTS

Code lists are important to uniquely convey the semantics of elements in electronic documents such as the country codes, currency codes, and the payment units. All of the surveyed

document standards provide default code lists and allow them to be modified and/or extended to support local codes.

As shown in Table 2.1, EDI provides codes for structuring of the message artifacts (e.g. segment codes). Furthermore, UN/EDIFACT recommends *ISO Country Code*, *Currency Code*, *Numerical Representation of Dates, Times, Periods of Time* and *UN/LOCODE* [32]. EDI also allows implementers to convey their own local or external codes through the use of two data elements, 1131 [86] and 3055 [87].

UN/CEFACT defines five code lists: *Country Codes*, *Subdivision Codes*, *Currency Codes*, *BinaryObject Mime Codes* and *Unit Codes*.

UBL 2.0 uses *Currency Codes*, *BinaryObject Mime Codes* and *Unit Codes* from UN/CEFACT and enumerates them in its schemas to validate attribute values. The other code lists used in UBL are not enumerated in the schema expressions. Instead of enumerating the codes in the XSD schemas, UBL uses a common base type called *CodeType*, which is an extension of "xsd:normalizedString", for all elements expressing values from code lists. As described in Section 2.4.1.3, UBL allows the users to implement their own local/external codes.

For use of code lists, OAGIS defines two "xsd:simpleType" for each coded *Field*: (1) an enumeration type, which lists the codes to be used and (2) a "xsd:simpleType" which is a union of that enumeration type and the "xsd:normalizedString" as explained in Section 2.5.1.3. With this mechanism, the implementers can use their own local/external code lists.

In GS1 XML, there are two types of code lists, external and internal. External code lists are defined and maintained by other standard bodies outside GS1 XML. The internal code lists are those developed and maintained within the GS1 System. They are defined as "xsd:enumeration" and imported into the business document schema that use them as described in Section 2.6.2.

2.7.1.3 THE USE OF NAMESPACES

Generally, the namespaces in XML are used for avoiding name conflicts. The document standards make additional use of the namespace mechanism as follows: UBL achieves categorization of documents through namespaces, OAGIS identifies the extensions through namespaces (Section 2.5.1) and GS1 XML gives context to the both original documents and extended

documents through the namespaces as described in Section 2.6.1.

2.7.1.4 NAMING AND DESIGN RULES

The naming and design rules specify how to name and structure the artifacts, how to put relations between the artifacts and how to use data types for the artifacts. UN/CCL uses ISO 11179 naming rules, which identify the artifacts in *Object Class*, *Property Term* and *Representation Term* format as described in Section 2.3. UBL 2.0 uses UBL 2.0 Naming and Design Rules, which are based on the CCTS terms such as ABIE, ASBIE and BBIE. Furthermore, these rules specify how to represent the artifacts such as ABIEs, ASBIEs and BBIEs in XML schemas. For example, for every ABIE, a "xsd:complexType" must be defined and the name of this complexType must be in upper camel case (UCC) format (UCC capitalizes the first character of each word and compounds the name such as "AccountType"). OAGIS 9.0 applies naming and design rules based on Applied Technology Group XML Syntax (ATG2) Naming and Design Rules (NDR) [4]. Note that UN/CEFACT ATG2 NDR are based on UBL 2.0 NDR.

GS1 XML first designs its information model in UML, before creating the corresponding XML schemas. GS1 XML uses its own UML to XSD conversion rules to generate their XML schemas and to name them.

2.7.1.5 ANALYSIS OF DOCUMENT DESIGN PRINCIPLES

The differences with respect to document design principles as analyzed in this section result in considerable differences in document instances from different standards. As an example, in Figure 2.23, the OAGIS 9.0 "AddressBaseType" Component and GS1 XML "NameAndAddressType" document elements are compared. As it is clear from this figure, there are differences in the element names, the element positions and structures as well as in the use of code lists.

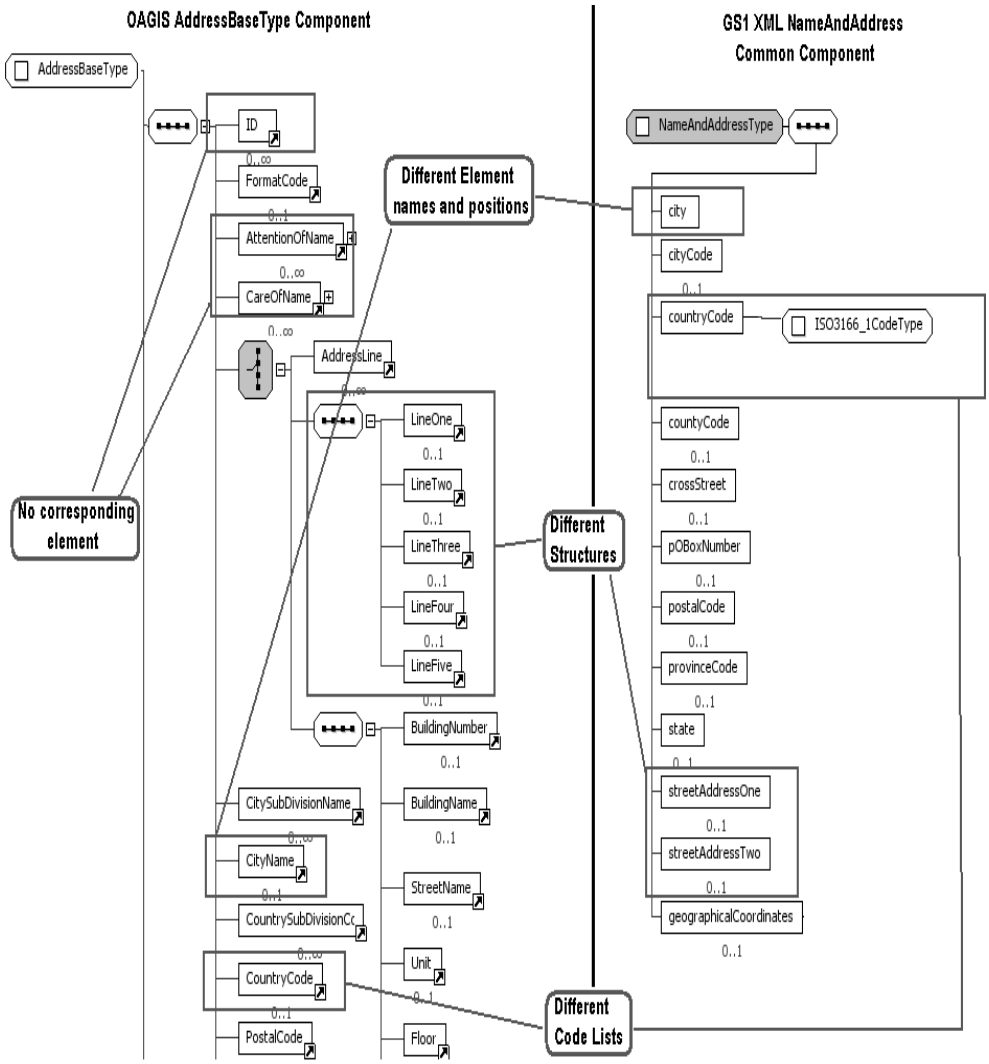


Figure 2.23: An Example Comparing Related Parts of OAGIS BOD 9.0 and GS1 XML Documents

2.7.2 CUSTOMIZATION AND EXTENSIBILITY

Any document interoperability standard faces two challenges. First, the standard needs to be extensible to allow definition of information that is not contained in the standard's artifacts because no standard can contain all of the data needed in every environment. Secondly, to be able to address a particular constraint in a specific context, it should be possible to customize the standard's artifacts according to a context.

Table 2.2 presents a summary of how the standards addressed in this thesis handle customization and extensibility. Note that conformant customizations are also extensible.

EDI addresses the customization through a subsetting mechanism to cover the requirements of a specific context. The EDI messages are subsetted first through industry Implementation Guides (IG), which are then subsetted into trading partner IGs, and into departmental IGs.

Extensibility in EDI is difficult because the EDI systems are highly static and inflexible: introducing a new type or changing an existing type of business document is a complex process. Such changes require the modification of translation software and must be validated in the related EDI committees.

In UN/CEFACT CCTS, a *Core Component* is designed to be context-independent and is customized to one of the eight contexts defined by UN/CEFACT to become a *Business Information Entity (BIE)*. The possible business contexts that can be used are defined to be: *Business Process Context; Product Classification Context; Industry Classification Context; Geopolitical Context; Business Process Role Context; Supporting Role Context; System Capabilities Context* and *Official Constraints Context*.

UN/CEFACT CCTS supports extensibility as follows: if users cannot find proper components in the *Core Component Library* to model their documents, they can create and publish new core components. In other words, UN/CEFACT CCTS thrives on extensibility by allowing users to define core components with possible future harmonizations and removal of redundancies.

UBL 2.0 allows customization through (1) *UBLExtensions* element, (2) subsetting by removing optional information entities that are not needed, and (3) putting constraints to the elements as described in Section 2.4.1.1. On the other hand, the users can extend the UBL 2.0

schemas through the mechanisms described in Section 2.4.1.2.

In OAGIS BODs, there is no formal mechanism to handle user specific constraints. However, the users are free to restrict an already existing BOD as they wish and share it with other partners.

OAGIS provides two mechanisms to extend its specifications as detailed in Section 2.5.1:

- *UserArea* Extensions: *UserArea* extensions provide an optional element within each OAGIS defined *Component* that may be used by an implementer to carry any necessary additional information. This area is of type "xsd:any", which means any valid XML instance can be inserted in this area without modifying the OAGIS standard XML Schemas (XSDs).
- *Overlay* Extensions: *Overlay* extensions allow users to extend an OAGIS BOD, *Noun* and *Component* to meet their own needs, even adding new BODs, *Verbs*, *Nouns* and *Components* where necessary. It is also possible for users to provide additional constraints in their own XSL constraints, which may then be applied to OAGIS document instances. The *Overlay* extension mechanism is used when the implementers have more complex customization requirements than a few additional elements.

Every document in GS1 XML is used in a business context, and in GS1 XML, there are three context categories: *Business Process*, *Industry Sector* and *Geopolitical contexts* as described in Section 2.6.1.1.

GS1 XML supports extensibility of its document schemas. Starting from release 2.0, there is an element called "extension" at the end of each business document XML schema where additional context-specific information that is not defined by GS1 XML can be inserted. This element is of type "xsd:any", which allows the users to insert any XML data to the exchanged instance documents without changing their standard XML Schema.

Before starting to exchange GS1 XML instances with other parties, each organization that requires additional elements in their documents publishes their extensions to the "Extended Attributes" section of the Global Data Dictionary (GDD) Web site. When a sender wishes to send a message to a receiver, the sender first checks whether the receiver has an extension by consulting the GDD Web site.

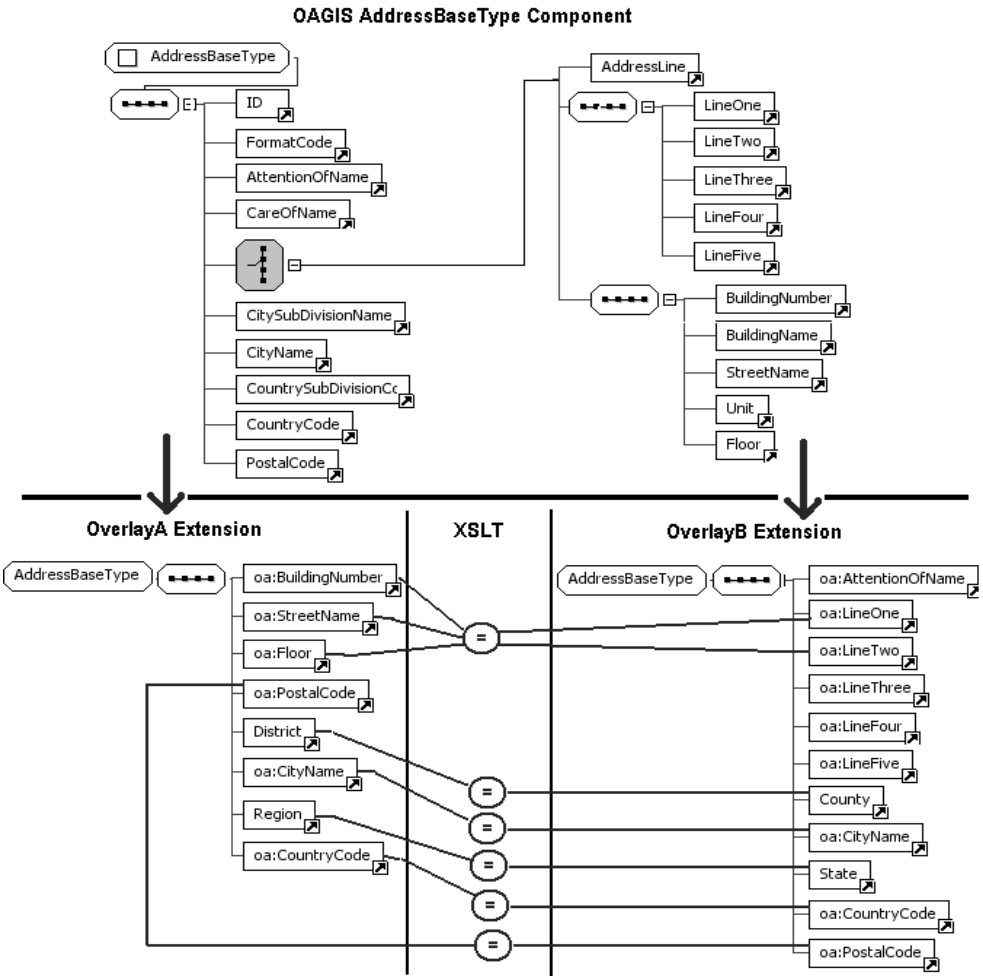


Figure 2.24: Example XSL Transformations necessary to map between two different *Overlay* extensions in OAGIS BODs

2.7.2.1 ANALYSIS OF CUSTOMIZATION AND EXTENSIBILITY

Customization and extensibility affect how the documents are processed. There are two cases to be considered:

- In the first case, if the parties use the same document schema with the same extensions and customizations, a two-phase validation at the receiving end is applied: In the first phase, the incoming document instance is validated against the common XSD schema. If the document instance passes the first phase, in the second phase it is checked against the rules, which specify additional domain specific constraints on the values of the elements in the instance. Generally, the rules are specified through XSL [97] or Schematron languages [64]. If the instance passes both of the phases successfully, it is delivered to the processing business application.

- In the second case, when two enterprises use different customizations or extensions of the same document schema, the schema changes need to be mapped to each other through manually provided XSL Transformations. For instance, Figure 2.24 shows the XSL Transformations necessary to map between two different example *Overlay* extensions in OAGIS BODs. A classification of problems and solutions using XSL transformations to convert business documents is given in [93].

  Once the transformations are applied, the document instance goes through the two-phase validation as described for the first case.

2.7.3 COVERAGE OF OTHER LAYERS OF INTEROPERABILITY

Document interoperability is only one of the layers in the interoperability stack. The other layers of interoperability include the transport protocol, the message header and the business processes. A detailed survey of Business-to-Business (B2B) interactions in general is given in [40] where a survey of the main techniques, systems, products, and standards for B2B interactions are presented together with a set of criteria for assessing them.

The standards covered in this survey do not enforce any specific transport protocol. However, some of them recommend certain transport protocols: GS1 XML recommends the use of EDIINT AS1 [17] and AS2 [18] transport protocols, which define a minimum set of pa-

rameters and options to enable secure/reliable transport for the exchange of EDI or XML data. EDIINT-AS1 is based upon SMTP and EDIINT-AS2 is based on HTTP. Among them, AS2 is the transport protocol of choice. However, the exchange of GS1 XML documents is not limited to these standards. OAGIS is currently moving in the Web Service technology direction, although any technology can be used to transport BODs.

The document standards first analyze the relevant business processes or scenarios before deciding on the document components. For example, through the analysis of an invoicing business process, it may be revealed that a component is necessary to represent the "tax amount" in the invoice. Hence, "Tax Amount" is defined as a component that can be discovered and reused in any business document. However, no formal business process specification is provided by the standards surveyed in this thesis. Yet, it s worth mentioning that there is work, called Universal Business Process [80], for defining UBL 1.0 processes through ebBP 2.0 [15]; however, currently it is only informative.

All of the standards (except for UBL and UN/CCL) provide message header information to be conveyed to the transport protocol header. The EDIFACT message headers are the Interchange Control Header Segment, UNB [30] and the X12 Interchange Control Header, ISA [29]. The *Application Area* in an OAGIS BOD is used to convey configuration information from application software to transport software. GS1 XML *StandardBusinessDocumentHeader* (SBDH) carries transport related information from application software to transport software just as in the case of OAGIS *Application Area*.

2.7.3.1 ANALYSIS OF LAYERS OF INTEROPERABILITY ADDRESSED

The surveyed standards do not specify a transport protocol but provide configuration information for the transport protocol message header.

Refraining from specifying other levels of interoperability has the advantage that it allows a wide variety of implementation techniques to be used and hence provides ease in implementation. However, the differences in the implementation techniques may cause interoperability problems.

2.7.4 INDUSTRY RELEVANCE

EDI, being an early horizontal standard, is being used in several industry domains. For example, financial and monetary systems like *Society for Worldwide Interbank Financial Telecommunication* [74] and *Electronic Funds Transfer* [19] use EDI. Furthermore, all airplane booking and ticketing operations are done over EDIFACT through the *International Air Transport Association* system [28].

Contrary to popular belief, electronic business interoperability is still achieved heavily through EDI based messages, and EDI use is growing 3 to 5 percent every year [92]. It seems large organizations will continue to use EDI for the foreseeable future mostly due to the existing infrastructure investments.

UN/CEFACT CCTS is gaining widespread adoption by standards organizations. As already mentioned, a number of standardization efforts have taken up CCTS Methodology, including UBL, GS1 XML, OAGIS, CIDX and PIDX in addition to UN/CEFACT's own Core Component Library (CCL).

The merits of CCTS for improving interoperability have also been noticed by industry and governments. For example, the German Government has made a formal announcement identifying CCTS as the future data standard for domestic affairs [11].

One of the first companies to support UN/CEFACT CCTS methodology and core components in their products is SAP [63]. SAP Global Data Types (GDTs) form the basis of Business Objects and Enterprise Services. All leaf elements of these SAP GDTs are based on *Core Component Types* and *Data Types* [70, 71].

UBL is being adopted by several communities around the world, especially in electronic government applications. The U.S. Department of the Navy (DON) designed their XML Naming and Design Rules around UBL 2.0 NDR.

The first government to use UBL Invoice is Denmark. The use of UBL Invoice is realized through the "Offentlig Information Online UBL (OIOUBL)" Project and has been mandated by law for all public-sector businesses [51] in Denmark. Also in Sweden, the National Financial Management Authority recommended UBL Invoice customized to Sweden, namely, Svefaktura for all government use [72].

Following the success of Danish and Swedish examples, representatives from Denmark, Norway, Sweden, UK, Finland and Iceland have created a Northern European Subset (NES) [43] for UBL to ensure interoperability among these countries.

In the USA, the Department of Transportation has developed a UBL based pilot project for a demonstration of state-of-the-art electronic commerce in a real-world setting [91].

OAGIS BODs are being used in more than 40 countries and in more than 38 industries [49]. The fact that OAGIS allows BODs to be extended by a vertical industry helps with its extensive use. The vertical standards based on OAGIS BODs include AiAG [2], Odette [50], STAR [68], and Aftermarket [1] in the automotive industry. Other standards bodies focused on human resources, chemical, and aerospace industries also use OAGIS BODs.

There are products based on OAGIS BODs such as Oracle E-Business Suite [52], where OAGIS BODs are implemented as Web Services. As another example, IBM WebSphere Commerce service interfaces are defined using the OAGIS message structure [62].

GS1 XML is being used in more than twenty countries and in more than twenty industries all over the world. GS1 is a business solution partner of many companies, including Oracle, Siemens and Philips. The GS1 standards are also leveraged in SAP business solutions packages [63].

2.8 DESCRIPTION LOGICS

Description logics (DLs) [5] are a family of knowledge representation languages that can be used to represent the knowledge of an application domain in a structured way. The name "description logics" come from, on the one hand, the important notions of the application domain are described through concept "descriptions", which are expressions that are built from atomic concepts and atomic roles; on the other hand, DLs differ from their predecessors, such as semantic networks and frames, in that DLs are based on a formal and logic-based semantics.

There are two types of concept descriptions: terminological and assertional. The terminological descriptions, called the TBox, describe the relevant notions of an application domain by stating properties of concepts (classes) and roles (properties), and relationships between

them - it corresponds to the schema in a database setting. On the other hand, the assertional descriptions, called the ABox, is used to describe a concrete situation by stating properties of individuals - it corresponds to the data in a database setting.

Description logic systems use these descriptions to automatically organize class descriptions in a taxonomic hierarchy and automatically classify instances into classes whose definitions are satisfied by the features of the instance. Specifically, description logic reasoners provide two key capabilities: 1) class subsumption, where a class C1 subsumes another class C2 if its definition includes a superset of the instances included in C2; 2) instance recognition, where an instance belongs to a class if the instance's features (roles and role values) satisfy the definition of the class. Description logic systems also have mechanisms to detect inconsistent definitions.

Considering their expressivity, there are a number of DL varieties. The basic DL is *ALC*, which stands for "Attributive concept Language with Complements". *ALC* includes the following constructors: conjunction, discjunction, negation, existential restriction and value restriction. More expressive DLs are obtained by including additional constructors to *ALC*. Considering the naming scheme, the addition of the new constructor results in the appending of a corresponding letter to *ALC* word. For example, if number restrictions, which is identified with *N* letter, is added, the name becomes *ALCN*. However, for expressive DLs, starting with the basic DL *ALC* would lead to quite long names. For this reason, the letter *S* is often used as an abbreviation for the basic DL consisting of *ALC* extended with transitive roles. The letter *H* represents subroles (role Hierarchies), *O* represents nominals (nOminals), *I* represents inverse roles (Inverse), *N* represent number restrictions (Number), and *Q* represent qualified number restrictions (Qualified). For example, *SHIQ* DL has the ability to express role hierarchies, inverse roles and qualified number restrictions in addition to the constructors that *ALC* supports.

One prominent application of DLs is as the formal foundation for ontology languages. Examples of DL based ontology languages include OIL, DAML+OIL and OWL [54], ontology language standard developed by the W3C Web Ontology Working Group.

High quality ontologies are crucial for many applications, and their construction, integration, and evolution greatly depends on the availability of a well-defined semantics and powerful reasoning tools. Since DLs provide for both, they are ideal candidates for ontology languages.

In this thesis, the UN/CEFACT based document standards' schemas are converted to OWL DL ontologies automatically. This helps to run the operations (new subsumption hierarchy computation, consistency checking and instance classification) that Description Logics provide on these ontologies through reasoners. The DL operation that is used mostly in this thesis is new subsumption hierarchy computation. With this operation, the DL reasoner discovers the implicit relations among the classes, which corresponds to *Business Information Entities* of different standards. And these relations helps to provide interoperability of the standards.

It should be noted that, all the knowledge that the DLs can provide could easily be represented by formulae of first-order predicate logic. However, the main reason for using DLs rather than predicate logic is that DLs are carefully tailored such that they combine interesting means of expressiveness with decidability of the important reasoning problems.

2.9 DESCRIPTION LOGICS REASONERS

Currently, there are the following Description Logics reasoners in the literature: Racer Pro [56], KAON2 [37], Fact++ [21] and Pellet [55]. A survey [39] investigates the resoners considering their OWL support, correctness, efficiency, interface capabilities and inference services. The survey concludes that no system, except RacerPro and KAON2, is able to correctly solve at least those tests which lay within the language fragment that the tools claim to support in full. And to some extend KAON2 is not application ready since it fails very often with "out of memory errors" or require significant processing time for language constructs, which are typically in real-world models such as cardinality restrictions. Pellet and FaCT++ do have some serious bugs which result in incorrect answers. In addition to the survey, in the scope of the thesis, the above mentioned reasoners are investigated in terms of their efficiency. Only Racer Pro could answer to the harmonized ontology without "out of memory error". Therefore, in this thesis Racer Pro is used as the Description Logics Reasoner.

2.10 ONTOLOGY AND WEB ONTOLOGY LANGUAGE - OWL

Web Ontology Language (OWL) [54] is a semantic markup language for publishing and sharing ontologies on the World Wide Web. OWL builds upon the Resource Description Framework (RDF) [59]. The complementary RDF Vocabulary Description Language, RDF Schema

(RDFS) [60] standard describes how to use RDF to describe RDF vocabularies.

OWL provides three decreasingly expressive sublanguages:

- **OWL Full** is meant for users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees. It is unlikely that any reasoning software will be able to support complete reasoning for OWL Full.
- **OWL DL** supports those users who want the maximum expressiveness while retaining computational completeness (all conclusions are guaranteed to be computable) and decidability (all computations will finish in finite time). OWL DL is so named due to its correspondence with description logics, which form the formal foundation of OWL. The DL corresponding to the OWL DL ontology language is *SHOIN*.
- OWL Lite supports those users primarily needing a classification hierarchy and simple constraints. OWL Lite is equivalent to *SHIN(D)*.

Within the scope of this thesis, only OWL DL constructs are considered and in the rest of the document, "OWL" is used to mean "OWL DL" unless otherwise stated. OWL describes the structure of a domain in terms of classes and properties. The list of OWL language constructs is as follows:

- OWL Lite Constructs:
  - RDF Schema Features: Class (Thing, Nothing), rdfs:subClassOf, rdf:Property, rdfs:subPropertyOf, rdfs:domain, rdfs:range, Individual
  - (In)Equality: equivalentClass, equivalentProperty, sameAs, differentFrom, AllDifferent, distinctMember
  - Property Characteristics: ObjectProperty, DatatypeProperty, inverseOf, TransitiveProperty, SymmetricProperty, FunctionalProperty, InverseFunctionalProperty
  - Property Restrictions: Restriction, onProperty, allValuesFrom, someValuesFrom
  - Restricted Cardinality: minCardinality (only 0 or 1), maxCardinality (only 0 or 1), cardinality (only 0 or 1)
  - Class Intersection: intersectionOf

Constructor	DL Syntax	Example
intersectionOf	$C_1 \sqcap \ldots \sqcap C_n$	Human ⊓ Male
unionOf	$C_1 \sqcup \ldots \sqcup C_n$	Doctor ⊔ Lawyer
complementOf	$\neg C$	¬Male
oneOf	$\{x_1 \ldots x_n\}$	{john, mary}
allValuesFrom	$\forall P.C$	∀hasChild.Doctor
someValuesFrom	$\exists r.C$	∃hasChild.Lawyer
hasValue	$\exists r.\{x\}$	∃citizenOf.{USA}
minCardinality	$(\geqslant n\ r)$	(⩾ 2 hasChild)
maxCardinality	$(\leqslant n\ r)$	(⩽ 1 hasChild)
inverseOf	r^-	hasChild$^-$

Figure 2.25: OWL Constructors

  - Versioning: versionInfo, priorVersion, backwardCompatibleWith, compatibleWith, DeprecatedClass, DeprecatedProperty,
  - Annotation Properties: rdfs:label, rdfs:comment, rdfs:seeAlso, rdfs:isDefinedBy, AnnotationProperty, OntologyProperty
  - Datatypes: xsd datatype

- OWL DL Constructs::
  - Class Axioms: oneOf, dataRange, disjointWith, equivalentClass (applied to class expressions), rdfs:subClassOf (applied to class expressions)
  - Boolean Combinations of Class Expressions: unionOf, complementOf, intersectionOf
  - Arbitrary Cardinality: minCardinality, maxCardinality cardinality
  - Filler Information: hasValue

Furthermore, in Figures 2.25 and 2.26, how OWL constructors and axioms are described using DL syntax are shown.

2.11 A BRIEF INTRODUCTION TO SPARQL

SPARQL [66] is a query language for RDF graphs. It is similar to Structured Query Language (SQL) and queries are written against the triples of RDF graph. The SPARQL uses the RDF

Axiom	DL Syntax	Example
subClassOf	$C_1 \sqsubseteq C_2$	Human $\sqsubseteq$ Animal $\sqcap$ Biped
equivalentClass	$C_1 \equiv C_2$	Man $\equiv$ Human $\sqcap$ Male
subPropertyOf	$P_1 \sqsubseteq P_2$	hasDaughter $\sqsubseteq$ hasChild
equivalentProperty	$P_1 \equiv P_2$	cost $\equiv$ price
disjointWith	$C_1 \sqsubseteq \neg C_2$	Male $\sqsubseteq \neg$Female
sameAs	$\{x_1\} \equiv \{x_2\}$	$\{$Pres_Bush$\} \equiv \{$G_W_Bush$\}$
differentFrom	$\{x_1\} \sqsubseteq \neg\{x_2\}$	$\{$john$\} \sqsubseteq \neg\{$peter$\}$
TransitiveProperty	P transitive role	hasAncestor is a transitive role
FunctionalProperty	$\top \sqsubseteq (\leqslant 1\ P)$	$\top \sqsubseteq (\leqslant 1$ hasMother$)$
InverseFunctionalProperty	$\top \sqsubseteq (\leqslant 1\ P^-)$	$\top \sqsubseteq (\leqslant 1$ isMotherOf$^-)$
SymmetricProperty	$P \equiv P^-$	isSiblingOf $\equiv$ isSiblingOf$^-$

Figure 2.26: OWL Axioms

view of an OWL ontology. Therefore, it does not benefit from the semantic described in an OWL ontology very effectively. A recent work, called SPARQL-DL [67], is initiated to enhance the expressive power of SPARQL for OWL-DL ontologies. In SPARQL-DL the queries are formalized against the class hierarchy of an OWL-DL ontology. The initiative is very new and as it becomes mature, the SPARQL queries might be migrated to SPARQL-DL. In this thesis, query templates in SPARQL are formulated to facilitate the discovery and reuse of document components in the Harmonized Ontology.

2.12 CONCLUSIONS

Today, an enterprise's competitiveness is to a large extent determined by its ability to seamlessly interoperate with others, and electronic document standards play an important role in this.

Although all the document standards surveyed in this thesis (with the exception of EDI) are based on UN/CEFACT CCTS Methodology, their analysis reveals that there are considerable differences in the resulting document schemas. This is mostly because the standards like OAGIS BODs and GS1 XML existed long before UN/CEFACT CCTS Methodology was proposed, and therefore these standards adapted their existing document schemas rather than starting from fresh. However, all of these standards are still developing, and their future versions may become more harmonized.

In fact, by observing that the divergent and competing approaches to electronic document standardization threatens intersectoral coherence in the field of electronic business, four major standard bodies, namely, the International Electrotechnical Commission (IEC), the International Organization for Standardization (ISO), the International Telecommunication Union (ITU) and the United Nations Economic Commission for Europe (UNECE) signed a "Memorandum of Understanding" to specify a framework of cooperation [42]. In the year 2000, they established a Memorandum of Understanding Meeting Group for eBusiness standards harmonization. Up to now, OAGIS 9.1, UBL 2.0 and UN/CCL have achieved a negligible level of harmonization. However, the harmonization needs to be extended to the upper level artifacts such as the BBIEs and the ABIEs. However having many document standards weakens interoperability.

Given the large number of electronic business document standards, conformance to one of these standards or implementing a combination of them will not solve the interoperability problem; there will always be some companies using a different, incompatible document standards.

Therefore, although the electronic document standards developed so far proved to be very useful for industry and government applications, further efforts are needed for their harmonization and semantic interoperability, and this is the focus of this thesis.

Table 2.1: Document Design Principles

	Document Artifacts	**Use of CCTS Methodology**	**Use of Codelists**	**Use of namespaces**	**Naming and Design Rules**
EDI	Interchange, Message, Segment, Element	Not used	UN/EDIFACT recommends a number of code lists. Local and external codes are also allowed	Not used	UN/EDIFACT Syntax Rules (ISO 9735) or X12.5 and X12.6 Syntax rules
UN/-CCL	Uses CCTS based document artifacts such as Core Component Types and BIEs	Fully based on CCTS Methodology	Defines five code lists: *Country Codes*, *Subdivision Codes*, *Currency Codes*, *BinaryObject Mime Codes* and *Unit Codes*	It is syntax-independent	ISO 11179-5
UBL 2.0	Uses CCTS Artifacts	Fully based on CCTS Methodology	Through a common base type called *CodeType* "xsd:-normalizedString"	Mostly for document categorization	UBL 2.0 Naming and Design Rules
OAGIS 9.0	BODs, Application Areas, Nouns, Verbs, Components, Compounds, Fields	Fields are UDT and CCT based. Some Components are UN/CEFACT ABIE based	Defines two "xsd:simpleType" for each coded *Field*	To identify the *Overlay* extension elements	UN/CEFACT ATG2 Naming and Design Rules
GS1 XML	SBDH, Transactions, Commands, Documents	Use the CCTS methodology to generate its own document artifacts	External Code Lists; Internal Code Lists defined through "xsd:-enumeration"	The namespaces indicate the document context	GS1 XML's UML to XSD conversion rules

Table 2.2: Customization and Extensibility

	Customization	**Extensibility**
EDI	Subsetting EDI documents through context specific Implementation Guidelines.	Introduction of new types of business documents which has to be validated through related EDI Committees.
UN/CCL	Core Components are customized according to eight contexts to create BIEs.	New components can be published to the Core Component Library.
UBL 2.0	Conformant customization through "UBLExtensions" element, or subsetting or placing constraints on the value space.	Compatible customization by reusing the largest suitable aggregation from the UBL Library.
OAGIS 9.0	No formal methodology for defining user specific customizations	Through *User Area* and *Overlay* extensions.
GS1 XML	Through the following three contexts: Business Process, Industry sector, Geopolitical	Through the "extension" element at the end of each document schema.

CHAPTER 3

ONTOLOGY BASED SEMANTIC INTEROPERABILITY OF ELECTRONIC BUSINESS DOCUMENT STANDARDS

Businesses need to exchange data with their trading partners to execute transactions. The partners conforming to the same electronic document standard can interoperate. However, there are very many electronic document standards and therefore there is an interoperability problem among the partners, who conform to different standards. As mentioned previously, UN/CEFACT Core Component Technical Specification (CCTS) is an important landmark in providing a framework to achieve electronic business document interoperability by defining the semantic properties of document artifacts.

As already mentioned, a number of standardization efforts have taken up CCTS Methodology, including UBL, GS1 XML, and OAGIS, in addition to UN/CEFACT's own Core Component Library (CCL). These standards are widely used in e-Government and e-Business applications all over the world.

Although all of these standards are CCTS based, they are not interoperable. The analysis provided in Chapter 2 reveals that there are considerable differences in their document design principles, the use of code lists and the XML namespaces, how they use the CCTS methodology and how they handle extensibility and customization. Furthermore, the current accepted practice of storing the document artifacts in spreadsheets does not facilitate to develop automated semantic interoperability support tools.

Towards solving the interoperability problem, the apporach in this thesis is basically as follows.

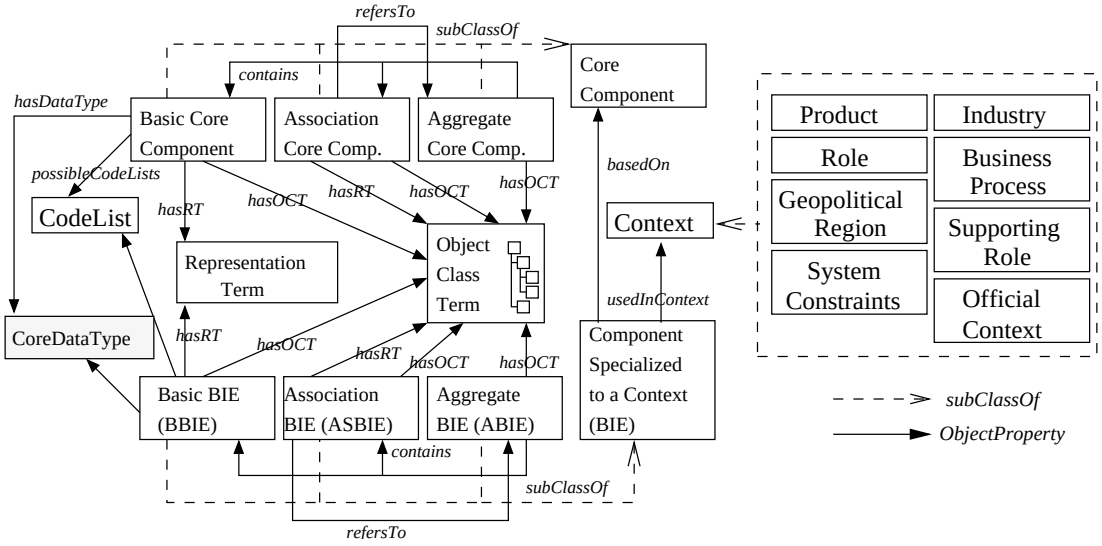


Figure 3.1: The Upper Ontology for the Semantics Exposed by the CCTS Framework

1. As already described, the CCTS defines meaning of document components at various dimensions starting with the data types used in document artifacts extending to the way the document artifacts are composed. To be able to relate the similar document artifacts of different document content models through the semantic properties they share with the CCTS, the CCTS semantics is first explicated using the Web Ontology Language (OWL) and define an upper CCTS ontology as shown in Figure 3.1. This upper ontology gives the CCTS artifacts as ontology classes together with their properties and the relationships among them. Then the upper ontologies for the other CCTS based standards such as UBL, OAGIS 9.1 and GS1 XML, are defined again as upper ontologies and relate their corresponding classes to the CCTS upper ontology as shown in Figure 3.2.

2. At the lower level, the semantics of document schemas from the prominent CCTS based standards are explicated. A document schema ontology is developed for each standard to describe the actual document artifacts as the subclasses of its own upper ontology classes (Figure 3.3). The similarities among document schema ontology classes of different document standards are established through both the semantic properties they share and the semantic equivalences established in the upper ontologies. The full OWL ontology of the semantics described in this section is available at [90].

3. After that some semantics related with the different usages of document data types

in different document schemas are explicated to obtain some desired interpretations by means of informal semantics. The intention is to give the reasoner the same information that the humans use in transforming document schemas into one another.

4. Then through a Description Logics (DL) reasoner, a Harmonized Ontology is computed. The Harmonized Ontology gives the specified as well as the computed equality and subsumption relations among the classes of both the upper ontologies and the document schema ontologies. The Harmonized Ontology is useful for three purposes:

   - It helps to discover equivalence of structurally similar document artifacts between two document schemas.
   - For translating such document artifacts through automatically generated XSLT rules.
   - Query templates (SPARQL and Reasoner based queries) are formulated to facilitate the discovery and reuse of document components using the Harmonized Ontology.

5. Finally, further heuristic rules are provided to identify the similarities between semantically similar but structurally different document components. The semantic properties of the CCTS based document artifacts help discovering the equivalences of structurally similar and semantically equivalent elements. However different document standards use core components in different structures. Semantic properties of document artifacts are not enough to find the similarity of the structurally different but semantically equivalent document artifacts; possible differences in structures must be provided as heuristics to enhance the practical uses of the specified semantics. Note that for defining heuristics to handle structurally different document artifacts, the Description Logic is not sufficient but more general purpose Predicate Logic Rules are needed.

This chapter is focused on the definition of the upper and document schema ontologies, and is organized as follows: Section 3.1 describes how the semantics of CCTS based document content models are explicated (i.e. how to generate the upper ontologies). In Sections 3.2, 3.3 and 3.4 the methodology to explicate the semantics of each document schema through document schema ontologies conforming to its own upper ontology is explicated. Section 3.5 presents harmonizing the ontologies of the document standards. In the next chapters (Chapter

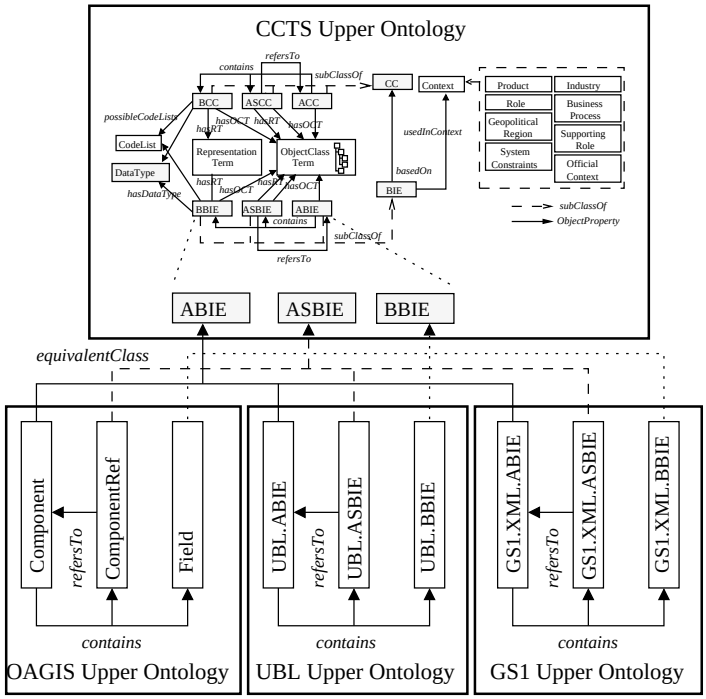


Figure 3.2: An Overview of the Upper Ontologies and their Relationships

4 and 5), the heuristics to discover semantically similar but structurally different document artifacts and the XSLT support are presented.

3.1 EXPLICATING THE SEMANTICS OF CCTS BASED DOCUMENT CONTENT MODELS

The semantics specified by the CCTS approach and given in spreadsheets are explicated basically as follows:

- *The semantics implied by the properties of the document components*: The semantics of *Core Components* implied by their properties as defined by the CCTS are explicated as follows:
  - Each aggregate document component (ACCs and ABIEs) has a *hasObjectClassTerm* (abbreviated as *hasOCT*) object property, whose range is the *Object Class Term* class, as shown in Figure 3.1. As already mentioned in Section 2.3.2, the aggregate components, created from the same core component by restricting them to

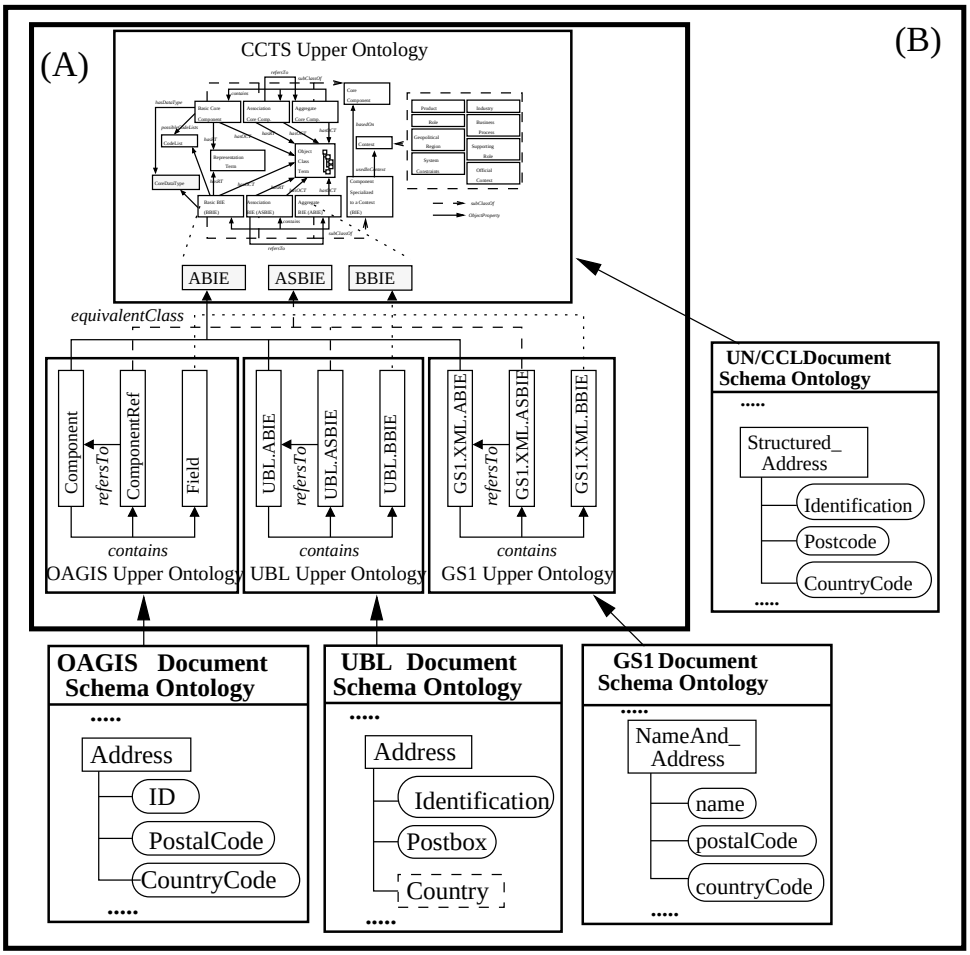


Figure 3.3: An Overview of the Upper Ontologies together with the Document Schema Ontologies

different contexts, share the same *Object Class Term*. Hence, this property contributes to determine similar document artifacts in different standards. As an example, UN/CCL has an aggregate document component called "Structured_Address" and UBL has "Address". Both of these document components have the same *Object Class Term*. Hence, this semantic property helps to find the similarity of these two document components. Note however that the *Object Class Term* cannot be the only semantics that determine the similarity of the document components and different standards may use different *Object Class Terms* for the same document component. For example, the *Object Class Terms* of two similar aggregate components are "Monetary Total" in UBL and "Monetary Summation" in UN/CCL. Therefore the *hasOCT* property is used together with other document properties, either explicitly defined or inferred, to determine document component similarity.

– Each basic document component (BCCs and BBIEs) also has a *hasObjectClassTerm* object property, whose range is the *Object Class Term* class and has a *hasRepresentationTerm* (abbreviated as *hasRT*) object property, whose range is the *Representation Term* class, as shown in Figure 3.1. As an example, UN/CCL has a basic document component "Identification", whose *Representation Term* is "Identifier". Likewise, UBL has a basic document component called "Identifier" whose *Representation Term* is also "Identifier". Hence, this semantic property contributes to find the similarity of these two document components. Again, different standards may use different *Representation Terms* and the similarity of this term alone cannot determine the similarity of document artifacts. Therefore the *hasRT* property is used together with other document properties.

– Each association document component (ASCCs and ASBIEs) has a *hasOCT* object property, whose range is the *Object Class Term* class and a *hasRT* object property, whose range is the *Object Class Term* class of the aggregate document component it refers to, as shown in Figure 3.1. As an example, UN/CCL's aggregate document component "Seller_Party" has an association document component called "Postal", whose *Representation Term* is "Address". On the other hand, UBL's "Party" aggregate component has an association document component called "PostalAddress", whose *Representation Term* is also "Address", helping to identify their similarity.

– The *Property Terms* of document components used in different standards show a wide diversity. For example, UBL uses "Postal_Zone" whereas UN/CCL uses "Postcode" for the same element (Figure 3.7). Hence, using the property terms as semantics does not help the reasoner with its purpose of discovering the similar elements. Therefore, the property terms are not used in the reasoning process. However, when matching document components through their semantic properties, a number of false positives may appear. In my approach, the *Property Terms* are used to reduce the number of false-positives while post-processing the result set. In the post-processing, each component in the inferred result set is assigned a similarity value, which is computed by comparing their *Property Terms* lexically. For the computation of the similarity value, the algorithm in [69] is used. For example, the lexical similarity value between "Postal_Zone" and "Postcode" turns out to be 0.7. The document components in the inferred result set are presented to

a human user together with their similarity value.

- *The semantics implied by the Business Information Entities*: *Business Information Entities* (BIEs) are derived from *Core Components* to be used in a certain context and this semantics is expressed through two OWL object properties: the *basedOn* object property indicates that the *Core Component* from which the BIE is derived and the *usedInContext* object property indicates the context of the BIE, as shown in Figure 3.1. The BIEs also inherit some of the properties of their corresponding *Core Components*, but it should be noted that there is no subclass relationship among them, since a BIE, being a restriction of a *Core Component* to a context, may not inherit all the properties of that *Core Component*. The Basic Core Components (BCCs), the Association Core Components (ASCCs) and the Aggregate Core Components (ACCs) are specialization of the Core Components and therefore they are defined as subclasses of the Core Components (CCs). Likewise, The Basic Business Information Entities (BBIEs), the Association Business Information Entities (ASBIEs) and the Aggregate Business Information Entities (ABIEs) are specialization of the Business Information Entities and therefore they are defined as subclasses of the Business Information Entities (BIEs).

- *Data type semantics*: CCTS provides a fixed set of reusable *Core Component Types (CCT)* and *Data Types (DT)* (which are also termed as *Core Data Types (CDTs)*) such as *Amount*, *Identifier*, or *Measure* for consistent business value representation. The *Core Data Type* semantics is explicated through the *CoreComponentType* class. For each of the 14 CDTs, a corresponding OWL class is created and inserted as the subclass of *CoreComponentType* class.

- *The semantics exposed by the contexts, in which document artifacts are used*: CCTS has established predefined context categories, like geopolitical region, industry or business process that identify the usage meaning of a document artifact. To explicate this semantics, first an OWL class is created for context concept, called *Context* and the context categories defined by UN/CEFACT are defined as subclasses of the *Context* class.

- *The semantics exposed by the use of the code lists*: The code lists are used to convey the meaning of the values in the elements of the document artifacts. An OWL class for the code list concept is created and for its classification the identified context categories are used.

- *The semantics implied by the structure of the Document Components*: As already mentioned, the aggregate document components are composed of either basic document components or association document components. This semantics is described through the *contains* OWL object property of an aggregate document component which denotes the basic document components and the association document components it contains. Each association document component has an OWL object property called *refersTo* whose range is the aggregate document component it refers.

In the following subsections, more details on the definition of these mentioned semantics is described.

3.1.1 SPECIFICATION OF THE SEMANTICS EXPOSED BY THE CCTS FRAMEWORK THROUGH OWL

This section specifies how the existing semantics in the CCTS Framework document artifacts can be explicated by using OWL constructs so that this semantics can later be used in an automated manner to discover useful implicit relationships among the document artifacts of other CCTS based standards.

3.1.1.1 EXPLICATING SEMANTICS THROUGH CORE DATA TYPES (CDT)

First lets provide some insight on why there is a need to explicate the data type semantics: UN/CEFACT CCTS defines 14 CCTs (which are also termed as *Core Data Types (CDT)*. When two document artifacts use the same CCT, this can be considered as a hint towards these artifacts meaning the same thing if their other semantic properties also match.

The *Core Component Type* semantics is explicated through the *CoreComponentType* class. UN/CEFACT CCTS defines 14 CCTs and for each of them, a corresponding class is created and inserted as the subclass of *CoreComponentType* as follows:

```
<owl:Class rdf:ID="CoreComponentType" />
  <owl:Class rdf:ID="Amount.Type">
    <rdfs:subClassOf rdf:resource="#CoreComponentType"/>
  </owl:Class>
  <owl:Class rdf:ID="BinaryObject.Type">
```

```
    <rdfs:subClassOf rdf:resource="#CoreComponentType"/>
  </owl:Class>
  <owl:Class rdf:ID="Code.Type">
    <rdfs:subClassOf rdf:resource="#CoreComponentType"/>
  </owl:Class>
  <owl:Class rdf:ID="Date.Type">
    <rdfs:subClassOf rdf:resource="#CoreComponentType"/>
  </owl:Class>
  <owl:Class rdf:ID="DateTime.Type">
    <rdfs:subClassOf rdf:resource="#CoreComponentType"/>
  </owl:Class>
  <owl:Class rdf:ID="Identifier.Type">
    <rdfs:subClassOf rdf:resource="#CoreComponentType"/>
  </owl:Class>
  <owl:Class rdf:ID="Indicator.Type">
    <rdfs:subClassOf rdf:resource="#CoreComponentType"/>
  </owl:Class>
  <owl:Class rdf:ID="Measure.Type">
    <rdfs:subClassOf rdf:resource="#CoreComponentType"/>
  </owl:Class>
  <owl:Class rdf:ID="Numeric.Type">
    <rdfs:subClassOf rdf:resource="#CoreComponentType"/>
  </owl:Class>
  <owl:Class rdf:ID="Quantity.Type">
    <rdfs:subClassOf rdf:resource="#CoreComponentType"/>
  </owl:Class>
  <owl:Class rdf:ID="Text.Type">
    <rdfs:subClassOf rdf:resource="#CoreComponentType"/>
  </owl:Class>
  <owl:Class rdf:ID="Percent.Type">
    <rdfs:subClassOf rdf:resource="#CoreComponentType"/>
  </owl:Class>
....
```

3.1.1.2 EXPLICATING SEMANTICS THROUGH CONTEXT

The context in which a document artifact is used gives it a certain semantics. Therefore if two document arifacts have related contexts and if their other semantic properties are related, this gives a hint on their their possible equivalence. There is an OWL class for context concept, called *Context* as follows:

```
<owl:Class rdf:ID="Context" />
```

The context categories defined by UN/CEFACT are defined as subclasses of *Context* class as follows:

```
<owl:Class rdf:ID="BusinessProcessContext" >
  <rdfs:subClassOf rdf:resource="#Context"/>
</owl:Class>

<owl:Class rdf:ID="GeopoliticalContext" >
  <rdfs:subClassOf rdf:resource="#Context"/>
</owl:Class>

<owl:Class rdf:ID="IndustryContext" >
  <rdfs:subClassOf rdf:resource="#Context"/>
</owl:Class>

<owl:Class rdf:ID="ProductContext" >
  <rdfs:subClassOf rdf:resource="#Context"/>
</owl:Class>

<owl:Class rdf:ID="RoleContext" >
  <rdfs:subClassOf rdf:resource="#Context"/>
</owl:Class>

<owl:Class rdf:ID="OfficialContext" >
  <rdfs:subClassOf rdf:resource="#Context"/>
</owl:Class>

<owl:Class rdf:ID="SupportingRoleContext" >
  <rdfs:subClassOf rdf:resource="#Context"/>
</owl:Class>

<owl:Class rdf:ID="SystemConstraintsContext" >
  <rdfs:subClassOf rdf:resource="#Context"/>
</owl:Class>
```

For each *Context* subclass, such as “IndustryContext”, an ontology can be defined based on the taxonomies or classifications already used by the industry such as Universal Standard Product and Service Specification (UNSPSC) [89] or Standard International Trade Classification (SITC) [65].

3.1.1.3 EXPLICATING SEMANTICS THROUGH CODE LISTS

The code lists are important to identify the meaning of a BCC or BBIE. As an example, assume that two document standards name a BBIE differently. However if BBIEs use the same code list or use code lists for the same purpose, there is a possibility that they are similar. It should be noted that code lists used for a BCC or BBIE can vary according to context. Therefore, the classification of code lists is also important. For the classification

categories, the identified context categories can be used. There is an OWL class for *Codelist* concept, called *CodeList*:

```
<owl:Class rdf:ID="CodeList" />
  <owl:Class rdf:ID="BusinessProcessCodeList" >
    <rdfs:subClassOf rdf:resource="#CodeList"/>
  </owl:Class>
  <owl:Class rdf:ID="GeopoliticalCodeList" >
    <rdfs:subClassOf rdf:resource="#CodeList"/>
  </owl:Class>
  <owl:Class rdf:ID="IndustryCodeList" >
    <rdfs:subClassOf rdf:resource="#CodeList"/>
  </owl:Class>
  <owl:Class rdf:ID="ProductCodeList" >
    <rdfs:subClassOf rdf:resource="#CodeList"/>
  </owl:Class>
  <owl:Class rdf:ID="RoleCodeList" >
    <rdfs:subClassOf rdf:resource="#CodeList"/>
  </owl:Class>
  <owl:Class rdf:ID="OfficialCodeList" >
    <rdfs:subClassOf rdf:resource="#CodeList"/>
  </owl:Class>
  <owl:Class rdf:ID="SupportingRoleCodeList" >
    <rdfs:subClassOf rdf:resource="#CodeList"/>
  </owl:Class>
  <owl:Class rdf:ID="SystemConstraintsCodeList" >
    <rdfs:subClassOf rdf:resource="#CodeList"/>
  </owl:Class>
```

Once the *CodeList* subclasses are defined, the specific code lists in use are defined as a subclass of the related context. Some examples are provided as follows:

```
  <owl:Class rdf:ID="iso-ch.3166.1999" >
    <rdfs:subClassOf rdf:resource="#GeopoliticalCodeList"/>
  </owl:Class>

  <owl:Class rdf:ID="ntis-gov.naics.1997" >
    <rdfs:subClassOf rdf:resource="#IndustryCodeList"/>
  </owl:Class>

  <owl:Class rdf:ID="unspsc-org.unspsc.3-1" >
    <rdfs:subClassOf rdf:resource="#ProductCodeList"/>
  </owl:Class>
```

3.1.1.4 EXPLICATING SEMANTICS OF CORE COMPONENTS

There are a number of terms giving meaning to the CCs. When such semantics is explicated in an ontology, it may help to find similarities in document artifacts from different document schemas. For example, if two document artifacts have the same *Object Class Term*, this may give a hint on their similarity.

A CCTS Core component is expressed as an OWL class as follows:

```
<owl:Class rdf:ID="CoreComponent"/>
```

The following OWL classes are defined to represent these terms as follows:

```
<owl:Class rdf:ID="ObjectClassTerm" />
<owl:Class rdf:ID="RepresentationTerm" />
```

A CCTS Basic Core Component (BCC) is defined as an OWL class to have the following object properties *hasDataType*, *hasObjectClassTerm*, *hasRepresentationTerm*, and *possible-CodeLists*:

```
  <owl:Class rdf:ID="BasicCoreComponent" >
    <owl:equivalentClass>
      <owl:Class>
<owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#CoreComponent"/>
  <owl:Restriction>
    <owl:onProperty rdf:resource="#hasDataType"/>
    <owl:allValuesFrom>
      <owl:Class rdf:about="#DataType"/>
    </owl:allValuesFrom>
  </owl:Restriction>
  <owl:Restriction>
    <owl:onProperty rdf:resource="#hasObjectClassTerm"/>
    <owl:allValuesFrom>
      <owl:Class rdf:about="#ObjectClassTerm"/>
    </owl:allValuesFrom>
  </owl:Restriction>
   <owl:Restriction>
    <owl:onProperty rdf:resource="#hasRepresentationTerm"/>
    <owl:allValuesFrom>
      <owl:Class rdf:about="#RepresentationTerm"/>
    </owl:allValuesFrom>
```

```
  </owl:Restriction>
  <owl:Restriction>
    <owl:onProperty rdf:resource="#possibleCodeLists"/>
    <owl:allValuesFrom>
      <owl:Class rdf:about="#CodeList"/>
    </owl:allValuesFrom>
  </owl:Restriction>
</owl:intersectionOf>
      </owl:Class>
    </owl:equivalentClass>
  </owl:Class>
```

ACCs may contain BCCs and ASCCs, and ACCs only have Object Class Terms. A CCTS Aggregate Core Component is defined as an OWL class to have the following object properties *contains* and *hasObjectClassTerm* as follows:

```
<owl:Class rdf:ID="AggregateCoreComponent" >
    <owl:equivalentClass>
      <owl:Class>
<owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#CoreComponent"/>
  <owl:Restriction>
    <owl:onProperty rdf:resource="#contains"/>
    <owl:allValuesFrom>
      <owl:Class>
<owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#BasicCoreComponent"/>
  <owl:Class rdf:about="#AssociationCoreComponent"/>
</owl:intersectionOf>
      </owl:Class>
    </owl:allValuesFrom>
  </owl:Restriction>
  <owl:Restriction>
    <owl:onProperty rdf:resource="#hasObjectClassTerm"/>
    <owl:allValuesFrom>
      <owl:Class rdf:about="#ObjectClassTerm"/>
    </owl:allValuesFrom>
  </owl:Restriction>
</owl:intersectionOf>
      </owl:Class>
    </owl:equivalentClass>
  </owl:Class>
```

A CCTS Association Core Component is defined as an OWL class to have the following object properties *refersTo*, *hasObjectClassTerm*, and *hasAssociatedObjectClassTerm*:

```
    <owl:Class rdf:ID="AssociationCoreComponent" >
    <owl:equivalentClass>
      <owl:Class>
<owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#CoreComponent"/>
  <owl:Restriction>
    <owl:onProperty rdf:resource="#refersTo"/>
    <owl:allValuesFrom>
      <owl:Class rdf:about="#AggregateCoreComponent"/>
    </owl:allValuesFrom>
  </owl:Restriction>
  <owl:Restriction>
    <owl:onProperty rdf:resource="#hasObjectClassTerm"/>
    <owl:allValuesFrom>
      <owl:Class rdf:about="#ObjectClassTerm"/>
    </owl:allValuesFrom>
  </owl:Restriction>
  <owl:Restriction>
  <owl:Restriction>
    <owl:onProperty rdf:resource="#hasAssociatedObjectClassTerm"/>
    <owl:allValuesFrom>
      <owl:Class rdf:about="#ObjectClassTerm"/>
    </owl:allValuesFrom>
  </owl:Restriction>
</owl:intersectionOf>
      </owl:Class>
    </owl:equivalentClass>
  </owl:Class>
```

Note that, as mentioned previously the *Property Terms* is not used in the ontology definition since the usage of property terms show a great degree of variances and hence do not contribute to discovering similarities among document artifacts. However, the *Property Term* is used to decrease the number of false-positives.

3.1.1.5 EXPLICATING SEMANTICS OF BUSINESS INFORMATION ENTITIES (BIE)

The semantics of a BIE is given by the core component from which it is derived and the context it is constrained to. A BIE is expressed as an OWL class as follows:

```
<owl:Class rdf:ID="BusinessInformationEntity"/>
```

The *BasicBusinessInformationEntity* class is based on *BasicCoreComponentClass* and this is expressed as follows:

```
<owl:ObjectProperty rdf:ID="basedOn">
    <rdfs:domain rdf:resource="#BusinessInformationEntity"/>
    <rdfs:range rdf:resource="#CoreComponent"/>
  </owl:ObjectProperty
```

A BIE is a CC used in a context and there is an *owl:ObjectProperty* called *usedInContext*. This object property has *BusinessInformationEntity* class as its domain and *Context* class as its range as follows:

```
<owl:ObjectProperty rdf:ID="usedInContext">
    <rdfs:domain rdf:resource="#BusinessInformationEntity"/>
    <rdfs:range rdf:resource="#Context"/>
  </owl:ObjectProperty>
```

Just like a *BasicCoreComponent* it is derived from, a BIE has Object Properties for its data type, naming terms and possible code lists but there is no subclass relationship among them since a BIE, being a restriction of a Core Component to a context, may not inherit all the properties of that Core Component.

A CCTS *BasicBusinessInformationEntity (BIE)* is defined as an OWL class to have the following object properties *hasDataType*, *hasObjectClassTerm*, *hasRepresentationTerm*, and *possibleCodeLists* as follows:

```
<owl:Class rdf:ID="BasicBusinessInformationEntity" >
    <owl:equivalentClass>
      <owl:Class>
<owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#BusinessInformationEntity"/>
  <owl:Restriction>
    <owl:onProperty rdf:resource="#basedOn"/>
    <owl:allValuesFrom>
      <owl:Class rdf:about="#BasicCoreComponent"/>
    </owl:allValuesFrom>
  </owl:Restriction>
  <owl:Restriction>
    <owl:onProperty rdf:resource="#hasDataType"/>
    <owl:allValuesFrom>
      <owl:Class rdf:about="#DataType"/>
    </owl:allValuesFrom>
  </owl:Restriction>
  <owl:Restriction>
    <owl:onProperty rdf:resource="#hasObjectClassTerm"/>
    <owl:allValuesFrom>
```

```
      <owl:Class rdf:about="#ObjectClassTerm"/>
    </owl:allValuesFrom>
  </owl:Restriction>
   <owl:Restriction>
    <owl:onProperty rdf:resource="#hasRepresentationTerm"/>
    <owl:allValuesFrom>
      <owl:Class rdf:about="#RepresentationTerm"/>
    </owl:allValuesFrom>
  </owl:Restriction>
  <owl:Restriction>
    <owl:onProperty rdf:resource="#possibleCodeLists"/>
    <owl:allValuesFrom>
      <owl:Class rdf:about="#CodeList"/>
    </owl:allValuesFrom>
  </owl:Restriction>
</owl:intersectionOf>
      </owl:Class>
    </owl:equivalentClass>
  </owl:Class>
```

An *AssociationBusinessInformationEntity* (ASBIE) is defined as an OWL class to have the following object properties *refersTo*, *hasObjectClassTerm*, *hasAssociatedObjectClassTerm*:

```
<owl:Class rdf:ID="AssociationBusinessInformationEntity" >
    <owl:equivalentClass>
      <owl:Class>
<owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#BusinessInformationEntity"/>
  <owl:Restriction>
    <owl:onProperty rdf:resource="#basedOn"/>
    <owl:allValuesFrom>
      <owl:Class rdf:about="#AssociationCoreComponent"/>
    </owl:allValuesFrom>
  </owl:Restriction>
  <owl:Restriction>
    <owl:onProperty rdf:resource="#refersTo"/>
    <owl:allValuesFrom>
      <owl:Class rdf:about="#AggregateBusinessInformationEntity"/>
    </owl:allValuesFrom>
  </owl:Restriction>
  <owl:Restriction>
    <owl:onProperty rdf:resource="#hasObjectClassTerm"/>
    <owl:allValuesFrom>
      <owl:Class rdf:about="#ObjectClassTerm"/>
    </owl:allValuesFrom>
  </owl:Restriction>
  <owl:Restriction>
```

```
    <owl:onProperty rdf:resource="#hasAssociatedObjectClassTerm"/>
    <owl:allValuesFrom>
      <owl:Class rdf:about="#ObjectClassTerm"/>
    </owl:allValuesFrom>
  </owl:Restriction>
  <owl:Restriction>
</owl:intersectionOf>
      </owl:Class>
    </owl:equivalentClass>
  </owl:Class>
```

An *AggregateBusinessInformationEntity* (ABIE) is defined as an OWL class to have the following object properties *contains*, *basedOn*, and *hasObjectClassTerm* as follows:

```
<owl:Class rdf:ID="AggregateBusinessInformationEntity" >
    <owl:equivalentClass>
      <owl:Class>
<owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#BusinessInformationEntity"/>
  <owl:Restriction>
    <owl:onProperty rdf:resource="#contains"/>
    <owl:allValuesFrom>
      <owl:Class>
<owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#BasicBusinessInformationEntity"/>
  <owl:Class rdf:about="#AssociationBusinessInformationEntity"/>
</owl:intersectionOf>
      </owl:Class>
    </owl:allValuesFrom>
  </owl:Restriction>
  <owl:Restriction>
    <owl:onProperty rdf:resource="#basedOn"/>
    <owl:allValuesFrom>
      <owl:Class rdf:about="#AggregateCoreComponent"/>
    </owl:allValuesFrom>
  </owl:Restriction>
  <owl:Restriction>
    <owl:onProperty rdf:resource="#hasObjectClassTerm"/>
    <owl:allValuesFrom>
      <owl:Class rdf:about="#ObjectClassTerm"/>
    </owl:allValuesFrom>
  </owl:Restriction>
</owl:intersectionOf>
      </owl:Class>
    </owl:equivalentClass>
  </owl:Class>
```

3.1.1.6 EXPLICATING THE SEMANTICS OF CCL ARTIFACTS

To be able to determine the semantically similar document artifacts at the schema level, the semantics of each document schema is explicated conforming to its own upper ontology. The relationships among the document schema ontology classes are established through reasoning process by using the explicit relationships defined among the upper ontology classes.

The semantics of UN/CEFACT Core Component Library (CCL) artifacts are explicated conforming to the CCTS Upper Ontology defined. The generation of ontologies from the artifacts defined in CCL conforming to the CCTS upper ontology as follows: To create the ontology classes corresponding to the CCL artifacts which are given in MS Excel spreadsheets, the CCL spreadsheets are first converted to a custom XML format by using XML Map mechanism of MS Excel. Then, through a piece of software developed, the necessary OWL classes conforming to the specified CCTS upper ontology are created from this XML file.

An example on how ”Structured_Address.Details” artifact of CCL is represented in the Harmonized Ontology conforming to the CCTS Upper Ontology is as follows:

```
<owl:Class rdf:ID="Structured_Address.Details">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
<owl:Class rdf:about="#AggregateBusinessInformationEntity"/>
<owl:Restriction>
  <owl:onProperty rdf:resource="#hasObjectClassTerm"/>
  <owl:allValuesFrom>
    <owl:Class rdf:about="#Address"/>
  </owl:allValuesFrom>
</owl:Restriction>
<owl:Restriction>
  <owl:onProperty rdf:resource="#usedInContext"/>
  <owl:allValuesFrom>
    <owl:Class rdf:about="#Context"/>
  </owl:allValuesFrom>
</owl:Restriction>
<owl:Restriction>
  <owl:onProperty rdf:resource="#contains"/>
  <owl:allValuesFrom>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
<owl:Class rdf:about="#Structured_Address.Identification.Identifier"/>
<owl:Class rdf:about="#Structured_Address.Postcode.Code"/>
```

```
<owl:Class rdf:about="#Structured_Address.BuildingName.Text"/>
<owl:Class rdf:about="#Structured_Address.StreetName.Text"/>
<owl:Class rdf:about="#Structured_Address.CityName.Text"/>
<owl:Class rdf:about="#Structured_Address.Country.Identifier"/>
<owl:Class rdf:about="#Structured_Address.CitySub-DivisionName.Text"/>
<owl:Class rdf:about="#Structured_Address.CountryName.Text"/>
<owl:Class rdf:about="#Structured_Address.CountrySub-DivisionName.Text"/>
<owl:Class rdf:about="#Structured_Address.BlockName.Text"/>
<owl:Class rdf:about="#Structured_Address.PlotIdentification.Text"/>
      </owl:intersectionOf>
    </owl:Class>
  </owl:allValuesFrom>
</owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
```

The following listing provides an example on how ”Structured_Address.Identification.Identifier” artifact of CCL is represented in the Harmonized Ontology conforming to the CCTS Upper Ontology:

```
<owl:Class rdf:ID="Structured_Address.Identification.Identifier">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
<owl:Class rdf:about="#BasicBusinessInformationEntity"/>
<owl:Restriction>
  <owl:onProperty rdf:resource="#hasObjectClassTerm"/>
  <owl:allValuesFrom>
    <owl:Class rdf:about="#Address"/>
  </owl:allValuesFrom>
</owl:Restriction>
<owl:Restriction>
  <owl:onProperty rdf:resource="#hasRepresentationTerm"/>
  <owl:allValuesFrom>
    <owl:Class rdf:about="#Identifier"/>
  </owl:allValuesFrom>
</owl:Restriction>
<owl:Restriction>
  <owl:onProperty rdf:resource="#usedInContext"/>
  <owl:allValuesFrom>
    <owl:Class rdf:about="#Context"/>
  </owl:allValuesFrom>
</owl:Restriction>
```

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#hasDataType"/>
  <owl:allValuesFrom>
    <owl:Class rdf:about="#Identifier.Type"/>
  </owl:allValuesFrom>
</owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
```

3.2 EXPLICATING THE SEMANTICS OF CCTS BASED DOCUMENT SCHEMAS - GS1 UPPER ONTOLOGY

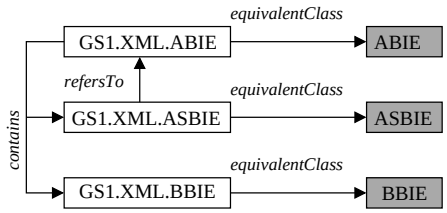


Figure 3.4: The Upper Ontology for the Semantics Exposed by the GS1 XML Document Standard

Figure 3.4 shows the upper ontology for the GS1 XML document standard and as shown in this figure GS1 classes are related with the corresponding CCTS classes by using the *owl:equivalentClass* property.

A GS1 BBIE is defined as an OWL class named *GS1.XML.BBIE* and it is declared equivalent to the BBIE class defined in CCTS upper ontology as follows:

```
<owl:Class rdf:ID="GS1.XML.BBIE">
  <owl:equivalentClass rdf:resource="#BasicBusinessInformationEntity"/>
</owl:Class>
```

A GS1 ABIE is defined as an OWL class named *GS1.XML.ABIE* and it is declared equivalent to the ABIE class defined in CCTS upper ontology. The *contains* Object Property of the *GS1.XML.ABIE* class is restricted to *GS1.XML.BBIE* and *GS1.XML.ASBIE* as follows:

```
<owl:Class rdf:ID="GS1.XML.ABIE">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
<owl:Class rdf:about="#AggregateBusinessInformationEntity"/>
<owl:Restriction>
  <owl:onProperty rdf:resource="#contains"/>
  <owl:allValuesFrom>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
<owl:Class rdf:about="#GS1.XML.BBIE"/>
<owl:Class rdf:about="#GS1.XML.ASBIE"/>
      </owl:intersectionOf>
```

```
    </owl:Class>
  </owl:allValuesFrom>
</owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
```

A GS1 ASBIE is defined as an OWL class named *GS1.XML.ASBIE* and it is declared equivalent to the ASBIE class defined in CCTS upper ontology. The *refersTo* Object Property of the *GS1.XML.ASBIE* class is restricted to *GS1.XML.ABIE* as follows:

```
<owl:Class rdf:ID="GS1.XML.ASBIE">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
<owl:Class rdf:about="#AssociationBusinessInformationEntity"/>
<owl:Restriction>
  <owl:onProperty rdf:resource="#refersTo"/>
  <owl:allValuesFrom>
    <owl:Class rdf:about="#GS1.XML.ABIE"/>
  </owl:allValuesFrom>
</owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
```

3.2.1 EXPLICATING THE SEMANTICS OF GS1 DOCUMENT SCHEMAS

GS1 XML defines the *Business Information Entites* either in “pdf” Business Message Specifications or through the Global Data Dictionary (GDD). The GDD is a Web accessible registry, where a component is queried by its name. When the “%” character is entered to the search box, all of the BIEs are returned to the user. Through the browser, only the entity names and the component that contains the entity is displayed, however in the HTML source, the type of the entity (e.g. ABIE, ASBIE or BBIE), the id of the entity and the id of the ABIE it belongs to are also available. In order to create the OWL ontology corresponding to the GS1 XML artifacts, the HTML code is processed and the created classes are inserted to the ontology as the subclasses. As an example, the HTML code shown in the following fragment is for “companyNumber” concept.

```
<tr><td width='375' valign='top' rowspan='7'>companyNumber</td>
<td><a href="javascript:doDisplayDoc('2.1.0', 'AIDC: GS1 Company Prefix', '723', '6383',
'BBIE')">AIDC: GS1 Company Prefix</a></td></tr>
<td><a href="javascript:doDisplayDoc('2.1.0', 'AIDC: Global Location Number', '724', '6383',
'BBIE')">AIDC: Global Location Number</a></td></tr>
<td><a href="javascript:doDisplayDoc('2.1.0', 'AIDC: Global Service Relation Number', '725',
'6383', 'BBIE')">AIDC: Global Service Relation Number</a></td></tr>
<td><a href="javascript:doDisplayDoc('2.1.0', 'AIDC: GlobalReturnableAssetIdentifier', '726',
'6383', 'BBIE')">AIDC: GlobalReturnableAssetIdentifier</a></td></tr>
<td><a href="javascript:doDisplayDoc('2.1.0', 'AIDC: GlobalIndividualAssetIdentifier', '727',
'6383', 'BBIE')">AIDC: GlobalIndividualAssetIdentifier</a></td></tr>
<td><a href="javascript:doDisplayDoc('2.1.0', 'AIDC: Global Document Type Identifier', '728',
'6383', 'BBIE')">AIDC: Global Document Type Identifier</a></td></tr>
```

As shown in this listing, the “companyNumber” has as its id “6383” and it is a BBIE. Furthermore, it exists in components numbered from 723 to 728.

In order to create GS1 XML OWL ontology, the HTML code is processed and the created classes are inserted to the ontology as the subclasses of GS1 *Business Information Entity*.

The following listing provides an example on how “NameAndAddress.Details” artifact of GS1 is represented conforming to the GS1 Upper Ontology.

```
<owl:Class rdf:ID="NameAndAddress.Details">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
<owl:Class rdf:about="#GS1.XML.ABIE"/>
<owl:Restriction>
  <owl:onProperty rdf:resource="#hasObjectClassTerm"/>
  <owl:allValuesFrom>
    <owl:Class rdf:about="#Address"/>
  </owl:allValuesFrom>
</owl:Restriction>
<owl:Restriction>
  <owl:onProperty rdf:resource="#usedInContext"/>
  <owl:allValuesFrom>
    <owl:Class rdf:about="#Context"/>
  </owl:allValuesFrom>
</owl:Restriction>
<owl:Restriction>
  <owl:onProperty rdf:resource="#contains"/>
  <owl:allValuesFrom>
    <owl:Class>
```

```
      <owl:intersectionOf rdf:parseType="Collection">
<owl:Class rdf:about="#city"/>
<owl:Class rdf:about="#cityCode"/>
<owl:Class rdf:about="#countryCode"/>
<owl:Class rdf:about="#countyCode"/>
<owl:Class rdf:about="#crossStreet"/>
<owl:Class rdf:about="#currency"/>
<owl:Class rdf:about="#languageOfTheParty"/>
<owl:Class rdf:about="#name"/>
<owl:Class rdf:about="#pOBoxNumber"/>
<owl:Class rdf:about="#postalCode"/>
<owl:Class rdf:about="#provinceCode"/>
<owl:Class rdf:about="#state"/>
<owl:Class rdf:about="#streetAddressOne"/>
<owl:Class rdf:about="#streetAddressTwo"/>
<owl:Class rdf:about="#geographicalCoordinates"/>
      </owl:intersectionOf>
    </owl:Class>
  </owl:allValuesFrom>
</owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
```

The following provides an example on how ”Address” artifact of GS1 is represented conforming to the GS1 Upper Ontology.

```
<owl:Class rdf:about="#Address">
  <rdfs:subClassOf rdf:resource="#ObjectClassTerm"/>
</owl:Class>
<owl:Class rdf:ID="city">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
<owl:Class rdf:about="#GS1.XML.BBIE"/>
<owl:Restriction>
  <owl:onProperty rdf:resource="#hasObjectClassTerm"/>
  <owl:allValuesFrom>
    <owl:Class rdf:about="#Address"/>
  </owl:allValuesFrom>
</owl:Restriction>
<owl:Restriction>
  <owl:onProperty rdf:resource="#hasRepresentationTerm"/>
  <owl:allValuesFrom>
```

```
    <owl:Class rdf:about="#Text"/>
  </owl:allValuesFrom>
</owl:Restriction>
<owl:Restriction>
  <owl:onProperty rdf:resource="#usedInContext"/>
  <owl:allValuesFrom>
    <owl:Class rdf:about="#Context"/>
  </owl:allValuesFrom>
</owl:Restriction>
<owl:Restriction>
  <owl:onProperty rdf:resource="#hasDataType"/>
  <owl:allValuesFrom>
    <owl:Class rdf:about="#Text.Type"/>
  </owl:allValuesFrom>
</owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
```

3.3 EXPLICATING THE SEMANTICS OF CCTS BASED DOCUMENT SCHEMAS - UBL UPPER ONTOLOGY

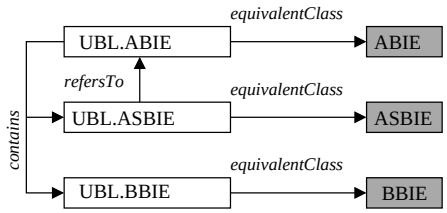


Figure 3.5: The Upper Ontology for the Semantics Exposed by the UBL XML Document Standard

Figure 3.5 shows the upper ontology for the UBL document standard and in this figure UBL classes are related with the corresponding CCTS classes by using the *owl:equivalentClass* property.

A UBL BBIE is defined as an OWL class named *UBL.BBIE* and it is declared equivalent to the BBIE class defined in CCTS upper ontology as follows:

```
<owl:Class rdf:ID="UBL.BBIE">
  <owl:equivalentClass rdf:resource="#BasicBusinessInformationEntity"/>
</owl:Class>
```

A UBL ABIE is defined as an OWL class named *UBL.ABIE* and it is declared equivalent to the ABIE class defined in CCTS upper ontology. The *contains* Object Property of the *UBL.ABIE* class is restricted to *UBL.BBIE* and *UBL.ASBIE* as follows:

```
<owl:Class rdf:ID="UBL.ABIE">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
<owl:Class rdf:about="#AggregateBusinessInformationEntity"/>
<owl:Restriction>
  <owl:onProperty rdf:resource="#contains"/>
  <owl:allValuesFrom>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
<owl:Class rdf:about="#UBL.BBIE"/>
<owl:Class rdf:about="#UBL.ASBIE"/>
      </owl:intersectionOf>
```

```
    </owl:Class>
  </owl:allValuesFrom>
</owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
```

A UBL ASBIE is defined as an OWL class named *UBL.ASBIE* and it is declared equivalent to the ASBIE class defined in CCTS upper ontology. The *refersTo* Object Property of the *UBL.ASBIE* class is restricted to *UBL.ABIE* as follows:

```
<owl:Class rdf:ID="UBL.ASBIE">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
<owl:Class rdf:about="#AssociationBusinessInformationEntity"/>
<owl:Restriction>
  <owl:onProperty rdf:resource="#refersTo"/>
  <owl:allValuesFrom>
    <owl:Class rdf:about="#UBL.ABIE"/>
  </owl:allValuesFrom>
</owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
```

3.3.1 EXPLICATING THE SEMANTICS OF UBL DOCUMENT SCHEMAS

The semantics of UBL artifacts are explicated conforming to the UBL Upper Ontology defined. In UBL, the BIEs are provided in MS Excel spreadsheets. To create UBL artifacts ontology conforming to the specified UBL upper ontology, the UBL spreadsheets are first converted to a custom XML format by using XML Map mechanism of MS Excel. Then, the necessary OWL classes are created from this XML file and populated in the OWL ontology.

The following fragment provides an example on how ”Address.Details” artifact of UBL is represented conforming to the UBL Upper Ontology.

```
<owl:Class rdf:ID="Address.Details">
```

```
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
<owl:Class rdf:about="#UBL.ABIE"/>
<owl:Restriction>
  <owl:onProperty rdf:resource="#hasObjectClassTerm"/>
  <owl:allValuesFrom>
    <owl:Class rdf:about="#Address"/>
  </owl:allValuesFrom>
</owl:Restriction>
<owl:Restriction>
  <owl:onProperty rdf:resource="#usedInContext"/>
  <owl:allValuesFrom>
    <owl:Class rdf:about="#Context"/>
  </owl:allValuesFrom>
</owl:Restriction>
<owl:Restriction>
  <owl:onProperty rdf:resource="#contains"/>
  <owl:allValuesFrom>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
<owl:Class rdf:about="#Address.Identifier"/>
<owl:Class rdf:about="#Address.AddressTypeCode.Code"/>
<owl:Class rdf:about="#Address.AddressFormatCode.Code"/>
<owl:Class rdf:about="#Address.Postbox.Text"/>
<owl:Class rdf:about="#Address.Floor.Text"/>
<owl:Class rdf:about="#Address.Room.Text"/>
<owl:Class rdf:about="#Address.StreetName.Name"/>
....
<owl:Class rdf:about="#Address.CountrySubentity.Text"/>
<owl:Class rdf:about="#Address.CountrySubentityCode.Code"/>
<owl:Class rdf:about="#Address.Region.Text"/>
<owl:Class rdf:about="#Address.District.Text"/>
<owl:Class rdf:about="#Address.TimezoneOffset.Text"/>
<owl:Class rdf:about="#Address.AddressLine"/>
<owl:Class rdf:about="#Address.Country"/>
<owl:Class rdf:about="#Address.LocationCoordinate"/>
      </owl:intersectionOf>
    </owl:Class>
  </owl:allValuesFrom>
</owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
```

The below listing provides an example on how “Address.Identifier” artifact of UBL is represented conforming to the UBL Upper Ontology.

```
<owl:Class rdf:ID="Address.Identifier">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
<owl:Class rdf:about="#UBL.BBIE"/>
<owl:Restriction>
  <owl:onProperty rdf:resource="#hasObjectClassTerm"/>
  <owl:allValuesFrom>
    <owl:Class rdf:about="#Address"/>
  </owl:allValuesFrom>
</owl:Restriction>
<owl:Restriction>
  <owl:onProperty rdf:resource="#hasRepresentationTerm"/>
  <owl:allValuesFrom>
    <owl:Class rdf:about="#Identifier"/>
  </owl:allValuesFrom>
</owl:Restriction>
<owl:Restriction>
  <owl:onProperty rdf:resource="#usedInContext"/>
  <owl:allValuesFrom>
    <owl:Class rdf:about="#Context"/>
  </owl:allValuesFrom>
</owl:Restriction>
<owl:Restriction>
  <owl:onProperty rdf:resource="#hasDataType"/>
  <owl:allValuesFrom>
    <owl:Class rdf:about="#Identifier.Type"/>
  </owl:allValuesFrom>
</owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
```

3.4 EXPLICATING THE SEMANTICS OF CCTS BASED DOCUMENT SCHEMAS - OAGIS 9.1 UPPER ONTOLOGY

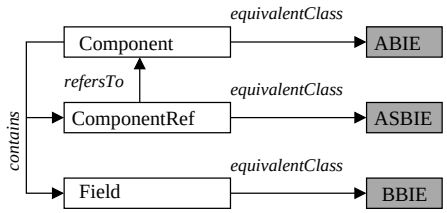


Figure 3.6: The Upper Ontology for the Semantics Exposed by the OAGIS XML Document Standard

Although GS1 XML and UBL use the same terminology for document artifacts such as ABIE as the CCTS framework, OAGIS names its document components differently. Figure 3.6 shows the upper ontology for the OAGIS 9.1 document standard and how OAGIS 9.1 classes are related with the corresponding CCTS classes by using the *owl:equivalentClass* property. An OAGIS 9.1 *Component* corresponds to *ABIE* class in CCTS upper ontology; *ComponentRef* corresponds to *ASBIE* and *Field* corresponds to *BBIE*.

An OAGIS 9.1 *Field* is defined as an OWL class named *Field* and it is declared equivalent to the BBIE class defined in CCTS upper ontology as follows:

```
<owl:Class rdf:ID="Field">
  <owl:equivalentClass rdf:resource="#BasicBusinessInformationEntity"/>
</owl:Class>
```

An OAGIS 9.1 *Component* is defined as an OWL class named *Component* and it is declared equivalent to the ABIE class defined in CCTS upper ontology. The *contains* Object Property of the *Component* class is restricted to *Field* and *ComponentRef* as follows:

```
<owl:Class rdf:ID="Component">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
<owl:Class rdf:about="#AggregateBusinessInformationEntity"/>
<owl:Restriction>
  <owl:onProperty rdf:resource="#contains"/>
  <owl:allValuesFrom>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
```

```
<owl:Class rdf:about="#Field"/>
<owl:Class rdf:about="#ComponentRef"/>
      </owl:intersectionOf>
    </owl:Class>
  </owl:allValuesFrom>
</owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
```

An OAGIS 9.1 *ComponentRef* is defined as an OWL class named *ComponentRef* and it is declared equivalent to the ASBIE class defined in CCTS upper ontology. The *refersTo* Object Property of the *ComponentRef* class is restricted to *Component* as follows:

```
<owl:Class rdf:ID="ComponentRef">
  <owl:equivalentClass rdf:resource="#AssociationBusinessInformationEntity"/>
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
<owl:Class rdf:about="#AggregateBusinessInformationEntity"/>
<owl:Restriction>
  <owl:onProperty rdf:resource="#refersTo"/>
  <owl:allValuesFrom>
    <owl:Class rdf:about="#Component"/>
  </owl:allValuesFrom>
</owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
```

3.4.1 EXPLICATING THE SEMANTICS OF OAGIS 9.1 DOCUMENT SCHEMAS

The semantics of OAGIS 9.1 artifacts are explicated conforming to the OAGIS Upper Ontology defined. OAGIS provides the XSD schemas of its *Components* and *Fields* (e.g. Components.xsd and Fields.xsd) and does not name its components according to ISO 11179 Part 5 . Therefore a special adapter is developed to generate the OAGIS document schema ontology as follows: In OAGIS XSD Schemas, each *Component* is represented with an element declaration and a corresponding type declaration. For example, the “Address” *Component* shown below contains the following element and type declarations.

```
<xsd:element name="Address" type="AddressType"/>

<xsd:complexType name="AddressType">
<xsd:complexContent>
<xsd:extension base="AddressBaseType">
<xsd:sequence>
<xsd:element ref="UserArea" minOccurs="0"/>
</xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="AddressBaseType" abstract="true">
<xsd:annotation>
<xsd:documentation source="http://www.openapplications.org/oagis/9">Address aseType provides
the information about the address or semantic address of an asociated entity.</xsd:documentation>
</xsd:annotation>
<xsd:sequence>
<xsd:element ref="ID" minOccurs="0" maxOccurs="unbounded"/>
<xsd:element ref="FormatCode" minOccurs="0"/>
<xsd:element ref="AttentionOfName" minOccurs="0" maxOccurs="unbounded"/>
<xsd:element ref="CareOfName" minOccurs="0" maxOccurs="unbounded"/>
..
<xsd:element ref="PostalCode" minOccurs="0"/>
<xsd:element ref="Status" minOccurs="0"/>
<xsd:element ref="Preference" minOccurs="0"/>
</xsd:sequence>
<xsd:attribute name="languageCode" type="LanguageCodeContentType" use="optional"/>
<xsd:attribute name="type" type="NormalizedStringType"/>
</xsd:complexType>
```

Furthermore, several components may have the same complex type:

```
<xsd:element name="BillingAddress" type="AddressType"/>
<xsd:element name="OwnerAddress" type="AddressType"/>
<xsd:element name="RemitLocationPostalAddress" type="AddressType">
```

When constructing the OAGIS document schema ontology, for each element declaration, one ontology class is created. An example is shown below, where the OAGIS “Address.Details” artifact is represented conforming to the OAGIS 9.1 Upper Ontology.

```
    <owl:Class rdf:ID="Address.Details">
        <owl:equivalentClass>
```

```
            <owl:Class>
                <owl:intersectionOf rdf:parseType="Collection">
                    <owl:Class rdf:about="#Component"/>
                    <owl:Restriction>
                        <owl:onProperty rdf:resource="#hasObjectClassTerm"/>
                        <owl:allValuesFrom>
                            <owl:Class rdf:about="#Address"/>
                        </owl:allValuesFrom>
                    </owl:Restriction>
                    <owl:Restriction>
                        <owl:onProperty rdf:resource="#usedInContext"/>
                        <owl:allValuesFrom>
                            <owl:Class rdf:about="#Context"/>
                        </owl:allValuesFrom>
                    </owl:Restriction>
                    <owl:Restriction>
                        <owl:onProperty rdf:resource="#contains"/>
                        <owl:allValuesFrom>
                            <owl:Class>
                                <owl:intersectionOf rdf:parseType="Collection">
                                    <owl:Class rdf:about="#AddressLine.AddressLine"/>
                                    <owl:Class rdf:about="#AttentionOfName.NameType"/>
                                    <owl:Class rdf:about="#BuildingName.NameType"/>
                                    <owl:Class rdf:about="#BuildingNumber.TextType"/>
                                    <owl:Class rdf:about="#CareOfName.NameType"/>
                                    <owl:Class rdf:about="#CityName.NameType"/>
                                    <owl:Class rdf:about="#CitySubDivisionName.NameType"/>
                                    <owl:Class rdf:about="#CountryCode.CountryCodeType"/>
.....
                                    <owl:Class rdf:about="#PostOfficeBox.TextType"/>
                                    <owl:Class rdf:about="#Preference.Preference"/>
                                    <owl:Class rdf:about="#Status.Status"/>
                                    <owl:Class rdf:about="#StreetName.NameType"/>
                                    <owl:Class rdf:about="#Unit.TextType"/>
                                </owl:intersectionOf>
                            </owl:Class>
                        </owl:allValuesFrom>
                    </owl:Restriction>
                </owl:intersectionOf>
            </owl:Class>
        </owl:equivalentClass>
    </owl:Class>
```

As shown above, the *Component* “Address.Details” is defined as common component implying that it is used in the general “Context”. OAGIS does not provide the *Object Class Terms*

for its Components. However a closer investigation reveals that the names of the complex types in OAGIS give the information captured by the Object Class Terms of CCTS. Hence, the complex type names are used as the Object Class Terms of OAGIS artifact and they are obtained by simply dropping the suffix “Type” from the element’s complex type name. For example, as shown above, the Object Class Term for Address *Component* is “Address”.

However, some of the OAGIS *Components* are defined based on CCL Core Components. In such cases, the Object Class Terms of the corresponding Core Component is used. For example, there is a *Component* called “ProjectReference” which is of type “ProjectReferenceType” as follows:

```
<xsd:element name="ProjectReference" type="ProjectReferenceType"/>
```

“ProjectReferenceType” is derived from “ProjectBaseType” and “ProjectBaseType” is based on “ProjectABIEType”. In this case, the Object Class Term for “ProjectReference” is “Project” as follows:

```
<xsd:complexType name="ProjectReferenceType">
<xsd:complexContent>
<xsd:extension base="ProjectBaseType">
<xsd:sequence>
<xsd:element ref="ActivityID" minOccurs="0"/>
<xsd:element ref="UserArea" minOccurs="0"/>
</xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="ProjectBaseType">
<xsd:complexContent>
<xsd:extension base="ProjectABIEType"/>
</xsd:complexContent>
</xsd:complexType>
```

In OAGIS, a *Component* is composed of *Fields* and/or *ComponentReferences*. *Fields* are at the leaf level and they are based on CCTS Core Component Types. All the OAGIS *Fields* are defined in “Fields.xsd” document. They are inserted to the OAGIS Document Schema Ontology as follows: In XSDs, each *Field* is defined with an element and a corresponding

type declaration. The type declaration usually points to the Core Component Type. For example, the “PostalCode” field is of type “Code Type”:

```
<xsd:element name="PostalCode" type="CodeType">
```

This field is inserted to the ontology as a new class, which is a restriction on *Field* class. An example on how ”PostalCode.CodeType” field of OAGIS 9.1 is represented conforming to the OAGIS 9.1 Upper Ontology is shown below. The representation term of this class is “Code” and the data type of this class is “Code.Type”.

```
<owl:Class rdf:ID="PostalCode.CodeType">
        <owl:equivalentClass>
            <owl:Class>
                <owl:intersectionOf rdf:parseType="Collection">
                    <owl:Class rdf:about="#Field"/>
                    <owl:Restriction>
                        <owl:onProperty
rdf:resource="#hasRepresentationTerm"/>
                        <owl:allValuesFrom>
                            <owl:Class rdf:about="#Code"/>
                        </owl:allValuesFrom>
                    </owl:Restriction>
                    <owl:Restriction>
                        <owl:onProperty rdf:resource="#usedInContext"/>
                        <owl:allValuesFrom>
                            <owl:Class rdf:about="#Context"/>
                        </owl:allValuesFrom>
                    </owl:Restriction>
                    <owl:Restriction>
                        <owl:onProperty rdf:resource="#hasDataType"/>
                        <owl:allValuesFrom>
                            <owl:Class rdf:about="#Code.Type"/>
                        </owl:allValuesFrom>
                    </owl:Restriction>
                </owl:intersectionOf>
            </owl:Class>
        </owl:equivalentClass>
    </owl:Class>
```

Considering the *Component References*, they are inserted to the ontology as new classes, which are restrictions of *ComponentRef* class. For example, the “Preference.Preference” *Component Reference* is inserted as shown below. It should be noted that it is used in general *Context* and refers to “Preference.Details” Component.

```
<owl:Class rdf:ID="Preference.Preference">
        <owl:equivalentClass>
            <owl:Class>
                <owl:intersectionOf rdf:parseType="Collection">
                    <owl:Class rdf:about="#ComponentRef"/>
                    <owl:Restriction>
                        <owl:onProperty rdf:resource="#hasObjectClassTerm"/>
                        <owl:allValuesFrom>
                            <owl:Class rdf:about="#Address"/>
                        </owl:allValuesFrom>
                    </owl:Restriction>
                    <owl:Restriction>
                        <owl:onProperty rdf:resource="#hasRepresentationTerm"/>
                        <owl:allValuesFrom>
                            <owl:Class rdf:about="#Preference"/>
                        </owl:allValuesFrom>
                    </owl:Restriction>
                    <owl:Restriction>
                        <owl:onProperty rdf:resource="#usedInContext"/>
                        <owl:allValuesFrom>
                            <owl:Class rdf:about="#Context"/>
                        </owl:allValuesFrom>
                    </owl:Restriction>
                    <owl:Restriction>
                        <owl:onProperty rdf:resource="#refersTo"/>
                        <owl:allValuesFrom>
                            <owl:Class rdf:about="#Preference.Details"/>
                        </owl:allValuesFrom>
                    </owl:Restriction>
                </owl:intersectionOf>
            </owl:Class>
        </owl:equivalentClass>
    </owl:Class>
```

3.5 HARMONIZING THE ONTOLOGIES OF THE DOCUMENT STANDARDS

When a DL reasoner runs through the upper ontologies together with the document schema ontologies defined, the resulting inferred (harmonized) ontology gives the correspondences between the document artifacts of the CCTS based standards. The established relationships can be directly indicating that the two document artifacts are equivalent or in subsumption relationship. Furthermore, the relationship can be indirect through CCL, that is, the document artifacts of two different standards may both be a superclass (or subclass) of a CCL artifact and the relationship between them can only be established through CCL.

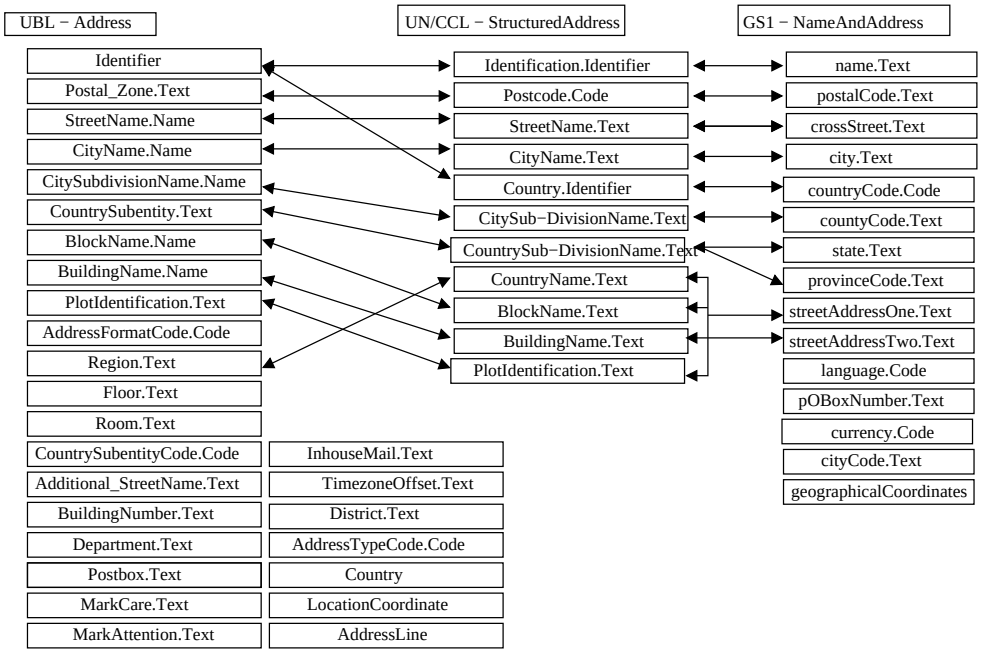


Figure 3.7: The Semantic Equivalences among the BBIEs of UBL-Address, CCL-Structured Address and GS1-NameAndAddress Discovered through the Harmonized Ontology

In this section, an example is presented on how the correlation of three document components is established in the Harmonized Ontology through the DL reasoning process. In order to facilitate the description of the reasoning process used in this example, first the ontology descriptions are expressed as specified in this document through their corresponding logical expressions. Table 3.1 gives the logical expressions corresponding to “CCL Structured Address ABIE” and Tables 3.2, 3.3 and 3.4 give logical expressions corresponding to “UBL Address ABIE”.

Table 3.1: UN/CCL - “Structured Address” ABIE Asserted Definition

1. Structured_Address.Details ≡ AggregateBusinessInformationEntity ∧ ∀ contains Structured_Address.Identification.Identifier ∧ Structured_Address.Postcode.Code ∧ Structured_Address.BuildingName.Text ∧ Structured_Address.StreetName.Text ∧ Structured_Address.CityName.Text ∧ Structured_Address.Country.Identifier ∧ Structured_Address.CitySub-DivisionName.Text ∧ Structured_Address.CountryName.Text ∧ Structured_Address.CountrySub-DivisionName.Text ∧ Structured_Address.BlockName.Text ∧ Structured_Address.PlotIdentification.Text ∧ ∀ hasObjectClassTerm Address ∧ ∀ usedInContext Context
2.Structured_Address.BlockName.Text ≡ BasicBusinessInformationEntity ∧ ∀ hasDataType Text.Type ∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm Text ∧ ∀ usedInContext Context
3.Structured_Address.BuildingName.Text ≡ BasicBusinessInformationEntity ∧ ∀ hasDataType Text.Type ∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm Text ∧ ∀ usedInContext Context
4.Structured_Address.CityName.Text ≡ BasicBusinessInformationEntity ∧ ∀ hasDataType Text.Type ∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm Text ∧ ∀ usedInContext Context
5.Structured_Address.CitySub-DivisionName.Text ≡ BasicBusinessInformationEntity ∧ ∀ hasDataType Text.Type ∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm Text ∧ ∀ usedInContext Context
6.Structured_Address.Country.Identifier ≡ BasicBusinessInformationEntity ∧ ∀ hasDataType Identifier.Type ∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm Identifer ∧ ∀ usedInContext Context
7.Structured_Address.CountryName.Text ≡ BasicBusinessInformationEntity ∧ ∀ hasDataType Text.Type ∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm Text ∧ ∀ usedInContext Context
8.Structured_Address.CountrySub-DivisionName.Text ≡ BasicBusinessInformationEntity ∧ ∀ hasDataType Text.Type ∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm Text ∧ ∀ usedInContext Context
9.Structured_Address.Identification.Identifier ≡ BasicBusinessInformationEntity ∧ ∀ hasDataType Identifier.Type ∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm Identifer ∧ ∀ usedInContext Context
10.Structured_Address.PlotIdentification.Text ≡ BasicBusinessInformationEntity ∧ ∀ hasDataType Text.Type ∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm Text ∧ ∀ usedInContext Context
11.Structured_Address.Postcode.Code ≡ BasicBusinessInformationEntity ∧ ∀ hasDataType Code.Type ∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm Code ∧ ∀ usedInContext Context
12.Structured_Address.StreetName.Text ≡ BasicBusinessInformationEntity ∧ ∀ hasDataType Text.Type ∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm Text ∧ ∀ usedInContext Context

Table 3.2: UBL “Address” ABIE Asserted Definition - Part 1

13.Address.Details ≡ UBL.ABIE ∧ ∀ contains Address.Identifier ∧ Address.AddressTypeCode.Code ∧ Address.AddressFormatCode.Code ∧ Address.Postbox.Text ∧ Address.Floor.Text ∧ Address.Room.Text ∧ Address.StreetName.Name ∧ Address.Additional_StreetName.Name ∧ Address.BlockName.Name ∧ Address.BuildingName.Name ∧ Address.BuildingNumber.Text ∧ Address.Inhouse_Mail.Text ∧ Address.Department.Text ∧ Address.MarkAttention.Text ∧ Address.MarkCare.Text ∧ Address.PlotIdentification.Text ∧ Address.CitySubdivisionName.Name ∧ Address.CityName.Name ∧ Address.Postal_Zone.Text ∧ Address.CountrySubentity.Text ∧ Address.CountrySubentityCode.Code ∧ Address.Region.Text ∧ Address.District.Text ∧ Address.TimezoneOffset.Text ∧ Address.AddressLine ∧ Address.Country ∧ Address.LocationCoordinate ∧ hasObjectClassTerm Address ∧ usedInContext Context

14.Address.Additional_StreetName.Name ≡ UBL.BBIE ∧ ∀ hasDataType Name.Type ∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm Name ∧ ∀ usedInContext Context

15.Address.AddressFormatCode.Code ≡ UBL.BBIE ∧ ∀ hasDataType Code.Type ∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm Code ∧ ∀ usedInContext Context

16.Address.AddressTypeCode.Code ≡ UBL.BBIE ∧ ∀ hasDataType Code.Type ∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm Code ∧ ∀ usedInContext Context

17.Address.BlockName.Name ≡ UBL.BBIE ∧ ∀ hasDataType Name.Type ∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm Name ∧ ∀ usedInContext Context

18.Address.BuildingName.Name ≡ UBL.BBIE ∧ ∀ hasDataType Name.Type ∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm Name ∧ ∀ usedInContext Context

19.Address.BuildingNumber.Text ≡ UBL.BBIE ∧ ∀ hasDataType Text.Type ∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm Text ∧ ∀ usedInContext Context

20.Address.CityName.Name ≡ UBL.BBIE ∧ ∀ hasDataType Name.Type ∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm Name ∧ ∀ usedInContext Context

21.Address.CitySubdivisionName.Name ≡ UBL.BBIE ∧ ∀ hasDataType Name.Type ∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm Name ∧ ∀ usedInContext Context

22.Address.CountrySubentity.Text ≡ UBL.BBIE ∧ ∀ hasDataType Text.Type ∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm Text ∧ ∀ usedInContext Context

23.Address.CountrySubentityCode.Code ≡ UBL.BBIE ∧ ∀ hasDataType Code.Type ∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm Code ∧ ∀ usedInContext Context

24.Address.Department.Text ≡ UBL.BBIE ∧ ∀ hasDataType Text.Type ∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm Text ∧ ∀ usedInContext Context

25.Address.District.Text ≡ UBL.BBIE ∧ ∀ hasDataType Text.Type ∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm Text ∧ ∀ usedInContext Context

26.Address.Floor.Text ≡ UBL.BBIE ∧ ∀ hasDataType Text.Type ∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm Text ∧ ∀ usedInContext Context

27.Address.Identifier ≡ UBL.BBIE ∧ ∀ hasDataType Identifer.Type ∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm Identifer ∧ ∀ usedInContext Context

28.Address.Inhouse_Mail.Text ≡ UBL.BBIE ∧ ∀ hasDataType Text.Type ∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm Text ∧ ∀ usedInContext Context

Table 3.3: UBL “Address” ABIE Asserted Definition - Part 2

29.Address.MarkAttention.Text ≡ UBL.BBIE ∧ ∀ hasDataType Text.Type ∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm Text ∧ ∀ usedInContext Context
30.Address.MarkCare.Text ≡ UBL.BBIE ∧ ∀ hasDataType Text.Type ∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm Text ∧ ∀ usedInContext Context
31.Address.PlotIdentification.Text ≡ UBL.BBIE ∧ ∀ hasDataType Text.Type ∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm Text ∧ ∀ usedInContext Context
32.Address.Postal_Zone.Text ≡ UBL.BBIE ∧ ∀ hasDataType Text.Type ∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm Text ∧ ∀ usedInContext Context
33.Address.Postbox.Text ≡ UBL.BBIE ∧ ∀ hasDataType Text.Type ∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm Text ∧ ∀ usedInContext Context
34.Address.Region.Text ≡ UBL.BBIE ∧ ∀ hasDataType Text.Type ∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm Text ∧ ∀ usedInContext Context
35.Address.Room.Text ≡ UBL.BBIE ∧ ∀ hasDataType Text.Type ∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm Text ∧ ∀ usedInContext Context
36.Address.StreetName.Name ≡ UBL.BBIE ∧ ∀ hasDataType Name.Type ∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm Name ∧ ∀ usedInContext Context
37.Address.TimezoneOffset.Text ≡ UBL.BBIE ∧ ∀ hasDataType Text.Type ∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm Text ∧ ∀ usedInContext Context
38.Address.AddressLine ≡ UBL.ASBIE ∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm AddressLine ∧ ∀ refersTo AddressLine.Details ∧ ∀ usedInContext Context
39.Address.Country ≡ UBL.ASBIE ∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm Country ∧ ∀ refersTo Country.Details ∧ ∀ usedInContext Context ∧ ∀
40.Address.LocationCoordinate ≡ UBL.ASBIE ∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm LocationCoordinate ∧ ∀ refersTo LocationCoordinate.Details ∧ ∀ usedInContext Context
41.AddressLine.Details ≡ UBL.ABIE ∧ ∀ contains AddressLine.Line.Text ∧ ∀ hasObjectClassTerm AddressLine ∧ ∀ usedInContext Context
42.AddressLine.Line.Text ≡ UBL.BBIE ∧ ∀ hasDataType Text.Type ∧ ∀ hasObjectClassTerm AddressLine ∧ ∀ hasRepresentationTerm Text ∧ ∀ usedInContext Context
43.Country.Details ≡ UBL.ABIE ∧ ∀ contains Country.IdentificationCode.Code ∧ Country.Name ∧ ∀ hasObjectClassTerm Country ∧ ∀ usedInContext Context
44.Country.IdentificationCode.Code ≡ UBL.BBIE ∧ ∀ hasDataType CountryIdentification_Code.Type ∧ ∀ hasObjectClassTerm Country ∧ ∀ hasRepresentationTerm Code ∧ ∀ usedInContext Context
45.Country.Name ≡ UBL.BBIE ∧ ∀ hasDataType Name.Type ∧ ∀ hasObjectClassTerm Country ∧ ∀ hasRepresentationTerm Name ∧ ∀ usedInContext Context
46.LocationCoordinate.Details ≡ UBL.ABIE ∧ ∀ contains LocationCoordinate.CoordinateSystemCode.Code ∧ LocationCoordinate.Latitude_Degrees.Measure ∧ LocationCoordinate.Latitude_Minutes.Measure ∧ LocationCoordinate.LatitudeDirectionCode.Code ∧ LocationCoordinate.Longitude_Degrees.Measure ∧ LocationCoordinate.Longitude_Minutes.Measure ∧ LocationCoordinate.LongitudeDirectionCode.Code ∧ ∀ hasObjectClassTerm LocationCoordinate ∧ ∀ usedInContext Context

Table 3.4: UBL “Address” ABIE Asserted Definition - Part 3

47.LocationCoordinate.CoordinateSystemCode.Code ≡ UBL.BBIE ∧ ∀ hasDataType Code.Type ∧ ∀ hasObjectClassTerm LocationCoordinate ∧ ∀ hasRepresentationTerm Code ∧ ∀ usedInContext Context
48.LocationCoordinate.Latitude_Degrees.Measure ≡ UBL.BBIE ∧ ∀ hasDataType Measure.Type ∧ ∀ hasObjectClassTerm LocationCoordinate ∧ ∀ hasRepresentationTerm Measure ∧ ∀ usedInContext Context
49.LocationCoordinate.Latitude_Minutes.Measure ≡ UBL.BBIE ∧ ∀ hasDataType Measure.Type ∧ ∀ hasObjectClassTerm LocationCoordinate ∧ ∀ hasRepresentationTerm Measure ∧ ∀ usedInContext Context
50.LocationCoordinate.LatitudeDirectionCode.Code ≡ UBL.BBIE ∧ ∀ hasDataType LatitudeDirection_Code.Type ∧ ∀ hasObjectClassTerm LocationCoordinate ∧ ∀ hasRepresentationTerm Code ∧ ∀ usedInContext Context
51.LocationCoordinate.Longitude_Degrees.Measure ≡ UBL.BBIE ∧ ∀ hasDataType Measure.Type ∧ ∀ hasObjectClassTerm LocationCoordinate ∧ ∀ hasRepresentationTerm Measure ∧ ∀ usedInContext Context
52.LocationCoordinate.Longitude_Minutes.Measure ≡ UBL.BBIE ∧ ∀ hasDataType Measure.Type ∧ ∀ hasObjectClassTerm LocationCoordinate ∧ ∀ hasRepresentationTerm Measure ∧ ∀ usedInContext Context
53.LocationCoordinate.LongitudeDirectionCode.Code ≡ UBL.BBIE ∧ ∀ hasDataType LongitudeDirection_Code.Type ∧ ∀ hasObjectClassTerm LocationCoordinate ∧ ∀ hasRepresentationTerm Code ∧ ∀ usedInContext Context

Table 3.5: The Assertion Related with the different Usage of Datatypes

54.Name.Type ≡ Text.Type 74.Code.Type ≡ Text.Type ≡ Identifier.Type

The Harmonized Ontology contains the fact that “CCL Structured Address ABIE” is a subclass of “UBL Address ABIE” giving a much needed correspondence. Similarly, Table 3.7 gives the logical expressions corresponding to “GS1 NameAndAddress ABIE” and Table 3.5 gives one of the additional assertions for data types.

Table 3.8 gives the inferred equalities and subsumptions in the Harmonized Ontology.

Table 3.6: Inferred Equalities/Subsumptions between UN/CCL "Structured Address" and UBL "Address" in the Harmonized Ontology

Inferred Relations	**The Facts used in Computing the Inferred Relations**
Structured_Address.CityName.Text ≡ Address.CityName.Name	Def.4, Def.20, Def.54
Structured_Address.CitySub-DivisionName.Text ≡ Address.CitySubdivisionName.	Def.5, Def.21, Def.54
Structured_Address.StreetName.Text ≡ Address.StreetName.Name	Def.12, Def.36, Def.54
Structured_Address.Identification.Identifier ≡ Address.Identifier	Def.9, Def.27
Structured_Address.Postcode.Code ≡ Address.Postal_Zone.Text	Def.11, Def.32
Structured_Address.Country.Identifier ≡ Address.Identifer	Def.6, Def.27
Structured_Address.CountrySub-DivisionName.Text ≡ Address.CountrySubentity.Text	Def.8, Def.22
Structured_Address.CountryName.Text ≡ Address.Region.Text	Def.7, Def.34
Structured_Address.BlockName.Text ≡ Address.BlockName.Name	Def.2, Def.17, Def.54
Structured_Address.BuildingName.Text ≡ Address.BuildingName.Name	Def.3, Def.18, Def.54
Structured_Address.PlotIdentification.Text ≡ Address.PlotIdentification.Text	Def.10, Def.31
Address.Details *is a subclass of* Structured_Address.Details	Def.13, Def.1 (since Address.Details contain more than Structured_Address)

Table 3.7: GS1 “NameAndAddress” ABIE Asserted Definition

56.NameAndAddress.Details ≡ GS1.XML.ABIE ∧ ∀ contains city ∧ cityCode ∧ countryCode ∧ countyCode ∧ crossStreet ∧ currency ∧ languageOfTheParty ∧ name ∧ pOBoxNumber ∧ postalCode ∧ provinceCode ∧ state ∧ streetAddressOne ∧ streetAddressTwo ∧ geographicalCoordinates ∧ ∀ hasObjectClassTerm Address ∧ ∀ usedInContext Context
57.city ≡ GS1.XML.BBIE ∧ ∀ hasDataType Text.Type ∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm Text ∧ ∀ usedInContext Context
58.cityCode ≡ GS1.XML.BBIE ∧ ∀ hasDataType Text.Type∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm Text ∧ ∀ usedInContext Context
59.countryCode ≡ GS1.XML.BBIE ∧ ∀ hasDataType Code.Type ∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm Code ∧ ∀ usedInContext Context
60.countyCode ≡ GS1.XML.BBIE ∧ ∀ hasDataType Text.Type∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm Text ∧ ∀ usedInContext Context
61.crossStreet ≡ GS1.XML.BBIE ∧ ∀ hasDataType Text.Type∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm Text ∧ ∀ usedInContext Context
62.currency ≡ GS1.XML.BBIE ∧ ∀ hasDataType Code.Type ∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm Code ∧ ∀ usedInContext Context
63.languageOfTheParty ≡ GS1.XML.BBIE ∧ ∀ hasDataType Code.Type ∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm Code ∧ ∀ usedInContext Context
64.name ≡ GS1.XML.BBIE ∧ ∀ hasDataType Text.Type∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm Text ∧ ∀ usedInContext Context
65.pOBoxNumber ≡ GS1.XML.BBIE ∧ ∀ hasDataType Text.Type∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm Text ∧ ∀ usedInContext Context
66.postalCode ≡ GS1.XML.BBIE ∧ ∀ hasDataType Text.Type∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm Text ∧ ∀ usedInContext Context
67.provinceCode ≡ GS1.XML.BBIE ∧ ∀ hasDataType Text.Type∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm Text ∧ ∀ usedInContext Context
68.state ≡ GS1.XML.BBIE ∧ ∀ hasDataType Text.Type∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm Text ∧ ∀ usedInContext Context
69.streetAddressOne ≡ GS1.XML.BBIE ∧ ∀ hasDataType Text.Type∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm Text ∧ ∀ usedInContext Context
70.streetAddressTwo ≡ GS1.XML.BBIE ∧ ∀ hasDataType Text.Type∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm Text ∧ ∀ usedInContext Context
71.geographicalCoordinates ≡ GS1.XML.ASBIE ∧ ∀ hasObjectClassTerm Address ∧ ∀ hasRepresentationTerm GeographicalCoordinates ∧ ∀ refersTo GeographicalCoordinates.Details ∧ ∀ usedInContext Context
72.GeographicalCoordinates.Details ≡ GS1.XML.ABIE ∧ ∀ contains latitude ∧ longitude ∧ ∀ hasObjectClassTerm GeographicalCoordinates ∧ ∀ usedInContext Context

Table 3.8 gives the inferred equalities and subsumptions in the Harmonized Ontology.

Table 3.8: Inferred Equalities/Subsumptions between UN/CCL “Structured Address” and GS1 “NameAndAddress” in the Harmonized Ontology

Inferred Relations	**The Facts used in Computing the Inferred Relations**
Structured_Address.CityName.Text ≡ city	Def.4, Def.57
Structured_Address.City-SubdivisionName.Text ≡ countyCode	Def.5, Def.59
Structured_Address.StreetName.Text ≡ crossStreet	Def.12, Def.61
Structured_Address.Identification.Identifier ≡ name	Def.9, Def.64, Def. 55
Structured_Address.Postcode.Code ≡ postalCode	Def.11, Def.66, Def.55
Structured_Address.CountrySubDivisionName.Text ≡ provinceCode	Def.8, Def.67
Structured_Address.Country.Identifier ≡ countryCode	Def.6, Def.60, Def. 55
Structured_Address.CountrySub-DivisionName.Text ≡ state	Def.8, Def.68
Structured_Address.CountryName.Text ≡ streetAddressOne ≡ streetAddressTwo	Def.7, Def.69, Def.70
Structured_Address.BlockName.Text ≡ streetAddressOne ≡ streetAddressTwo	Def.2, Def.69, Def.70
Structured_Address.BuildingName.Text ≡ streetAddressOne ≡ streetAddressTwo	Def.3, Def.69, Def.70
Structured_Address.PlotIdentification.Text ≡ streetAddressOne ≡ streetAddressTwo	Def.10, Def.69, Def.70
NameAndAddress.Details *is a subclass of* Structured_Address.Details	Def.56, Def.1 (since NameAndAddressDetails contain more BBIEs than Structured_Address.Details)

As a summary, Table 3.6 and Table 3.8 give how the equivalences given in Figure 3.7 are computed. Additionally, as shown in Table 3.8, the reasoner discovers that “GS1-NameAnd Address.Details” *is a subclass of* “UN/CCL-Structured_Address.Details” and as shown in Table 3.6, and that “UBL-Address.Details” *is a subclass of* “UN/CCL-Structured_Address.-Details” because the “UBL-Address.Details” and the “GS1-NameAndAddress.Details” both contain more elements than the “UN/CCL-Structured_Address.Details” document artifact.

CHAPTER 4

PROVIDING HEURISTICS TO DISCOVER STRUCTURALLY DIFFERENT DOCUMENT ARTIFACTS

The semantic properties of the CCTS based document artifacts help discovering the equivalences of structurally and semantically similar artifacts. However, different document standards use *Core Components* in different structures. For example, an "Address" component of one standard may be totally different than the "Address" of another standard. Semantic properties of document artifacts are not enough to find the similarity of the structurally different but semantically equivalent document artifacts. In other words, DL reasoners discover relations among both semantically and structurally similar components. However, document standards generate their components in different structures. For example, in CCL, "Primary_Identification. Identifier" BBIE in included directly into "Buyer_Party" ABIE. However, in UBL, the BBIE used for party identification is in "PartyIdentification" ABIE, which is associated to "Party" ABIE through "Party.PartyIdentification" ASBIE. Therefore, the reasoner cannot find the relation between "Party" and "BuyerParty".

More specifically, description logics cannot find the relations between these structurally different components. Therefore, heuristics should be provided to find these relations. In my approach, there are two types of heuristics:

1. Heuristics given in description logics: These heuristics are mostly auxiliary in nature and they are inserted into the upper and document schema ontologies in creation time. They are for resolving the different usages of CCTS Data Types and putting the object class terms into a hierarchy to help finding the equivalent BBIEs at different structural levels. They are described in Sections 4.1 and 4.2, respectively.

2. Heuristics given in predicate logics: These heuristics are used to find relations between structurally different but semantically similar components and they are described in Section 4.3.

4.1 HEURISTICS TO HELP RESOLVING THE DIFFERENT USAGES OF CCTS DATA TYPES

Different document standards use CCTS Data Types differently. For example, “Code.Type” can be used to specify the datatype of a BBIE in one standard and “Text.Type” can be used for the same BBIE in another standard and yet “Identifier.Type” in another standard. This knowledge in real world is expressed through class equivalences as shown below so that not only the humans but also the reasoner knows about it.

```
<owl:Class rdf:ID="Code.Type">
 <owl:equivalentClass rdf:resource="#Text.Type"/>
 <rdfs:subClassOf rdf:resource="#CoreComponentType"/>
</owl:Class>
<owl:Class rdf:ID="Name.Type">
 <owl:equivalentClass rdf:resource="#Text.Type"/>
 <rdfs:subClassOf rdf:resource="#CoreComponentType"/>
</owl:Class>
<owl:Class rdf:ID="Identifier.Type">
 <owl:equivalentClass rdf:resource="#Text.Type"/>
 <rdfs:subClassOf rdf:resource="#CoreComponentType"/>
</owl:Class>
```

Obviously such rules will not be needed if the document standards use CCTS methodology in the same way. But in reality, the existing CCTS based document standards have such different usages and to discover similar document artifacts in different standards, the reasoner needs this information. It should be noted that such an assertion may produce some false positives, that is, finding two unrelated document artifacts to be similar. However, such false positives are in limited numbers, since many other semantic properties of the document artifacts are compared to find similarities. The false positives, when they happen, need to be sorted out manually. Since my purpose is to develop a support tool for humans to use rather than a completely automated process, the human intervention is necessary to eliminate the remaining false positives if there are any.

4.2 A HEURISTIC TO HELP FINDING THE EQUIVALENT BBIES AT DIFFERENT STRUCTURAL LEVELS

A problem in finding the document artifacts with similar information content in two different document schemas is that the semantically similar artifacts may appear at structurally different positions. For example, two semantically equivalent aggregate document components, which belong to different document standards, may have their basic document components, which are also semantically equivalent, at different structural levels as shown in Figure 4.1 (A) and (B). These cases prevent description logics reasoners to discover the relationship between them. The problem is how to inform the reasoner in an automated way that such document components are considered similar.

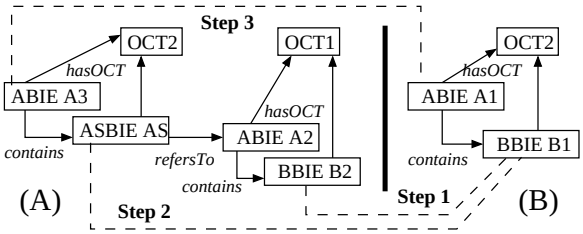


Figure 4.1: Example structural difference

First, note that an *Object Class Term* in CCTS denotes that a document component belongs to an abstract class, although not in ontological terms. In explicating the semantics of the CCTS, ontology classes are created for document components and associate them with their corresponding *Object Class Term* classes through *owl:hasOCT* property. Consider the following two document components:

- A basic component “BBIE B2” with an *Object Class Term* OCT1 is referred to through an association component from an aggregate component with an *Object Class Term* OCT2 in one document schema as shown in Figure 4.1 (A),
- A basic component “BBIE B1”, which is semantically equivalent to “BBIE B2”, directly appears under an aggregate component with an *Object Class Term* OCT2 in another schema as shown in Figure 4.1 (B).

To handle such a structural difference, a subsumption relation is established among their *Object Class Term* clasess by declaring that OCT1 is a subclass of OCT2. The intention is that

; Assume that there are two BBIEs, whose names are $name_1$ and $name_2$,
; respectively.
BBIE($name_1$, oct_1, rt_1, $context_1$, $dataType_1$) ∧
BBIE($name_2$, oct_2, rt_2, $context_2$, $dataType_2$) ∧
; If their "Object Class Terms" are equal or subclass of each other
(∨ ($oct_1 \equiv oct_2$) (oct_1 *is a subclass of* oct_2) (oct_2 *is a subclass of* oct_1)) ∧
; And if their "Representation Terms" are equal or subclass of each other
(∨ ($rt_1 \equiv rt_2$) (rt_1 *is a subclass of* rt_2) (rt_2 *is a subclass of* rt_1)) ∧
; And if their "Contexts" are equal or subclass of each other
(∨ ($context_1 \equiv context_2$) ($context_1$ *is a subclass of* $context_2$) ($context_2$
is a subclass of $context_1$)) ∧
; And if their "Data Types" are equal or subclass of each other
(∨ ($dataType_1 \equiv dataType_2$) ($dataType_1$ *is a subclass of* $dataType_2$)
($dataType_2$ *is a subclass of* $dataType_1$))
⇒
; Then these BBIEs are equal
$name_1 \equiv name_2$

Figure 4.2: The Rule for Discovery of two Semantically Similar Basic Document Components

a basic document component semantically similar to "BBIE B2" might as well be regarded as a part of an aggregate component whose *Object Class Term* is OCT2 to let the reasoner discover such equivalences.

Note that once such an assertion is made, then the reasoner can recursively trace the association components at any depth.

This semantics is extracted when the document schemas are processed to create the corresponding OWL ontologies through a software developed which automatically asserts a subsumption hierarchy among the *Object Class Term* classes of such document artifacts. In fact, the harmonized ontology given in [90] already contains all such relationships.

4.3 HEURISTICS TO FIND RELATIONSHIPS BETWEEN SEMANTICALLY SIMILAR BUT STRUCTURALLY DIFFERENT DOCUMENT ARTIFACTS

A very common structural difference in semantically similar document artifacts is that although some of the semantic properties of a document artifact "A" is the subclass of the corresponding properties of the document artifact "B", some other properties of "A" are the super classes of the corresponding attributes of "B". As an example, a document artifact

; Assume that there are two ASBIEs, whose names are $name_1$ and $name_1$,
; respectively.
ASBIE($name_1$, oct_1, rt_1, $context_1$, $refersTo_1$) $\wedge$
ASBIE($name_2$, oct_2, rt_2, $context_2$, $refersTo_2$) $\wedge$
; If their "Object Class Terms" are equal or subclass of each other
($\vee$ ($oct_1 \equiv oct_2$) (oct_1 *is a subclass of* oct_2) (oct_2 *is a subclass of* oct_1)) $\wedge$
; And if their "Representation Terms" are equal or subclass of
; each other
($\vee$ ($rt_1 \equiv rt_2$) (rt_1 *is a subclass of* rt_2) (rt_2 *is a subclass of* rt_1)) $\wedge$
; And if their "Contexts" are equal or subclass of each other
($\vee$ ($context_1 \equiv context_2$) ($context_1$ *is a subclass of* $context_2$) ($context_2$
is a subclass of $context_1$)) $\wedge$
; And if they "refesTo" the same ABIE or the referred ABIEs subclass of
; each other
($\vee$ ($refersTo_1 \equiv refersTo_2$) ($refersTo_1$ *is a subclass of* $refersTo_2$) ($refersTo_2$
is a subclass of $refersTo_1$))
$\Rightarrow$
; Then these ASBIEs are equal
$name_1 \equiv name_2$

Figure 4.3: The Rule for Discovery of two Semantically Similar Association Document Components

A's context may be a subclass of document artifact B's context but the subclass relationship among their other properties may be in the reverse direction. As another example, as shown in lower part of Figure 4.7, the relations between the values of "Party. Postal_Address. Address" and "Buyer_Party. Postal. Structured_Adress" ASBIEs' "usedInContext" and "refersTo" properties are in reverse direction. For discovering the similarity of document artifacts, it is not important if the direction of the subsumption relations among the corresponding semantic properties of the document artifacts is the same. In other words, when the purpose is to find out whether these artifacts are similar, it is not important whether the direction of the subsumption relationship is different among their corresponding attributes.

Furthermore, the heuristics are categorized according to structural differences that can occur among different document artifacts as follows:

- *Heuristics to Discover Structurally Different Basic Document Components (BBIEs)*: If the semantic properties of two basic document components are pair wise equivalent or subclasses of each other, these basic document components are considered to be similar. The rule in Figure 4.2 expresses this heuristics. When this rule fires, it establishes an

; Assume there are the following artifacts.
ASBIE($name_1$, oct_1, rt_1, $context_1$, $refersTo_1$) ∧
ABIE($name_2$, oct_2, $containsSet_2$, $context_2$, $bieCount_2$) ∧
BBIE($name_3$, oct_3, rt_3, $context_3$, $dataType_3$) ∧
BBIE($name_4$, oct_4, rt_4, $context_4$, $dataType_4$) ∧
; If ASBIE $name_1$, "refersTo" ABIE $name_2$
$refersTo_1 \equiv name_2 \wedge$
; And if BBIE $name_3$ is in the ContainsSet of ABIE $name_2$
$name_3 \in containsSet_2 \wedge$
; And if BBIE ABIE $name_3$ is equals to BBIE $name_4$
$name_3 \equiv name_4$
⇒
; Then BBIE $name_4$ is equals to BBIE $name_1$ $name_4 \equiv name_1$

Figure 4.4: The Rule for Discovery of Semantic Similarity between a Basic Document Component and an Association Document Component

owl:equivalentClass property between these two basic document components.

- *Heuristics to Discover Structurally Different Association Document Components (ASBIEs)*: If the semantic properties of two association document components are pair wise equivalent or subclasses of one another, these association document components are considered to be equivalent. The rule given in Figure 4.3 states this heuristics and when this rule fires, it establishes an *owl:equivalentClass* property between these two association document components.

- *Heuristics to Discover Structurally Different Association Document Component (ASBIE) and Basic Document Component Pairs (BBIE)*: Consider two basic document components, "BBIE B1" and "BBIE B2", which belong to different standards and whose semantic equivalence is established through the harmonized ontology as indicated in Figure 4.1 Step 1. Assume that "BBIE B1" is in an aggregate document component "ABIE A1" whose object class term is OCT2 as shown in Figure 4.1 (B). Assume further that "BBIE B2" is in an aggregate document component "ABIE A2" as shown in Figure 4.1 (A). Another aggregate document component "ABIE A3" whose object class term is OCT2, refers to "ABIE A2" through the association document component "ASBIE AS" (Figure 4.1 (A)). There is a possibility that the association document component "ASBIE AS" is semantically equivalent to the basic document component "BBIE B1" as shown in Figure 4.1 Step 2.

; Assume that there are two ABIEs, whose names are $name_1$ and $name_2$
$ABIE(name_1, oct_1, containsSet_1, context_1, bieCount_1) \wedge$
$ABIE(name_2, oct_2, containsSet_2, context_2, bieCount_2) \wedge$
; If their "Object Class Terms" are equal or subclass of each other
$(\vee\ (oct_1 \equiv oct_2)\ (oct_1$ *is a subclass of* $oct_2)\ (oct_2$ *is a subclass of* $oct_1)) \wedge$
; And if their "ContainsSet" are equal or subclass of each other
$(\vee\ (containsSet_1 \equiv containsSet_2)\ (containsSet_1 \subset containsSet_2)$
$(containsSet_2 \subset containsSet_1)) \wedge$
; And if their "Contexts" are equal or subclass of each other
$(\vee\ (context_1 \equiv context_2)\ (context_1$ *is a subclass of* $context_2)$
$(context_2$ *is a subclass of* $context_1))$
$\Rightarrow$
; Then these ABIEs are equal.
$name_1 \equiv name_2$

Figure 4.5: The Rule for Discovery of two Aggregate Document Components having Semantically Similar Content

The rule given in Figure 4.4 states this heuristics and when this rule fires, it establishes an *owl:equivalentClass* property between the association document component "ASBIE AS" and basic document component "BBIE B1". Note that once this equivalence is established in the harmonized ontology and the reasoner is executed again it may establish the equivalence of the aggregate components (Figure 4.1 Step 3) by triggering the *Heuristics to Discover Structurally Different Aggregate Document Components (ABIEs)* rule.

- *Heuristics to Discover Structurally Different Aggregate Document Components (ABIEs)*: The structural differences that can occur in aggregate document components are more complex, because each of them may contain a different number of basic and association document components, some of which may be semantically equivalent, some may not. To be able to better express the different cases that may appear, the *ContainsSet* of an aggregate document component is defined to be the set of all of its document components. The *ContainsSet* is in fact the set of document components in the range of the *contains* property of an aggregate document component. The *ContainsSets* of two aggregate document components may be equal; may have a non-null intersection; may be in subset relationships or may be disjoint of each other. If the *ContainsSets* are not disjoint, heuristics are provided to discover their similarity. Note that the other semantic properties of two aggregate document components must be similar for these

rules to fire.

- *Case 1: The ContainsSets of two aggregate document components are equivalent or in subset relationship*: Considering all the semantic properties of two aggregate document components, if each of them is pair wise equivalent or subclasses of one another, and their *ContainsSets* are the same or subsets of each other, these aggregate document components are considered to be similar. The rule in Figure 4.5 states this heuristics. When this rule fires, it establishes an *owl:equivalentClass* property between these two aggregate document components.

- *Case 2: The ContainsSets of two aggregate document components have a non-null intersection*: The semantic properties of two aggregate document components may be equivalent and their *ContainsSet* may have a non-null intersection. What is provided is a *similarityConstant* that the user may set. As an example, if the user considers that two aggregate document components are similar when 60% of their document components are similar then he can set the *similarityConstant* to "0.6". Hence, when all the semantic properties of two aggregate document components are either pair wise equivalent or subclasses of one another, and the document components in their *ContainsSet* sets are *similarityConstant* percent equivalent, these aggregate document components are considered to be similar. The rule given in Figure 4.6 states this heuristics and when it fires, it establishes an *owl:equivalentClass* property between these two aggregate document components.

These rules are defined in predicate logic using JESS Rules [35] and execute them through JESS Rule Engine and carry the results back to the harmonized ontology. Note that the rules defined produce further OWL class equivalences in the harmonized ontology. After this step, the DL-Reasoner is executed again to compute the new equality/subsumption relations. The discovered equivalences are presented to the domain expert or the user for his approval. This process may be repeated until the user is satisfied with the discovered similarities.

; Assume that there are two ABIEs, whose names are $name_1$ and $name_2$
$ABIE(name_1, oct_1, containsSet_1, context_1, bieCount_1) \wedge$
$ABIE(name_2, oct_2, containsSet_2, context_2, bieCount_2) \wedge$
; If BIE $name_3$ is in the ContainsSet of ABIE $name_1$
$name_3 \in containsSet_1 \wedge$
; And if BIE $name_4$ is in the ContainsSet of ABIE $name_2$
$name_4 \in containsSet_2 \wedge$
; And if BIE $name_3$ and BIE $name_4$ are equal or subclass of each other
$(\vee (name_3 \equiv name_4) (name_3$ *is a subclass of* $name_4) (name_3$ *is a subclass of* $name_4))$
$\Rightarrow$
; Then increase count by 1
($count = $count +1) $\wedge$
; And if the similarity between their ContainsSet is greater then
; similarityConstant then these ABIEs are equal.
(IF (($count / ($bieCount$_1$ + $bieCount$_2$ - $count)) > $similarityConstant))
THEN $name_1 \equiv name_2$

Figure 4.6: The Rule for Discovery ofg two Semantically Similar Aggregate Document Components

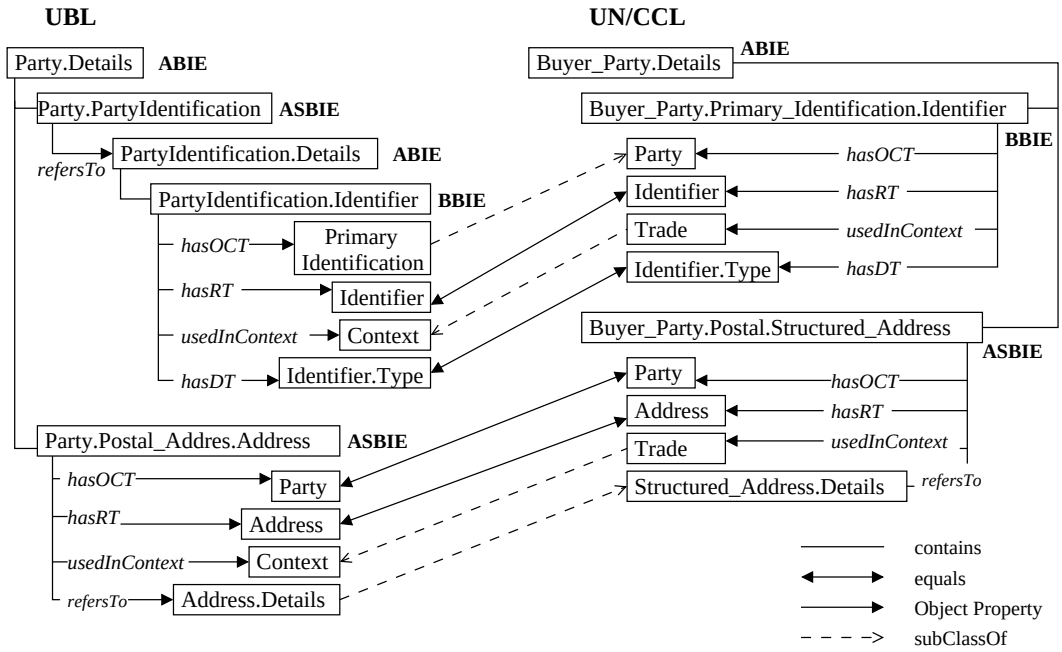


Figure 4.7: UBL's Party ABIE and CCL's Buyer_Party ABIE

4.4 AN EXAMPLE ON THE USE OF THE HARMONIZED ONTOLOGY AND THE PROVIDED HEURISTICS

In this section, a real life example is presented to explain the use of the harmonized ontology and the provided heuristics. Consider the two example document artifacts given in Figure

Table 4.1: The Relationship among the Semantic Properties of two Example Basic Document Components

"PartyIdentification. Identifier" of UBL	"Buyer_Party. Primary_Identification. Identifier" of UN/CCL	
PrimaryIdentification	Party	"PrimaryIdentification" *Object Class Term* is a subclass of "Party" *Object Class Term* as described in Section 4.2
Identifier	Identifier	Both Basic Document Components have the same *Representation Term*
Context	Trade	"Trade" context class is a subclass of the *Context* top class
Identifier.Type	Identifier.Type	Both Basic Document Components have the same *Data Type*

4.7. A human looking at this figure can immediately tell the similarity between the UBL "Party.Details" and UN/CCL "Buyer_Party.Details". The aim is to discover such similarities in an automated way.

Given the two document artifacts in Figure 4.7, the Description Logics reasoner first discovers the similarities of the semantic properties of the following basic document components: the UBL's "PartyIdentification.Identifier" and the UN/CCL's "Buyer_Party.Primary_Identification.Identifier" as shown in Table 4.1.

Note that the subsumption relationship between "PrimaryIdentification" and "Party" classes is in the opposite direction of the subsumption relationship between *Context* and "Trade". However, through the class equivalence asserted to the harmonized ontology with the execution of the rule given in Figure 4.2, the Description Logics reasoner establishes the relationship between the UBL's "PartyIdentification.Identifier" and the UN/CCL's "Buyer_Party.Primary_Identification.Identifier" as being equivalent. Furthermore, the reasoner using the class equivalence inferred by the rule given in Figure 4.4, establishes the fact that the "Buyer_Party.Primary_Identification.Identifier" is equivalent to the "Party.PartyIdentification" association document component.

Similarly, for the UBL's "Party.Postal_Address.Address" association document component and the UN/CCL's "Buyer_Party.Postal.Structured_Address" association document component, the reasoner establishes the equivalences and the subsumption relationships among the document artifacts as shown in Table 4.2 in the harmonized ontology.

In Table 4.2, the direction of the subsumption relationship between the UBL "Address.-Details" and the UN/CCL "Structured_Address.Details" classes is in the opposite direction of the subsumption relationship between the *Context* and the "Trade". However, since the class equivalence relationship is already asserted to the harmonized ontology with the execution of the rule given in Figure 4.3, the Description Logics reasoner establishes the relationship between the UBL's "Party.Postal_Address.Address" and the UN/CCL's "Buyer_Party.Postal.-Structured_Address" as being equivalent classes. In Figure 4.7, for the sake of simplicity some of the basic document components and association document components, namely, the UBL "Party.Details" and the UN/CCL "Buyer_Party.Details" are not shown. When the Description Logics reasoner considers these extra basic and association document components, the equivalence among the semantic properties becomes as shown in Figure 4.8. In other words, the set of basic and association document components of the UBL "Party.Details" is a superset of document components of UN/CCL "Buyer_Party.Details".

The relationships shown in Figure 4.8 together with the rule in Figure 4.5 trigger the semantic equivalence of the UBL "Party.Details" and the UN/CCL "Buyer_Party.Details" document artifacts.

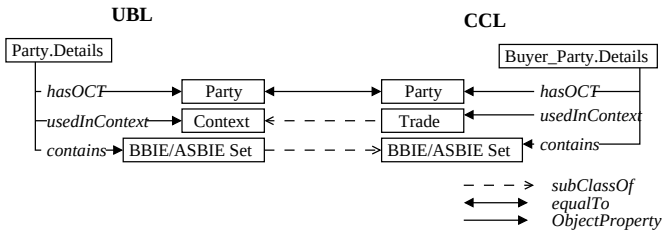


Figure 4.8: UBL's Party ABIE and CCL's Buyer_Party ABIE

Table 4.2: The Relationship among the Semantic Properties of two Example Association Document Components

"Party. Postal_Address. Address" of UBL	"Buyer_Party. Postal. Structured_Address" of UN/CCL	
Party	Party	Both Association Document Components have the same *Object Class Term*
Address	Address	Both Association Document Components have the same *Representation Term*
Context	Trade	"Trade" context class is a subclass of the *Context* top class
Address. Details	Structured_ Address. Details	"Address. Details" *is a subclass of* "Structured_ Address. Details"

CHAPTER 5

AUTOMATED XSLT GENERATION SUPPORT

The harmonized ontology gives the specified as well as the computed equality and subsumption relations among the classes of both the upper ontologies and the document schema ontologies. In order to translate document instances between different document schemas, this knowledge should be used to determine the relationship between the elements in the document instances. In other words, the equivalences discovered are carried to the data level, where the document instances are described in XML. The instance level translation is achieved using XSLT and the XSLT definitions are generated automatically for the identified equivalences through the following process:

- While constructing the upper ontologies, the relations between the ontology classes and XSD schema elements are identified and the corresponding XPath expressions are generated.
- To obtain the XSLT definition from a document artifact to another:
  - The harmonized ontology classes that correspond to document artifacts in the document schemas are matched.
  - Then, the XPath expressions for the identified classes are retrieved.
  - Finally, XSLT expressions are generated automatically from these XPath expressions using the equivalences among document artifacts discovered through the harmonized ontology.

The extra elements, for which the methodology cannot establish the corresponding elements in the target document schema (either because there is no corresponding element or there is

such an element but the methodology is not able to find the related element) are inserted into the extension part of the target document schema. The domain experts can further handcraft the XSLT definitions.

5.1 AN EXAMPLE: TRANSLATING UBL “ADDRESS.dETAILS” TO GS1 “NAME AND ADDRESS”

In this section, the translation process is described through an example.

5.1.1 OBTAINING THE XPATH EXPRESSIONS FOR UBL “ADDRESS” ABIE AND FOR ITS BBIES/ASBIES AUTOMATICALLY

The ABIEs in UBL are represented with “xsd:complexType” definitions in XSDs. The “xsd:-complexType” of “Address.Details” ABIE is “AddressType”. In the annotation part of the UBL XSD schemas, the dictionary entry names of the BIEs are also provided. As mentioned previously, the class names for BIEs in the document schema ontologies are generated from the dictionary entry names. Therefore, this information is directly used in constructing the XPath expressions for UBL ontology classes (i.e. this information implicitly describes the ontology class). In the below fragment, a part of UBL “AddressType” XSD declaration is given.

```
<xsd:complexType name="AddressType">
    <xsd:annotation>
      <xsd:documentation>
        <ccts:Component>
          <ccts:ComponentType>ABIE</ccts:ComponentType>
          <ccts:DictionaryEntryName>Address. Details</ccts:DictionaryEntryName>
          <ccts:Definition>Information about a structured address.</ccts:Definition>
          <ccts:ObjectClass>Address</ccts:ObjectClass>
        </ccts:Component>
      </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element ref="cbc:ID" minOccurs="0" maxOccurs="1">
        <xsd:annotation>
          <xsd:documentation>
            <ccts:Component>
              <ccts:ComponentType>BBIE</ccts:ComponentType>
```

```
            <ccts:DictionaryEntryName>Address. Identifier</ccts:DictionaryEntryName>
            <ccts:Definition>An identifier for a specific address within a
            scheme of registered addresses.</ccts:Definition>
            <ccts:Cardinality>0..1</ccts:Cardinality>
            <ccts:ObjectClass>Address</ccts:ObjectClass>
            <ccts:PropertyTerm>Identifier</ccts:PropertyTerm>
            <ccts:RepresentationTerm>Identifier</ccts:RepresentationTerm>
            <ccts:DataType>Identifier. Type</ccts:DataType>
            <ccts:AlternativeBusinessTerms>DetailsKey</ccts:AlternativeBusinessTerms>
          </ccts:Component>
        </xsd:documentation>
      </xsd:annotation>
    </xsd:element>
...
    <xsd:element ref="LocationCoordinate" minOccurs="0" maxOccurs="1">
      <xsd:annotation>
        <xsd:documentation>
          <ccts:Component>
            <ccts:ComponentType>ASBIE</ccts:ComponentType>
            <ccts:DictionaryEntryName>Address. Location Coordinate</ccts:DictionaryEntryName>
            <ccts:Definition>An association to Location Coordinate.</ccts:Definition>
            <ccts:Cardinality>0..1</ccts:Cardinality>
            <ccts:ObjectClass>Address</ccts:ObjectClass>
            <ccts:PropertyTerm>Location Coordinate</ccts:PropertyTerm>
            <ccts:AssociatedObjectClass>Location Coordinate</ccts:AssociatedObjectClass>
          </ccts:Component>
        </xsd:documentation>
      </xsd:annotation>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>
```

The XPath language itself refers to XML elements, not to XSD complex types. Therefore, when generating the XPath expression for the “Address.Details” ABIE, the XML elements that refer to the “AddressType” are collected. In UBL XSDs, the XML elements that refer to “AddressType” are as follows:

```
<xsd:element name="RegistrationAddress" type="AddressType"/>
<xsd:element name="PostalAddress" type="AddressType"/>
<xsd:element name="OriginAddress" type="AddressType"/>
<xsd:element name="JurisdictionRegionAddress" type="AddressType"/>
<xsd:element name="DespatchAddress" type="AddressType"/>
<xsd:element name="DeliveryAddress" type="AddressType"/>
<xsd:element name="ApplicableTerritoryAddress" type="AddressType"/>
<xsd:element name="ApplicableAddress" type="AddressType"/>
```

```
<xsd:element name="Address" type="AddressType"/>
```

Hence, the XPath expression for “Address” ABIE is as follows:

```
#Address.Details --> //(Address | ApplicableAddress | ApplicableTerritoryAddress |
DeliveryAddress | DespatchAddress | JurisdictionRegionAddress | OriginAddress |
PostalAddress | RegistrationAddress)
```

This XPath expression states that “Address.Details” ontology class corresponds to Address, ApplicableAddress, ApplicableTerritoryAddress, DeliveryAddress, DespatchAddress, JurisdictionRegionAddress, OriginAddress, PostalAddress and RegistrationAddress XML elements in a UBL XML document. The “//” in the beginning of the XPath expression states that the element can be at any depth in the XML document. The XPath expressions given below for the BBIEs and ASBIEs in the “Address.Details” ABIE are generated by concatenating the corresponding XML elements names to the above XPath expression.

```
#Address.Identifier --> //(Address | ApplicableAddress | ApplicableTerritoryAddress |
DeliveryAddress | DespatchAddress |JurisdictionRegionAddress | OriginAddress |
PostalAddress | RegistrationAddress)/ID
...
#Address.LocationCoordinate --> //(Address | ApplicableAddress |
ApplicableTerritoryAddress | DeliveryAddress | DespatchAddress |
JurisdictionRegionAddress | OriginAddress | PostalAddress |
RegistrationAddress)/LocationCoordinate
...
#LocationCoordinate.Details --> //LocationCoordinate (LocationCoordinateType)
#LocationCoordinate.CoordinateSystemCode.Code --> //LocationCoordinate/CoordinateSystemCode
#LocationCoordinate.Latitude\_Degrees.Measure --> //LocationCoordinate/LatitudeDegreesMeasure
#LocationCoordinate.Latitude\_Minutes.Measure --> //LocationCoordinate/LatitudeMinutesMeasure
#LocationCoordinate.LatitudeDirectionCode.Code --> //LocationCoordinate/LatitudeDirectionCode
#LocationCoordinate.Longitude\_Degrees.Measure --> //LocationCoordinate/LongitudeDegreesMeasure
#LocationCoordinate.Longitude\_Minutes.Measure --> //LocationCoordinate/LongitudeMinutesMeasure
#LocationCoordinate.LongitudeDirectionCode.Code --> //LocationCoordinate/LongitudeDirectionCode
```

In the above example, XPath expressions for “AddressLine”, “Country” and “LocationCoordinate” ABIEs are also provided for the sake of completeness.

5.1.2 OBTAINING XPath EXPRESSIONS FOR GS1 “NAMEANDADDRESS” ABIE AND FOR ITS BBIES

The ABIEs are represented in GS1 XML’s XSD schemas through “xsd:complexType” declarations. Furthermore, in GS1 XML, the “xsd:complexType” name of an ABIE can be identified by concatenating “Type” keyword to the ABIE name. Therefore, the complex type name of “NameAndAddress” is “NameAndAddressType”. The next step is to find the declaration of this type. In GS1 XML, for all ABIEs there is either a separate XSD file (which can be identified by concatenating “.xsd” extension to the ABIE’s name, e.g., “NameAndAddress” has “NameAndAddress.xsd”) or the ABIE’s “xsd:complexType” declaration is in the parent ABIEs XSD file (e.g. “GeographicalCoordinates” ABIE’s declaration is in “NameAndAddress.xsd”). Therefore, the complex type “NameAndAddressType” is declared in “NameAndAddress.xsd” file, as follows:

```
<xsd:complexType name="NameAndAddressType">
    <xsd:sequence>
      <xsd:element name="city">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string">
            <xsd:maxLength value="35"/>
            <xsd:minLength value="1"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:element>
..
      <xsd:element name="geographicalCoordinates"
type="eanucc:GeographicalCoordinatesType" minOccurs="0">
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
```

As mentioned previously, the XPath language itself refers to XML elements, not to XSD complex types. Therefore, when generating the XPath expression for the “NameAndAddress” ABIE, the XML elements that refer to the “NameAndAddressType” should be identified. In GS1 XSDs, one of the XML elements that refer to “NameAndAddressType” is “shipToNonCommercial” as follows:

```
<xsd:element name="shipToNonCommercial" type="eanucc:NameAndAddressType">
</xsd:element>
```

The XPath expression for “NameAndAddress” ABIE is as follows:

```
#NameAndAddress.Details  --> //shipToNonCommercial
```

Furthermore, the XPath expressions for the BBIEs in “NameAndAddress” can be obtained by concatenating the corresponding element name to the above XPath expression. The XPath expressions for the BBIEs of “NameAndAddress” are as follows:

```
#city --> //shipToNonCommercial/city
#cityCode --> //shipToNonCommercial/cityCode
#countryCode --> //shipToNonCommercial/countryCode/countryISOCode
#countyCode --> //shipToNonCommercial/countyCode
#crossStreet --> //shipToNonCommercial/crossStreet
#currency --> //shipToNonCommercial/currency
#languageOfTheParty --> //shipToNonCommercial/languageOfTheParty
#name --> //shipToNonCommercial/name
#pOBoxNumber --> //shipToNonCommercial/pOBoxNumber
#postalCode --> //shipToNonCommercial/postalCode
#provinceCode --> //shipToNonCommercial/provinceCode
#state --> //shipToNonCommercial/state
#streetAddressOne --> //shipToNonCommercial/streetAddressOne
#streetAddressTwo --> //shipToNonCommercial/streetAddressTwo
#geographicalCoordinates --> //shipToNonCommercial/geographicalCoordinates

#GeographicalCoordinates.Details --> //geographicalCoordinates
#latitude --> //geographicalCoordinates/latitude
#longitude --> //geographicalCoordinates/longitude
```

For the sake of completeness, the XPath expressions of “GeographicalCoordinates” are also provided.

5.1.3 CONSTRUCTING THE XSLT DEFINITIONS

In constructing the XSLT definitions, the generated XPath expressions for the ontology classes and the semantic equivalences discovered are used.

As an example, Figure 3.7 gives the semantic equivalences among the BBIEs shown below:

```
#Address.Identifier --> #name, #countryCode
#Address.Postal_Zone.Text --> #postalCode
#Address.StreetName.Name --> #crossStreet
```

```
#Address.CityName.Name --> #city
#Address.CitySubdivisionName.Name --> #countyCode
#Address.CountrySubentityCode.Code --> #provinceCode, #state
#Address.BlockName.Name --> #streetAddressOne, #streetAddressTwo
#Address.BuildingName.Name--> #streetAddressOne,#streetAddressTwo
#Address.PlotIdentification.Text --> #streetAddressOne, #streetAddressTwo
#Address.Region.Text --> #streetAddressOne, #streetAddressTwo
```

The XSLT Definition given below is constructed using the equivalences given above and the generated XPath expressions to convert an UBL “Address” ABIE instance to GS1 “NameAndAddress” instance.

```
<xsl:stylesheet>
<xsl:output method="xml" encoding="utf-8" indent="yes" omit-xml-declaration="yes"/>
<xsl:template match="/">
<xsl:for-each select="//cac:Address|//cac:ApplicableAddress|
//cac:ApplicableTerritoryAddress|//cac:DeliveryAddress|//cac:DespatchAddress|
//cac:JurisdictionRegionAddress|//cac:OriginAddress|//cac:PostalAddress|
//cac:RegistrationAddress">
<xsl:call-template name="nameAndAddress"/>
</xsl:for-each>
</xsl:template>
<xsl:template name="nameAndAddress">
<xsl:element name="eanucc:shipToNonCommercial">
<xsl:element name="city">
<xsl:value-of select="cbc:CityName"/> </xsl:element>
<xsl:element name="countryCode">
<xsl:element name="countryISOCode">
<xsl:value-of select="cbc:ID"/> </xsl:element>
</xsl:element>
<xsl:element name="countyCode">
<xsl:value-of select="cbc:CitySubdivisionName"/> </xsl:element>
<xsl:element name="crossStreet">
<xsl:value-of select="cbc:StreetName"/> </xsl:element>
<xsl:element name="name">
<xsl:value-of select="cbc:ID"/> </xsl:element>
<xsl:element name="postalCode">
<xsl:value-of select="cbc:PostalZone"/> </xsl:element>
<xsl:element name="provinceCode">
<xsl:value-of select="cbc:CountrySubentityCode"/> </xsl:element>
<xsl:element name="state">
<xsl:value-of select="cbc:CountrySubentityCode"/> </xsl:element>
<xsl:element name="streetAddressOne">
<xsl:value-of select="cbc:BlockName|cbc:BuildingName|cbc:PlotIdentification|cbc:Region"/>
</xsl:element>
```

```
<xsl:element name="streetAddressTwo">
<xsl:value-of select="cbc:BlockName|cbc:BuildingName|cbc:PlotIdentification|cbc:Region"/>
</xsl:element> </xsl:element> </xsl:template>
</xsl:stylesheet>
```

Assume that the user has the UBL “Address” instance as follows:

```
<cac:Address>
<cbc:ID>UBL1</cbc:ID>
<cbc:AddressTypeCode>UBLAddressTypeCode</cbc:AddressTypeCode>
<cbc:AddressFormatCode>UBLAddressFormatCode</cbc:AddressFormatCode>
<cbc:Postbox>UBLPostbox</cbc:Postbox>
<cbc:Floor>UBL2</cbc:Floor>
<cbc:Room>UBL3</cbc:Room>
<cbc:StreetName>UBLStreetName</cbc:StreetName>
<cbc:AdditionalStreetName>UBLAdditionalStreetName</cbc:AdditionalStreetName>
<cbc:BlockName>UBLBlockName</cbc:BlockName>
<cbc:BuildingName>UBLBuildingName</cbc:BuildingName>
<cbc:BuildingNumber>UBL4</cbc:BuildingNumber>
<cbc:InhouseMail>UBLInhouseMail</cbc:InhouseMail>
<cbc:Department>UBLDepartment</cbc:Department>
<cbc:MarkAttention>UBLMarkAttention</cbc:MarkAttention>
<cbc:MarkCare>UBLMarkCare</cbc:MarkCare>
<cbc:PlotIdentification>UBLPlotIdentification</cbc:PlotIdentification>
<cbc:CitySubdivisionName>UBLCitySubdivisionName</cbc:CitySubdivisionName>
<cbc:CityName>UBLCityName</cbc:CityName>
<cbc:PostalZone>UBLPostalZone</cbc:PostalZone>
<cbc:CountrySubentity>UBLCountrySubentity</cbc:CountrySubentity>
<cbc:CountrySubentityCode>UBLCountrySubentityCode</cbc:CountrySubentityCode>
<cbc:Region>UBLRegion</cbc:Region>
<cbc:District>UBLDistrict</cbc:District>
<cbc:TimezoneOffset>UBLTimezoneOffset</cbc:TimezoneOffset>
<cac:AddressLine>
<cbc:Line>UBLLine</cbc:Line>
</cac:AddressLine>
<cac:Country>
<cbc:IdentificationCode>UBLIdentificationCode</cbc:IdentificationCode>
<cbc:Name>UBLName</cbc:Name>
</cac:Country>
<cac:LocationCoordinate>
<cbc:CoordinateSystemCode>UBLCoordinateSystemCode</cbc:CoordinateSystemCode>
<cbc:LatitudeDegreesMeasure unitCode="04">UBL0.0</cbc:LatitudeDegreesMeasure>
<cbc:LatitudeMinutesMeasure unitCode="04">UBL0.0</cbc:LatitudeMinutesMeasure>
<cbc:LatitudeDirectionCode>UBLLatitudeDirectionCode</cbc:LatitudeDirectionCode>
<cbc:LongitudeDegreesMeasure unitCode="04">UBL0.0</cbc:LongitudeDegreesMeasure>
<cbc:LongitudeMinutesMeasure unitCode="04">UBL0.0</cbc:LongitudeMinutesMeasure>
```

```
<cbc:LongitudeDirectionCode>UBLLongitudeDirectionCode</cbc:LongitudeDirectionCode>
</cac:LocationCoordinate>
</cac:Address>
```

After applying the XSLT definitions to the example UBL “Address” Instance given above, the GS1 “NameAndAddress” instance in the following fragment is obtained.

```
<eanucc:shipToNonCommercial>
<city>UBLCityName</city>
<countryCode> <countryISOCode>1</countryISOCode>
</countryCode>
<countyCode>UBLCitySubdivisionName</countyCode>
<crossStreet>UBLStreetName</crossStreet>
<name>1</name>
<postalCode>UBLPostalZone</postalCode>
<provinceCode>UBLCountrySubentityCode</provinceCode>
<state>UBLCountrySubentityCode</state>
<streetAddressOne>UBLBlockName UBLBuildingName UBLPlotIdentification UBLRegion</streetAddressOne>
<streetAddressTwo>UBLBlockName UBLBuildingName UBLPlotIdentification UBLRegion</streetAddressTwo>
</eanucc:shipToNonCommercial>
<extension>
<cbc:AddressTypeCode>UBLAddressTypeCode</cbc:AddressTypeCode>
<cbc:AddressFormatCode>UBLAddressFormatCode</cbc:AddressFormatCode>
<cbc:Postbox>UBLPostbox</cbc:Postbox>
<cbc:Floor>UBL2</cbc:Floor>
<cbc:Room>UBL3</cbc:Room>
<cbc:AdditionalStreetName>UBLAdditionalStreetName</cbc:AdditionalStreetName>
<cbc:BuildingNumber>UBL4</cbc:BuildingNumber>
<cbc:InhouseMail>UBLInhouseMail</cbc:InhouseMail>
<cbc:Department>UBLDepartment</cbc:Department>
<cbc:MarkAttention>UBLMarkAttention</cbc:MarkAttention>
<cbc:MarkCare>UBLMarkCare</cbc:MarkCare>
<cbc:CountrySubentity>UBLCountrySubentity</cbc:CountrySubentity>
<cbc:District>UBLDistrict</cbc:District>
<cbc:TimezoneOffset>UBLTimezoneOffset</cbc:TimezoneOffset>
<cac:AddressLine> <cbc:Line>UBLLine</cbc:Line> </cac:AddressLine>
<cac:Country>
<cbc:IdentificationCode>UBLIdentificationCode</cbc:IdentificationCode>
<cbc:Name>UBLName</cbc:Name> </cac:Country>
<cac:LocationCoordinate>
<cbc:CoordinateSystemCode>UBLCoordinateSystemCode</cbc:CoordinateSystemCode>
<cbc:LatitudeDegreesMeasure unitCode="04">UBL0.0</cbc:LatitudeDegreesMeasure>
<cbc:LatitudeMinutesMeasure unitCode="04">UBL0.0</cbc:LatitudeMinutesMeasure>
<cbc:LatitudeDirectionCode>UBLLatitudeDirectionCode</cbc:LatitudeDirectionCode>
<cbc:LongitudeDegreesMeasure unitCode="04">UBL0.0</cbc:LongitudeDegreesMeasure>
```

```
<cbc:LongitudeMinutesMeasure unitCode="04">UBL0.0</cbc:LongitudeMinutesMeasure>
<cbc:LongitudeDirectionCode>UBLLongitudeDirectionCode</cbc:LongitudeDirectionCode>
</cac:LocationCoordinate>
</extension>
```

It should be noted that the unmapped elements are inserted to the extension part of the document.

5.2 DOCUMENT COMPONENT DISCOVERY SUPPORT

Document component discovery is very important for the following reasons:

- When creating a new document type say a “Planning Document in UBL”, it is necessary to find the already existing document components in UBL to be reused. For the document artifacts that do not exist in UBL, CCL must be searched to find the corresponding components.
- Additionally, when a user wants to transform a document artifact in one standard into another, if s/he cannot obtain a mapping from the Harmonized Ontology, s/he may wish to query the Harmonized Ontology to discover the corresponding artifacts manually.

Currently document artifacts are mostly stored in spread sheets. Also, there is an initiative, called UN/CEFACT Registry Implementation Specification, for storing/querying CCTS articfacts. However, all of these mechanisms only support keyword-based queries. For example, UN/CEFACT Registry Implementation Specification allows the users to query Aggregate Business Information Entities (ABIE) according to ABIE’s name, definition, business term, propery term, object class term and the context values, where the ABIE used. Keyword-based queries fail short in the following respects:

- The users usually may not guess the exact keyword for querying.
- It is not possible to query BIEs/CCs based on the components they contain although this type of information is very useful. For intance, a user cannot issue a query to return the ABIEs which contain a “BBIE A”, a “BBIE B” and an “ASBIE C”.

- Keyword-based queries cannot make use of the class hierarchy. For example, assume that the user is looking for a Business Information Entity (BIE) which is related with USA geopolitical context. If the BIE is related with “North America” context node, this BIE is not returned to the user.

Furthermore, the implicit semantic relationships provided by the Harmonized Ontology are very useful for certain type of queries. For this reason, the possible type of queries are identified and they are formulated either with SPARQL (if the query does not necessitate reasoning) or in OWL.

The Harmonized Ontology can be queried for discovering the document components and the following types of queries are identified:

- Keyword Queries: This type of queries returns the user the BIEs/CCs whose name or description includes a given keyword.
- Type Queries: The type queries allow the users to query BIEs/CCs based on their Data Types, to query BIEs according to their CCs or to query ASBIEs/ASCCs according to their source/target ABIEs/ACCs.
- Structural Queries: This type of queries allows the users to search for CCs/BIEs according to their structure. For example, the user can query BCCs/BBIEs in a given ACC/ABIE, can query ACCs/ABIEs that contain a BCC/ASCC or BIE/ASBIE expression. The expression can be composed of using logical operators AND, OR or NOT.
- Context Queries: The context queries are used for discovering BIEs, which are used in a specified context set or set expression (e.g. Return BIEs which are used in context set S and not used in context set S’)
- Equivalence/Similarity Queries: Most of the time, the user would like to obtain CCs/BIEs similar to a user-defined CC/BIE. In other words, the user specifies the desired content and issues this content as a query against the Harmonized Ontology.

5.2.1 SPARQL QUERIES

For keyword, type and structural queries no further reasoning is needed than what is present in the Harmonized Ontology. Therefore for these three types of queries it is enough to formulate them using SPARQL for efficiency.

An example Keyword Query is as given below. This query retrieves the CCs or BIEs that contain “address” (Case-insensitive) in their “label” or “comment” elements.

```
PREFIX  rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX  owl:  <http://www.w3.org/2002/07/owl#>
PREFIX  rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT  ?subject
WHERE
  { ?subject  rdfs:label    ?label ;
              rdfs:comment  ?comment .
    FILTER ( regex(str(?label), "^address", "i") || regex(str(?comment), "^address", "i") )
  }
```

An example Type Query is as given below. This query retrieves BIEs (from the Harmonized Ontology) that have “basedOn” property whose range is restricted to a class whose CCs can only come from “Price.Details”. In other words, this query retrieves BIEs derived from “Price.Details” CC.

```
PREFIX  rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX  owl:  <http://www.w3.org/2002/07/owl#>
PREFIX  rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX  uo:   <http://144.122.230.79/ontology/UpperOnt.owl#>

SELECT  ?subject
WHERE
  { ?subject  owl:equivalentClass  ?temp4 .
    ?temp4    owl:intersectionOf   ?temp3 .
    ?temp3    rdf:rest             ?temp2 .
    ?temp2    rdf:rest             ?temp1 .
    ?temp1    rdf:first            ?temp .
    ?temp     rdf:type             owl:Restriction ;
              owl:onProperty       uo:basedOn ;
              owl:allValuesFrom    uo:Price.Details .
  }
```

An example Structural Query is as given below. This query retrieves those ASCCs and/or ASBIEs (from the Harmonized Ontology) that have “refersTo” property whose range is restricted to a class whose ACCs and/or ABIEs can only come from “Period.Details”. In other words, this query retrieves ASCCs and/or ASBIEs that refer to “Period.Details” ACCs and ABIEs.

```
PREFIX  rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX  owl:  <http://www.w3.org/2002/07/owl#>
PREFIX  rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX  uo:   <http://144.122.230.79/ontology/UpperOnt.owl#>

SELECT  ?subject
WHERE
  { ?subject  owl:equivalentClass  ?temp  .
    ?temp     owl:intersectionOf   ?temp0 .
    ?temp0    rdf:rest             ?temp1 .
    ?temp1    rdf:rest             ?temp2 .
    ?temp2    rdf:rest             ?temp3 .
    ?temp3    rdf:first            ?temp4 .
    ?temp4    owl:allValuesFrom    uo:Period.Details ;
              owl:onProperty       uo:refersTo ;
              rdf:type             owl:Restriction .
  }
```

An example Context query is as given below. This query retrieves BIEs from the Harmonized Ontology that have “usedInContext” property whose range is restricted to a context that whose values can only come from “Trade”. In other words, this query retrieves BIEs that are used in “Trade” context.

```
PREFIX  rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX  owl:  <http://www.w3.org/2002/07/owl#>
PREFIX  rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX  uo:   <http://144.122.230.79/ontology/UpperOnt.owl#>

SELECT  ?subject
WHERE
  { ?subject  owl:equivalentClass  ?temp  .
    ?temp     owl:intersectionOf   ?temp1 .
    ?temp1    rdf:rest             ?temp2 .
    ?temp2    rdf:rest             ?temp3 .
    ?temp3    rdf:rest             ?temp4 .
    ?temp4    rdf:rest             ?temp5 .
```

```
    ?temp5    rdf:first            ?temp6 .
    ?temp6    owl:allValuesFrom    uo:Trade ;
              owl:onProperty       uo:usedInContext ;
              rdf:type             owl:Restriction .
  }
```

5.2.2 QUERIES THAT REQUIRE REASONING SUPPORT

Some of the Context and all of the Equivalence type of queries require reasoning support and they are formulated as new class expressions in the OWL Ontology and the result is obtained by computing the new inferred ontology through the reasoner. In other words, the reasoner classifies the newly introduced class and computes its relationships to all the related classes.

For example, assume the user would like to obtain the BIEs used in North America but not in Mexico. The query is as follows:

```
<owl:Class rdf:ID="Query">
    <owl:equivalentClass>
      <owl:Class>
<owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#BusinessInformationEntity"/>
  <owl:Restriction>
    <owl:onProperty rdf:resource="#usedInContext" />
    <owl:allValuesFrom>
      <owl:Class>
<owl:complementOf>
  <owl:Class rdf:about="#iso.ch.3166.1999.MX"/>
</owl:complementOf>
      </owl:Class>
    </owl:allValuesFrom>
  </owl:Restriction>
  <owl:Restriction>
    <owl:onProperty rdf:resource="#usedInContext" />
    <owl:allValuesFrom>
      <owl:Class rdf:about="#iso.ch.3166.1999.North.America"/>
    </owl:allValuesFrom>
  </owl:Restriction>
</owl:intersectionOf>
      </owl:Class>
    </owl:equivalentClass>
  </owl:Class>
```

A DL reasoner computes the new hierarchy and relates the similar classes (BIEs) to the Query class through subClassOf or equivalentClass constructs. The class given below is the result of the query, as the reasoner puts the Query as the subclass of the class in the inferred hierarchy.

```
<owl:Class rdf:ID="AssetExpense_AccountingAccount.Details">
    <owl:equivalentClass>
      <owl:Class>
<owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:ID="BusinessInformationEntity"/>
  <owl:Restriction>
    <owl:onProperty rdf:resource="#usedInContext"/>
    <owl:allValuesFrom>
  <owl:Class rdf:about="#iso.ch.3166.1999.US"/>
    </owl:allValuesFrom>
  </owl:Restriction>
</owl:intersectionOf>
      </owl:Class>
    </owl:equivalentClass>
  </owl:Class>
```

Reasoner support is needed to formulate the similarity/equivalence queries. Assume that a user would like to obtain an Account ACC which has a Type, AmountType and Identifier. Such a query is formulated as follows:

```
<owl:Class rdf:ID="Query">
    <owl:equivalentClass>
      <owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#AggregateCoreComponent"/>
          <owl:Restriction>
            <owl:onProperty rdf:resource="#contains"/>
            <owl:allValuesFrom>
              <owl:Class>
                <owl:intersectionOf rdf:parseType="Collection">
                  <owl:Class rdf:about="#AccountingAccount.Identification.Identifier"/>
                  <owl:Class rdf:about="#AccountingAccount.Type.Code"/>
                  <owl:Class rdf:about="#AccountingAccount.AmountType.Code"/>
                </owl:intersectionOf>
              </owl:Class>
            </owl:allValuesFrom>
          </owl:Restriction>
        </owl:intersectionOf>
      </owl:Class>
    </owl:equivalentClass>
```

```
  </owl:Class>
```

The reasoner puts a subclass of relation from the AccountingAccount.Details ACC, shown below, to the Query as the result has more BCCs than the Query class.

```
<owl:Class rdf:ID="AccountingAccount.Details">
    <owl:equivalentClass>
      <owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#AggregateCoreComponent"/>
          <owl:Restriction>
            <owl:onProperty>
              <owl:ObjectProperty rdf:ID="contains"/>
            </owl:onProperty>
            <owl:allValuesFrom>
              <owl:Class>
                <owl:intersectionOf rdf:parseType="Collection">
                  <owl:Class    rdf:about="#AccountingAccount.Identification.Identifier"/>
                  <owl:Class  rdf:about="#AccountingAccount.SetTrigger.Code"/>
                  <owl:Class rdf:about="#AccountingAccount.Type.Code"/>
                  <owl:Class rdf:about="#AccountingAccount.AmountType.Code"/>
                </owl:intersectionOf>
              </owl:Class>
            </owl:allValuesFrom>
          </owl:Restriction>
        </owl:intersectionOf>
      </owl:Class>
    </owl:equivalentClass>
  </owl:Class>
```

CHAPTER 6

SYSTEM ARCHITECTURE AND IMPLEMENTATION RESULTS

In this chapter, the overall system architecture, a use case of the system and the implementation results are presented. The overall system architecture is presented in Section 6.1. In Section 6.2, a use case to generate mappings between GS1 XML Planning documents [26] and UBL Collaborative Planning, Forecasting and Replenishment documents [76] is described. Finally, the implementation results and the performance of the system is described in Section 6.3.

6.1 SYSTEM ARCHITECTURE AND EVALUATION OF THE IMPLEMENTATION

The overall system framework shown in Figure 6.1 puts together all the parts described and demonstrates how the specified semantics can be used.

6.1.1 THE FRAMEWORK

The framework consists of a harmonized ontology developed as described in Sections 3.1, 3.2, 3.3 and 3.4 (The harmonized ontology and how to construct this harmonized ontology is submitted as a specification to the OASIS SET TC [46]); a repository to store the XPath expressions and the XSLT rules; a wrapper to convert the harmonized ontology to the corresponding set of predicate logic facts; additional rules for expressing the defined heuristics; a DL reasoner and a predicate logic rule engine. Figure 6.1 shows the system components:

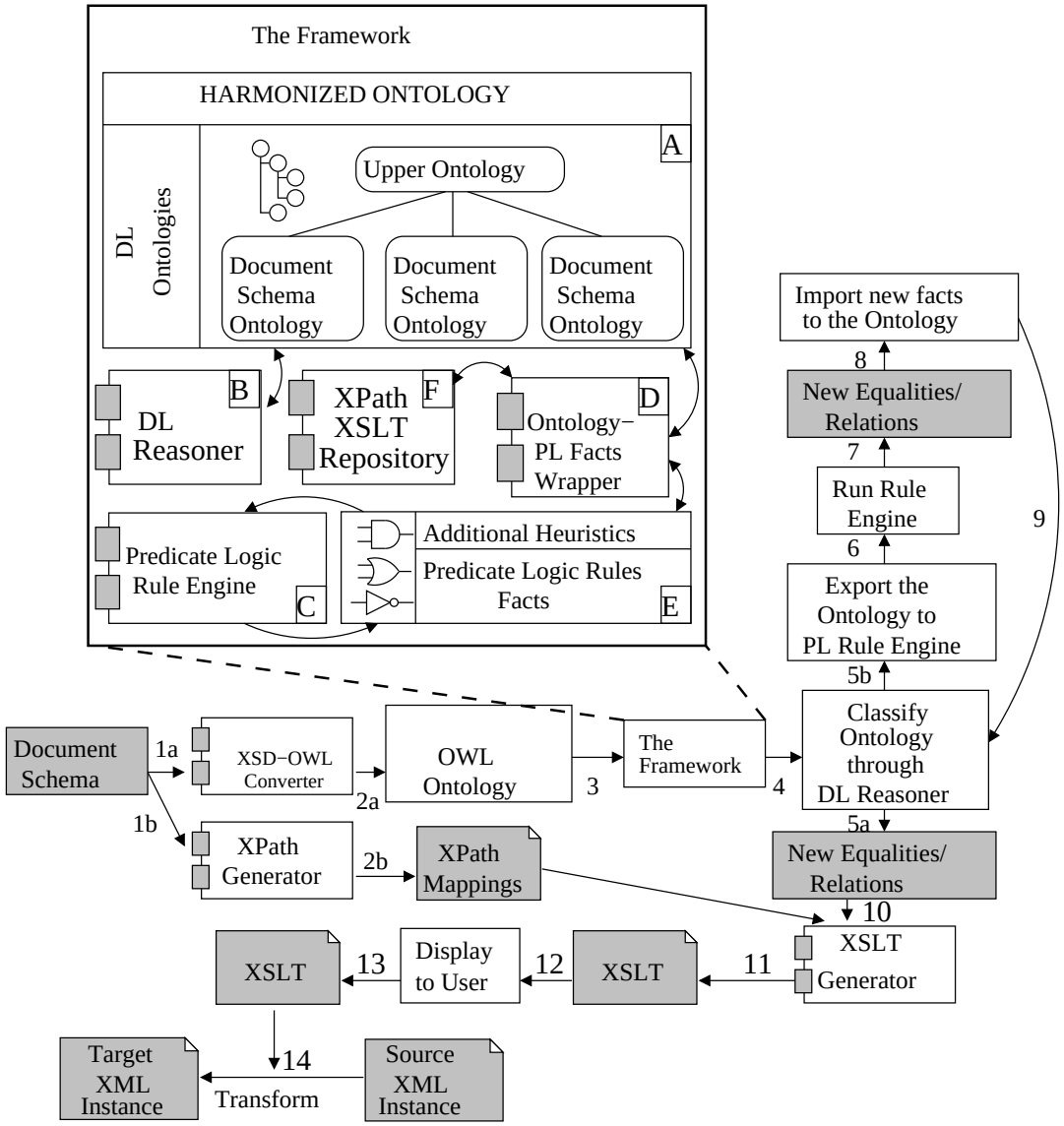


Figure 6.1: The Overall Framework and the Steps of Document Instance Translation

A. *Harmonized Ontology*: This ontology contains two types of OWL-DL ontologies: (1) the Upper Ontology and (2) the Document Schema Ontologies. As mentioned before, the Upper Ontology describes the CCTS artifacts, such as *Business Information Entities* and *Core Components*, as generic classes. The Document Schema Ontologies, on the other hand, describe the BIEs generated by the CCTS based electronic business document standards. There is a Document Schema Ontology for each electronic business document standard. The Document Schema Ontologies are defined conforming to the generic classes defined in the Upper Ontology. The harmonized ontology is obtained by running the DL-Reasoner against the Upper Ontology and Document Schema

Ontologies.

B. *DL-Reasoner*: A Description Logic (DL) Reasoner is used to identify the equivalence and subsumption relations in the harmonized ontology. As the DL reasoner, Racer Pro 1.9.2 Beta [56] is used. The discovered similarities among the document artifacts are then used to generate XSLT definitions for transforming between different electronic business document standards' XML Instances.

C. Predicate Logic Rule Engine: In some cases, the Description Logic is not sufficient to find relations between document artifacts. Therefore, in these cases, generic heuristics in the form of Predicate Logic Rules are used. The JESS Rule Engine [35] is used to execute the heuristics to find additional relations among the Document Schema Ontology classes.

D. Ontology-PL Facts Wrapper: The document artifacts are represented through OWL classes and properties in the harmonized ontology and they are represented as facts in the Predicate Logic Rule Engine. This wrapper converts the OWL definitions to facts definitions, which are then asserted to the rule engine. After the rule engine executes the rules on the facts, new facts are inferred, which represent equivalence relationship among classes. The wrapper converts the newly obtained fact definitions back to OWL class equivalences to be inserted to the harmonized ontology.

E. Additional Heuristics: These heuristics are given through the Predicate Logic Rules to identify the relations among the document artifacts, which cannot be identified through Description Logic.

A number of tools are developed to support this framework:

1. The XSD-OWL Converter: This component converts a CCTS based document schema into Document Schema Ontology as described in Sections 3.2, 3.3 and 3.4.

2. The XPath Generator Tool: The XPath Generator extracts the correspondences between the XSD Schema elements in the document schemas and the OWL classes in the Document Schema Ontologies through XPath expressions. These expressions are then used to generate the XSLT definitions.

3. The XSLT Generator Tool: This component generates the XSLT definitions by using the XPath definitions and the newly computed equivalence/ subsumption relations. These XSLT definitions are used in the transformation between two XML instances conforming to different electronic business document standards.

4. Ontology-PL Facts Wrapper as explained above.

6.1.2 THE DOCUMENT INSTANCE TRANSLATION THROUGH THE FRAMEWORK

This framework is used to transform a source XML document instance to a target XML document instance. If the document schema ontologies corresponding to these document instances are already a part of the harmonized ontology, the corresponding XSLT transformations can directly be generated. If the source, or the target or both of these document schema ontologies are not yet a part of the harmonized ontology, first they must be inserted through the following procedure:

- First the OWL Document Schema Ontology conforming to the OWL structure described in Sections 3.2, 3.3 and 3.4 are created for the XSD Document Schemas (2a) by using the XSD-OWL Converter.
- In the mean time, the XPath Generator Tool is used to keep track of the correspondences among the XSD elements and OWL classes to generate XPath Mappings (2b).
- The OWL ontologies are inserted to the harmonized ontology (3).
- The ontologies are classified with the DL Reasoner and the harmonized ontology is computed (5). In this step new equality/subsumption relations are computed (if there are any) (5a).
- At the same time, the OWL definitions are converted to Predicate Logic facts by using the Ontology-PL Facts Wrapper Tool and asserted to the Predicate Logic Rule Engine (5b).
- The rule engine is executed and new relations are identified as facts (7).
- These facts are converted to OWL Definitions through Ontology-PL Facts Wrapper (8).

- The newly generated OWL Definitions are appended to the harmonized ontology and the DL-Reasoner is executed again to compute new equality/subsumption relations (9). The steps from 5 to 9 are executed repeatedly until the harmonized ontology reaches a certain maturity level (i.e., no further equivalences are computed by the DL reasoner and Predicate Logic Rule Engine).

- The equality/subsumption definitions and the XPath Mappings are input to XSLT Generator to produce XSLT Definitions automatically (11).

- The XSLT Definitions are displayed to the user for further editing (13).

- The XSLTs are used to transform to XML Instances conforming to different standards (14).

6.2 USE CASE: iSURF INTEROPERABILITY SERVICE UTILITY FOR COLLABORATIVE PLANNING, FORECASTING AND REPLENISHMENT

As mentioned in Section 6.3, this harmonized ontology together with the heuristic rules developed are used within the scope of the ICT-213031-iSURF [34] project to generate mappings between GS1 XML Planning documents [26] and UBL Collaborative Planning, Forecasting and Replenishment documents [76]. The planning documents used in the iSURF Project are: “Event”, “Exception Criteria”, “Exception Notification”, “Forecast”, “Forecast Revision”, “Performance History”, “Product Activity”, “Retail Event”, “Trade Item Information Request” and “Trade Item Location Profile”. As a result, 20 XSLT documents are generated by following the approach described in this thesis. The documents are available at iSURF Project Web site [99].

The user interface (client) of the iSURF ISU is Web based and is developed using Adobe Flex. The entrance page is shown in Figure 6.2. In this page, the user selects the document type to be mapped, the source standard and the target standard. In this example, assume that the user selects the Trade Item Location Profile as the message and assume further that the user wants to map from GS1 XML to UBL.

Having selected the document type, in the next step (Figure 6.3), the document models are presented to the user. On the left pane, the GS1 Trade Item Location Profile and on the right

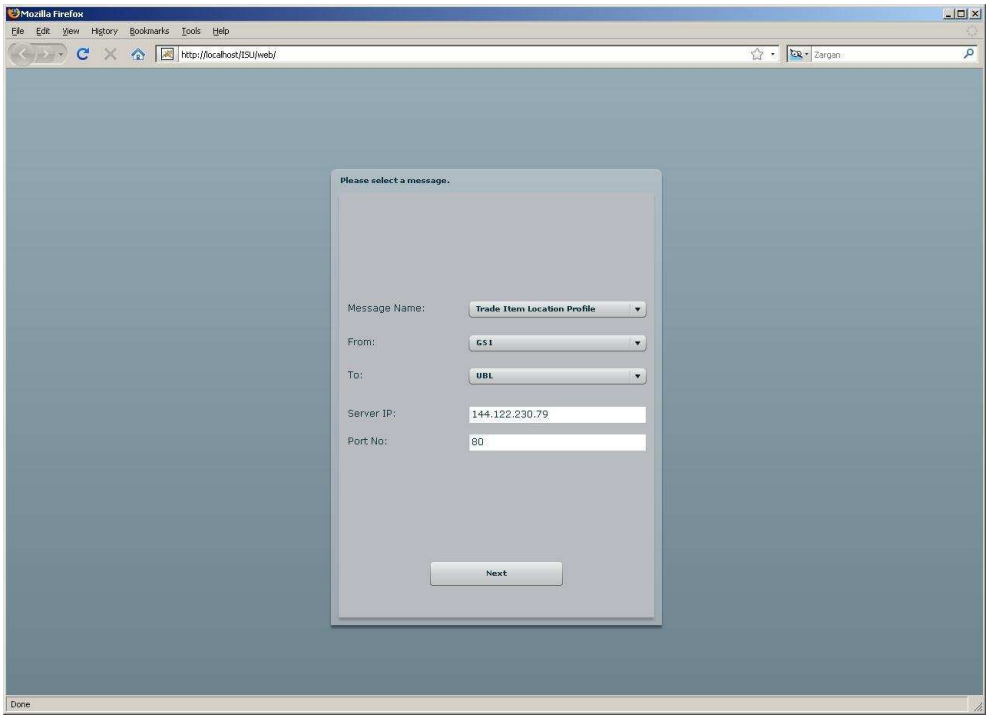


Figure 6.2: iSURF ISU Entrance Page

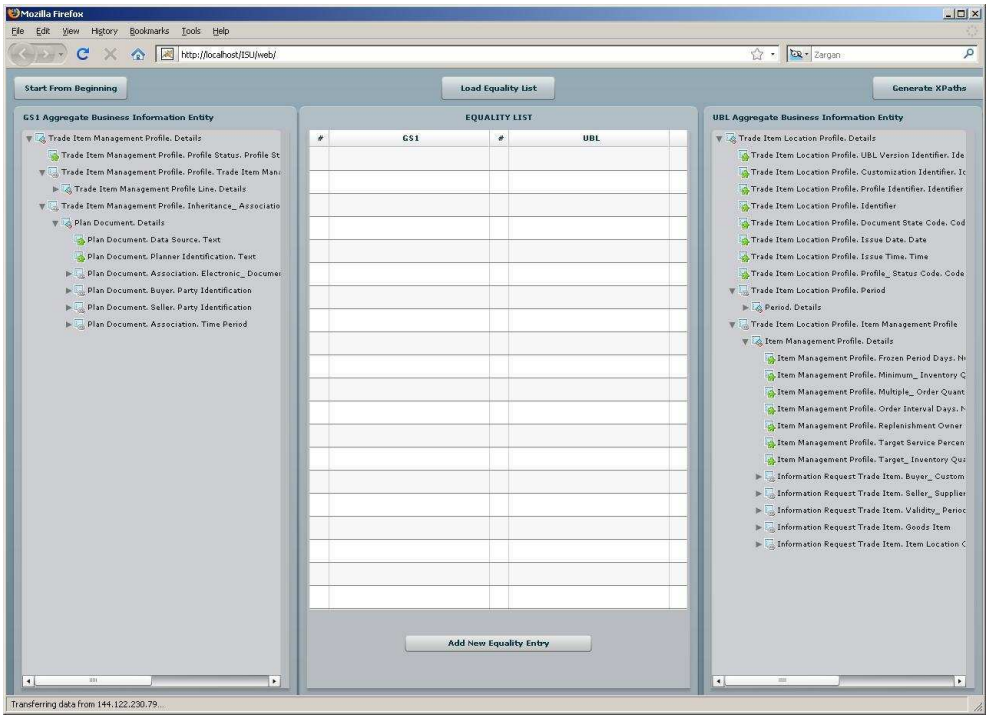


Figure 6.3: Document Content Models

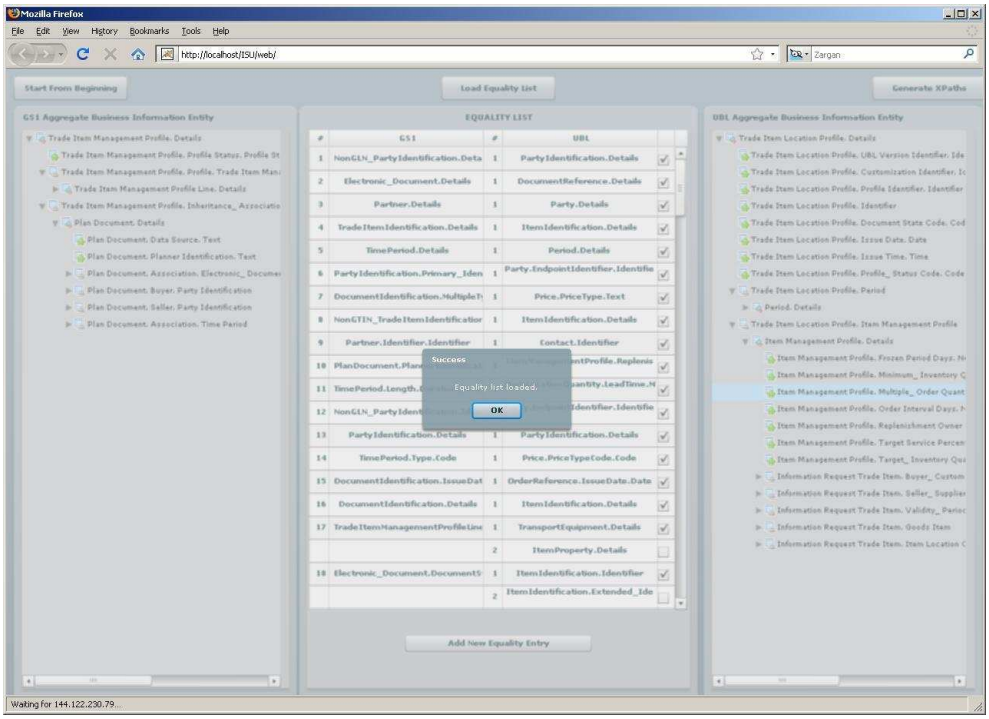


Figure 6.4: Equalities are loaded to the middle pane

pane, the UBL Trade Item Location Profile is presented to the user. On the other hand, in the middle pane, the discovered equalities will be presented to the user.

When the user clicks “Load Equality List” button, the client connects to the iSURF ISU, which in turn directs the ISU to run the description logic reasoner and Jess rule engine. Upon receiving the discovered equalities from the reasoner and rule engine, the ISU sends them to the client. The client displays the equalities to the user in the middle pane as shown in Figure 6.4.

The user can see the elements in the equality in their corresponding document tree by right clicking the equality and selecting “Find Equivalences” item as shown in Figure 6.5. For example, in Figure 6.6, the GS1 XML’s “TimePeriod.Length.Duration_Measure” and UBL’s “Period.Duration.Measure” elements are shown in their document trees.

At this point the user can inspect the equalities and identify whether they are right. If a discovered equality is false-positive, the user can deselect the checkbox right next to the equality. Furthermore, the user can add new equalities by using the “Add New Equality” button. Having completed the examination step, the user clicks the “Generate XPaths” button

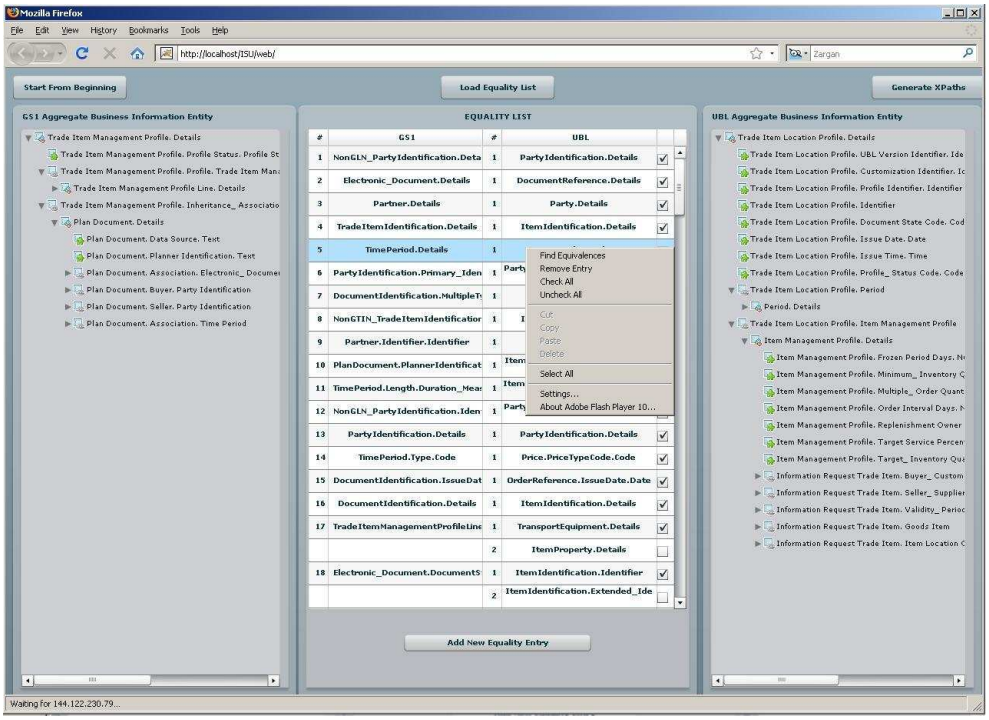


Figure 6.5: Find equivalances item

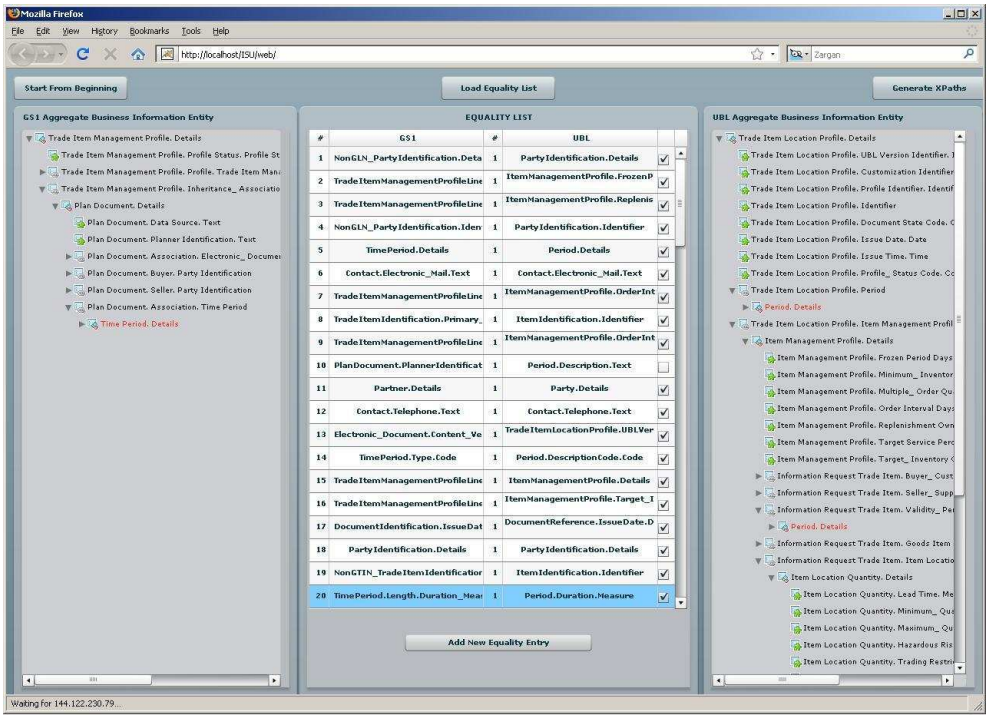


Figure 6.6: Discovered Equivalences in the corresponding document trees

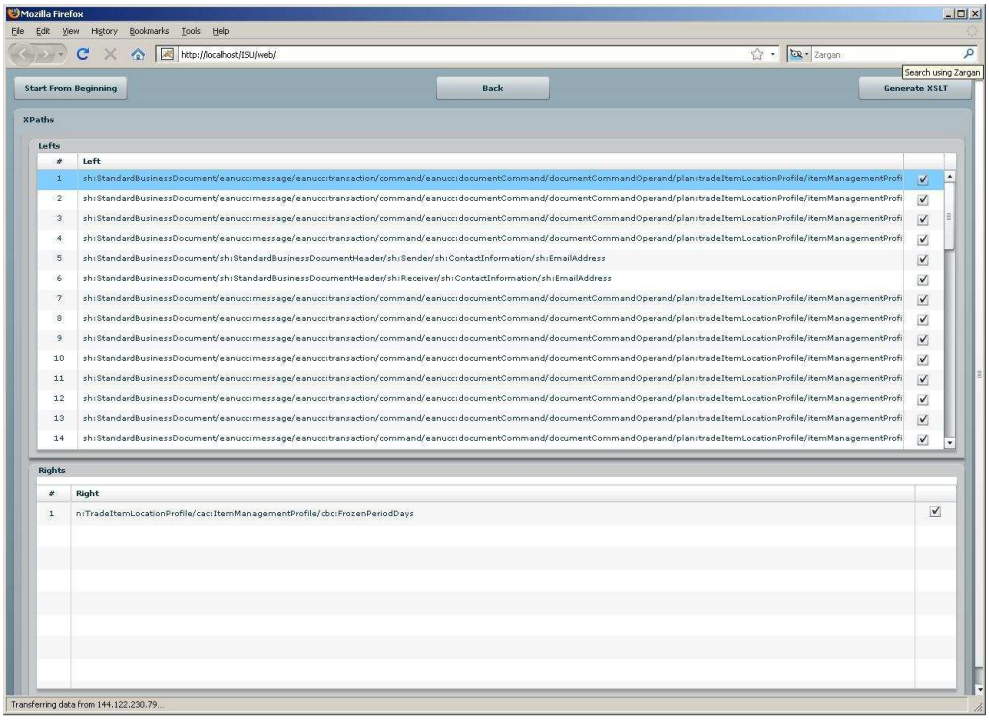


Figure 6.7: Identitfied XPaths

to proceed with the next step, where the XPaths of the elements in the discovered equalities are presented to the user as shown in Figure 6.7.

After that the user clicks the “Generate XSLT” button to generate the XSLT definition (Figure 6.8) to be used to map an instance GS1 Trade Item Location instance to UBL Trade Item Location Instance. It should be noted that the user may further edit the XSLT with any XSLT Editor. In Appendix A.1 the generated XSLT document is presented. Furthermore, in Appendix A.2 the XSLT document for reverse translation is provided.

6.3 THE IMPLEMENTATION AND PERFORMANCE OF THE SYSTEM

The current version of the harmonized ontology, available from [90], contains the ontological representations of:

- All of the CCs and BIEs in UN/CCL 07B.
- All of the BIEs in the common library of UBL 2.0.

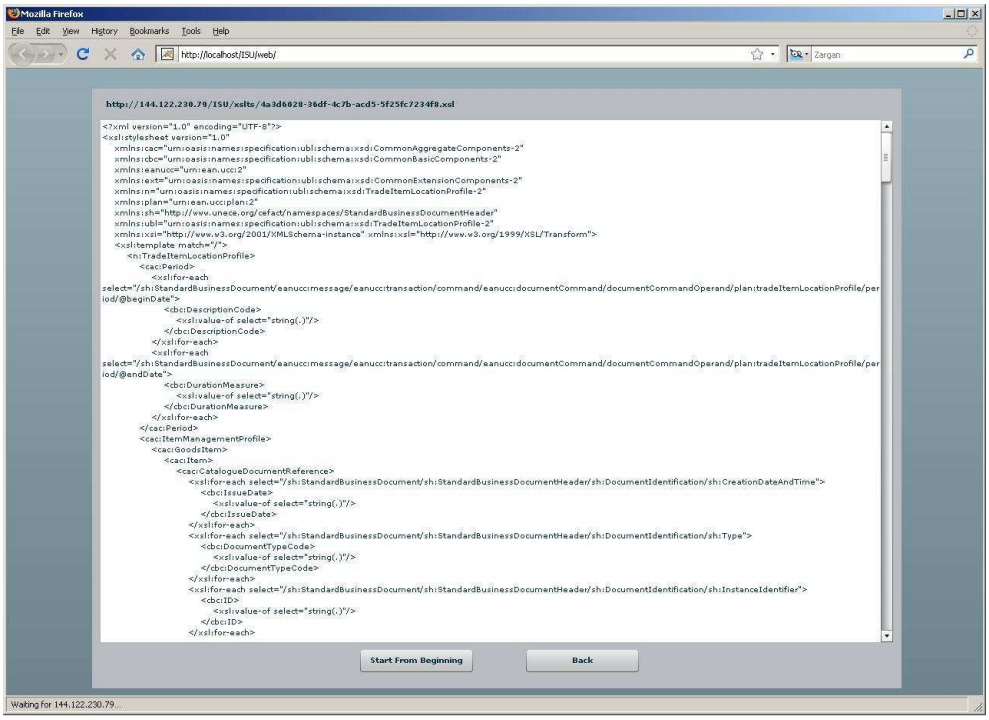


Figure 6.8: Generated XSLT

- All of the OAGIS 9.1 Common Components and Fields.
- All of the elements in the common library of GS1 XML.

There are about 3400 Named OWL Classes and 11000 Restriction Definitions in the harmonized ontology.

This harmonized ontology together with the heuristic rules developed are used within the scope of the ICT-213031-iSURF [34] project to generate mappings between GS1 XML Planning documents [26] and UBL Collaborative Planning, Forecasting and Replenishment documents [76].

The planning documents used in the iSURF Project are: “Event”, “Exception Criteria”, “Exception Notification”, “Forecast”, “Forecast Revision”, “Performance History”, “Product Activity”, “Retail Event”, “Trade Item Information Request” and “Trade Item Location Profile”. The success rate in discovering the document components with similar information content is 88.10% for aggregate document components (ABIEs) and 89.33% for basic document components (BBIEs). The rate of false positives is about 10%.

Related with performance, an issue that needs to be addressed is whether the gain in automation justifies the resources needed to develop the ontological representation of the document schemas. In order to reduce this cost, as already described, automated tool support is provided to create OWL definitions of the document schemas. Additionally, by conforming to a standard ontological representation and hence having all the document schema ontologies in a common pool, the users of the harmonized ontology only need to create a document schema ontology if it is not already in the harmonized ontology and benefit from all the existing connections when they do so.

Another issue related with performance is the computational complexity of the reasoning process involved. On a PC with 2GB RAM, the Racer Pro 1.9.2 Beta reasoner [56], it takes about 120 seconds to compute the harmonized ontology. The harmonized ontology will be re-computed in limited cases:

- When the predicate rule engine is run to generate more equivalent classes, or,
- When a new document schema is introduced to the system, or,
- When a new CCTS based upper document ontology is introduced to the system

Therefore this performance is considered satisfactory.

CHAPTER 7

CONCLUSIONS AND THE FUTURE WORK

Today, an enterprise's competitiveness is, to a large extent, determined by its ability to seamlessly interoperate with others, and the electronic document standards play an important role in this. However, given the large number of electronic business document standards, conformance to one of these standards or implementing a few of them will not solve the interoperability problem; there will always be some companies using a different, incompatible document standard. Therefore, there is a need for semantic tools to support electronic document interoperability.

Semantics is domain specific knowledge and the success of the developed tools heavily depends on the amount of domain knowledge available. In this respect, UN/CEFACT CCTS has achieved an important milestone and provided the semantics of the electronic business documents.

In this thesis, this semantics is explicated and show how to use it for semi-automated translation of document schema instances from different standards. Both the success rate in discovering the similarities of document artifacts and the performance of the system are promising for industry take-up.

A possible obstacle to industry take-up of the described mechanisms may stem from the different representations of the same meaning. For example, what one representation may choose as a subclass, may be represented as an object property in another representation. When this happens, harmonizing different ontological representations will become a challenge. Therefore, we have initiated a standardization effort for the representation of CCTS semantics and have formed the OASIS "Semantic Support for Electronic Business Document Interoperability (SET)" Technical Committee [45]. In this way, that is, having a common semantic descrip-

tion of the CCTS based document standards, it will be possible to collect all descriptions in a common pool and when the semantics of a new document schema is introduced to the pool, it will benefit from the already existing connections in the harmonized ontology. We anticipate that this initiative will increase the industry take-up of the work. Although the framework is tested in collaborative planning domain, it is generic and it can be applied to all domains having CCTS based document standards. The current status of the heuristics has achieved a promising success rate. Once a further insight is gained to a large number of document schemas, additional heuristics may be developed as a future work.

REFERENCES

[1] Automotive Aftermarket Industry Association. http://www.aftermarket.org/Home.asp (2009).

[2] Automotive Industry Action Group. http://www.aiag.org/ (2009).

[3] N. Anicic, N. Ivezic, "Semantic Web Technologies for Enterprise Application Integration", ComSIS, Computer Science and Information Systems, International Journal, Volume 02 , Issue 01, June 2005 (2005).

[4] UN/CEFACT Applied Technology Group (ATG) XML Syntax, XML Naming and Design Rules, http://www.uncefactforum.org/ATG/Documents/ATG/Downloads/-XMLNamingAndDesignRulesV2.0.pdf (2009).

[5] Franz Baader, Ian Horrocks, Ulrike Sattler, "Chapter 3: Desription Logics", http://www.cs.man.ac.uk/ horrocks/Publications/download/2007/BaHS07a.pdf (2009).

[6] Brown and Reynolds, Strategy for production and maintenance of standards for interoperability within and between service departments and other healthcare domains, CEN/TC 251 Health Informatics, CEN/TC 251/N00-047.

[7] CBL. Common Business Library, http://xml.coverpages.org/cbl.html (2009).

[8] Chemical Industry Data Exchange, http://www.cidx.org/ (2009).

[9] Collaborative Planning, Forecasting and Replenishment (CPFR ©) Guidelines , http://www.vics.org/ (2009).

[10] CLR TC, The OASIS Code List Representation Technical Committee, http://www.oasis-open.org/committees/codelist (2009).

[11] Crawford, M. Crawford, Core Components Adoption On The Rise, https://www.sdn.sap.com/irj/sdn/weblogs?blog=/pub/wlg/5395.

[12] cXML, Commerce XML, http://cxml.org/ (2009).

[13] EAN, European Article Number, http://en.wikipedia.org/wiki/European_Article_Number/ (2009).

[14] EANCOM, European Article Number Communication, http://www.gs1.org/productssolutions/ecom/eancom/ (2009).

[15] ebBP, ebXML Business Process, http://docs.oasis-open.org/ebxml-bp/2.0.4/OS/ (2009).

[16] EDI, Electronic Data Interchange, http://en.wikipedia.org/wiki/Electronic_Data-_Interchange (2009).

[17] EDIINT-AS1, T. Harding, R. Drummond, C. Shih, MIME-based Secure Peer-to-Peer Business Data Interchange over the Internet, RFC 3335, Sept 2002, http://www.ietf.org/rfc/rfc3335.txt (2009).

[18] EDIINT-AS2, D. Moberg, R. Drummond, MIME-Based Secure Peer-to-Peer Business Data Interchange Using HTTP, Applicability Statement 2 (AS2), RFC 4130, July 2005, http://www.ietf.org/rfc/rfc4130.txt (2009).

[19] EFT, Electronic Funds Transfer, http://en.wikipedia.org/wiki/Electronic_funds_transfer (2009).

[20] EPCglobal, Electronic Product Code Global, http://www.gs1.org/productssolutions/-epcglobal/ (2009).

[21] FaCT++ Resoner, http://owl.man.ac.uk/factplusplus/ (2009).

[22] GDSN, GS1 Global Data Synchronisation Network, http://www.gs1.org/productssolutions/gdsn/ (2009).

[23] GS1, Global Standard One, http://www.gs1.org/ (2009).

[24] Global Standard One, Global Data Dictionary, http://gdd.gs1.org/ (2009).

[25] Global Standard One XML, http://www.gs1.org/productssolutions/ecom/xml/ (2009).

[26] GS1 XML Plan Documents, http://www.gs1.org/services/gsmp/kc/ecom/xml/-plan_grid.html (2009).

[27] HL7, Health Level 7, http://www.hl7.org/ (2009).

[28] IATA, International Air Transport Association, http://www.iata.org/index.htm (2009).

[29] ICH, ANSI ASC X12 ISA Interchange Control Header Segment, http://www.rawlinsecconsulting.com/x12tutorial/x12syn.html (2009).

[30] ICHS, UN/EDIFACT UNB Interchange Header Segment, http://www.unece.org/trade/edifact/untdid/d422_s.htm (2009).

[31] Institute of Electrical and Electronics Engineers. IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries, New York (1990).

[32] ISO Codes, International Standards Organization Codes, http://www.unece.org/cefact/codesfortrade/codes_index.htm (2009).

[33] ISO/IEC 11179-5: Naming and identification principles, http://standards.iso.org/ittf/PubliclyAvailableStandards/c035347_ISO_IEC_11179-5_2005(E).zip (2009).

[34] IST-213031 iSURF, An Interoperability Service Utility for Collaborative Supply Chain Planning across Multiple Domains Supported by RFID Devices, http://www.srdc.com.tr/isurf/ (2009).

[35] Jess: Java Rule Engine, http://herzberg.ca.sandia.gov/ (2009).

[36] Kabak Y., Dogac A. “A Survey and Analysis of Electronic Business Document Standards”. Accepted for publication in ACM Computing Surveys (2008).

[37] KAON2 Resoner, http://kaon2.semanticweb.org/ (2009).

[38] Knowledge Interchange Format (KIF), http://logic.stanford.edu/kif/kif.html (2009).

[39] Thorsten Liebig, “Reasoning with OWL - System Support and Insights”, Technical Report, September 2006.

[40] B. Medjahed, B. Benatallah, A. Bouguettaya, A.H.H. Ngu, A.K. Elmagarmid, Business-to-Business Interactions: Issues and Enabling Technologies, The International Journal on Very Large Data Bases, Vol. 12, No. 1, May 2003.

[41] MIT-AutoID, Auto-ID Labs at MIT, http://autoid.mit.edu/cs/ (2009).

[42] MoU, Memorandum of Understanding on electronic business between IEC, ISO, ITU, and UN/ECE, http://www.itu.int/ITU-T/e-business/files/mou.pdf (2009).

[43] NES, UBL Northern European Subset, http://www.nesubl.eu/ (2009).

[44] OAGi, Open Applications Group, http://www.openapplications.org/ (2009).

[45] OASIS Semantic Support for Electronic Business Document Interoperability (SET) TC, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=set (2009).

[46] OASIS SET TC Specification, Semantic Representations of the UN/CEFACT CCTS-based Electronic Business Document Artifacts, http://www.oasis-open.org/committees/download.php/29436/20080924SemanticRepresentationOfDocumentArtifacts.doc (2008).

[47] Ontolingua, http://ksl.stanford.edu/software/ontolingua/ (2009).

[48] Open Applications Group Integration Specification 9.0, http://www.openapplications.org/downloads/oagis/loadfrm9.htm (2009).

[49] Open Applications Group (OAGi) at 10 Years: A Look Back and Forward, http://webservices.sys-con.com/read/47282.htm (2009).

[50] Organisation for Data Exchange by Tele Transmission in Europe, http://www.odette.org/html/home.htm (2009).

[51] Offentlig Information Online UBL, http://www.oio.dk/dataudveksling/ehandel/-hoeringer/oioubl (2009).

[52] Oracle Corporation, http://www.oracle.com/products/middleware/docs/oracle_-ebs_and_soa.pdf (2009). http://www.oracle.com/technology/products/applications/-integration/1147_EBS_and_SOA.ppt (2009).

[53] OTA, OpenTravel Alliance, http://www.opentravel.org/ (2009).

[54] Web Ontology Language, http://www.w3.org/2004/OWL/ (2009).

[55] Pellet Reasoner, http://clarkparsia.com/pellet/ (2009).

[56] RacerPro: DL Reasoner, http://www.racer-systems.com/ (2009).

[57] PIDX, Petroleum Industry Data Exchange, http://www.pidx.org/ (2009).

[58] E. Rahm, P. A. Bernstein “A survey of approaches to automatic schema matching”, The VLDB Journal, Vol. 10, 2001, pp. 334-350 (2001).

[59] Resource Description Framework (RDF), http://www.w3.org/TR/rdf-concepts/ (2009).

[60] RDF Schema (RDFS), http://www.w3.org/TR/rdf-schema/ (2009).

[61] RosettaNet. http://www.rosettanet.org/ (2009).

[62] M. Rowell, The Open Applications Group Integration Specification, http://www.ibm.com/developerworks/xml/library/x-oagis/ (2009).

[63] SAP, SAP - Systemanalyse und Programmentwicklung, http://www.sap.com/index.epx (2009).

[64] Schematron, http://www.ldodds.com/papers/schematron_xsltuk.html (2009).

[65] SITC, http://unstats.un.org/unsd/cr/registry/regcst.asp?Cl=14 (2009).

[66] SPARQL Query Language for RDF, http://www.w3.org/TR/rdf-sparql-query/ (2009).

[67] SPARQL-DL Implementation Experience, http://tw.rpi.edu/wiki/index.php/Special:-Browse/SPARQL-DL_Implementation_Experience (2009).

[68] Standards for Technology in Automotive Retail, http://www.starstandard.org/ (2009).

[69] Giorgos Stoilos, Giorgos Stamou and Stefanos Kollias, “A String Metric for Ontology Alignment”, Lecture Notes in Computer Science, Volume 3729/2005.

[70] Stuhec, Gunther Stuhec, How to Solve the Business Standards Dilemma - The CCTS based Core Data Types, https://www.sdn.sap.com/irj/sdn/go/portal/prtroot/docs/library/uuid/500db5c9-0e01-0010-81aa-d73cdd30df9a,

[71] Stuhec2, Gunther Stuhec, How to Solve the Business Standards Dilemma: The Context Driven Business Exchange, https://www.sdn.sap.com/irj/servlet/prt/portal/prtroot/docs/library/uuid/a6c5dce6-0701-0010-45b9-f6ca8c0c6474,

[72] Swedish Invoice, http://www.svefaktura.se/SFTI_Basic_Invoice20051130_EN/-SFTI%20Basic%20Invoice_1.0/index.html (2009).

[73] SWRL: A Semantic Web Rule Language, http://www.w3.org/Submission/SWRL/ (2009).

[74] SWIFT, Society for Worldwide Interbank Financial Telecommunication, http://www.swift.com/ (2009).

[75] Universal Business Language, http://www.oasis-open.org/committees/ubl/ (2009).

[76] UBL CPFR Documents, http://www.oasis-open.org/committees/download.php/28979/-UBLforCPFR.zip (2008).

[77] UBL NDR, Universal Business Language Naming and Design Rules, http://docs.oasis-open.org/ubl/os-UBL-2.0/doc/ndr/NDR-checklist.pdf (2009).

[78] UBL-SBS, Universal Business Language Small Business Subcommittee, http://www.oasis-open.org/committees/sc_home.php?wg_abbrev=ubl-sbsc (2009).

[79] UBLSchemas, Universal Business Language 2.0 Schemas, http://docs.oasis-open.org/ubl/os-UBL-2.0/ (2009).

[80] UBP, Universal Business Process, http://docs.oasis-open.org/ubl/cs-UBL-1.0-SBS-1.0/universal-business-process-1.0-ebBP/ (2009).

[81] UCC, Uniform Code Council, http://www.uc-council.org/ (2009).

[82] UN/CCL, United Nations Core Component Library, http://www.unece.org/cefact/codesfortrade/unccl/CCL07A.xls (2009).

[83] UN/CEFACT Core Components Technical Specification, http://www.unece.org/cefact/ebxml/CCTS_V2-01_Final.pdf (2009).

[84] UN/CEFACT Registry Implementation Specification, http://www.unece.org/cefact/documents/reg_specific_vor6.zip (2009).

[85] UN/EDIFACT, United Nations Directories for Electronic Data Interchange for Administration, Commerce and Transport, http://www.unece.org/trade/untdid/welcome.htm (2009).

[86] UN/EDIFACT 1131, UN/EDIFACT 1131 Data Element, Code list identification code, http://www.unece.org/trade/untdid/d00a/tred/tred1131.htm (2009).

[87] UN/EDIFACT 3055, UN/EDIFACT 3055 Data Element, Code list responsible agency code, http://www.unece.org/trade/untdid/d00a/tred/tred3055.htm (2009).

[88] UN/SBDH, UN/CEFACT Standard Business Document Header Technical Specification, http://www.gs1.org/docs/gsmp/xml/sbdh/CEFACT_SBDH_TS_version1.3.pdf (2009).

[89] UNSPSC, Product Classification, http://www.unspsc.org/ (2009).

[90] Harmonized Ontology, http://www.srdc.metu.edu.tr/iSURF/OASIS-SET-TC/ontology/HarmonizedOntology.owl (2009).

[91] US/DOT, US Department of Transportation UBL Implementation, http://www.oasis-open.org/committees/ubl/faq.php (2009).

[92] Ken Vollmer, B2B Integration Trends, Forrester, http://www.forrester.com/Research/Document/Excerpt/0,7211,42735,00.html (2009)

[93] E. Wustner, T. Hotzel, P. Buxmann, Converting Business Documents: A Classification of Problems and Solutions using XML/XSLT, Proc. of the 4th IEEE Intl Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems (WECWIS), 2002.

[94] X12, EDI ANSI X12, http://www.x12.org/ (2009).

[95] xCBL, XML Common Business Library, http://www.xcbl.org/ (2009).

[96] XML, Extensible Markup Language, http://www.w3.org/XML/ (2009).

[97] Extensible Stylesheet Language, http://www.w3.org/Style/XSL/ (2009).

[98] XSL Transformations (XSLT), http://www.w3.org/TR/xslt (2009).

[99] Generated Mappings, http://www.srdc.com.tr/isurf/documents/Mappings.rar (2009).

[100] Y. Yarimagan, A. Dogac “Semantics Based Customization of UBL Document Schemas”, Journal of Distributed and Parallel Databases, Springer-Verlag, Volume 22, Numbers 2-3 / December 2007, pp. 107-131 (2007).

[101] Y. Yarimagan, A. Dogac “A Semantic based Solution for the Interoperability of UBL Schemas”. IEEE Internet Computing, to appear (2008).

[102] Y. Ye, D. Yang, Z. Jiang, L. Tong “Ontology-based semantic models for supply chain management”, The International Journal of Advanced Manufacturing Technology, Published online, Springer London, May 2007 (2007).

APPENDIX A

GENERATED XSLT DOCUMENTS

A.1 THE XSLT FILE FOR TRANSLATING THE TRADE ITEM LOCATION PROFILE INSTANCES FROM GS1 XML TO UBL

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet>
 <xsl:namespace-alias stylesheet-prefix="n" result-prefix="#default"/>
 <xsl:output method="xml" encoding="UTF-8" indent="yes"/>
 <xsl:template match="/n2:documentCommand">
  <n:TradeItemLocationProfile>
   <xsl:attribute name="xsi:schemaLocation">
    <xsl:value-of select="'urn:oasis:names:specification:ubl:schema:xsd:TradeItemLocationProfile-2
UBL-TradeItemLocationProfile-2.1.xsd'"/>
   </xsl:attribute>
   <xsl:variable name="Vvar2_const" select="''"/>
   <cbc:ID>
    <xsl:value-of select="string($Vvar2_const)"/>
   </cbc:ID>
   <xsl:for-each select="documentCommandOperand">
    <xsl:for-each select="plan:tradeItemLocationProfile">
     <xsl:for-each select="@documentStatus">
      <xsl:variable name="Vvar150_documentStatus_string" select="string(.)"/>
      <cbc:DocumentStateCode>
       <xsl:value-of select="string($Vvar150_documentStatus_string)"/>
      </cbc:DocumentStateCode>
     </xsl:for-each>
    </xsl:for-each>
   </xsl:for-each>
   <xsl:for-each select="documentCommandOperand">
    <xsl:for-each select="plan:tradeItemLocationProfile">
     <xsl:for-each select="@creationDateTime">
      <xsl:variable name="Vvar159_creationDateTime_dateTime" select="string(.)"/>
      <cbc:IssueDate>
       <xsl:value-of select="string(substring-before(
```

```
string($Vvar159_creationDateTime_dateTime), 'T'))"/>
      </cbc:IssueDate>
     </xsl:for-each>
    </xsl:for-each>
   </xsl:for-each>
   <xsl:for-each select="documentCommandOperand">
    <xsl:for-each select="plan:tradeItemLocationProfile">
     <xsl:for-each select="@creationDateTime">
      <xsl:variable name="Vvar170_creationDateTime_dateTime" select="string(.)"/>
      <cbc:IssueTime>
       <xsl:value-of select="string(substring-after(
string($Vvar170_creationDateTime_dateTime), 'T'))"/>
      </cbc:IssueTime>
     </xsl:for-each>
    </xsl:for-each>
   </xsl:for-each>
   <xsl:for-each select="documentCommandOperand">
    <xsl:for-each select="plan:tradeItemLocationProfile">
     <xsl:for-each select="profileStatus">
      <xsl:variable name="Vvar181_profileStatus_string" select="string(.)"/>
      <cbc:ProfileStatusCode>
       <xsl:value-of select="string($Vvar181_profileStatus_string)"/>
      </cbc:ProfileStatusCode>
     </xsl:for-each>
    </xsl:for-each>
   </xsl:for-each>
   <xsl:for-each select="documentCommandOperand">
    <xsl:for-each select="plan:tradeItemLocationProfile">
     <xsl:for-each select="period">
      <cac:Period>
       <xsl:for-each select="@beginDate">
<xsl:variable name="Vvar194_beginDate_date" select="string(.)"/>
<cbc:StartDate>
 <xsl:value-of select="$Vvar194_beginDate_date"/>
</cbc:StartDate>
       </xsl:for-each>
       <xsl:for-each select="@endDate">
<xsl:variable name="Vvar198_endDate_date" select="string(.)"/>
<cbc:EndDate>
 <xsl:value-of select="$Vvar198_endDate_date"/>
</cbc:EndDate>
       </xsl:for-each>
      </cac:Period>
     </xsl:for-each>
    </xsl:for-each>
   </xsl:for-each>
   <xsl:for-each select="documentCommandOperand">
```

```
    <xsl:for-each select="plan:tradeItemLocationProfile">
     <xsl:for-each select="itemManagementProfile">
      <cac:ItemManagementProfile>
       <xsl:for-each select="frozenPeriodDays">
<xsl:variable name="Vvar207_frozenPeriodDays_integer"
 select="number(.)"/>
<cbc:FrozenPeriodDays>
 <xsl:value-of select="number($Vvar207_frozenPeriodDays_integer)"/>
</cbc:FrozenPeriodDays>
       </xsl:for-each>
       <xsl:for-each select="minimumInventory">
<xsl:variable name="Vvar211_minimumInventory_float" select="number(.)"/>
<cbc:MinimumInventoryQuantity>
 <xsl:value-of select="number($Vvar211_minimumInventory_float)"/>
</cbc:MinimumInventoryQuantity>
       </xsl:for-each>
       <xsl:for-each select="orderQuantityMultiple">
<xsl:variable name="Vvar215_orderQuantityMultiple_float"
 select="number(.)"/>
<cbc:MultipleOrderQuantity>
 <xsl:value-of select="number($Vvar215_orderQuantityMultiple_float)"/>
</cbc:MultipleOrderQuantity>
       </xsl:for-each>
       <xsl:for-each select="orderIntervalDays">
<xsl:variable name="Vvar219_orderIntervalDays_integer"
 select="number(.)"/>
<cbc:OrderIntervalDays>
 <xsl:value-of select="number($Vvar219_orderIntervalDays_integer)"/>
</cbc:OrderIntervalDays>
       </xsl:for-each>
       <xsl:for-each select="replenishmentOwner">
<xsl:variable name="Vvar225_replenishmentOwner_string"
 select="string(.)"/>
<cbc:ReplenishmentOwnerDescription>
 <xsl:value-of select="$Vvar225_replenishmentOwner_string"/>
</cbc:ReplenishmentOwnerDescription>
       </xsl:for-each>
       <xsl:for-each select="targetServiceLevel">
<xsl:variable name="Vvar229_targetServiceLevel_decimal"
 select="number(.)"/>
<cbc:TargetServicePercent>
 <xsl:value-of select="$Vvar229_targetServiceLevel_decimal"/>
</cbc:TargetServicePercent>
       </xsl:for-each>
       <xsl:for-each select="targetInventory">
<xsl:variable name="Vvar230_targetInventory" select="."/>
<xsl:for-each select="value">
```

```
 <xsl:variable name="Vvar239_value_decimal" select="number(.)"/>
 <cbc:TargetInventoryQuantity>
  <xsl:for-each select="$Vvar230_targetInventory/@unitOfMeasure">
   <xsl:variable name="Vvar236_unitOfMeasure_string"
select="string(.)"/>
   <xsl:attribute name="unitCode">
    <xsl:value-of select="string($Vvar236_unitOfMeasure_string)"/>
   </xsl:attribute>
  </xsl:for-each>
  <xsl:value-of select="$Vvar239_value_decimal"/>
 </cbc:TargetInventoryQuantity>
</xsl:for-each>
       </xsl:for-each>
       <xsl:for-each select="collaborativeTradeItem">
<xsl:for-each select="buyerLocation">
 <cac:BuyerCustomerParty>
  <cac:Party>
   <cac:PartyIdentification>
    <xsl:for-each select="gln">
     <xsl:variable name="Vvar245_gln_string"
select="string(.)"/>
     <cbc:ID>
      <xsl:value-of select="string($Vvar245_gln_string)"/>
     </cbc:ID>
    </xsl:for-each>
   </cac:PartyIdentification>
  </cac:Party>
 </cac:BuyerCustomerParty>
</xsl:for-each>
       </xsl:for-each>
       <xsl:for-each select="collaborativeTradeItem">
<xsl:for-each select="sellerLocation">
 <cac:SellerSupplierParty>
  <cac:Party>
   <cac:PartyIdentification>
    <xsl:for-each select="gln">
     <xsl:variable name="Vvar253_gln_string"
select="string(.)"/>
     <cbc:ID>
      <xsl:value-of select="string($Vvar253_gln_string)"/>
     </cbc:ID>
    </xsl:for-each>
   </cac:PartyIdentification>
  </cac:Party>
 </cac:SellerSupplierParty>
</xsl:for-each>
       </xsl:for-each>
```

```
       <cac:GoodsItem>
<cbc:ID>
 <xsl:value-of select="string($Vvar2_const)"/>
</cbc:ID>
<cac:Item>
 <cac:StandardItemIdentification>
  <xsl:for-each select="collaborativeTradeItem">
   <xsl:for-each select="product">
    <xsl:for-each select="gtin">
     <xsl:variable name="Vvar263_gtin_string"
select="string(.)"/>
     <cbc:ID>
      <xsl:value-of select="string($Vvar263_gtin_string)"/>
     </cbc:ID>
    </xsl:for-each>
   </xsl:for-each>
  </xsl:for-each>
 </cac:StandardItemIdentification>
</cac:Item>
       </cac:GoodsItem>
       <cac:ItemLocationQuantity>
<xsl:for-each select="orderingLeadTimeDays">
 <xsl:variable name="Vvar267_orderingLeadTimeDays_integer"
select="number(.)"/>
 <cbc:LeadTimeMeasure>
  <xsl:attribute name="unitCode">
   <xsl:value-of select="string('DAY')"/>
  </xsl:attribute>
  <xsl:value-of select="
number($Vvar267_orderingLeadTimeDays_integer)"/>
 </cbc:LeadTimeMeasure>
</xsl:for-each>
       </cac:ItemLocationQuantity>
      </cac:ItemManagementProfile>
     </xsl:for-each>
    </xsl:for-each>
   </xsl:for-each>
  </n:TradeItemLocationProfile>
 </xsl:template>
</xsl:stylesheet>
```

A.2 THE XSLT FILE FOR TRANSLATING THE TRADE ITEM LOCATION PROFILE INSTANCES FROM UBL TO GS1 XML

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet>
 <xsl:namespace-alias stylesheet-prefix="n" result-prefix="#default"/>
 <xsl:output method="xml" encoding="UTF-8" indent="yes"/>
 <xsl:template match="/n2:TradeItemLocationProfile">
  <n:documentCommand>
   <xsl:attribute name="xsi:schemaLocation">
    <xsl:value-of select="'urn:ean.ucc:2 TradeItemLocationProfileProxy.xsd'"/>
   </xsl:attribute>
   <xsl:variable name="Vvar1_firstSource" select="."/>
   <documentCommandOperand xmlns="">
    <plan:tradeItemLocationProfile>
     <xsl:for-each select="cbc:IssueDate">
      <xsl:variable name="Vvar114_IssueDate_date" select="string(.)"/>
      <xsl:for-each select="$Vvar1_firstSource/cbc:IssueTime">
       <xsl:variable name="Vvar118_IssueTime_time" select="string(.)"/>
       <xsl:attribute name="creationDateTime">
<xsl:value-of select="string(concat(concat(string
($Vvar114_IssueDate_date), 'T'),
string($Vvar118_IssueTime_time)))"/>
       </xsl:attribute>
      </xsl:for-each>
     </xsl:for-each>
     <xsl:for-each select="cbc:DocumentStateCode">
      <xsl:variable name="Vvar125_DocumentStateCode_normalizedString"
select="string(.)"/>
      <xsl:attribute name="documentStatus">
       <xsl:value-of select="string($Vvar125_DocumentStateCode_normalizedString)"/>
      </xsl:attribute>
     </xsl:for-each>
     <xsl:for-each select="cac:Period">
      <period>
       <xsl:for-each select="cbc:StartDate">
<xsl:variable name="Vvar134_StartDate_date" select="string(.)"/>
<xsl:attribute name="beginDate">
 <xsl:value-of select="$Vvar134_StartDate_date"/>
</xsl:attribute>
       </xsl:for-each>
       <xsl:for-each select="cbc:EndDate">
<xsl:variable name="Vvar138_EndDate_date"
select="string(.)"/>
<xsl:attribute name="endDate">
 <xsl:value-of select="$Vvar138_EndDate_date"/>
```

```
</xsl:attribute>
       </xsl:for-each>
      </period>
     </xsl:for-each>
     <xsl:for-each select="cbc:ProfileStatusCode">
      <xsl:variable name="Vvar141_ProfileStatusCode_normalizedString"
select="string(.)"/>
      <profileStatus>
       <xsl:value-of select="string($Vvar141_ProfileStatusCode_normalizedString)"/>
      </profileStatus>
     </xsl:for-each>
     <xsl:for-each select="cac:ItemManagementProfile">
      <itemManagementProfile>
       <xsl:for-each select="cbc:FrozenPeriodDays">
<xsl:variable name="Vvar148_FrozenPeriodDays_decimal"
select="number(.)"/>
<frozenPeriodDays>
 <xsl:value-of select="number($Vvar148_FrozenPeriodDays_decimal)"/>
</frozenPeriodDays>
       </xsl:for-each>
       <xsl:for-each select="cbc:MinimumInventoryQuantity">
<xsl:variable name="Vvar152_MinimumInventoryQuantity_decimal"
select="number(.)"/>
<minimumInventory>
 <xsl:value-of select="number($Vvar152_MinimumInventoryQuantity_decimal)"/>
</minimumInventory>
       </xsl:for-each>
       <xsl:for-each select="cbc:OrderIntervalDays">
<xsl:variable name="Vvar156_OrderIntervalDays_decimal"
select="number(.)"/>
<orderIntervalDays>
 <xsl:value-of select="number($Vvar156_OrderIntervalDays_decimal)"/>
</orderIntervalDays>
       </xsl:for-each>
       <xsl:for-each select="cbc:MultipleOrderQuantity">
<xsl:variable name="Vvar160_MultipleOrderQuantity_decimal"
select="number(.)"/>
<orderQuantityMultiple>
 <xsl:value-of select="number($Vvar160_MultipleOrderQuantity_decimal)"/>
</orderQuantityMultiple>
       </xsl:for-each>
       <xsl:for-each select="cbc:ReplenishmentOwnerDescription">
<xsl:variable name="Vvar166_ReplenishmentOwnerDescription_string"
select="string(.)"/>
<replenishmentOwner>
 <xsl:value-of select="$Vvar166_ReplenishmentOwnerDescription_string"/>
</replenishmentOwner>
```

```
       </xsl:for-each>
       <targetInventory>
<xsl:for-each select="cbc:TargetInventoryQuantity">
 <xsl:for-each select="@unitCode">
  <xsl:variable name="Vvar171_unitCode_token" select="string(.)"/>
  <xsl:attribute name="unitOfMeasure">
   <xsl:value-of select="string($Vvar171_unitCode_token)"/>
  </xsl:attribute>
 </xsl:for-each>
</xsl:for-each>
<xsl:for-each select="cbc:TargetInventoryQuantity">
 <xsl:variable name="Vvar177_TargetInventoryQuantity_decimal"
select="number(.)"/>
 <value>
  <xsl:value-of select="$Vvar177_TargetInventoryQuantity_decimal"/>
 </value>
</xsl:for-each>
       </targetInventory>
       <xsl:for-each select="cbc:TargetServicePercent">
<xsl:variable name="Vvar181_TargetServicePercent_decimal"
select="number(.)"/>
<targetServiceLevel>
 <xsl:value-of select="$Vvar181_TargetServicePercent_decimal"/>
</targetServiceLevel>
       </xsl:for-each>
       <collaborativeTradeItem>
<product>
 <xsl:for-each select="cac:GoodsItem">
  <xsl:for-each select="cac:Item">
   <xsl:for-each select="cac:StandardItemIdentification">
    <xsl:for-each select="cbc:ID">
     <xsl:variable name="Vvar189_ID_normalizedString"
select="string(.)"/>
     <gtin>
      <xsl:value-of select="string($Vvar189_ID_normalizedString)"/>
     </gtin>
    </xsl:for-each>
   </xsl:for-each>
  </xsl:for-each>
 </xsl:for-each>
</product>
<xsl:for-each select="cac:BuyerCustomerParty">
 <buyerLocation>
  <xsl:for-each select="cac:Party">
   <xsl:for-each select="cac:PartyIdentification">
    <xsl:for-each select="cbc:ID">
     <xsl:variable name="Vvar199_ID_normalizedString"
```

```
select="string(.)"/>
     <gln>
      <xsl:value-of select="string($Vvar199_ID_normalizedString)"/>
     </gln>
    </xsl:for-each>
   </xsl:for-each>
  </xsl:for-each>
 </buyerLocation>
</xsl:for-each>
<xsl:for-each select="cac:SellerSupplierParty">
 <sellerLocation>
  <xsl:for-each select="cac:Party">
   <xsl:for-each select="cac:PartyIdentification">
    <xsl:for-each select="cbc:ID">
     <xsl:variable name="Vvar209_ID_normalizedString"
select="string(.)"/>
     <gln>
      <xsl:value-of select="string($Vvar209_ID_normalizedString)"/>
     </gln>
    </xsl:for-each>
   </xsl:for-each>
  </xsl:for-each>
 </sellerLocation>
</xsl:for-each>
       </collaborativeTradeItem>
      </itemManagementProfile>
     </xsl:for-each>
    </plan:tradeItemLocationProfile>
   </documentCommandOperand>
  </n:documentCommand>
 </xsl:template>
</xsl:stylesheet>
```

VITA

PERSONAL INFORMATION

Surname, Name:	KABAK, YILDIRAY
Nationality:	Turkish (TC)
Date and Place of Birth:	20 March 1979, Adana
Marital Status:	Single
Phone:	+90 312 210 2076
e-Mail:	yildiray@srdc.com.tr

EDUCATION

Degree	Institution	Year of Graduation
MS	METU-Computer Engineering	2003
BS	METU-Computer Engineering	2001
High School	İsmail Safa Özler Anatolian High School, Ankara	1997

WORK EXPERIENCE

Year	Place	Enrollment
2008-present	SRDC, Reseach, Development and Consultancy Ltd.	Researcher
2001-2008	Software Research and Development Center, METU	Researcher
2000-2001	Software Research and Development Center, METU	Part Time Software Developer

FOREIGN LANGUAGES

Advanced English, Intermediate German

PUBLICATIONS

1. Kabak Y., Dogac A., Toroslu I.H. Semantic Interoperability of the Electronic Business

Document Standards submitted for publication to IEEE Transactions on Knowledge and Data Engineering (TKDE).

2. Kabak Y., Dogac A. A Survey and Analysis of Electronic Business Document Standards accepted for publication in ACM Computing Surveys. (Science Citation Index Core, Impact Factor: 11.286).

3. Kabak Y., Olduz M., Laleci G. B. Namli T., Bicer V., Radic N., Dogac A. A Semantic Web Service Based Middleware for the Tourism Industry Book Chapter, to appear.

4. Dogac A., Yildirim A., Kabak Y., Laleci G. B., Ocalan C., Bilen M. Design and Implementation of the eInvoice Interoperability Profile of the Revenue Administration of Turkey eChallanges Conference, October 2009, Istanbul, Turkey.

5. Kabak Y., Dogac A., Ocalan C., Cimen S., Laleci G. B. iSURF Semantic Interoperability Service Utility for Collaborative Planning, Forecasting and Replenishment eChallanges Conference, October 2009, Istanbul, Turkey.

6. Dogac A., Laleci G. B., Olduz M., Kabak Y., Okcan A., Tasyurt I. An Interoperability Service Utility for Collaborative Supply Chain Planning Presented in 14th International Conference on Concurrent Enterprising (ICE 2008), Lisbon, Portugal.

7. Kabak Y., Dogac A., Kose I., Akpinar N., Gurel M., Arslan Y., Ozer H., Yurt N., Ozcam A., Kirici S., Yuksel M., Sabur E. The Use of HL7 CDA in the National Health Information System (NHIS) of Turkey 9th International HL7 Interoperability Conference (IHIC) 2008, Crete, Greece, October, 2008 pp. 49-55.

8. Kose I., Akpinar N., Gurel M., Arslan Y., Ozer H., Yurt N., Kabak Y., Dogac A. Turkey's National Health Information System (NHIS) in the Proc. of the eChallanges Conference, Stockholm, October 2008, pp. 170-177.

9. Dogac A., Kabak Y., Namli T., Okcan A., Collaborative Business Process Support in eHealth: Integrating IHE Profiles through ebXML Business Process Specification Language IEEE Transactions on Information Technology in Biomedicine, Vol.12, No.6, November 2008, pp. 754-762. (Science Citation Index Core, Impact Factor: 1.787).

10. Dogac A., Namli T., Okcan A., Laleci G., Kabak Y., Eichelberg M. Key Issues of Technical Interoperability Solutions in eHealth and the RIDE Project eChallenges Conference, The Hague, The Netherlands, October 2007

11. Della Valle E., Cerizza D., Celino I., Dogac A., Laleci G., Kabak Y., Okcan A., Gulderen O., Namli T., Bicer V. , An eHealth Case Study Book Chapter in ”Semantic Web Services: Concepts, Technologies, and Applications”, Studer, Rudi; Grimm, Stephan; Abecker, Andreas (Eds.), 2007, Approx. 15 p., 100 illus., Hardcover, ISBN: 978-3-540-70893-3, Due: April 5, 2007, Springer.

12. Dogac A., Kabak Y., Laleci G., Najmi F., Mattocks C., Pollock J., Wallace E. ebXML Registry Profile for Web Ontology Language (OWL) OASIS ebXML Registry Technical Committee approved Committee Draft.

13. Laleci G., Dogac A., Akcay B., Olduz M., Yuksel M., Orhan U., Tasyurt I., Sen T., Kabak Y., Namli T., Gulderen O., Okcan A. SAPHIRE: A semantic Web service based Clinical guideline deployment infrastructure exploiting IHE XDS eChallenges Conference, Barcelona, Spain, October 2006. Published in: Exploiting the Knowledge Economy: Issues, Applications, Case Studies, Paul Cunningham and Miriam Cunningham (Eds), IOS Press, 2006 Amsterdam, ISBN: 1-58603-682-3.

14. Dogac A., Laleci G., Kirbas S., Kabak Y., Sinir S., Yildiz A., Gurcan Y. Artemis: Deploying Semantically Enriched Web Services in the Healthcare Domain Information Systems Journal (Elsevier), Volume 31, Issues 4-5, June-July 2006, pp.321-339 (Science Citation Index Core, Impact Factor: 1.827).

15. Dogac A., Laleci G., Kabak Y., Unal S., Beale T., Heard S., Elkin P., Najmi F., Mattocks C., Webber D., Kernberg M. Exploiting ebXML Registry Semantic Constructs for Handling Archetype Metadata in Healthcare Informatics International Journal of Metadata, Semantics and Ontologies, Volume 1, No. 1, 2006.

16. Della Valle E., Cerizza D., Bicer V., Kabak Y., Laleci G., Lausen H. The Need for Semantic Web Service in the eHealth W3C workshop on Frameworks for Semantics in Web Services, 2005.

17. Bicer V., Laleci G., Dogac A., Kabak Y. Providing Semantic Interoperability in the Healthcare Domain through Ontology Mapping eChallenges 2005, October 2005, Ljubljana, Slovenia.

18. Bicer V., Laleci G., Dogac A., Kabak Y. Artemis Message Exchange Framework: Semantic Interoperability of Exchanged Messages in the Healthcare Domain ACM Sig-

mod Record, Vol. 34, No. 3, September 2005 (Science Citation Index Expanded, Impact Factor: 1.759).

19. Dogac A., Kabak Y., Laleci G. C. Mattocks, F. Najmi, J. Pollock Enhancing ebXML Registries to Make them OWL Aware Distributed and Parallel Databases Journal, Springer-Verlag, Vol. 18, No. 1, July 2005, pp. 9-36. (Science Citation Index Expanded, Impact Factor: 1.785).

20. Dogac A., Kabak Y., Laleci G., Sinir S., Yildiz A., Tumer A. SATINE Project: Exploiting Web Services in the Travel Industry eChallenges 2004 (e-2004), 27 - 29 October 2004, Vienna, Austria.

21. Dogac A., Kabak Y., Laleci G., Sinir S., Yildiz A., Kirbas S., Gurcan Y. Semantically Enriched Web Services for Travel Industry ACM Sigmod Record, Vol. 33, No. 3, September 2004. (Science Citation Index Expanded, Impact Factor: 1.759).

22. Laleci G., Kabak Y., Dogac A., Cingil I., Kirbas S., Yildiz A., Sinir S., Ozdikis O., Ozturk O. A Platform for Agent Behavior Design and Multi Agent Orchestration Agent-Oriented Software Engineering (AOSE-2004) Workshop, the Third International Joint Conference on Autonomous Agents & Multi-Agent Systems (AAMAS 2004), New York City, New York - July 19, 2004 (Science Citation Index Expanded, Impact Factor: 01.515).

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

24. Dogac A., Laleci G., Kabak Y. Context Frameworks for Ambient Intelligence eChallenges 2003, October 2003, Bologna, Italy.

25. Dogac A., Kabak Y., Laleci G. A Semantic-Based Web Service Composition Facility for ebXML Registries 9th International Conference of Concurrent Enterprising, Espoo, Finland, June 2003.

26. Dogac A., Tambag Y., Pektas S., Laleci G., Kurt G., Toprak S., Kabak Y., ”An ebXML Infrastructure Implementation through UDDI Registries and RosettaNet PIPs”, ACM SIGMOD International Conference on Management of Data, Madison, Wisconsin, USA, June 2002.

27. Dogac A., Laleci G., Kabak Y., Cingil I. Exploiting Web Service Semantics: Taxonomies vs. Ontologies IEEE Data Engineering Bulletin, Vol. 25, No. 4, December 2002, http://www.research.microsoft.com/research/db/debull/issues-list.htm

28. Mitkas P., Symeonidis A., Kechagias D., Athanasiadis I., Laleci G., Kurt G., Kabak Y., Acar A., Dogac A. An Agent Framework for Dynamic Agent Retraining: Agent Academy e2002: European Commission's e-Business and e-Work Annual Conference, Czech Republic, October 2002.

29. Dogac A., Cingil I., Laleci G., Kabak Y. Improving the Functionality of UDDI Registries through Web Service Semantics 3rd VLDB Workshop on Technologies for E-Services (TES-02), Hong Kong, China, August 23-24, 2002.

30. Dogac A., Laleci G., Kurt G., Kabak Y., Acar A. A Platform for Semantically Enriched Mobile Services in Proc. of the First International Conference on Mobile Business, Athens, Greece, July 2002.

Number of Citations Received: 25 (Based on ISI Web of Science)