



Interoperability between Heterogeneous and Distributed Biodiversity Data Sources in Structured Data Networks

by

Rathinasabapathy Sundaravadivelu

A thesis submitted in partial fulfilment
of the requirements for the degree of
Doctor of Philosophy
in
Computer Science

School of Computer Science & Informatics
Cardiff University
August 2010

DECLARATION

This work has not previously been accepted in substance for any degree and is not concurrently submitted in candidature for any degree.

Signed.....(candidate) Date....**31-May-2010**.....

STATEMENT 1

This thesis is being submitted in partial fulfillment of the requirements for the degree of PhD.

Signed.....(candidate) Date..... **31-May-2010**.....

STATEMENT 2

This thesis is the result of my own independent work/investigation, except where otherwise stated. Other sources are acknowledged by explicit references.

Signed.....(candidate) Date..... **31-May-2010**.....

STATEMENT 3

I hereby give consent for my thesis, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Signed.....(candidate) Date..... **31-May-2010**.....

*To my loving wife,
Neela*

ACKNOWLEDGEMENTS

First and Foremost, I would like to express my sincere gratitude to my supervisors, Dr. Richard J. White, Dr. Andrew C. Jones, and Prof. W. Alex Gray for their kind support, expert guidance and encouragement through out my time in this research work. I am also grateful for their careful reading and constructive comments on this thesis and our joint papers. Without their guidance I could not have achieved this.

My immense thanks to the supervisors for giving me the funding for the study and an opportunity to work as research developer in the European Network of Biodiversity Information (ENBI) project to develop an Interoperable system. It is my privilege to gain both the research and development experience from my supervisors, in software engineering and biodiversity informatics.

I would like to thank the paper referees whose valuable comments on my published papers have added to the success of this project.

Special thanks are due to the late Prof. N. J. Fiddian and the members of the school for their help, especially Mrs. Margaret Evans who has helped me with travel related issues, Dr. P. A. Munn for her help in arranging appointments with my supervisors and Mrs. Helen Williams for her help in administrative issues.

I would like to thank Mr R. Evans and Dr. J. R. Davies for their technical assistance in setting up computer systems and email accounts that are vital for my research work. I would also like to thank Miss H.R. Phillips, Mrs P.K. Ryder and Mrs E.A. Slater for giving me jobs such as tutorial, lab supervision and marking of the papers.

I would also like to express my thanks to my friends and fellow research students in the School of Computer Science & Informatics at Cardiff University for their useful discussions on the research topics and help.

I am indebted to my wife for her endurance and unconditional support which provided vital encouragement during the period of my PhD study. Without her love and devotion, this research would have been impossible.

Abstract

The extensive capturing of biodiversity data and storing them in heterogeneous information systems that are accessible on the internet across the globe has created many interoperability problems. One is that the data providers are independent of others and they can run systems which were developed on different platforms at different times using different software products to respond to different needs of information. A second arises from the data modelling used to convert the real world data into a computerised data structure which is not conditioned by a universal standard. Most importantly the need for interoperation between these disparate data sources is to get accurate and useful information for further analysis and decision making.

The software representation of a universal or a single data definition structure for depicting a biodiversity entity is ideal. But this is not necessarily possible when integrating data from independently developed systems. The different perspectives of the real-world entity when being modelled by independent teams will result in the use of different terminologies, definition and representation of attributes and operations for the same real-world entity.

The research in this thesis is concerned with designing and developing an interoperable flexible framework that allows data integration between various distributed and heterogeneous biodiversity data sources that adopt XML standards for data communication. In particular the problems of scope and representational heterogeneity among the various XML data schemas are addressed.

To demonstrate this research a prototype system called BUFFIE (Biodiversity Users' Flexible Framework for Interoperability Experiments) was designed using a hybrid of Object-oriented and Functional design principles. This system accepts the query information from the user in a web form, and designs an XML query. This request query is enriched and is made more specific to data providers using the data provider information stored in a repository. These requests are sent to the different heterogeneous data resources across the internet using HTTP protocol. The responses received are in varied XML formats which are integrated using knowledge mapping rules defined in XSLT & XML. The XML mappings are derived from a biodiversity domain knowledgebase defined for schema mappings of different data exchange protocols. The integrated results are presented to users or client programs to do further analysis.

The main results of this thesis are: (1) A framework model that allows interoperation between the heterogeneous data source systems. (2) Enriched querying improves the accuracy of responses by finding the correct information existing among autonomous, distributed and heterogeneous data resources. (3) A methodology that provides a foundation for extensibility as any new network data standards in XML can be added to the existing protocols. The presented approach shows that (1) semi automated mapping and integration of datasets from the heterogeneous and autonomous data providers is feasible. (2) Query enriching and integrating the data allows the querying and harvesting of useful data from various data providers for helpful analysis.

Contents

Abstract	v
CHAPTER 1	1
Introduction	1
1.1 Introduction to the Research.....	1
1.2 Motivation for Interoperable Solutions	4
1.3 Statement of the Problem	5
1.4 Hypothesis and Aims	7
1.5 Objectives of the Research.....	7
1.6 Contribution of the Research.....	8
1.7 Organisation of the Thesis.....	9
CHAPTER 2	12
Background	12
2.1 Introduction	12
2.2 Interoperability	13
2.2.1 Technical Interoperability	13
2.2.2 Syntax and Structural Interoperability	13
2.2.3 Semantic Interoperability	15
2.2.4 Causes of Interoperability Issues	15
2.3 Approaches to Interoperability	16

2.4	Biodiversity Data Domain.....	20
2.5	Evolution of Data Communication Standards in Biodiversity	22
2.5.1	Dublin Core.....	23
2.5.2	Earlier Standards and Protocols of Biodiversity Data	24
2.5.3	Darwin Core V2.....	25
2.5.4	ABCD Standard	26
2.5.5	Taxonomic Concept Transfer Schema.....	27
2.5.6	TAPIR	27
2.5.7	SPICE Common Data Model.....	28
2.5.8	Ontologies in Biodiversity	28
2.6	Biodiversity Information Community Networks	29
2.6.1	Global Biodiversity Information Facility.....	30
2.6.2	European Network for Biodiversity Information.....	30
2.6.3	Taxonomic Database Working Group	31
2.6.4	LIFEWATCH	31
2.7	Summary of Background Study	32
CHAPTER 3.....		33
Relevant Technologies and Interoperability Projects in Biodiversity Data		33
3.1	XML Standards	34
3.1.1	XML Schema	35
3.2	API for XML Processing.....	36
3.3	XSLT	38
3.3.1	XPath.....	39
3.4	Web Services.....	39
3.5	Ontology and OWL tools	40
3.5.1	Protégé	42
3.6	Object Oriented Design in Programming.....	43
3.6.1	Microsoft .NET Framework.....	44
3.6.2	Java Framework	45
3.7	Functional Programming Model	46
3.7.1	Language Integrated Query (LINQ)	48
3.7.2	LINQ to XML	48
3.7.3	LINQ to Entities.....	49
3.8	Database Management Systems	49
3.9	Related Works of Interoperability in Biodiversity	50
3.9.1	BUFFIE – v1.0.....	54
3.9.2	GBIF - Infrastructure	54

3.9.3	Global Earth Observation System of Systems	55
3.9.4	Distributed Dynamic Diversity Databases for Life.....	56
3.9.5	Life Science Identifiers	56
3.10	Summary Analysis of Relevant Technologies and Projects	57
CHAPTER 4		60
The System Design and Framework Model		60
4.1	Introduction	61
4.2	An overview of our approach.....	63
4.2.1	Abstraction of Problem Domain	66
4.3	System Design of Prototype	67
4.3.1	Requirements of the Prototype System.....	69
4.4	Heterogeneity Issues in the BUFFIE System	70
4.5	Use-Case of Interoperability in the BUFFIE Application.....	72
CHAPTER 5		77
BUFFIE Architecture and Operation		77
5.1	Introduction	77
5.2	System Architecture for Interoperability in Biodiversity Networks	78
5.3	Generating the Queries for Heterogeneous Providers	83
5.4	Enriching the User Query.....	85
5.5	Architecture for User Query Enrichment	88
5.5.1	Example of Query Enrichment in BUFFIE.....	89
CHAPTER 6		92
The Query Response Retrieval and Transformation Process		92
6.1	Introduction	92
6.2	Response Data Integration Strategy from Heterogeneous Providers	93
6.3	Schema Matching Model	95
6.4	Biodiversity Data Transformation Architecture.....	97
6.5	Functional Approach for Schema Integration	98
6.5.1	XSLT Library for Schema Mapping.....	99
6.5.2	Transformation Functions in DKB	105
6.6	Example of Data Transformation in BUFFIE	106
CHAPTER 7		109
The BUFFIE Implementation		109
7.1	Introduction	109
7.2	Implementation Principles in BUFFIE System.....	112
7.3	Query Processing in the Business Logic Layer.....	112
7.3.1	Buffie Core.....	113

7.3.2	Buffie Services and Utils	114
7.4	The Data Access Layer of the Prototype	118
7.5	The Presentation Layer Prototype	120
7.6	BUFFIE System Tested with Data Providers	121
7.7	BUFFIE System's Interoperation with Linnaeus II	126
CHAPTER 8		131
Evaluation & Discussion		131
8.1	Introduction	131
8.2	Evaluation	132
8.2.1	Functionality of the BUFFIE Framework	133
8.2.2	Extensibility of the Framework Model	135
8.2.3	Architecture of BUFFIE Framework	136
8.2.4	Domain Knowledge Base (DKB)	137
8.2.5	Applications of the BUFFIE Common Access System	138
8.2.6	Implementation and Verification of the BUFFIE System	139
8.3	Discussion	140
8.3.1	Verifications of Goals Achieved	142
8.4	Applicability and Limitations	143
CHAPTER 9		145
Summary, Conclusion and Future Work		145
9.1	Thesis Summary	145
9.1.1	Publications from Thesis	147
9.2	Conclusions	148
9.3	Future work	149
Bibliography		151
Appendix A		167
Mapping between Darwin Core (DWCV2) and ABCD (BioCASE) Concepts...		167
Appendix B		174
C # code for BUFFIE Framework's Core Components and Services		174
Appendix C		196
SQL code for BUFFIE Database and Entity Data Model Services		196
Appendix D		226
XSLT Templates from Domain Knowledge Base		226

List of Figures

Figure 1.1: Spectrum of Biodiversity Databases and target population of this research.	6
Figure 3.1: Structure of an XML document.....	35
Figure 3.2: XML Schema Definition.	36
Figure 3.3: Data Communication in Web services	39
Figure 3.4: Microsoft .NET framework Architecture.	45
Figure 3.5: Java Framework Architecture.....	46
Figure 4.1: DiGIR Architecture.	62
Figure 4.2: Schematic representation of Interoperability in BUFFIE.....	64
Figure 4.3: Conceptual Design of the BUFFIE Common Access System.	68
Figure 4.4: Differing Scope of Biodiversity Data in Communication Standards.	71
Figure 4.5: Differing Views of representation of Data.	72
Figure 4.6: Request message structure for Darwin Core standard Provider.....	73
Figure 4.7: Request message for ABCD standard Provider.	74

Figure 4.8: Response for Species named “ Acicula lineata ” from Darwin Core Provider.....	75
Figure 4.9: Response for “ Acicula lineata ” from ABCD Provider ZOBODAT.	76
Figure 5.1: Multi Layered, Web based Service oriented architecture.	79
Figure 5.2: Architecture of XML data mapping process.	82
Figure 5.3: Conceptual view of Query Generation in BUFFIE.	84
Figure 5.4: Sample XML request message for data provider.	85
Figure 5.5: Enrich a concept by the generalization of its values.	87
Figure 5.6: Architecture for Query Enrichment.....	88
Figure 5.7: AJAX web page with species scientific name and data.	89
Figure 5.8: XML result from the synonym web service for species name.	90
Figure 5.9: results of synonym web service call in the application.	91
Figure 6.1: Conceptual view of heterogeneous response integration in BUFFIE.	94
Figure 6.2: Schema Mapping Assertion Model.	96
Figure 6.3: Biodiversity Data Transformation using Schema matching.....	97
Figure 6.4: Sample Response from a Darwin Core Provider.....	103
Figure 6.5: Example of a source response message from Darwin core provider.....	107
Figure 6.6: Example of the transformed xml message in ABCD format.....	108
Figure 7.1: Layered Implementation of BUFFIE Architecture.	111
Figure 7.2: BuffieCore classes from framework Business domain.	113
Figure 7.3: BuffieServices classes.	114
Figure 7.4: Buffie Utils Classes.	115
Figure 7.5: Sample Request XML schema created by BuffieServices.	116
Figure 7.6: DomainKnowledgeBase Implemented as XSLT files in config folders.	118
Figure 7.7; Entity Data Model for BuffieDatabase.....	119
Figure 7.8: Query Design Page.	120

Figure 7.9: Query Results Page.	121
Figure 7.10: Common Name: “Fruit-Fig”	122
Figure 7.11: UserQuery stored in the Buffie database.....	123
Figure 7.12: Heterogeneous data-providers information.	123
Figure 7.13: XML Request and Response messages in Buffie system.	124
Figure 7.14: AustrianZobo data provider. (returns 3 records in a BioCASE data format).....	124
Figure 7.15: New York Botanical Garden from USA, Herbarium data provider.....	125
Figure 7.16: RealJardin Botanico data provider from Spain.	125
Figure 7.17: Merged results stored in Buffie system.	126
Figure 7.18: BUFFIE used by Linnaeus II to connect to providers databases.....	127
Figure 7.19: BUFFIE demonstration with species data.	128
Figure 7.20: BUFFIE demonstration with Linnaeus II.	129
Figure 7.21: Heterogeneous data merged using BUFFIE system used by client application.....	130

List of Tables

Table 5.1: Synonym service Providers information	90
Table 6.1: Sample atomic-level match.....	100
Table 6.2: Full and Partial structural match.....	101
Table 6.3: Transformation Match Cardinalities.....	102
Table 6.4: Full and Partial structural match.....	104

Acronyms

ABCD	Access for Biodiversity Collection Data
API	Application Programming Interface
ASP	Active Server Pages
BUFFIE	Biodiversity Users' Flexible Framework for Interoperability Experiments.
CDM	Common Data Model
CODATA	Committee on Data for Science and Technology
COM	Component Object Model
DBMS	Database Management System
DOM	Document Object Model
DwC	Darwin Core
DTD	Document Type Definition
EDM	Entity Data Model
ENBI	European Network for Biodiversity Information
FDBS	Federated Database System

GBIF	Global Biodiversity Information Facility
GUI	Graphical User Interface
HTML	Hyper Text Markup Language
JDBC	Java Database Connectivity
JDOM	Java Document Object Model
JSP	Java Server Pages
JXC	Java XML Connectivity
KB	Domain Knowledgebase
LITCHI	Logic-based Integration of Taxonomic Conflicts in Heterogeneous Information Systems
LINQ	Language-Integrated Query
MDBS	Multi-database System
OLE	Object Linking and Embedding
SAX	Simple API for XML
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SGML	Standard Generalized Markup Language
SQL	Structured Query Language
StAX	Streaming API for XML
TAPIR	TDWG Access Protocol for Information Retrieval
TDWG	Taxonomic Databases Working Group
UDDI	Universal Description, Discovery and Integration
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WSDL	Web Service Description Language
XML	eXtensible Markup Language
XSD	XML Schema Definition
XSLT	eXtensible Stylesheet Language Transformation

CHAPTER 1

Introduction

1.1 Introduction to the Research

The research described in this thesis is concerned with achieving interoperability of distributed and heterogeneous biodiversity databases by creating a novel and flexible framework that uses the synergies of the web-based service-oriented architecture, extensible processing logic and knowledge of the data domain stored in XML and an XSLT repository. This approach helps in preserving the local autonomy of the data providers and still enables the users to have interoperable common access to the data from varied networks of data resources. Interoperability issues of heterogeneous and distributed databases are highly challenging, as they have to be resolved to the level of the initial requirements of interoperability, restrictions of technology and the dynamic

nature of the biodiversity data involved in this research. Real world biodiversity data are being increasingly digitised and stored in digital formats [1] and applying computer science technologies on these data would reveal useful information from the data. This includes efficient organising of the data in a structured format, querying these data simultaneously from heterogeneous and distributed sources and combined analysing to derive useful knowledge and present the information from data for decision making.

Expressing real world data using computer data types is challenging and the majority of the data that are digitized are stored in relational databases. If more data are made available for analysis by the system then this would improve the statistical significance of the information derived from them, which can be more reliable and useful. Interoperation is required to access and integrate the data from multiple resources but when autonomous data resources are delivering their data in different formats, it only compounds the problem.

Standards have been introduced to represent the data that are provided on the communication network of biodiversity data providers [2], so that it could be understood by another system that is aware of the standard. The eXtensible Markup Language (XML) is often used to describe the data; though not the best data structure for every possible data domain, it proves to be the most generally adequate one for the text based data communication over computer networks e.g. internet applications. The main advantages of using XML are in providing the metadata of the data in the structure used to contain the data and the universal standardisation of XML by the World Wide Web Consortium [3]. The standards defined using XML schemas allow the data sources and consumers to communicate with each other in the data provider networks thereby resolving the interoperability issue considerably. Many research projects such as Species 2000, MaNIS, BioCASE, etc. have made the first step in interoperability process by providing common access to a set of data providers by adapting to one of the many XML Schemas to represent different kinds of data for communication [4], [5], [6].

The issues of interoperating between data-provider networks that follow different data communication standards are at another level which are researched in this thesis. This research aims at achieving both structural and semantic interoperability between these different XML interchange formats (schemas) using a new framework model formed by combining suitable service oriented architecture, extensible processing logic and knowledge of the data domain captured in ontology.

A service-oriented architecture is essentially a collection of services whose goal is to achieve loose coupling among interacting software agents. The communication can either involve data exchange or to coordinate some activity [7]. Extensible processing design means the system should be able to adapt to acceptable and predictable future changes in data interoperation such as extension to communication protocol standards with relative ease.

An Ontology is an explicit specification of a conceptualization and it can be understood as an intentional semantic structure which encodes the implicit rules constraining the structure of a piece of reality [8]. A Knowledge base is a repository of related information about a particular domain and can be a machine-readable resource for the dissemination of information. The knowledge base and ontology are built for a specific purpose and represent specific knowledge of a problem domain about concepts and their interrelations [9]. Both Ontologies and Knowledge bases are used to create some kind of integration schema by deriving the domain knowledge and this process is called knowledge fusion [10].

Our approach assumes the existence of appropriate domain knowledge for biodiversity data concepts, but to demonstrate the framework model we will use a purpose built prototype domain knowledge base. This thesis explains the interoperability issues of the XML schemas used in the biodiversity domain and shows how they can be addressed by applying the proposed framework which is implemented using the extensibility of object oriented languages (Java, .NET) and eXtensible Stylesheet Language Transformation (XSLT).

Biodiversity means the diversity or variety of plants and animals and other living things in a particular area or region [11], [12]. Heterogeneity in biodiversity data is not only a result of non-standardized data capture but also due to the wide variety of data sets in biodiversity and new sources of information such as genetic sequences in bioinformatics studies. The representation of biodiversity data is evolving and the species to which the data refer are named using the principles of taxonomy. Taxonomy is based on a classification procedure used for hierarchically describing the organisms into groups on the basis of perceived shared characteristics, reflecting postulated evolutionary relationships between these groups [13]. Taxonomic classifications represent an evolving hypothesis rather than static descriptions of organisms and can reflect the views of the person assessing the information at a given time. Hence taxonomic identification and the unambiguous labelling of these groups is becoming a significant problem for the integration and comparison of the diverse datasets for analysis across all fields of biology [14]. When these taxonomic values are expressed in XML schemas for communication across the networks, different XML standards have evolved that are followed by groups of data providers forming biodiversity networks. Please refer to chapter 3 for a detailed discussion of biodiversity data and XML standards related to it. This research focuses upon the interoperability issues that exist between these biodiversity networks and provides a framework model that can be extensible to accommodate the changes that may happen in the near future.

1.2 Motivation for Interoperable Solutions

The differences in data capturing, storing, software execution platforms and interfaces used by autonomous and distributed data sources have created heterogeneity in data communication. Interoperating between these systems is essential to generate information from these data. The concept of intelligent integration of data rests on the premise that a suitable framework model with knowledge of the data domain is needed to integrate the factual observed data into useful information [15]. Biological data is complex but computer science can provide a solution to analyse these data that can be useful to scientists, environmentalists, natural resource managers and policy-makers of government and other organisations and academic researchers. The need

for interoperability and common access to biodiversity information that is stored on a large number of biodiversity databases distributed around the globe is highlighted now that more emphasis is given to protect the environment of the planet [16], [17].

All living organisms are interdependent for their existence, and form relationships and ecosystems which constitute the web of life on the planet Earth. A country's prosperity is directly related to its natural resources and moreover for mankind to exist into the future it is very vital to understand and conserve the wide diversity of all organisms. In the context of the climate change problem caused by human activities on the planet the need to monitor the factors affecting biodiversity loss in order to mitigate them becomes important. Common access to biodiversity data held in distributed heterogeneous databases is thus important to researchers, academics, industries, and conservationists. The objective of interoperation here is to access data from different data resources and increase the value of information accessible, in terms of quality and quantity while the common access should provide a secure access by authenticating the users who are accessing the information. Bringing together the large volume of biodiversity data available in heterogeneous and distributed databases is impossible without appropriate supporting technology. This research focuses on achieving the interoperability of the XML data structures in a novel way by designing an extensible framework architecture.

1.3 Statement of the Problem

Interoperability is the ability of two or more systems or software components to exchange information and to use the information that has been exchanged. Interoperability in general is concerned with the capability of differing information systems to communicate [18]. Several different levels of interoperability are to be addressed among the group of biodiversity data resources. Broadly they can be classified as technical interoperability, syntactic & structural interoperability of data and semantic interoperability of data which are explained further in section 2.2. The technical interoperability is concerned with the hardware and software platforms of computers for communication e.g. internet. The structural and semantic issues concerning biodiversity data are very complex due to the nature of the data. This data is digitally represented in different forms that allow communication among

computers. XML standards are used predominantly by organisations such as the Global Biodiversity Information Facility (GBIF) which indexes a huge amount of data, currently more than 200 million collection records from hundreds of databases. This is about 20% [19; 20] of the digitised data existing in biodiversity data resources. The spectrum of data providers that form the target population of this research is illustrated in the Figure 1.1.

The available interoperable systems in biodiversity only allow interoperation within a particular network of data resources and also this would limit extensibility such as the ability to include new types of data as they become available. Chapters 2 and 3 include a survey of relevant biodiversity information systems in interoperability. Developing a universal and continually updated schema that can accommodate the new and evolving data structure schemas in biodiversity is one way of approaching the problem, but this would heavily influence the autonomy of the data providers who will have to continuously update their systems for these changes. This limitation is researched in this work and an extensible solution is proposed.

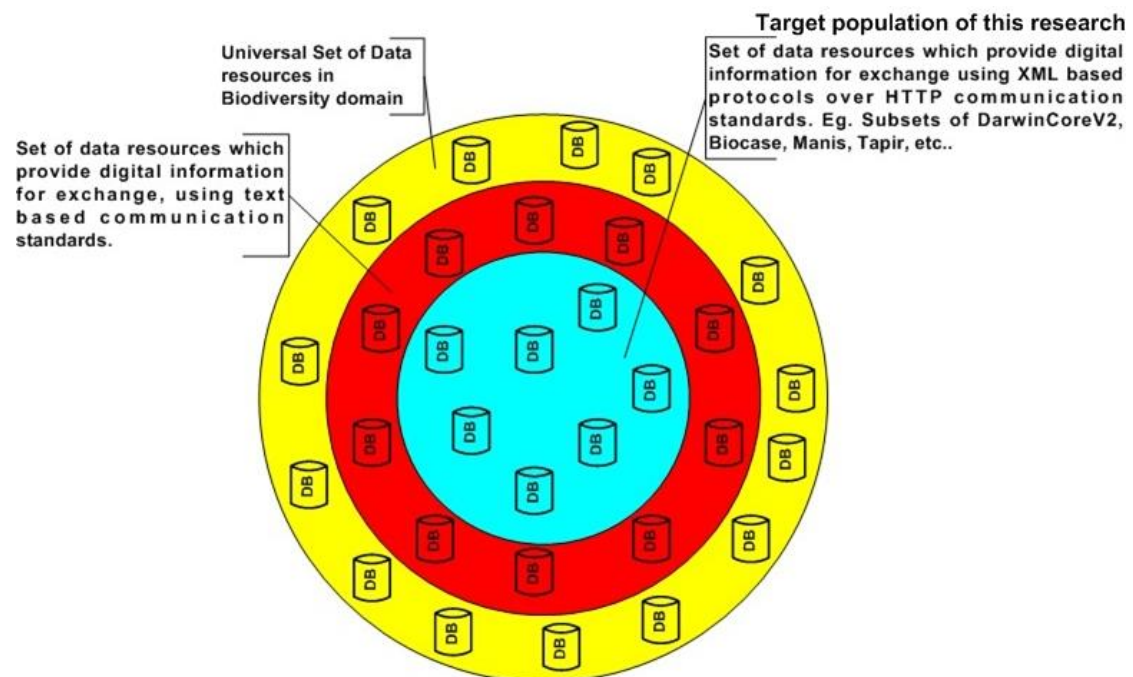


Figure 1.1: Spectrum of Biodiversity Databases and target population of this research.

1.4 Hypothesis and Aims

This research focuses on achieving interoperability in heterogeneous and distributed biodiversity databases which are already able to communicate using XML standards for data over HTTP connections. The hypothesis is:-

“Interoperability among distributed, autonomous and heterogeneous biodiversity databases can be achieved by developing a new framework that exploits the synergies of combining multi-layered service oriented system architecture, domain knowledge expressed in a knowledgebase designed using XML and XSLT, and object-oriented functional design of components.”

In particular, our approach uses the query enrichment process and heterogeneous responses integration using the information from a knowledgebase about the domain, to enhance the visibility and interoperability of the available information in heterogeneous biodiversity databases. This thesis presents a new framework that can provide structural/syntactical and semantic interoperability among biodiversity networks. Please refer to chapters 4 through to 9 which demonstrate this hypothesis.

1.5 Objectives of the Research

The objectives of this research are to design and develop a suitable framework model for achieving structural and semantic interoperability between heterogeneous and distributed data providers and demonstrate it using a prototype system. This will be designed after analysing the available approaches towards interoperability in the biodiversity domain. A new system design and architecture is devised, which would help in achieving the following objectives.

1. To design, develop and implement a suitable framework (BUFFIE - Biodiversity Users' Flexible Framework For Interoperability Experiments)
2. To design the components and integrate the services required for BUFFIE to perform the interoperation process, e.g. building the query enriching modules and data schema integration using XSLT templates that carry out the data transformation and knowledgebase of taxonomical concepts used for data exchange expressed in XML format.

3. Web based prototype application development using Java and .NET technologies to verify the claim made in the hypothesis (section 1.4) using the test datasets.

These objectives explained here are later revisited in the evaluation and discussion of chapter 8 to demonstrate that they have been accomplished. Please refer to chapter 8 for the scope and limitations, while achieving the above stated objectives.

1.6 Contribution of the Research

The importance of this research lies in presenting a suitable framework and demonstrating that the system achieves interoperability of text based, mainly XML structured data delivered by distributed heterogeneous data providers. This would verify the claim made in the hypothesis. Unlike the current approaches this research provides a flexible and pragmatic approach to achieve the interoperability between the existing structured data networks in the biodiversity domain by applying the software engineering and functional programming techniques in the distributed query process. In our approach the extensibility is built into the middleware system that does the interoperation without affecting the existing data providers and also allowing new data providers to join into this network seamlessly. The primary contributions of our work include:

- Presenting a suitable framework (BUFFIE v2.0) for Biodiversity Users that allows interoperability through a common access system for data querying from heterogeneous data resources using XML based communication protocols.
- We introduce a query enriching process aiming to maximize the success in finding information for a query, using synonym web services. Also the responses from the data providers are integrated using both structural and semantic matching of the data before presenting to the user.

- A combination of both object-oriented and functional approaches in design of the architecture used for the BUFFIE core and the domain knowledge base respectively. This allows the extensibility in the framework to add or remove a new data schema in the system or to use the core system with a different data domain by plugging in a new domain knowledgebase.
- A web based software application (prototype) system is developed to demonstrate the ideas explored in the thesis. A production version of the application (BUFFIE v1.0) has been hosted, live on the “Veenai” server at Cardiff, which was used by the client programs like Linnaeus II from ETI, Amsterdam, Holland and BioGis-Israel information System [21] from Jerusalem for accessing and harvesting species based information from participating heterogeneous and distributed data providers.

The interoperability issues in the biodiversity data domain spread over a vast expanse. In our research, we would like to define the scope and the boundaries to the level of interoperability that we aim to achieve. For example only those data providers who were communicating using XML based standards were considered in the design of the framework. Another example is that the level of semantic interoperability is semi-automatic and is proportional to the knowledge rules defined by the XSLT templates of the knowledge base. A limitation of this approach is that the developer of the knowledge base modules needs to be aware of the relevant biodiversity data concepts and will have to continuously update their systems to accommodate new and evolving data structure schemas as biodiversity studies progress. This process of developing the domain knowledge and the limitations and the possible extensions for future work are explained in sections 8.4 and 9.3.

1.7 Organisation of the Thesis

This section shows an overview of the thesis organisation. The first chapter has presented an introduction to the research undertaken, motivation for the research, the hypothesis to be tested and highlights the aims and objectives of the research and its original contributions.

Chapter 2: *Background*

This chapter explains about interoperability and its various types that relates to this research and states the general causes of interoperability from various viewpoints. It then moves on to describe the different approaches to resolve interoperability. An overview of the biodiversity data domain and various data communication standards used to achieve interoperability are discussed.

Chapter 3: *Relevant Technologies and Interoperability Projects in Biodiversity Data.*

This chapter presents an overview of related software engineering technologies and XML communication standards used for common access systems and the existing levels of interoperability in biodiversity data communication. It gives an overview of the research projects in heterogeneous and distributed biodiversity data sources, and analyses of their relation to this research project.

Chapter 4: *The System Design and Framework Model*

This chapter introduces the conversion of ideas in the thesis to a prototype system; it describes the approach and the overview of the design of the prototype system and justifies why it is more relevant than other possible approaches. An example of the issues of data heterogeneity and an interoperability test from the BUFFIE system are described.

Chapter 5: *BUFFIE Architecture and Operation*

This chapter details the multi-layered, Web-based service-oriented architecture of the BUFFIE System along with its components including *BuffieCore*, *BuffieServices*, *BuffieUtils* and *DomainKnowledgeBase* and their responsibilities. The conceptual and logical architecture of the subsystems, such as query enrichment and query generation are described. The processing logic (algorithm) for query enrichment and heterogeneous multiple query generation are explained by running through an example.

Chapter 6: *The Query Retrieval and Translation Process*

In this chapter, we detail the query response retrieval and response data transformation process in the BUFFIE framework. We introduce the query retrieval process by asynchronous multithreading over the HTTP protocol and then the conceptual view of heterogeneous response integration in BUFFIE and the logic used for integration. The response integration strategy and the Schema matching model are described. We then present a functional approach for schema integration using XSLT technology and this process will be shown using an example of data transformation in the BUFFIE system.

Chapter 7: *The BUFFIE Implementation*

This chapter covers the implementation of the BUFFIE system. We will present a brief description of BUFFIE v1.0 that was implemented and tested and currently used in the biodiversity domain. More emphasis is given to the current version of BUFFIE v2.0 that is implemented on Microsoft .NET 3.5 framework using the Visual Studio 2008/2010 integrated development tool. Different details on the implementation of the components in the three layers namely, the middleware business logic layer, data access layer and the presentation layer will be shown.

Chapter 8: *Evaluation & Discussion*

This chapter focuses on the evaluation of the two versions of the BUFFIE prototype system and assesses the functionality and flexibility of the framework's architecture. Here, we discuss the verifications of the contributions achieved by the BUFFIE system deployed on live servers. We will state the various application areas of the system and also its limitations.

Chapter 9: *Summary, Conclusion and Future work*

This chapter concludes the thesis with the summary of our accomplishments in this research and the related issues that can be considered in the future work.

CHAPTER 2

Background

2.1 Introduction

In this chapter the background of the concepts like different interoperability classifications, approaches and architectures that are related to heterogeneous and distributed databases in general are discussed. The various causes of interoperability issues from different viewpoint such as software communication and control mechanisms, data modelling in computer systems and the type of processing components used are presented. Then the chapter focuses on elucidating the most relevant and related researches that were carried out in the biodiversity informatics domain. The different approaches for solving the general interoperability issues of distributed and heterogeneous data resources were analysed with respect to the heterogeneity of data. Then all the related projects in the biodiversity domain were reviewed with specific emphasis to the first outcome of this research (BUFFIE v1.0) and how this work will harmonise into the research of other related works.

2.2 Interoperability

Interoperability is one of the most critical and much researched issues of any information domain, as there is often the need to use information stored on autonomously managed multiple heterogeneous systems. Interoperability is the ability of two or more systems or components to exchange information and to use the information that has been exchanged. Interoperability in general is concerned with the capability of differing information systems to communicate [22]. This communication may take various forms such as the transfer, exchange, transformation, mediation, migration or integration of information. From an implementation point of view interoperability is the ability of two or more software components to cooperate despite differences in programming language, data exchange interface, data model representation and execution platform. A user from a system should be able to access any data in a distributed database without having to know where or how the data object is physically stored [23]. These explanations of interoperability are more relevant to the context of the issues researched in this thesis, such as interoperability through a common access system that provides integrated information from distributed heterogeneous data resources that follow different data exchange standards. Several different levels of interoperability are to be addressed to achieve a working system. Broadly they can be classified as follows.

2.2.1 Technical Interoperability

This is concerned with integrating different computer networks operating on different platforms. An example is the Internet where many disparate networks communicate meaningfully using the TCP/IP protocols. This kind of interoperability can be achieved largely by selecting the appropriate hardware and software systems for the proposed application.

2.2.2 Syntax and Structural Interoperability

The data is represented in different forms or models across different systems. In the biodiversity domain, different schemas are used to represent data and for information

exchange between the data provider networks. These differences in schema/ metadata are characterized in structural differences, leading to structural interoperability issues. Examples are naming conflicts, entity-identifier conflicts, schema-isomorphism conflicts, generalization conflicts, aggregation conflicts and schematic inconsistencies [24].

In biodiversity data, naming conflicts occur when scientific names for a same species are assigned by different biologists not known to each other from different parts of the world, and also they might disagree about the taxonomy of a species. Entity-identifier conflicts are often caused by assigning different identifiers to the same concept in different data models. Schema-isomorphism conflicts occur when the same biological concept is described by different attributes. Generalisation conflicts result from different design choices for modelling related entity classes. For example a data model such as Access to Biological Collection Data (ABCD) [25] can have separate representations for Bacterial/Genus, Botanical/Genus, Viral/Genus and Zoological/Genus whereas another data model such as Darwin Core 2 [26] may have one “Genus” entity to collectively represent the different but related entities. Aggregation conflicts arise when an aggregation is used in one data model to identify a collection of entities in another data model. For example the entity with element name as “GatheringDateTime/ISODateTimeBegin” in ABCD is formed by concatenating Year, Month and Day collected entities of the Darwin Core model.

Schematic inconsistencies occur when the logical structure of elements in one data model are organized to form a different structure in another data model. This interoperability issue is at the application level that can be solved in some systems by enforcing data standards or by writing wrapper programs, which convert the data format into a format understandable by the system [27]. Schema mappings for disparate data models may result in achieving this interoperability. E.g. Microsoft BizTalk and the Altova XMLSpy suite are commercial tools used to create schema matching and data mappings using XML transformation between disparate systems [28], [29]. BUFFIE resolves structural interoperability among different communication protocols, as discussed in chapters 4 to 8.

2.2.3 Semantic Interoperability

This is one of the most daunting issues in interoperability. Semantic interoperability is the knowledge-level ability of information systems to exchange information on the basis of shared, pre-established and negotiated meanings of terms and expressions [18]. Even though the data are available from different systems where each system uses a standardized data model, these data can be more useful and can be integrated only when the integrating system is aware of (has knowledge to process) the information contained in the participating data models. BUFFIE v1.0 demonstrates the possibility of semantic interoperation as a proof of concept by using built-in ontology-like concepts using programming logic. More scalable domain specific knowledgebase are used in the current version of BUFFIE which is version 2.0. Another viewpoint of interoperability issues in biodiversity domain are the data interoperability and the systems interoperability.

2.2.4 Causes of Interoperability Issues

Interoperability issues are identified when complex software systems are integrated to access heterogeneous and distributed data using disparate components. Application Software systems have been developed by autonomous communities or individuals who use their own semantics to achieve their specified requirements in familiar and closed environments. Similarly the data collection and the definition of the collected data in a particular domain were carried out by disconnected set of individuals. Though more organisation and communication are being introduced in every domain to universally standardize the data collection process, differences in the data structure and semantics prevail. Applications cannot dictate the structure of data or the semantics of the data held in autonomous data resources. The main causes of interoperability can be traced down to the fundamental characteristics of the interacting systems design, architecture and the data structure and semantics. The main causes for interoperability issues from a software engineering analytical point of view include the following.

Control Mechanisms:- The control interaction between the different components of the system lead to interoperability problems [30] and this can be influenced by the coupling between the components of the system. This is mainly the communication methods between the components of a system and between independent but coordinating systems and hence relates mostly to the technical interoperability issues described in the previous section 2.2.1.

Data Topology: - This is about the data model that represents the concepts or entities of the data and also it defines the structure for the data that are used internally within the components of the system and for external communication between different systems. The interoperability problems caused by this are most prevalent in every data domain due to the dissimilar data representation formats provided by each data provider [31]. This relates to the structural or semantic interoperability types.

Process Synchronisation: - both synchronous and asynchronous style of communication can affect the data and control of the components which could create technical, structural and semantic interoperability problems [32].

In the biodiversity domain the research groups that collect data use different vocabularies, assumptions, methodologies and goals, and work under varying geographical locations and time periods. These factors result in multiple representation formats for the same real world data. The interoperability problems caused by the heterogeneity in biodiversity data representation and computer science technologies are discussed in the chapters 3 and 4.

2.3 Approaches to Interoperability

Achieving interoperability is a complex task comprising a balanced mixture of communication, cooperation and competition among the communities and the software systems in a particular data domain. Community networks were formed that includes the experts of a particular domain to share ideas, research issues and develop interoperable software systems and data communication standards that can allow the data interoperability. Some examples of such networks in biodiversity domain are ENBI (European Network for Biodiversity Information). NBN (The National

Biodiversity Network) and GBIF (Global Biodiversity Information Facility). More details on these networks and biodiversity projects are discussed in the next section 2.4. This section describes about the common approaches and the related technologies used to resolve the interoperability issue.

Federated Database system – is an integrated collection of completely functional and independent databases controlled by local administrators but cooperating with the federation by supporting global operations [33]. This federation can be either tightly coupled or loosely coupled. A tightly coupled federated system presents a predefined static view to the end-user. This is usually based on a global schema that accommodates the entire component schema and maintained by system administrator who makes all schematic and semantic integration decisions in advance. In a loosely coupled system the integration is dynamic. The user is responsible for the integration of data or the system has to provide a mechanism for performing the integration of data.

Client server architecture - provides the ability of two or more components to cooperate despite differences in interface, execution language and platform. Client server applications achieve systems interoperability, using interface standardisation by mapping client and server interfaces to a common representation and interface bridging which uses two-way maps between client and server [18]. The Common Object Request Broker Architecture [34], OMG's open, vendor-independent architecture and Microsoft's Component Object Model COM/OLE [35] realize interoperability using interface standardisation. The client server architecture restricts the autonomy and heterogeneity of distributed data sources as they all have to conform to either a client or a server component which also imposes a maintenance problem once when the system is scaled up.

Mediator systems - provide a remedy to client/server architecture as they recognize the autonomy and diversity of the data systems [36], [37]. A Mediator acts as an interchange component which translates data between two systems with different data schemas to information by applying knowledge about resources, semantic information of data and user requirements. The mediator handles an information exchange by

converting the user query into a source compatible query and executes the query. This result is converted back into user recognizable format. In short it acts as a semantic gateway between the systems allowing the user to view all the sources without concern for the differences in names and representations of data.

Multiple View Definition System (MVDS) - focuses on the architecture of software to achieve interoperability in heterogeneous multidatabases [38]. Providing a tool (typically automated) for user to define the integration views to infer information from multidatabases is a way of supporting interoperation among heterogeneous and autonomous databases. [39], [40] Ontologies with all participating schema components are not a complete solution as they will not provide complete information to the users to make a query to the heterogeneous databases. A canonical data model and an architecture using knowledge base as a mediator that stores the static and dynamic knowledge about the participating databases has proved to be one answer to the issue of interoperation. A variety of other approaches in developing mediator systems involve the use of:

- **Wrappers** – Wrapping is a method of permitting existing legacy software systems to communicate with the current systems. A wrapper program can be described in two parts, an adapter that provides extra functionality to an application and an encapsulation mechanism that binds the adapter to the application [41]. It provides the communication interface between application programs by converting the data as required. The interoperation ability depends on the levels of abstraction in design, extensibility and maintainability of the wrappers.
- **Data Warehouses** [42] – A data warehouse is a centralized repository of information extracted from multiple data sources. It can serve as an index or as a cleaned data gathered from different heterogeneous systems. The disadvantage of this approach is the difficulty of updating the data and to keep them in synchronization with the local databases as the participating database numbers are growing.

- **Metadata Repository systems** - In this system the queries are formulated dynamically with the use of an on-line global metadata dictionary [43]. The metadata information can be stored using schema maps, data type with description logics and ontologies to solve queries over multiple web-based information sources.
- **Shared Ontologies** - A common ontology approach is used to resolve the semantic heterogeneity in a particular domain by using the knowledge ontologies [44], [45], which contain deep domain knowledge and form a conceptual standard.

The different approaches described in this section are the main technologies and tools that are used by the software information systems to create an integrated querying infrastructure to access multiple, distributed and heterogeneous data resources. Each approach has made progress in achieving interoperability but still possesses some limitations. For example the limitations of Federated systems require a common data model that has to be understood by all the participating databases, or if the data model is varied then another layer of mediation between the data structures is required to achieve interoperability. Client-server architecture requires the bulk of the processing to be performed at the server side and also the clients are to be continuously maintained for any new changes on the server side. With the advent of web-based data communication, client-server architecture is less preferred in designing distributed systems due to the requirement of centralized maintenance of the system. Tools such as wrappers, metadata repository and ontologies are used to either convert or translate the data formats. The choice and the ability of these tools to interpret the data format affect the design and implementation of the multiple querying systems. Our research evolves from analysing these technologies and tools with consideration of the nature of the biodiversity data domain and real data sets of biodiversity data providers.

The technical details of the interoperability approach adopted are discussed further in section 4.2. With reference to the types of interoperability described in the section 2.2 of this chapter, this research deals with the structural and semantic interoperability

issues prevailing among the established data providers communities in the biodiversity data domain. To overcome the technical interoperability the system design and architecture of the framework use standard internet communication protocols. Syntactic and structural interoperability are addressed by the use of XML transformations. Most of the semantics of the biodiversity data concepts are captured using a knowledgebase that is part of the architecture, as an alternative to capturing the semantics of data using a data model technique such as RDF. The limitation of this approach is that the developer of the knowledgebase needs to be aware of biodiversity data concepts, which are discussed in the remainder of this chapter, and to maintain the knowledgebase as the relevant standards evolve.

2.4 Biodiversity Data Domain

Biodiversity data refers to the different life forms such as different plants, animals and micro organisms, the genes they contain and the ecosystem they form. These data reflect hundreds of millions of years of evolutionary history and hence their volume is huge and their nature is dynamic. Biodiversity is generally considered at three different levels namely genetic diversity, species diversity and ecosystem diversity [46]. Genetic diversity refers to the variation of the genes within species. Species diversity refers to the variety of species. Ecosystem diversity refers to variety of habitats, biotic communities, and ecological processes, as well as the tremendous diversity present within ecosystems in terms of habitat differences and the variety of ecological processes. By the 1750s Carl Linnaeus, a Swedish naturalist devised a structure to represent living organisms known as Linnaean taxonomy which uses a ranking scale (shown below with sample data of human beings):

Kingdom: Animalia

Phylum: Chordata

Class: Mammalia

Order: Primates

Family: Hominidae

Genus: *Homo*

Species: *sapiens*

Then “taxon” is a unit in a taxonomic system, such as species, genus, etc. And “species” is the basic lower unit of classification, consisting of a population or series

of populations of closely related and similar organisms [47], [48]. This classification based on differences in characteristics or genetics formed the basis for a more structured biodiversity data representation. Taxonomy is the science of identification and classification of organisms. It has strict rules, which all taxonomists follow while identifying, naming, and describing the species. With the expansion of knowledge in the domain many hierarchical levels are added to the taxonomic structure. The taxonomic data are classified into two groups such as “collection and observation data” and “nomenclature and taxonomic data”, as described below. There are different kinds of databases containing information about species, or more generally about taxa. Some of these databases contain information about classification and nomenclature, while some others contain information about characteristics, usages, conservation, and geographical distribution of organisms. The scope of biodiversity data has been expanding beyond classical or “biological” data. The ratification of the Convention on Biological Diversity (CBD) and the United Nations Environment Programme (UNEP) outlines the following eight characteristics of biodiversity data [49], [50].

- **Biological:** Information on ecosystem, species, and genetic resources.
- **Physical:** Information on physical factors such as climate, topography and hydrology that allows biological data to be placed within a physical context.
- **Socio-economic:** Information on socio-economic attributes such as population, population distribution and transport routes.
- **Cost and Benefits:** A value of biodiversity that takes into account the cost and benefits of management options.
- **Pressure and Threats:** Information on both potential and actual threats to biological diversity.
- **Sustainable management:** Information on current and past management activities particularly the use of biological resources.
- **Sources and Contacts:** Information models, standards and technologies, and appropriate agencies or experts who can be contacted.
- **Interrelationships:** Information on the interrelationship between and among species and ecosystems so as to forecast the effects of proposed actions.

The biodiversity data is represented in various formats such as physical samples, description of observations and is usually represented using documents with text and

images. The digitization of biodiversity data, performed using multiple medias such as text, images, videos and sound, is used to capture all the attributes of the biological data. The primary types of biodiversity data with respect to storing formats are explained in the following sections.

Collection and Observation Data:

Collection data, which are usually found in natural history museums, botanical gardens, and institutions holding microbial culture collections, contain information about biological organisms [51]. Observation data contain information on observation of an organism ideally at a specific geo-temporal location. The databases that hold these data are known as collection databases. The main information in these databases is about specimens, including the information specific to the specimen itself (e.g. taxonomic identification, sex, etc.), and the information about the collection event (date/time of collection, method of collection, etc.).

Nomenclature and Taxonomic Data:

Nomenclature data focuses on the list of names of a species and contain data relevant to a specific taxon. A comprehensive information model for designers of biological information systems [52] to record taxonomic and observation data from literature, field collecting and other sources has been proposed from research which will usually evolve into data standards. These databases are called taxonomic databases which may have variations in the representation of a real-world entity on different systems. This research attempts to resolve the interoperability issues prevailing in these taxonomic databases that can exchange the data in a specified XML format. The interoperability of these databases is concerned with heterogeneity of scope that refers to the fact that differing amounts and types of data are stored in the various databases; heterogeneity of representation refers to the terminology used, format, accuracy, range of values allowed and structural representation.

2.5 Evolution of Data Communication Standards in Biodiversity

A standard is a document approved by a recognized body that provides for common and repeated use, rules and guidelines for products or related processes and

production methods [53]. Non-governmental organisations such as ISO (International Organisation for Standardisation) act as a bridge that enables a consensus to be reached on standards and satisfies the reciprocal requirements of commercial and non-commercial needs of the community in general [54]. The ability of these organisations to accommodate larger participants and to provide universal service increases the scope and success of the interoperable standards. For example the ISO network is the world's largest developer and publisher of International Standards that has 162 member countries with a central secretariat for coordination. In the biodiversity domain organisations such as Governmental, Commercial, Natural history museums, Universities and other institutions are working together to form communities that develop standards, for exchanging data among them. Standards are a rule or requirement that is determined by a consensus opinion of the biodiversity data provider networks, experts in the data such as biologists and end-users. A standard provides a framework that is to be used consistently as a rule, guideline, or definition. Data communication standards are created to ensure that two or more independent data sources can collaborate in order to achieve compatibility. These standards support distributed querying and combining the distributed responses for a query. The success of a standard is based on the features such as simplicity in creation, easy maintenance, commonly understood semantics, international scope and extensibility. The use of standards will enable interoperability between different systems and can provide richer information for biodiversity research and analysis. The most commonly known data exchange standards in biodiversity domain are discussed in this section.

2.5.1 Dublin Core

Dublin Core is a metadata standard that defines an effective element set for describing a wide range of networked resources. The Dublin Core standard includes two levels: Simple and Qualified. Simple Dublin Core consists of fifteen elements such as Title, Author, Description, etc. Qualified Dublin Core includes three additional elements like Audience, Provenance and Rights Holder, as well as a group of element refinements also called as qualifiers that refine the semantics of the elements used for resource discovery [55]. Dublin Core is primarily used to describe digital resources. The semantics of the Dublin Core standard have been established by the Dublin Core

Metadata Initiative (DCMI) which is an open organisation comprising international, cross-disciplinary group of professionals from librarianship, computer science, text encoding, the museum community, and other related fields of scholarship and practice. The main principles of the Dublin Core standard are:

1. The One-to-One Principle. The metadata should describe one instance or version of a resource, rather than assuming that manifestations stand in for one another. For example if a original is reproduced as a copy instance then the relationship between the metadata for the original and the reproduction is part of the metadata description. It should assist the user in determining whether the original is needed or the reproduced instance will meet the user's requirement.
2. The Dumb-down Principle. According to this rule, a client should be able to ignore any qualifier and use the element value for discovery. Qualification is should be used only to refine but not to extend the semantic scope of a property.
3. Appropriate values. Context of application decides the best practice for a particular element or qualifier. In general an implementer of metadata cannot predict the type of interpreter but the design of metadata should be useful for discovery.

2.5.2 Earlier Standards and Protocols of Biodiversity Data

In some of the earlier biodiversity information systems the generic metadata element set of the cross-domain standards like Dublin Core or Z39.50 were used for data representation and exchange. In the biodiversity domain many standards and formats for representing data for exchange between software systems were developed. The Botanic Garden Conservation International (BGCI) organized international workshops and consulted with the experts in the biodiversity domain to develop a standard named as ITF2 (International Transfer Format for Botanic Garden Plant Records) [56]. This standard was mainly used for data transfer between botanic gardens. The 'Herbarium

Information Standards and Protocols for Interchange of Data' (HISPID) developed by a committee of representatives from all major Australian herbaria was first published in 1989 as a standard format for the interchange of electronic herbarium specimen information [57]. In 1989 Robert Allkin and Richard White in U.K [58] developed one of the earliest standard formats named XDF ("eXchange Data Format") that can be used for the definition and exchange of biological data sets. XDF is a text based, high-level language for describing biological data, with its own syntax and command vocabulary and it is very flexible for representing both taxon-based and specimen-based data. It was used for data representation and exchange in the implementation of the ILDIS (World Database of Legumes) project [59]. The organisation Biodiversity Information Standards (TDWG) is hosting a collection of most biodiversity standards in a repository and tracking the progress and development of these standards for the benefit of the users in biodiversity informatics.

2.5.3 Darwin Core V2

The Darwin Core (DwC) is a metadata standard specification of data concepts and structure intended to support the discovery, retrieval, and integration of information about organisms, their spatiotemporal occurrence, and the supporting evidence stored in collections either in physical or digital medium [60]. The primary goal of the Darwin Core is to provide a stable reference to standard terms about biodiversity, which can be used in a variety of contexts. The Darwin Core derives its vocabulary from community-based experience in data discovery, sharing, and integration, while its form is derived from the practices developed by the Dublin Core Metadata Initiative except where otherwise noted in the standard. Structures, data-typing, and constraints on the values of terms are meant to be implemented using representation-specific application profiles such as XML schemas. The standard consists of properties, elements, fields, concepts, the policy governing the maintenance of these terms, decisions resulting in changes to terms, the complete history of terms including detailed attributes, a generic application schema for use in the construction of new application schemas based on Darwin Core, a simple (flat) application schema for the use of these terms and a metafile schema to allow for the description of Darwin Core fielded text files. Most data resources include only the core data elements that are

likely to be available for the vast majority of specimen and observation records. This standard is utilized within both the Species Analyst and REMIB networks, among others. DwC is also a Global Biodiversity Information Facility (GBIF) approved data standard, and GBIF uses Darwin Core for harvesting data of specimen collections and observations data from organisations around the world to develop a catalogue of names of known organisms [61]. Darwin Core is the main biodiversity data standard used by many databases about natural history collections, living collections (i.e., zoological and botanical gardens), germplasm and genetic resource collections, and data sets produced from biodiversity survey and monitoring programs. Darwin Core supports the search and retrieval of descriptive information from these resources. The previous versions of DwC were integrated with the DiGIR communication protocol, but the recent version does not contain any references to the retrieval protocol making it appropriate to biodiversity data irrespective of the data exchange protocols such as HTTP, web service, etc...

2.5.4 ABCD Standard

The Access to Biological Collections Data (ABCD) Schema is an evolving comprehensive standard for the access to and exchange of data about specimens and observations in biodiversity [62]. The objective of ABCD Schema is to be comprehensive and highly structured in defining biodiversity concepts and to be compatible with other existing data standards. ABCD is the product of a joint TDWG and Committee on Data for Science and Technology (CODATA) initiative to develop a standard for distributed data retrieval from specimen collection databases. ABCD version 2.06 has been recommended by the TDWG meeting in November 2005 at St. Petersburg as the actual standard, and has since then been ratified by TDWG members. The schema supports data exchange for all kingdoms and for both specimen and observation records. The ABCD Schema is a GBIF approved data standard that incorporates DwC elements and it attempts to be comprehensive and highly structured, supporting data from a wide variety of databases [63]. Parallel structures exist so that either (or both) atomized data and free-text can be accommodated. Versions 1.2 and 2.06 are currently in use with the GBIF (Global Biodiversity

Information Facility) and BioCASE (Biological Collection Access Service for Europe) networks. ABCD is a step towards ontology for biological collections.

2.5.5 Taxonomic Concept Transfer Schema

The development of this standard was intended to solve problems that are inherent in the use of names to represent taxonomic concepts as required by the Rules of Nomenclature. These can be resolved by means of a richer representation, based on a name plus a reference to the definition of the concept. The taxonomic data providers, biologists, computer scientists and users identified the need for a common mechanism that would allow information interchange among them and with the users of varying expertise at TDWG Lisbon 2003. The aim of this standard is to adequately represent the data model of the data owners whilst facilitating the integration with different data models of taxonomy [(94)]. This was the main motivation towards the development of the Taxon Concept Schema (TCS) and later known as Taxonomic Concept Transfer Schema. The TCS schema was designed as an XML document that allows the representation of taxonomic concepts as defined in published taxonomic classifications, revisions and databases. This standard specifies the structure for valid XML documents to be used for the transfer of defined taxonomic taxon concepts, transfer GUIDs referring to defined taxon concepts or a mixture of the two. TCS documents are for transferring the definitions of taxon concepts, not for detailing observations of the defined concepts.

2.5.6 TAPIR

TDWG Access Protocol for Information Retrieval (TAPIR) specifies a standardized, stateless, HTTP transmittable, XML-based request and response protocol for accessing structured data that may be stored on any number of distributed databases of varied physical and logical structure [64]. TAPIR aims to integrate by extending features of the BioCASE and DiGIR protocols to create a new and more generic means of communication between client applications and data providers using the Internet. TAPIR was designed as a generic tool but was developed primarily for use with biodiversity and natural science collection data. The TAPIR task group is entrusted with liaising with other subgroups of TDWG and related biodiversity

standards group to ensure the applicability and effectiveness of the protocol for interoperability with other protocols. The TAPIR protocol is implemented using XML schemas [65].

2.5.7 SPICE Common Data Model

The SPICE Common Data Model (CDM) is a biodiversity communication protocol developed for the Species 2000 project. Species 2000 is a federation of database organisations working closely with users, taxonomists and sponsoring agencies [4]. The primary goal of this project is to create a validated checklist of all the world's species plants, animals, fungi and microbes. The SPICE project has developed a distributed computing engine that runs the Dynamic Checklist. A wrapper layer allows a number of species databases to be queried simultaneously to return a uniform list of results based on CDM. The conceptual basis of the SPICE distributed system is built around the SPICE Common Data Model (CDM) described in various documents (CDM v1.20, CDM v1.21). This specifies the SPICE Protocol by which the Common Access System (CAS) queries the list of connected databases, and the responses envisaged from these databases. The Spice CDM is implemented using XML schema and the current version of Spice is 5.0 [66]. The CDM XML schema provides six type of request with each type having a specific XML schema for a request and response message. Please refer to section 3.9 for more details on the Species 2000 project and its approach towards achieving interoperability.

2.5.8 Ontologies in Biodiversity

Complementing the development of data standards, Ontologies have been developed by some research projects to facilitate data interoperability through semantic mediation between different data formats, as described further in section 3.5. An ontology is a formal specification of a set of concepts and their interrelationships in some application domain, such as biodiversity. It is the knowledge or concept map useful in reasoning about the relationships among concepts and among data that pertains to those concepts. The TDWG Ontology working group and the SEEK ontology project have been the main research works in this area for biodiversity knowledge:

- The TDWG Ontology group develops the Biodiversity Informatics Core Ontology that comprises classes mainly derived from the 4 TDWG XML schemas namely ABCD, Darwin Core, SDD and TCS [67]. These schemas were analyzed to determine the high level concepts that should act as a core for a larger TDWG ontology to be developed by the biodiversity community.
- The Science Environment for Ecological Knowledge (SEEK) [68] is a system developed for storing, integrating, transforming and analysing ecological and biodiversity data. One of the primary goals of the SEEK project aims to build an internet based architecture for data storage, sharing, access and analysis and to achieve interoperability of data through semantic mediation in which automatic transformation of data can occur according to the information system. The Knowledge Representation Working Group of SEEK project develops a framework of foundational ontologies for biodiversity and ecological applications.

2.6 Biodiversity Information Community Networks

The biodiversity data community is the collection of people that are concerned with the development of standards for storing and for exchanging information in biodiversity data. This community is composed of biologists, taxonomists, librarians, zoologists, entomologists, ecologists, librarians, geneticists, information analysts, software engineers, developers and users of the data. Developing such community networks for the biodiversity domain fosters the activities of conducting workshops, meetings and publications on biodiversity data. Community of Interest group provides a platform to produce a consensus definition for the data exchanged between participants and also promotes interoperability between the information systems. Communities create data panels with a lead person coordinating the activity and they extract the shared knowledge necessary for the data interoperability from the larger community and propose the standards. Various community groups produce and maintain common data representations that are organized into ontologies, abstract schemas, and definitive XML schema standards such as Darwin Core and ABCD. Section 2.5 describes the different standards for biodiversity data representation

produced by these communities of special interest in biodiversity data. This section is introduced because of its relevance to our research work and it describes the importance of these network communities in achieving data and systems interoperability in biodiversity domain. In biodiversity informatics domain the organisations discussed in the following sections were important in promoting interoperability of biodiversity data at the global and European regional levels.

2.6.1 Global Biodiversity Information Facility

The Global Biodiversity Information Facility (GBIF) is a multi-lateral initiative established by inter-governmental agreement between countries and based on a non-binding Memorandum of Understanding. GBIF's objective is to provide global biodiversity data, freely and universally available on the Internet. GBIF encourages the network of data providers to adhere established biodiversity standards thereby promoting interoperability. It also aims to provide the essential informatics infrastructure for biodiversity research and applications. The Informatics activities of the GBIF focus on developing a complete range of information technology infrastructure, architecture, services and tools to serve a fully functional network of users. GBIF's projects are based on existing and emerging standards and applications and take an active part in their development, in close collaboration with Biodiversity Information Standards (TDWG). Please refer to section for the GBIF Informatics initiative project in section 3.9.2.

2.6.2 European Network for Biodiversity Information

ENBI was a thematic network supported by the European Commission under the Fifth Framework Programme and contributing to the "Energy, environment and sustainable development" programme [69]. ENBI operated as a European contribution to the GBIF. ENBI follows the objective of GBIF by concentrating on databases at the European scale and on activities that need co-operation at a European level. The ENBI network is coordinated by the Zoological Museum of the University of Amsterdam, Cardiff University, ETI Bioinformatics Amsterdam, Hebrew University of Jerusalem, Biologiezentrum of Austria and many other institutions in Europe. ENBI investigates the potential of developing applications to use with biodiversity

data. Most of the participants in the ENBI network are the coordinating institutes of past and current EU biodiversity projects and the national GBIF-nodes. The interoperable application BUFFIE version 1 was demonstrated for the ENBI network in Stockholm, Sweden. The activities of ENBI are coordinated with those of the European Community Clearing-House Mechanism and the European Environmental Agency.

2.6.3 Taxonomic Database Working Group

The organisation known as Biodiversity Information Standards (TDWG) was earlier known as the Taxonomic Database Working Group. TDWG is a not for profit scientific and educational association that is affiliated with the International Union of Biological Sciences. TDWG community was formed to establish international collaboration among biological database projects and focuses on the development of standards for the exchange of biological/biodiversity data [70]. TDWG promotes global dissemination of information about the World's heritage of biological organisms and acts as a forum discussion through holding meetings and through publications. TDWG is maintaining an information lookup database on other biodiversity network communities and biodiversity projects known as 'Biodiversity Information Projects of the World' and 'Biodiversity Information Networks Database' respectively [71].

2.6.4 LIFEWATCH

LifeWatch is a network that aims to develop an "e-infrastructure" to support all aspects of research on the protection management and sustainable use of biodiversity by providing services for scientists and policy makers using biodiversity data [72]. This project supports the research needed to meet the European Union policy objectives on biodiversity and is a major part of the European contribution to the Global Earth Observation System of Systems (GEOSS). The initial phase builds this community by gathering the interested EU member and associated departments with the objective of preparing a cooperation agreement on the construction and maintenance of the LifeWatch research infrastructure. The participants of this community include Universiteit van Amsterdam, Netherlands Institute of Ecology,

Norwegian Institute for Nature Research, The Natural History Museum in London, Finnish Environment Institute, Swedish Research Council, Cardiff University and other institutions.

2.7 Summary of Background Study

- The types of interoperability relevant to this research are Technical, structural, and semantic.
- Data standards are the first step towards achieving interoperability for querying multiple data providers in biodiversity information systems.
- Communities of interest groups in biodiversity achieve systems interoperability by defining rules and adhering to a particular standard.

Only when the relevant people come together and interact to achieve a common goal, then the systems they develop can be designed to interoperate. The analysis of these standards, tools and biodiversity networks reveal that they vital and the first and primary step towards achieving interoperability between heterogeneous and distributed data providers. Our research work involves participation with these standards and the communities of networks in the domain of biodiversity.

CHAPTER 3

Relevant Technologies and Interoperability Projects in Biodiversity Data

The World Wide Web Consortium (W3C) coordinates in developing interoperable technologies, specifications, guidelines, software, and tools to lead data communication across the internet in the best possible way. In the biodiversity domain organisations such as Global Biodiversity Information Facility (GBIF) coordinate with the various government organisations and biodiversity networks such as TDWG, ENBI and national nodes to provide a common platform for the systems in biodiversity. In this chapter we present an overview of the different technologies related to this research, starting with an introduction of the XML standards and the technologies such as XSLT, web services, ontology and software design principles and frameworks. Then finally an overview of biodiversity data and the various XML communication protocols available in the biodiversity domain are described.

3.1 XML Standards

The data/information exchanges between different information systems in a computer network have been accomplished using a specified data model that is represented using text or binary formats. There was a keen and a universal effort to develop a data structure that could hold rich information about metadata, which is easy for storage and communication. Standard Generalized Markup Language (SGML) is an international standard for the definition of device and system independent methods for representing texts in an electronic form [73]. SGML was issued as an international standard (ISO 8879) in 1986. SGML provided some flexibility and was intended for semantic markup that would define the data it contains. It was, however, very complex and expensive for use in data exchange over the web [74]. Hyper Text Markup Language (HTML) which was fundamental for the World Wide Web (WWW) evolved from SGML. HTML is the publishing language of the WWW and it consists of markup tags that tell the web browser how to display the document/data [75]. Though HTML was good for data presentation it was not adequate for defining the data. Extensible Markup Language (XML) is a simple, very flexible text format derived from SGML and it facilitates to overcome the problem of universal data interchange between dissimilar systems [76], [77].

The richly structured documents created using XML can be transportable from one hardware and software environment to another without loss of information. XML is not itself a markup language, but a specification for defining markup languages which is very useful to create documents that can represent structural, presentational, and semantic information alongside content. Just because the data is defined in XML specification does not mean that it can be interoperable, It might make it easier for different client applications to create an import adapter or filter, but the real benefit will come if and when the network of providers or partners have agreed an XML standard for the data domain for e.g. Biodiversity species data documents. Standards are very important to achieve successful communication and data interoperability in domain networks. XML standards document structure can be defined by Document Type Definition (DTD) or XML schemas. The research reported in the present thesis aims at designing architecture and developing programs which take advantage of the

knowledge encapsulated in the XML document structure information, and can behave in a more intelligent fashion to present the answer for a users query.



Figure 3.1: Structure of an XML document.

3.1.1 XML Schema

Though XML document structures were better equipped for representing data than ordinary text files with delimiters, it could make more sense in data communication only if there is a way to define a set of rules for these structures. These rules could help in automatic validation of the data that is contained in the XML structure. Document Type Definition (DTD) and XML Schema provide a means for defining the structure, content and semantics of XML document. Unlike DTD, XML Schema is more powerful and written in XML specification and hence it is widely used. XML Schema defines the names of elements, data types, attributes, namespaces that can appear in a XML document. It also defines the relationship between the elements, the order and number of child element and values for elements and attributes [78], [79]. XML schema increases the security and consistency of the XML data communication. For example, when the client program and a data provider have particular expectations about the format of the XML message's content, then the XML schema helps to validate that standard. XML schema validates an XML document using a

parser. A validated XML document is said to confirm to the rules defined in the schema. XML Schema became W3C Recommendation 02, May 2001. The XML schema shown in figure 3.2 describes (or validates) the XML document shown in the figure 3.1.

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!--W3C Schema -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns="http://digir.net/schema/protocol/2003/1.0"
  xmlns:darwin="http://digir.net/schema/conceptual/darwin/2003/1.0"
  targetNamespace="http://digir.net/schema/protocol/2003/1.0" elementFormDefault="qualified">
  <xs:import namespace="http://digir.net/schema/conceptual/darwin/2003/1.0" schemaLocation=""/>
  <xs:element name="count">
  <xs:element name="destination">
  <xs:element name="equals">
  <xs:element name="filter">
  <xs:element name="header">
  <xs:element name="records">
  <xs:element name="request">
  <xs:element name="search">
  <xs:element name="sendTime">
  <xs:element name="source">
  <xs:element name="structure">
    <xs:complexType>
      <xs:attribute name="schemaLocation" use="required">
    </xs:complexType>
  </xs:element>
  <xs:element name="type">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="search"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:element>
  <xs:element name="version">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="1.0.0"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:element>
</xs:schema>

```

Figure 3.2: XML Schema Definition.

3.2 API for XML Processing

A special purpose program known as a parser can be used to process an XML document of a particular type and check that all the required elements for that document type are present and ordered as specified. More significantly, different documents of the same type can be processed in a uniform way. The XML structured data needs to be processed for using them in applications. These application programming interfaces (API) provide methods for reading, manipulating and storing the XML data. The two major types of XML API parsers are:

- Object model parsers which read the entire XML document, as Document Object Model (DOM) and construct an in-memory representation of the XML document [80].
- The Push and Pull parsers that simply read an XML document and return the data and structure of the document as Simple API for XML (SAX) [81].

The DOM API is designed to create a standard object oriented representation of XML documents. It is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents. The W3C DOM Working Group coordinates the direction and development concerning the evolution of the Document Object Model [82]. In DOM an in-memory representation of the complete XML document is created in a tree like structure for parsing. The programs and scripts can dynamically access and update the content, structure and style of XML documents using these API methods. The DOM API allows random access and navigation to the nodes in any direction that enables arbitrary modifications to the elements and its relationships in the tree structure. The downside of DOM API is that it uses more memory, but it is powerful and has many implementations. The limitation of DOM API is resolved in new approaches as in Apache's AXis Object Model (AXIOM) [83], which uses a StAX pull-parser for reading XML and only builds the tree representation of a document until the last node that was requested. Therefore, it does not need to read the complete document. JDOM (JSR-102) is an open source library and a Java API for processing XML document [84], [85]. It is similar to the DOM but specifically developed for the Java language. It uses the JAXP parser for handling XML and can integrate with DOM and SAX API implementations.

SAX (Simple API for XML) and StAX (Streaming API for XML) are event driven and serial I/O stream mechanisms for accessing XML documents. These APIs are the fastest and least memory consuming mechanisms for dealing with XML documents. SAX is a push parser that returns the data of the whole document in one stream and cannot be stopped. SAX is useful in applications that involve state independent processing where the processing of an element does not depend on the elements that

came before or after in the same document. The document is accessed sequentially and navigating to an earlier position or jumping ahead to a different position is not possible. StAX (Streaming API for XML) is a pull-parser specification built on the proposal of Java Community Process Program as defined in JSR 173 [86], which returns data only when requested to read the next node in a document. This is most useful in situations where the data is read and then processed by the application.

3.3 XSLT

Extensible Stylesheet Language (XSL) is developed by World Wide Consortium (W3C) as an XML- based style sheet language used for formatting or styling the XML document [87]. Interoperation of XML data is only possible, if there is a way to transform the XML documents from one structure into a different structure of XML document as required. This need drives the development of a sub-language called XSLT (XSL Transformations). XSLT is the most important part of XSL and it is used to transform a source XML document into another result XML document or another type of document such as HTML and XHTML. XSLT gives the ability to add/remove the attributes and elements from an XML document and can rearrange, sort and perform tests and make decisions about elements and attributes. XSLT uses the Xpath to find information and to navigate through the elements and attributes in XML document. The power of XSLT is to handle the data contained in XML in a programmatic way. Hence XML is now widely accepted as data representation syntax for communication over the internet, many research projects are using XSLT based infrastructure for transforming XML documents.

In this research approach we propose to use XSLT templates as one of the building blocks of the domain knowledge base. The functional programming methods and declarative style of XSLT, benefits our approach in deriving the metadata information from the XML structure and use that information for applying correct transformation function [88]. The modularity provided by the nature of XSLT programming [89] helps to create the separate knowledge base modules from the core framework of the information systems.

3.3.1 XPath

Xpath is a language that is defined mainly for discovering the information from an XML document. Xpath is a W3C standard which is used to navigate through the elements and attributes of an XML document [90]. Both XPath 1.0 and the most recent XPath 2.0 are expression language that process values that conform to the data model defined in XQuery/XPath Data Model (XDM). This data model provides a tree representation of XML documents and the atomic values such as integers, strings and booleans and sequences which contains the references to nodes and atomic values in an XML document. XPath uses path expressions to select nodes or node-sets from an XML document. XPath has built-in standard functions for string values, numeric values date and time comparison, node and QName manipulation, sequence manipulation, boolean values and other data types. Xpath is a major part of XSLT standard. XQuery also known as XML Query was designed to query XML data. XQuery is built on XPath expressions. XQuery 1.0 and XPath 2.0 share the same data model and support the same functions and operators.

3.4 Web Services

Web services are application components that provide interoperability between different software applications, running on a variety of platforms with various frameworks and are also called as utility computing [91], [92]. The W3C Web Services Activity group is designing the infrastructure, defining the architecture and creating the core technologies for Web services. The basic of web services platform is XML and HTTP. It uses XML to code and decode the data and Simple Object Access Protocol (SOAP) to transport it between the client and web service. SOAP is a platform independent XML based communication protocol between applications for sending messages via internet [93].



Figure 3.3: Data Communication in Web services

Independent programs that are providing simple services can be converted into web-applications using web service and they can interoperate with each other to deliver significantly improved value service. The Web Services approach provides an interoperable homogeneous runtime environment for the different applications by focusing on the design of well defined service interfaces [94]. Web services are developed as self-contained and self-describing components that are published by the owner and can be found by the clients across the internet using Universal Description, Discovery and Integration (UDDI) for use through the web. UDDI is a platform-independent framework for describing web service interfaces using Web Service Description Language (WSDL), discovering businesses, and integrating business services by using the Internet [95]. WSDL is a document written in XML which describes a web service. It specifies the location of the service and the operations (or methods) exposed by the service [96]. Other consortium such as OASIS (Organisation for the Advancement of Structured Information Standards) drives the development of more Web services standards for security, e-business, and standardisation efforts in the public sector and for domain/application-specific markets [97]. Global Biodiversity Information Facility (GBIF) provides a UDDI registry for the biodiversity data providers across the globe. GBIF UDDI registry service is available for all the GBIF national nodes where they can publish the web services [98].

3.5 Ontology and OWL tools

The abundance of inexpensive storage media, variety of data warehousing software and especially the popularity of the internet has made vast amounts of data available on digital format. This expansion in volume has made it difficult to analyze the data and to combine them properly to get the right information. Ontologies have been proposed as a solution for semantic data integration. These ontologies are constructed by capturing, representing and structuring the general relationships and semantic relations of the concepts in the domain [99]. An ontology is “an explicit specification of a conceptualization” [100]. A conceptualisation is an abstract, simplified view of the world that we wish to represent for some purpose. An ontology defines a common vocabulary for researchers to share information in a domain including machine interpretable definitions of basic concepts and relations among them. Another more technical definition of ontology in practical terms is that, ontology is a formal explicit

description of concepts in a domain of discourse where concepts are implemented as classes, roles are expressed as properties of each concept describing various features and attributes of the concept and role restrictions [101], [102]. Interoperability and data integration research projects aim to deliver computational ontologies that consist of logical axioms that relate terms of interest with specific purpose and scope in well-understood domains. However the limitation of Computational Ontology is that they cannot capture all real world semantics, but can express only the logical relations between terms in the domain [103]. The reasons why ontologies are much emphasized for data integration are that they allow sharing common understanding of the structure of information, enable reuse of domain knowledge, make domain assumptions explicit, separate domain knowledge from implementation knowledge and analyze domain knowledge. The five stages of ontology development are [104]:

- Specification of the purpose, scope and stakeholders of the ontology are identified.
- Conceptualization in which the organisation of acquired knowledge takes place. A conceptual model of the knowledge is represented in both tabular and graphical form.
- Formalization, which transforms these models of the conceptualization phase in to semi-formal models, this is the intermediate stage, where the information can still be easily understood by domain experts.
- Implementation, based on the models produced in the formalization phases, the ontology is implemented in the desired knowledge representation language.
- Maintenance, the final phase where corrections are made to the ontology, if needed.

The most prominent knowledge representation language used for building ontologies are Resource Description Framework (RDF) and Web Ontology Language (OWL) [105]. OWL provides additional vocabulary along with a formal semantics to represent data. OWL has three increasingly-expressive sublanguages: OWL Lite, OWL DL, and OWL Full.

3.5.1 Protégé

Protégé was developed by the Stanford Center for Biomedical Informatics Research at the Stanford University School of Medicine. Protégé is a free, open-source platform that provides a growing user community with a suite of tools to construct domain models and knowledge-based applications with ontologies [106]. Protégé helps users to construct domain ontologies, customise data entry forms, and enter data. The main components of Protégé implement a rich set of knowledge-modelling structures and actions that support the creation, visualization, and manipulation of ontologies in various representation formats. Protégé can be tailored to provide domain-friendly support for creating knowledge models and entering the data. Further, Protégé can be extended by way of a plug-in architecture and a Java-based Application Programming Interface (API) for building knowledge-based tools and applications.

The Protégé tool supports two main ways of modelling ontologies [107]:

- The Protégé-Frames editor builds and populates ontologies in accordance with the Open Knowledge Base Connectivity protocol (OKBC). In this model, an ontology is comprised of a set of classes organized in a subsumption hierarchy to represent a domain's main concepts, a set of slots associated to classes to describe their properties and relationships, and a set of instances of those classes.
- The Protégé-OWL editor builds ontologies for the Semantic Web, according to W3C's Web Ontology Language (OWL). "An OWL ontology may include more vocabulary about classes, properties and their instances. The OWL formal semantics specifies how to derive its logical consequences, from these ontologies. Other tools for ontology development include OilEd Apollo, OntoLingua, OntoEdit, RDFedit, WebODE, WebOnto, KAON and many more. Comparative studies on these tools are published in the survey paper by Dennis McLeod and Seongwook youn [108].

3.6 Object Oriented Design in Programming

By the early 1990 s Object-oriented design (OOD) evolved as a mainstream software application development. Object-oriented programming (OOP) is a programming paradigm that envisages a program as a set of interacting objects, each of which holds its own data and behaviour [109]. Object oriented modelling has been proposed as a solution to resolve differences in heterogeneous systems, as object orientation's principles can be used to present a unified interface [110], [111]. The core principles object oriented design involve finding pertinent objects, factoring them into classes at the right granularity, defining class interfaces and inheritance hierarchies, and establishing key relationships among them. The design should be specific to the problem at hand but also general enough to address future problems and requirements [112].

OOD favours low coupling of components in the system which means the components should be developed to the interface and not to an implementation. Another fundamental aspect of OOP is code reusability. This can be achieved using two routes namely white-box and black-box reusability. In white-box method the derived class inherits the code, context and some visibility of the parent class. Black-box method is based on object composition which is creating a new type that holds an instance of the base type through internal reference. This behaves as a wrapper class that delegates the call internally to the held instance of the class it enhances. The three more advanced design principles of object-oriented design are:

- The Open/Closed principle (OCP) which allows a module to be open for extension but closed for modification.
- Liskov's Substitution Principle (LSP) where subclasses should be substitutable for their base classes. This feature is polymorphism.
- The Dependency Inversion Principle (DIP) states high-level modules should not depend upon low level modules. Both should depend upon abstractions. Abstractions should not depend upon details. Details should depend upon abstractions.

OOD principles when used properly will deliver the benefits of the features such as encapsulation, modularity, polymorphism, inheritance and make the application code easier to read, test, extend and maintain. In the next section we discuss two of the most important software development frameworks for creating applications.

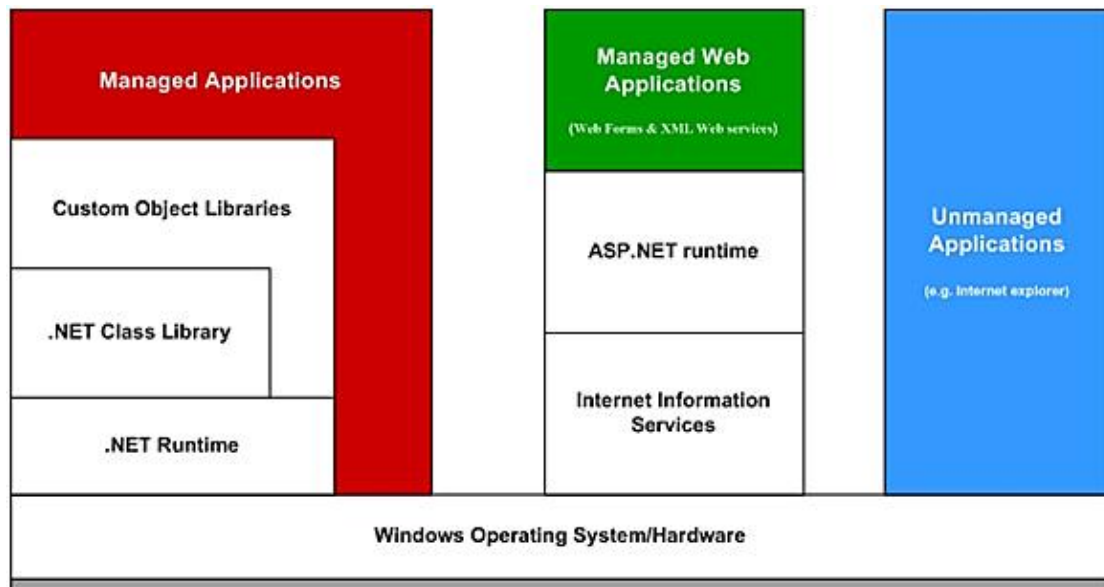
3.6.1 Microsoft .NET Framework

The .NET Framework is a software framework and an integral component of Microsoft Windows operating systems. It supports building and running the next generation of applications and XML Web services. The .NET Framework is designed to fulfil the following objectives [113]:

- Provides a consistent object-oriented programming environment irrespective of the object code location and its execution.
- Provides a code-execution environment that helps in effective software application deployment and avoids versioning conflicts.
- Provides a code-execution environment that promotes secure code execution.
- Provides a code-execution environment that eliminates the performance problems of scripted or interpreted environments.
- Consistent developer experience for both Windows-based applications and Web-based applications.
- Provides communication standards for codes based on the .NET Framework that can integrate with other applications.

The two main components of .NET framework are the common language runtime (CLR) and the .NET framework class library [114]. The CLR is the foundation of the .NET framework which is a runtime component that manages code at execution time, providing core services such as memory management, thread management, and remoting. CLR enforces strict type safety and other forms of code accuracy that

promote security and robustness. The class library, the other main component of the .NET Framework, is a comprehensive, object-oriented collection of reusable types that can be used to develop applications ranging from traditional command-line or graphical user interface (GUI) applications to web applications based on ASP.NET such as Web Forms and XML Web services.



*Figure 3.4: Microsoft .NET framework Architecture.
(source of Information from Microsoft website)*

3.6.2 Java Framework

The Java framework is predominantly an open source platform of the hardware or software environment in which a Java language program runs [115]. The two main components of the Java platform are:

- The Java Virtual Machine (JVM), Java programs are executed within JVM that converts the program into a byte code and which is then processed by the native operating system like Microsoft Windows, Linux, Solaris OS, and Mac OS. This helps the Java programs to be portable and interoperable as well.
- The Java Application Programming Interface (API), Java APIs are libraries of compiled code that is useful to create ready-made and customizable functionality to the programs and saves coding time. These are grouped into libraries of related classes and interfaces known as packages.

The Java framework incorporates a number of different APIs each providing a specific set of services to the application as shown in the figure 3.3. Java Platform Standard Edition (Java SE) development kit helps to write programs in three basic flavours: applets, applications, and servlets/ Java Server Pages technology (JSP) pages. Applets run in the JVM built into a web browser; applications run in the JVM installed on a computer system; and servlets/JSP run in the JVM installed on a web server such as Apache Tomcat.

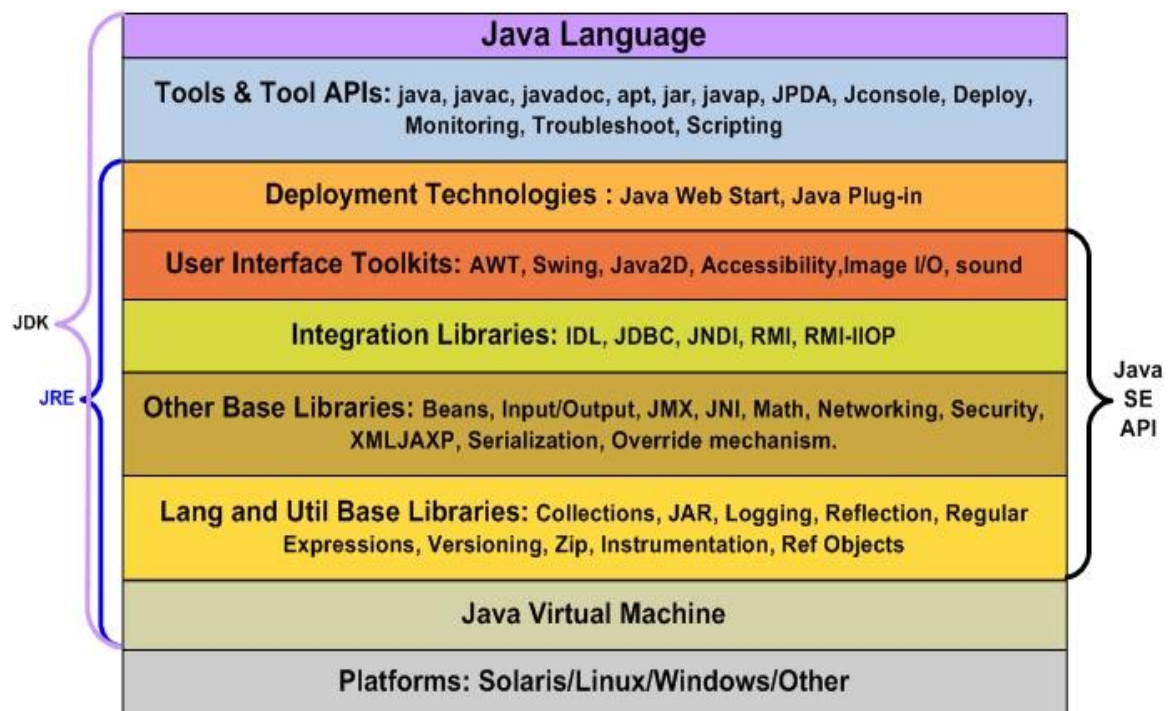


Figure 3.5: Java Framework Architecture.
(Source of Information from sun Java website)

3.7 Functional Programming Model

Functional programming is a fundamental style of computer programming to solve software engineering problems that treats computation logic as the evaluation of mathematical functions. Functional programming has its roots in the lambda calculus developed to investigate function definition, function application and recursion [116]. Functional programming approach is used for transforming the data structures like XML using XSLT functions. In functional transformation, a set of functions, define

how to transform a set of structured data from its original form into another form. The functions are:

- Self-contained, meaning functions can be freely ordered and rearranged without any interdependencies to the rest of the program. Pure transformations have no knowledge of or effect upon their environment. That is, the functions used in the transformation.
- Stateless, meaning that for the same input parameters the function or a set of functions will always result in the same output. These transformations do not store the result of previous execution.

The following are new language features of .NET C# 3.0 that are used to create functional programming that are more expressive, and easier to code, debug and maintain the applications [117]:

- Lambda expressions are a concise, functional syntax for writing anonymous methods. They are very useful for writing LINQ (please refer section 3.7.1) query expressions as they provide a very compact and type-safe approach that support higher-order functions that can be passed as arguments for subsequent evaluation.
- Anonymous types & implicit typing: Anonymous types are a feature of C#, which enable an unnamed class type to be declared and an object of that type to be instantiated at the same time without having to explicitly define a formal class declaration of the type. Implicit typing is the ability to determine the type of a variable in the absence of an explicit type declaration.
- Deferred execution and lazy evaluation: Deferred execution means that the evaluation of an expression is delayed until its resolved value is actually required. This can greatly improve performance when large data collections are manipulated, especially in programs that contain a series of chained queries or manipulations.

3.7.1 Language Integrated Query (LINQ)

Language-Integrated Query (LINQ) is a set of features in the .Net 3.5 framework that offers a powerful and consistent model for working with data across various kinds of data sources and formats [118], [119]. The C# 3.0 language can be used to write LINQ queries that introduce standard, easily-learned patterns for querying and updating data. The integrated development environment Visual Studio 2008/2010 includes LINQ provider assemblies, use the same basic coding patterns to query and transform data in .NET collections, SQL Server databases, ADO.NET Datasets, and XML documents. LINQ queries can be used on objects with IEnumerable or IEnumerable<(Of <(T)>>) collection directly without the use of an intermediate LINQ provider or API. LINQ queries offer three main advantages over traditional *foreach* loops:

- They are more concise and readable, especially when filtering multiple conditions.
- They provide powerful filtering, ordering, and grouping capabilities with a minimum of application code.
- They can be ported to other data sources with little or no modification.

The more the complexity of the operations performed on the data, the more benefit will be realized by using LINQ instead of traditional iteration techniques. The LINQ technologies of .NET 3.5 framework provides a consistent query experience for objects (LINQ to Objects), relational databases (LINQ to SQL and LINQ to ENTITIES), and XML (LINQ to XML).

3.7.2 LINQ to XML

XML is one of the prominent ways of data formatting in many scenarios. XML is used on the web, configuration files, applications and databases. LINQ to XML is an in-memory XML facility to provide XPath/XQuery functionality and a redesigned approach to programming with XML. It provides a programming interface using the in-memory document modification capabilities of the Document Object Model

(DOM), and supports LINQ query expressions [120], [121]. LINQ to XML provides a powerful approach to creating XML elements which is referred to as functional construction. This approach creates all or part of XML tree in a single instruction. LINQ to XML provides a full set of methods for manipulating XML like insert, delete, copy, and update XML content.

3.7.3 LINQ to Entities

Many web based applications are currently using relational databases for data persistence. These applications interact with database in a relational form that is specific to a particular type. The Entity Data Model (EDM) is a conceptual data model that can be used to model database schemas into objects of .NET classes, so that the applications can interact with the data as entities or objects. Language-Integrated Query (LINQ) provides support for the object layer exposed as entities by the ADO.NET through a provider. LINQ to Entities enables developers to write queries against the database from the same language used to build the business logic [122].

3.8 Database Management Systems

A database is an organized collection of data that is stored in a computer system. The database model structure is classified as hierarchical, network, relational and object models. Database management system (DBMS) is a computer software that is designed to assist in defining, maintaining and utilizing large collections of data stored in the database [123] . The first general-purpose DBMS was designed by Charles Bachman at General Electric in the early 1960s and was called the Integrated Data Store. By late 1960s IBM developed the Information Management System (IMS) DBMS based on hierarchical data model. In 1960 Edgar Codd at IBM's San Jose Research Laboratory designed the relational data model which is the most prominent basis for DBMS used at present. The Structured query language (SQL) for relational databases was initially developed by IBM and later became ANSI and ISO accredited standard. SQL is used for programming for managing the database.

DBMS provides many advantages [124]:

- Data Independence DBMS provides an abstract view of the data which hides the details of data representation and storage.
- Efficient Data Access - DBMS uses a variety of sophisticated techniques to store, update and retrieve data efficiently.
- Data Integrity and Security: - DBMS can enforce integrity constraints and also can enforce access controls that govern what data is visible to different classes of users.
- Data Administration - shared data among several users, can be managed easily by centralizing the administration
- Concurrent Access and Crash Recovery: A DBMS schedules concurrent accesses to the data for the users and protects them from the effects of system failures.
- Reduced Application Development Time: DBMS supports important functions common to many applications that are accessing the data which could save time in application development and testing.

Due to various advantages of DBMS systems they are used to resolve the interoperability issues of the data. The approach in which many databases systems provides a solution for shared access to heterogeneous files created by multiple autonomous applications in a centralized environment is called Multidatabase or federated systems [125], [126].

3.9 Related Works of Interoperability in Biodiversity

This section reviews the most relevant and related project works that were carried out in the biodiversity informatics domain, which is aiming to resolve the interoperability issues of the heterogeneous databases. Various research projects attempt to correlate the inherently heterogeneous biodiversity data in the domain by proposing methodologies for data collection and implementing standards for data modelling both at the conceptual and at physical level. The organisations like GBIF, TDWG in alliance with CODATA (Committee on Data for Science and Technology) and other biodiversity communities establish an international collaboration among the biological data providers and information system developers. They work in

collaboration and develop standards for communication and implement biodiversity information systems to access data from multiple data providers using various standards, architecture and technologies that would allow them to interoperate between different systems or databases. Earlier researches in biodiversity data can be classified into two areas based on data communication: file-based communication and XML-protocol-based approach. Most of the projects have used biodiversity standards to represent the data that could be understood with-in a network of data providers accessed by the application. The facility to structure the biodiversity data in a flat file using metadata has been found inadequate as it needs another layer of processing to infer information. This limits the use of flat files for encapsulating data. For example major Australian herbaria data providers network use a file based standard known as HISPID4. Another example is the ILDIS project where XDF [127] is used which is a file based representation with a formal definition language that can serve as a medium for defining biological data transfer formats for use between databases with incompatible formats. Dave Vieglais and others at University of Kansas Natural History Museum and Biodiversity Research Center are, involved in the Species Analyst research project aimed at developing standards and software tools that facilitate the data communication between 120 or more natural history collections databases, located all over the world. It uses the Z39.50 protocol and converts the result set into XML format [128]. The Z39.50 protocol is a client-server based protocol used for searching and retrieving information from remote databases, the main limitation is that it is pre-Web technology and is being mainly promoted in the library information domain [129]. In biodiversity domain groups of organisations are working together to form networks and implement common access systems and forming protocols more suitable for the data interoperability using XML based communications. Following projects described in this section use XML schema standards for data exchange.

The SPICE (Species 2000 Interoperability Co-ordination Environment) project main aim is to develop a suitable architecture that could provide a catalogue system consisting of all known species. The Species 2000 project provides a federation of individual databases coordinated by taxonomists, universities and other organisations [130]. The common access system aims eventually to provide a virtual checklist index

of all known species composed from segments held in the set of global species databases (GSDs). This system uses the SPICE protocol for data communication known as Common Data Model that consists of different type of requests. This architecture could allow many different data providers to interoperate by providing wrappers to translate from their own peculiar format into the common one. Thus this approach uses the protocol, tightly coupled database federation and data provider side wrappers to achieve greater interoperability. The Species 2000 project also provides programmatic access to find synonyms for species names through two Catalogue of Life web services namely Annual Checklist web service and Dynamic checklist SOAP web service [131].

LITCHI – The project “Logic-based Integration of Taxonomic Conflicts in Heterogeneous Information Systems” is concerned with the integration and maintenance of biodiversity databases. The constraints used to identify taxonomic conflicts in individual species database and in merged database are from established taxonomic practice. The LITCHI system can be used to resolve such conflicts incrementally on the databases from distinct sources. This system helps to resolve the taxonomic conflicts in individual and linked or merged species databases [132]. LITCHI has been implemented in two phases, Phase 1 was implemented as a standalone system using data files in the XDF format and in Phase 2 it is used by Species 2000 Europa project and adheres to the Species 2000 data standards. This does the integration of data using a set of consistency rules developed by biologists that produce XML cross-maps. This represents the relationships between the species that have been detected by the rules. In effect the cross-map is a knowledge-base or thesaurus.

The BioCASE Biodiversity Collection Access Service for Europe is to establish a web-based information service providing researchers with unified access to biological collection of Europe [133]. BioCASE use the ABCD XML schema which is a comprehensive data specification in biodiversity intended for data integration and communication. This approach uses database side wrappers that converts the database output to the required ABCD data set and the request are made in XML format using the HTTP protocol. This project uses the subset of ABCD protocol to define the

biodiversity data. TDWG Access Protocol for Information Retrieval (TAPIR) is based on HTTP, XML-based request and response protocol for accessing structured data that may be stored on multiple distributed databases with different physical and logical structure [64]. TAPIR was designed to be the successor of the Darwin Core and BioCASE Protocol. The aim here is to provide interoperable access to biodiversity and natural science collection data from data providers.

The Biodiversity World project (BDW) at Cardiff created a flexible and extensible web services-based Grid environment for biodiversity researchers to analyze the biodiversity data [134]. The BDW aims to provide the richness analysis, bioclimatic modelling and phylogenetic analysis on the heterogeneous biodiversity data. It uses the Triana workflow management tool for building and executing the workflows. A communication layer developed as Biodiversity Grid Interface (BGI) interfaces with the Grid resources and data sources. BDW data type is used for data representation and used by the components of the system. In this approach the heterogeneous data resources are exposed as web services using resource wrappers that could be invoked using HTTP requests. The data sets returned were in the format of XML documents. The metadata repository (MDR) component provides the information of the available data resources to the BDW system. The project realized a bioclimatic modelling workflow and thereby demonstrating the architecture to access heterogeneous data resources.

The Mammal Networked Information System (MANIS) is developed with the support from the National Science Foundation (USA) and seventeen North American institutions and their collaborators. The objectives of MANIS were to facilitate open access to combine specimen data through internet [135], MaNIS network uses an extension of Darwin Core standard for data communication between the participating data providers and avoids external maintenance of the network and centralized data management. The MaNIS network provides access to mammal specimen records from multiple and distributed and autonomous databases using a web based portal architecture and a universal data standard for all the participating data providers. The portal application sends requests for data to the provider software installed on computers at the participating institutions.

3.9.1 BUFFIE - v1.0

The BUFFIE v1.0 (Biodiversity Users Framework for Interoperability Experiments) project is a first phase of this research work and is based on the software engineering approach combining service oriented architecture, purpose-built-ontology (XML) and extensible processing methods. This research approach attempts to resolve the interoperability issue in the biodiversity area and the prototype system is developed and demonstrated in the European Network for Biodiversity Information ENBI [136]. Existing systems only allow searching data within a network community. BUFFIE is proposed as a solution to overcome this limitation by interoperating among various network communities at the syntax and semantic levels and improving data responses before presenting it to the user. BUFFIE v1.0 was designed, developed and tested using real species data provided from resources in Israel, The Netherlands and Austria. BUFFIE v1.0 was implemented on windows 2003 server and successfully demonstrated the interoperation of heterogeneous databases in the ENBI meeting at Stockholm in October 2005. The results of this research work has been published and continued to develop the next version BUFFIE v2.0, which includes more functionalities to achieve interoperability of biodiversity networks.

3.9.2 GBIF - Infrastructure

The Global Biodiversity Information Facility (GBIF) aims to provide free and open access to biodiversity data for any users across the Internet. GBIF initiative on informatics infrastructure for biodiversity research and applications is classified into six major components:

- **Publishing:** GBIF developed the Integrated Publishing Toolkit (IPT) as a software platform to publish biodiversity data on the Internet through the GBIF community network. IPT is a Java based application that manages three types of data: taxon primary occurrence data, taxonomic checklists and resource metadata. IPT allows the data providers to upload data in comma-separated and tab-delimited files to GBIF servers and also allows browsing and searching published data by end-users [137].

- **Discovery:** The Global Biodiversity Resources Discovery System (GBRDS) is an Internet-based registry that aims to create a single annotated index of biodiversity data publishers, institutions and collections, a schema repository and services. The GBRDS with its metadata catalogue and indexes, serves as a unified global entry point for the discovery of biodiversity resources, and also integrates the GBIF network with other systems [136].
- **Harvesting:** The Harvesting Index Toolkit (HIT) is an open-source, Java-based web application that builds an index of all biodiversity records into a central server at GBIF from a distributed network of data publishers.
- **Integrating:** GBIF provides a Data Portal as a proof of concept that a worldwide distributed network of biodiversity data providers can be linked together and made searchable from a single point of access. It allows searches on any taxon, country, or on a combination of parameters.
- **Retrieving and Analysis:** The GBIF portal provides a range of web services that can be used by other applications to directly access XML-formatted data. They are Taxon and Occurrence data service, Dataset metadata service, Data Provider metadata service and biodiversity community network metadata service. The data thus retrieved can also be used by other applications for analysis and to find more useful information on biodiversity data.

3.9.3 Global Earth Observation System of Systems

The Global Earth Observation System of Systems (GEOSS) project is an interoperability research trying to link the infrastructures of the climate change research and biodiversity research. The aim of this project is to realize an interoperable infrastructure based on service oriented architecture [138]. The GEOSS strategy is to use the advantages of existing systems and services and promoting interoperability through the adoption of a Service Oriented Architecture (SOA) framework approach based on established standards from bodies such as the International Organisation for Standardisation (ISO) and Open Geospatial Consortium (OGC). GEOSS overall system architecture consists of the following main logical

components: biodiversity data provider, climatological data provider, Catalog performing search operations on both biological and climatological datasets, Ecological niche modeller and graphical user interface. This project is a long term research started from 2005 and going on until 2015. The results of the pilot project has validated the need for international standards to support interoperability and developing mediation catalogue services using an open framework approach that manages the complexity of multi-disciplinary federated systems.

3.9.4 Distributed Dynamic Diversity Databases for Life

4D4Life (Distributed Dynamic Diversity Databases for Life) is a scientific data infrastructures project started by the European Commission's e-infrastructure programme [139]. The research activities of this project will establish the Catalogue of Life as a state of the art e-science facility, using service-based distributed architecture and by making it available for integration into analytical and synthetic distributed networks in the area of conservation, climate change, invasive species, molecular biodiversity and regulatory domains. It will create electronic taxonomic services like synonymy server, taxon name-change and other services that help any other systems in biodiversity domain to achieve better data interoperability.

3.9.5 Life Science Identifiers

The amount of biological data being created on computer databases is huge and biologists or bioinformaticians provide common access systems that have different ways to access the biodiversity information from multiple, distributed and heterogeneous databases. Due to the nature of this biodiversity data being dynamic and the data entity names or values can have synonyms, homonyms creates a problem while finding this data using a search query. To resolve the issues of naming and identifying data resources stored in multiple, distributed data stores, Life Science Identifiers (LSIDs) are used to uniquely reference each unit of data from a provider. LSIDs are persistent, location-independent, resource identifiers for uniquely naming biologically significant resources including species names, concepts, occurrences, genes or proteins, or data objects that encode information about them [140]. An LSID resolver is a software system that implements an agreed-upon LSID resolution

protocol to allow higher-level software to locate and access the data uniquely named by any LSID URN. The “server” side of this resolver solution is called an LSID authority. The client stacks and an example client, the LSID LaunchPad, are provided by the LSID Resolution Protocol Project [141]. LSID consists of the following five parts, each separated by a colon:

- Network Identifier (NID), i.e., the “urn:lsid:” label;
- Authority Identification, usually the root DNS name of the issuing authority;
- Namespace Identification chosen by the issuing authority;
- Object Identification unique in that namespace; and
- An optional Revision Id to represent versioning information.

LSID adoption in biodiversity domain is being encouraged by organisations like TDWG, GBIF, Species2000 and other data providers. This technique will contribute to resolve the interoperability problems, when trying to integrate data from multiple databases about making a decision on the uniqueness of the data retrieved.

3.10 Summary Analysis of Relevant Technologies and Projects

Organisations like GBIF and TDWG either participate or coordinate the majority of biodiversity informatics projects across the globe by providing a networking and communication platform. TDWG is mainly dealing with the biodiversity data standards and GBIF on the other hand integrates the informatics infrastructure of the different research projects and works with the biodiversity nodes of the different countries in the world [20]. Interoperability in Biodiversity information systems is an approach that involves multiple levels of research and problem solving like technological, data representation and communities of networks. European Network for Biodiversity Information (ENBI) contributes to the objectives of GBIF by providing a platform for European biodiversity data, information coordination, and exchange of information, priority setting and selected feasibility studies. The ENBI network community serves as a good platform for this research in terms of knowledge sharing and for implementing and testing the new prototype of framework in coordination with the data providers, clients and users. Reviewing the available

biodiversity data standards and the network communities in sections 2.5, 2.6 and the survey [142] of the related biodiversity informatics project in section 3.9 reveals the direction of the research in biodiversity informatics.

- The majority of the biodiversity data providers are publishing the data by participating into one or more network communities.
- Each biodiversity network community is aiming to achieve data interoperability by adhering to a particular data standard like Darwin Core, ABCD (BioCASE) or a Common Data Model like in SPICE 2000.
- Most of the biodiversity networks aim to achieve structural data interoperability by implementing wrappers at the data providers and converting the data representation to a common format used by that particular network. For example DiGIR providers like MaNIS network uses wrappers to convert the data to a Darwin Core standard, BioCASE network use wrappers to convert data to ABCD standard and SPICE 2000 project uses
- The main biodiversity informatics projects implement web based architecture using HTTP protocol and universal data representation in the network of data providers for distributed querying over multiple and heterogeneous databases.

Earlier researches in biodiversity projects using network communities have all implemented the first step in the interoperability process by adopting one of the established standards of interchange formats, to which all their data providers convert their data format. Reviewing the related projects have shown that one of the issue, is that the data providers have to implement wrappers to join a particular XML protocol, if they need to be included in a network of common access. Some of the researches are working to develop a comprehensive universal schema that should contain the available standards. For example TAPIR schema encloses both ABCD and Darwin Core standards. The current common access systems can only query the multiple data providers only if they are participating to the common standard of the network. Further research is needed for structural and semantic interoperation between different

biodiversity networks through a common access system that can improve the accuracy with which information can be retrieved and used for biodiversity research.

This research work approach aims at achieving structural and semantic interoperability between the networks of biodiversity data domain. It provides a framework which allows a new way of querying (enriched querying) to different biodiversity data providers. We propose that the common access prototype system should act like a middleware in the process of query and integration for XML-based data through semi-automatic structural and semantic schema matching to achieve interoperability between the data providers. The BUFFIE project particularly aims at solving a real world problem existing in the biodiversity domain and is explained in the following chapters.

CHAPTER 4

The System Design and Framework Model

This chapter introduces the project BUFFIE (Biodiversity Users Flexible Framework for Interoperability Experiments). Starting with a brief introduction to the motivation for this research, and following the discussion on various software engineering technologies and architectures from chapter 3 here we present relevant communication protocols used in biodiversity projects that deals with interoperability issues. Then we describe the overview of our approach and the heterogeneity issues in the BUFFIE system and present an example of the species data unit which is used throughout the thesis to evaluate, how the interoperability is accomplished.

4.1 Introduction

Interoperability of autonomous and heterogeneous data resources in biodiversity has been pioneered by organisations like GBIF, TDWG, ENBI etc... Data providers and users of the data are increasingly coordinating together to adopt a particular standard of data model representation and data communication protocols. In the context of this research domain, we define data interoperability as the ability to correctly interpret the data across biodiversity data providers across organisational boundaries. Independent data providers might use different data model to represent the biodiversity data. Unless a structural and semantic match is established between the concepts used in various data model, data interoperability cannot be achieved. This semantic knowledge is derived from the knowledge of expert biologist and developers of the domain. Interoperable systems that provide a common access interface for the users, apply the data integration technique to the various formatted data received from heterogeneous databases. A Common Access System has to insulate the application's end-users from the knowledge of the data structures and its different implementation across the varied databases. The standard data integration process would use a mediated schema and mapping rules which define the relationships of the concepts in data sources to the mediated schema. The general approach to design and develop a common access system follows two stages:

1. Based on the user query, the appropriate set of data resources are selected and generate the queries for each data resource.
2. Receive the response from the multiple data resources and perform necessary translation, filtering, merge the data and present the final answer to the user.

Typically, Common access systems for biodiversity information provide support for queries against a set of databases that adheres to exchange data using a particular protocol and data standard. The predominant biodiversity standards like Darwin Core and ABCD are used as federated schemas for databases that store occurrence records data. An occurrence record is data about observation of living beings that includes data on a species using taxonomical classification, location where the species were

observed or collected, by whom, when and how. Common access systems in biodiversity database networks use these federated schema standards and, in some cases, implement the DiGIR architecture for querying request and receiving responses. DiGIR (Distributed Generic Information Retrieval) aims at developing and testing a protocol for single point access to distributed data sources. DiGIR is an XML-based protocol with configurable federated schemas to support distributed data retrieval across one or more federation(s) of biological collection databases [143]. DiGIR was a project of the University of Kansas Natural History Museum and Biodiversity Research Center, California Academy of Sciences, and Museum of Vertebrate Zoology in Berkeley. DiGIR followed an open-source development using open standards and protocols like HTTP, XML, and UDDI. DiGIR has been adopted by several distributed networks, including GBIF, MaNIS, OBIS, and speciesLink, but its original inability to work with a completely independent XML-federated schema (e.g., ABCD) has led to a derivation of the protocol.

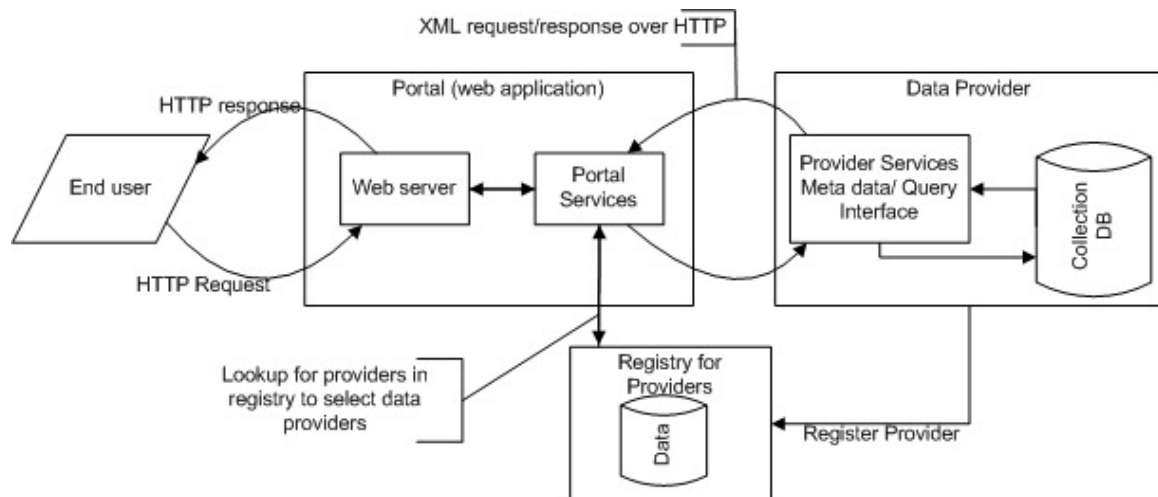


Figure 4.1: DiGIR Architecture.
(Source of Information sourceforge.net website and ENBI)

A survey of the biodiversity systems as discussed in chapters 2 and 3 would evidence the technologies like knowledge base, common data standards, shared ontologies, wrappers and web services as viable technologies or tools that can be used in unison to achieve interoperability among heterogeneous distributed biodiversity data sources. For example, common data standards using XML for data representation, tools like

ontologies and knowledge base to capture knowledge or information about the biodiversity data and configuration details of the system and architecture types like web based services were incorporated to develop our interoperable framework which is explained in the chapters 4, 5 and 6. Our specific objective is to test interoperability between federated XML schemas (e.g. Darwin Core, ABCD) by developing a prototype system using the BUFFIE framework and test some real data to prove that the interoperability can be achieved among networks using different federated XML schemas.

4.2 An overview of our approach

In general, developing common data access systems that should interoperate across distributed heterogeneous database systems requires addressing several complex issues of data matching and messaging processes involved. Currently in the biodiversity domain there is no common access system that can automatically interoperate with different types of network standards for e.g. DarwincoreV2, ABCD, and speciesCDM etc.

The novelty in our approach is in providing a flexible framework using software engineering techniques in contrast to building a universal global schema, thereby allowing any data providers to interoperate through a common access system irrespective of the data exchange standard they use. The synergies of web based Service oriented architecture, Domain Knowledgebase implemented using XML and XSLT and Object and Functional design of the Framework's Business rules are applied in this framework. The BUFFIE Framework derives interoperability from the heterogeneous and distributed data bases by using a web service oriented architecture, Knowledge of the domain expressed in XML/ XSLT (K) and Business logic (P) designed using J2EE and .NET which object and functional design. Figure4.2 shows a schematic representation of Interoperability in our approach.

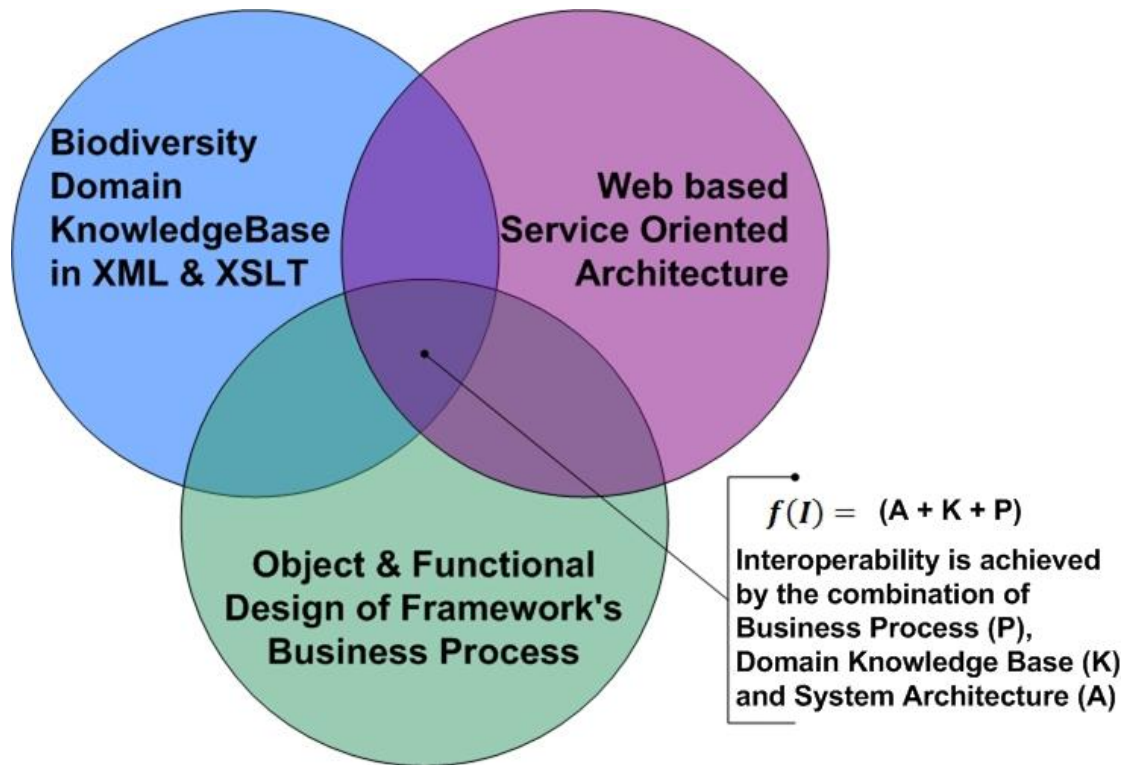


Figure 4.2: Schematic representation of Interoperability in BUFFIE.

In this thesis, we concentrate on the following two goals:

1. Establishing a novel framework (BUFFIE) that is practically useful and extensible for the biodiversity data providers and users to interoperate irrespective of the data network standards they are using.
2. Enriching the user queries to suit the data resources and integrating the data providers' responses using the XSLT templates.

Also we are restricting our scope of the implementation for this framework, to interoperate between databases that store biodiversity occurrence records and implement data communication using XML document structures. The common access system will provide the user with an integrated view over heterogeneous, distributed data sources that use XML documents for data exchange; such an integrated view will be best represented by XML because of the advantages of XML as an exchange model, such as rich expressiveness, clear notation and extensibility. The system will enable users to query its data sources using a tailor-made request messages based on the communication requirements of the provider. Due to the inherent nature of

biodiversity data there is never going to be a fully automatic approach to the problem of semantic data integration. We should be able to achieve a high degree of automation, which requires the expression of shared knowledge with some human intervention by using semantic mapping. Our approach recognises that there are significant differences in the XML messaging standards in biodiversity domain due to the semantics and syntax of data elements. We propose the application of "integration on Demand" [144] as a complement to "integration in advance" interoperable methodology.

To achieve integration we use a mechanism that expresses the relationships of the schema elements as a table of mappings (Appendix A). These mappings are produced by the biodiversity domain experts which is an important tool that helps in integrating the heterogeneous data from the providers. The data integration architecture followed was based on the mediator architecture. The system prototype is called BUFFIE (Biodiversity Users' Flexible Framework for Interoperability Experiments). It requires the effort of a computer developer to generate two set of knowledge base for the common access system. First is that BUFFIE should have knowledge about the data providers and their communication protocols. For example HTTP protocol or web services call for sending request and response to data sources. And second is that how the data concepts in a data model (elements in a XML schema) of a particular data provider compare with the data concepts of the other data models in the domain. This is known as concept mapping knowledge about the domain data concepts which could be produced from concept mapping tables published by biologists for the various XML standards. The biodiversity domain Knowledge base in Buffie (DKB) is like a XML metadata repository which needs to be maintained by continuous updating to assist the BUFFIE in query generation and messaging of the requests. The following steps describe a use case of how the BUFFIE system works for query processing and response integration:

1. The User is logged into the JSP/ASP.NET web application using an authentication system.
2. Users are presented with a query design page where they can select the search concept and search value.

3. User's query is enriched using AJAX to web services technology like finding the synonyms or the accepted names for the given search value.
4. The BUFFIE system will then consult the knowledgebase and select the list of providers suitable for this query, and the communication protocol format, data access point and other information required for messaging.
5. Based on the user's selection and the provider's information from the knowledgebase the system will create tailor made request messages to each provider according to their communication protocol.
6. This query is sent over to the data providers, over asynchronous threads.
7. The responses received from the data provider are validated and merged using the schema matching templates.
8. The integrated results are then transformed and presented to the users.
9. Alternatively the final integrated results are sent as a response to the web service clients if the query was initiated as a web service call.

The data flow of the application involves two main stages that are, enriching and generating multiple queries and response data integration and refining.

4.2.1 Abstraction of Problem Domain

This section lists the main abstract requirements of the data domains in, which this approach can be implemented to achieve integration and data interoperability.

- The data providers are independent and autonomous but should be willing to provide the data to the users through BUFFIE framework.
- Data providers should have web service interface for communication with BUFFIE framework through XML messages.
- Data providers should be part of an existing community that adopts one of the established XML data standard protocols of the data domain, otherwise they have to provide their data standards mapping to the BUFFIE framework.
- This approach would very much suit the data domain where the numbers of XML protocols are limited. If the number of the XML standards is larger then this problem can be resolved by using a central schema for routing the transformation.

- This approach also requires the existence of knowledge base that has information about the data providers, about their connection properties and the XML standards and transformation rule-sets. When these are not available a developer is required to capture the knowledge from the domain experts and design the different modules to implement the BUFFIE framework.

4.3 System Design of Prototype

The BUFFIE prototype is a new interoperable common access system that is developed to test the research idea of combining the software engineering technologies, architecture and domain knowledge base as described in section 4.2 and assumptions like the availability of biodiversity data providers' network that use XML data standards for communication. The objective of the system design is to deliver a system that is practical in querying multiple, heterogeneous and distributed data providers in an efficient way. The system was designed, so that the resultant application is extensible to accommodate future XML standards and interoperable capabilities. This application system allows data providers with different network standards to interoperate during a single querying process. For example a client should be able to access data from a DWCV2 data provider and an ABCD data provider and any other provider with a proprietary communication protocol simultaneously through a common access system. This common access system needs to have information about all the communication protocols that it deals with. The prediction of interoperability problems among the interacting components and the XML data standards are analyzed to make effective design decisions as well as which architecture to consider for development. The BUFFIE system design is based on a service oriented, web based n-tier architecture model, which includes presentation layer, business logic layer and data provider layer. Figure4.3 shows the logical design of various components in BUFFIE.

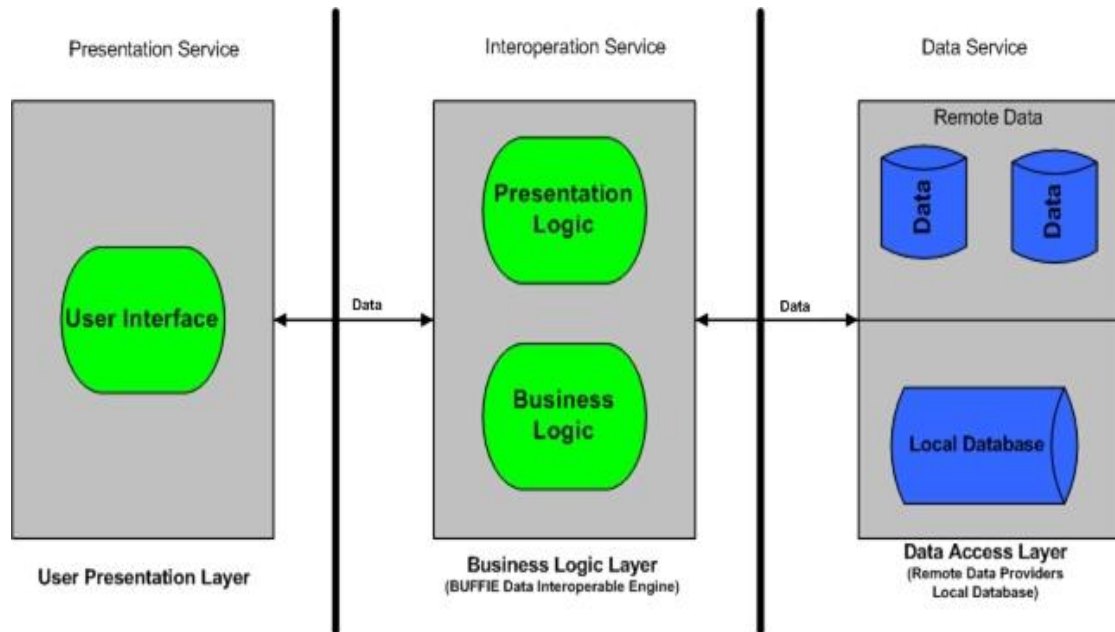


Figure 4.3: Conceptual Design of the BUFFIE Common Access System.

The *control topology* of the BUFFIE system determines the arrangement of the components according to the desired control interactions that can affect the complexity of the interactions among them. We have designed the components to be exposed as services, forming three logical layers that interact with XML messages. *Data topology* is the arrangement of the components in BUFFIE according to their required data interactions. The topology of the components can directly affect the transformation of the data for interchange [145]. For closely coupled components the data are encapsulated in objects and for other cases serialization of object data is used for data interactions.

The data service logical layer comprises of the local database of BUFFIE system and remote data providers which are autonomous, heterogeneous and distributed. Responses from the remote data providers are not controlled by BUFFIE system, due to this nature of data providers an asynchronous style of communication is preferred in the system design of BUFFIE. Two types of users are identified for the BUFFIE system; the common users who search for biodiversity information and institution users (researchers) who would like to harvest biodiversity information from heterogeneous data providers. The system provides a Web form interface and Web service interface for the users. The input for the system is either the form based data through web form or XML messages sent across web service methods, and similarly

the output of the system is also presented as html in a web page or XML messages to the client application. The initial version of this application has been developed on a java platform and released for the real-world users to test the data interoperation. The final version is to be developed on a Microsoft windows based platform with visual studio2008 and SQL server2008, both version will demonstrate the interoperation of biodiversity XML standards. The application components are designed such that the framework and its components could be used for different data domain with slight modifications to the application configuration settings about communication protocols, data sources information and with the inclusion of adequate knowledgebase about the data domain. Security modules were not implemented in the prototype though; it was included in the system design which might be useful for future live deployment of application.

4.3.1 Requirements of the Prototype System

The requirement of the prototype system is to demonstrate how interoperability has been achieved as a result of this application using a sample biodiversity species data modelled in XML data structure. For the BUFFIE system the interoperability requirement is to access data resources from multiple biodiversity networks like DWCV2 and ABCD and to present this data according to the users' preferences. The interoperability requirement for BUFFIE Framework involves the following:

- Deciding which XML data standards has to be incorporated to show the interoperation. For example like choosing data providers that use Darwin Core, ABCD (biocase) and SpeciesCDM data standards.
- Decide what would constitute interoperability; decide what level of interoperability would have to be achieved. For example, from single user query the system should generate multiple queries to heterogeneous providers and integrate all the valid responses using schema matching templates and produce the results.

- Perform testing by making measurements for interoperability. Evaluation metrics to decide whether we have achieved interoperability and the confidence in the result. For example user acceptance and implementation of BUFFIE system by the client programs, testing the results.

4.4 Heterogeneity Issues in the BUFFIE System

This research addresses two specific types of heterogeneity in the representation of the real world biodiversity data that is modelled using XML data structure on different databases of the data providers. The first type is concerned with the differences in the information represented by each XML biodiversity data standard. This is termed as “*heterogeneity of scope*”, which refers to the fact that differing amounts and types of information are represented by various data standards to express the species information. For example, in the BUFFIE system the data providers are autonomous and may have different data models (XML schema) to represent biodiversity data. Because independent development teams create these databases at the data providers’ end, each provider might adhere to a different XML data standard to capture the species information. Figure 4.4 show each provider uses different XML data communication standards. For instance Provider A and C uses Darwin Core XML standards, provider B uses ABCD data standard and provider D uses Species CDM data standard. Though all these data standards capture the core information about the species, ABCD standard allows a bigger scope to capture extra information about the species when compared to the Darwin Core and its variations. When the users model the data using one of the XML standards there could be differences in representing the aspects of the species data. These differences in the state and behaviour of the entities used in XML standards for species information can be thought as providing different views of the same species information.

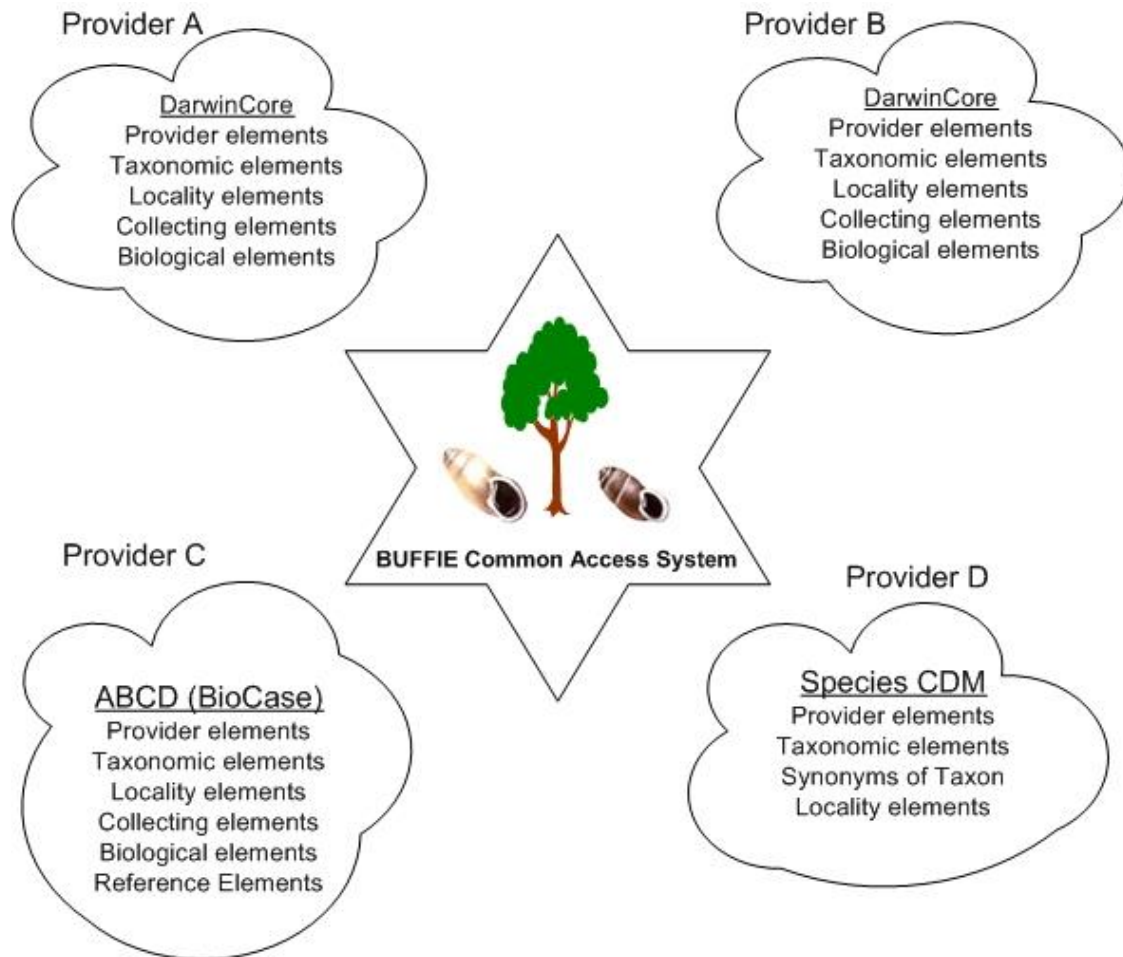


Figure 4.4: Differing Scope of Biodiversity Data in Communication Standards.

Even if more than one system provides the same view of the species that is being modelled using XML data standard, there may still be differences in the representation of that information across different standards. This type of “*heterogeneity of representation*” refers to the differences in the concept terminology used, format, accuracy, range of values allowed and structural representation of the included state and behavioural information. This difference in representation is illustrated in Figure 4.5 by providers A and C. Even though these standards (Darwin Core and ABCD) both represent the same real world biodiversity data, i.e. both capture species information under various categories like provider, taxonomic, locality, collecting, biological concepts; they each represent the information comprising that view in a different manner. For e.g. Darwin Core represents the collection time as separate elements in year, month, day, time whereas ABCD standard collects the same information in one element as “ISODateTime”. Another

example is for provider A and B collect the location information in latitudes and longitude coordinates but the range of accuracy varies among them, but provider D represents the same information in place names.

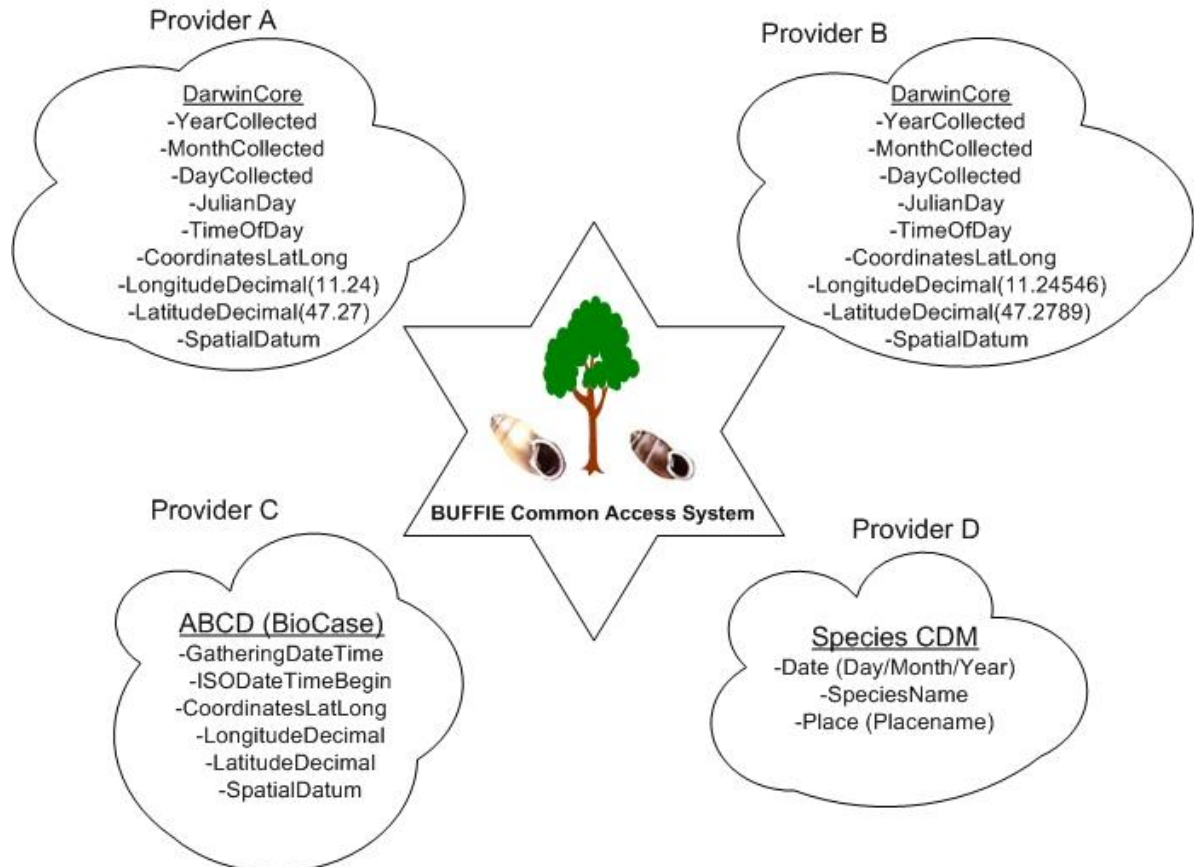


Figure 4.5: Differing Views of representation of Data.

4.5 Use-Case of Interoperability in the BUFFIE Application

To elucidate our interoperable approach for the XML data standards in biodiversity networks, this example data is discussed here. In the BUFFIE common access system most data providers either use Darwin Core or ABCD (BioCASE) data format for data communication. Both these standards represent the biodiversity information based on a species, but use different XML schemas to structure their data. For the same query about a particular species, these two standards have two different structures of request formats in XML message. For example let us discuss a query created from user to search for taxon information of a species scientific name known as “Acicula Lineata” (snail, Gastropod). This section will discuss the formats of two different request

messages and the corresponding two different responses from the providers in BUFFIE system.

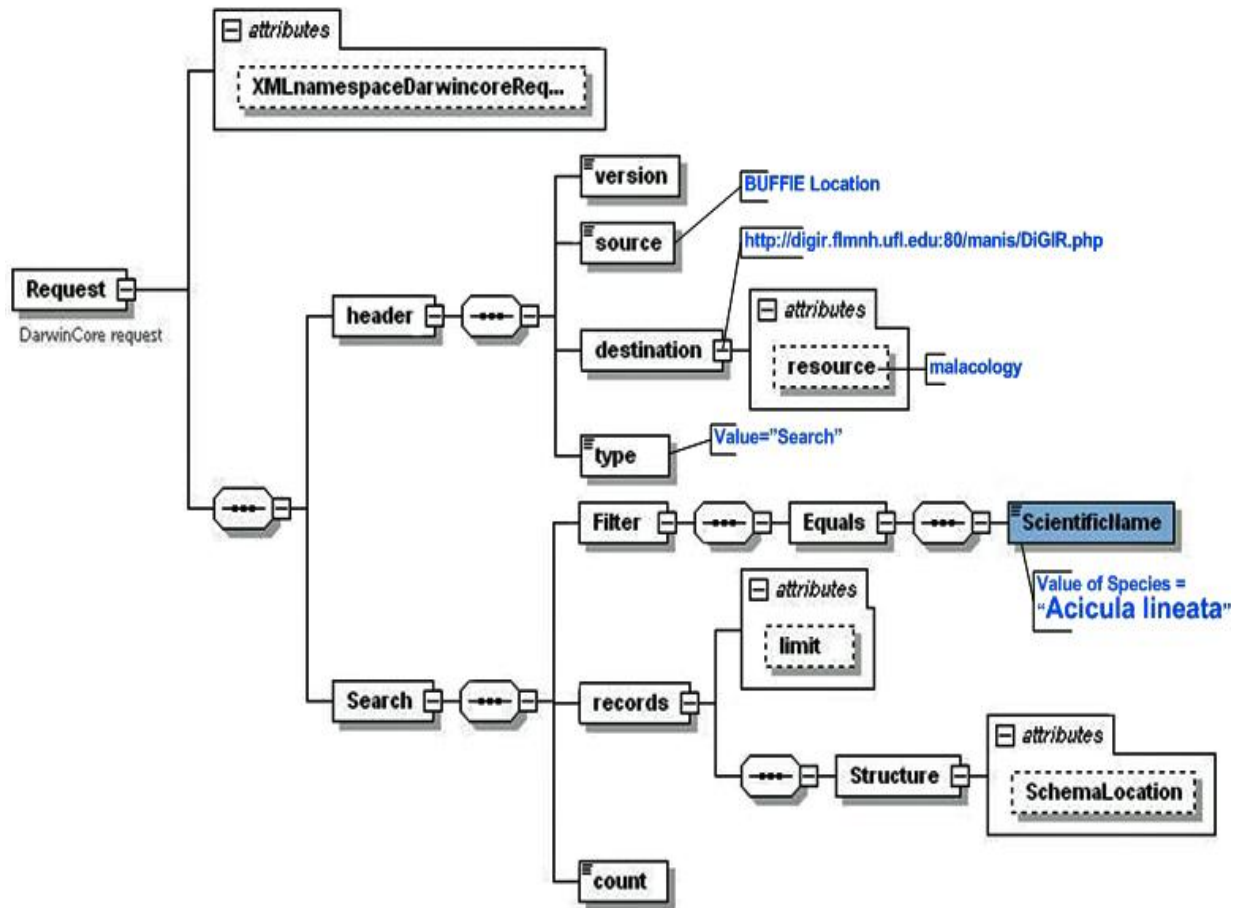


Figure 4.6: Request message structure for Darwin Core standard Provider.

BUFFIE generates the search request for species “Acicula Lineata” for Darwin Core providers based on the schema model as shown in figure 4.6 and similarly for ABCD providers as shown in Figure 4.7. These request messages are generated using the XSLT templates derived from the domain knowledgebase in BUFFIE that provides the knowledge about the data providers. The framework also provides a feature to limit or enhance the number of providers selected for the querying process. All the request messages generated by BUFFIE use following components:

- An attribute that qualifies the XML elements using a protocol specific XML namespace.
- Header part with the source and destination information along with the resource name (e.g. Malacology & ZOBODAT) and type of search.

- A search part which has the species scientific name (Acicula Lineata) that is to be searched.

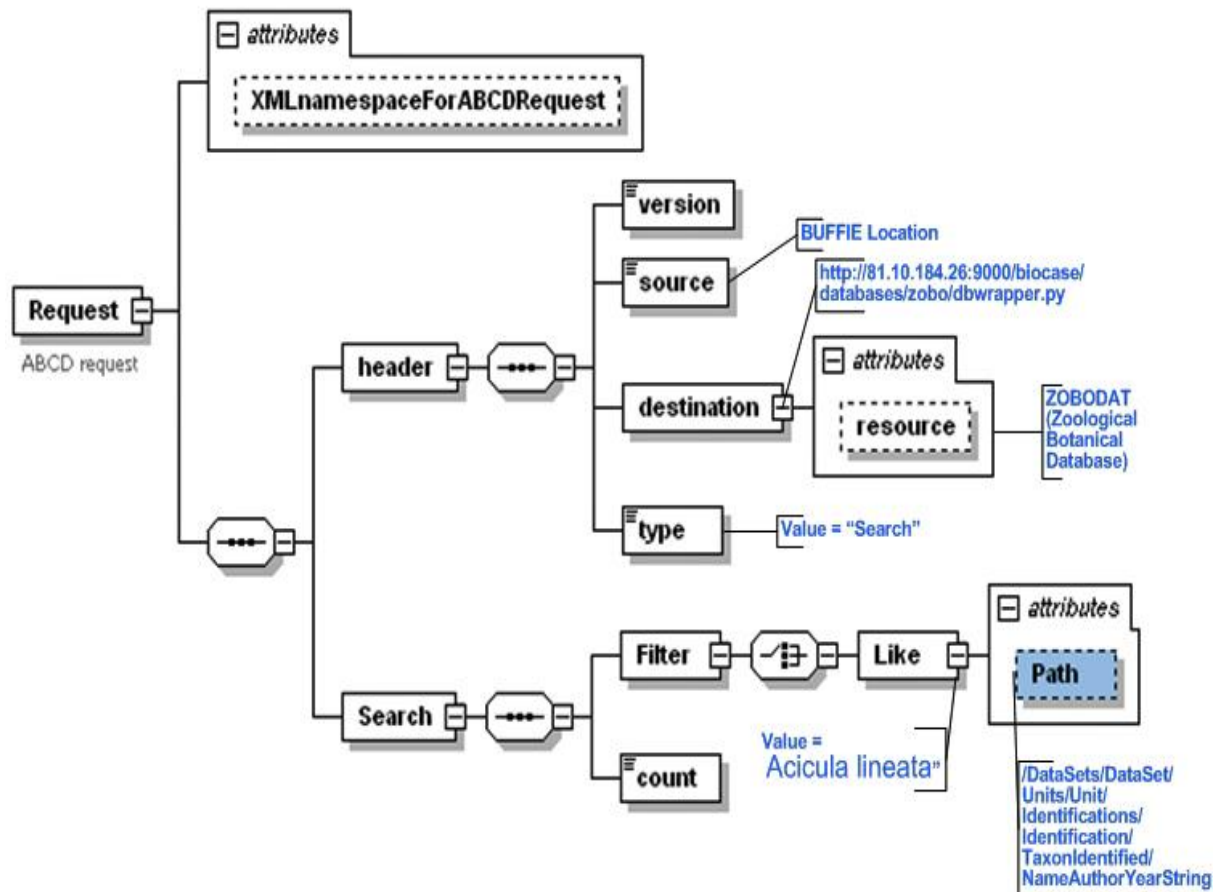


Figure 4.7: Request message for ABCD standard Provider.

All the biodiversity concepts described in XML standards used by the providers in BUFFIE system (like Darwin Core & ABCD) are integrated using XSLT templates. The integration logic implemented in this research work, as explained in section 6.5.2 were created by us after gaining the knowledge on biodiversity concepts from biologists and the concept mapping tables published by the domain experts as shown in Appendix A. Our objective in designing the framework is to create the ability to send asynchronous queries to the heterogeneous data providers and integrate the responses using the knowledge described using XSLT templates. The response message format from a Darwin Core provider for the species scientific name “Acicula Lineata” is shown in Figure 4.8, which has “m” number of records. The complete set of concepts in the Darwin Core XML structure is also included in appendix A. Figure 4.9 shows the responses from ABCD provider (ZOBODAT) that has “n” number of

records relating to species “Acicula Lineata” and for a full set of the XML structure please refer to images in section 7.6. These responses are received asynchronously and BUFFIE uses XSLT templates with integration logic and produces “m + n” number of records that are displayed to the end user of the system.

response	
header	
content	
xmlns:darwin	http://digir.net/schema/conceptual/darwin/2003/1.0
darwin:InstitutionCode	FLMNH
record ("M" number of records)	
darwin:CollectionCode	Invertebrate Zoology
darwin:CatalogNumberText	111354-Mollusca
darwin:ScientificName	Acicula lineata
darwin:TypeStatus	xsi:nil=true
darwin:Sex	xsi:nil=true
darwin:AgeClass	xsi:nil=true
darwin:Preparations	xsi:nil=true
darwin:Tissues	xsi:nil=true
darwin:Country	United Kingdom
darwin:StateProvince	England
darwin:County	Cumbria Co.
darwin:Island	xsi:nil=true
darwin:IslandGroup	xsi:nil=true
darwin:ContinentOcean	xsi:nil=true
darwin:Locality	ZZZ-038385
darwin:YearCollected	xsi:nil=true
darwin:MonthCollected	xsi:nil=true
darwin:DayCollected	xsi:nil=true
darwin:TimeCollected	xsi:nil=true
darwin:VerbatimCollectingDate	xsi:nil=true
darwin:Collector	xsi:nil=true
darwin:CollectorNumber	xsi:nil=true
darwin:FieldNumber	xsi:nil=true
darwin:FieldNotes	xsi:nil=true
darwin:JulianDay	xsi:nil=true
darwin:DecimalLatitude	xsi:nil=true
darwin:DecimalLongitude	xsi:nil=true
darwin:HorizontalDatum	xsi:nil=true
darwin:CoordinateUncertaintyInMeters	xsi:nil=true
darwin:OriginalCoordinateSystem	xsi:nil=true
darwin:VerbatimLatitude	xsi:nil=true
darwin:VerbatimLongitude	xsi:nil=true
darwin:MinimumElevationInMeters	xsi:nil=true
darwin:MaximumElevationInMeters	xsi:nil=true
darwin:VerbatimElevation	xsi:nil=true
darwin:MinimumDepthInMeters	xsi:nil=true
darwin:MaximumDepthInMeters	xsi:nil=true
darwin:VerbatimDepth	xsi:nil=true
darwin:Remarks	xsi:nil=true
darwin:Kingdom	xsi:nil=true
darwin:Phylum	xsi:nil=true
darwin:Class	Gastropoda
darwin:Order	Stylommatophora
darwin:Family	Ferussaciidae
darwin:Genus	Acicula
darwin:Species	lineata
darwin:Subspecies	xsi:nil=true
darwin:ScientificNameAuthor	Draparnaud
darwin:IdentificationModifier	xsi:nil=true
darwin:IdentifiedBy	xsi:nil=true
darwin:YearIdentified	xsi:nil=true
darwin:MonthIdentified	xsi:nil=true
darwin:DayIdentified	xsi:nil=true
darwin:IndividualCount	xsi:nil=true
darwin:GenBankNum	xsi:nil=true
darwin:OtherCatalogNumbers	xsi:nil=true
darwin:RelatedCatalogedItems	xsi:nil=true
darwin:CatalogNumberNumeric	xsi:nil=true
darwin:BasisOfRecord	xsi:nil=true
darwin:DateLastModified	

Figure 4.8: Response for Species named “Acicula lineata” from Darwin Core Provider.

The XSLT templates have the knowledge and logic to generate the resultant values from each schema element of various responses. For e.g. the Darwin Core has an element for the country name as:

`<darwin:Country>United Kingdom</darwin:Country>`

Whereas the ABCD response has the same information as follows:

```
<Country><ISO3Letter>AUT</ISO3Letter></Country>
```

These two elements have different element structure and values but refer to the same concept which is “country”, where the species is identified. Hence to integrate these values, the XSLT template use ISO country code lookups built into XSLT templates.

The screenshot displays an XML tree structure for an ABCD response. The root element is 'response', which contains a 'header' and 'content'. The 'content' element contains a 'DataSets' element, which in turn contains a 'DataSet' element. The 'DataSet' element contains a 'Units' element, which contains a 'Unit' element. The 'Unit' element contains an 'Identifications' element, which contains a 'TaxonIdentified' element. The 'TaxonIdentified' element contains a 'NameAuthorYearString' element with the value 'Acicula lineata Draparnaud'. The 'NameAuthorYearString' element contains a 'ScientificNameAtomized' element, which contains a 'Zoological' element. The 'Zoological' element contains 'Genus' (Acicula), 'SpeciesEpithet' (lineata), and 'Author Team Ori...' (Draparnaud). The 'Unit' element also contains a 'Gathering' element, which contains 'GatheringDateTime' (1962-05-01), 'GatheringSite' (Döbraschlucht N Pressegger S.), 'Country' (AUT), 'NamedAreas', 'SiteCoordinateSets', 'SiteCoordinates', 'CoordinatesLatLong' (LongitudeDecimal: 13.42, LatitudeDecimal: 46.64, SpatialDatum: WGS84, CoordinateErrorDistanceInMeters: 500), 'Altitude', and 'MeasurementAtomized' (MeasurementLowerValue: 900).

Figure 4.9: Response for “*Acicula lineata*” from ABCD Provider ZOBODAT.

Similarly the XSLT templates apply techniques like aggregation, atomizing, concatenation and substitution functions on biodiversity concepts and values present in the data structure to perform the data integration. A more detailed analysis of the integration process and examples are discussed in the following chapters through to the evaluation chapter.

CHAPTER 5

BUFFIE Architecture and Operation

5.1 Introduction

Software architecture is the description of the computational components of a program or system, the connectors that establish the interactions between the components and data, as well as principles and guidelines governing their design and evolution over time in order to achieve a desired set of architectural properties [146]. The fundamental characteristics of the architectures of the interacting components and connectors, data standards contribute to the architecture interoperability [147]. The integration strategy of this research is formulated by analysing the conflicts of components and biodiversity data characteristic values. This chapter describes the functionalities and the processing of the query request generated for multiple heterogeneous providers in the following steps:

- Architecture for interoperability.
- Query generation for multiple heterogeneous data providers
- Enriching the user query
- Heterogeneous issues resolved in this framework.

5.2 System Architecture for Interoperability in Biodiversity Networks

Application architecture is the process of defining a structured solution that meets all of the technical and operational requirements, while optimizing common quality attributes such as performance, security, and manageability [148]. Interoperation requires knowledge and intelligence as it distinguishes from the ordinary integration of data (which is usually syntactic) and databases and hence the proposed system architecture needs to accommodate these entities using a knowledge base. The novelty in this research is that it provides a new framework that helps in achieving interoperability among biodiversity data resources irrespective of the communication protocol and XML data schemas used by the data resources. Moreover our approach is designing an extensible framework rather than developing a universal schema for interoperation. This facilitates in non-intrusive future plug-ins and extensions. Previous works in this domain have resolved interoperation of heterogeneous and distributed database that adheres to a specific protocol (data schema) among the network. One of the objectives of this architecture is to maintain the data definition autonomy of the data providers' databases at the logical level and physical level.

The advance in our approach is achieving the interoperation of heterogeneous data resources by applying the Multi Layered, Web based Service oriented architecture and designing the business logic using Java and .NET components that use the knowledge of the data domain expressed in object oriented and functional programming components. Based on this, we mainly concentrate on realizing the network interoperability using the web service architecture and data interoperability using the LINQ to XML and XSLT components of the business layer.

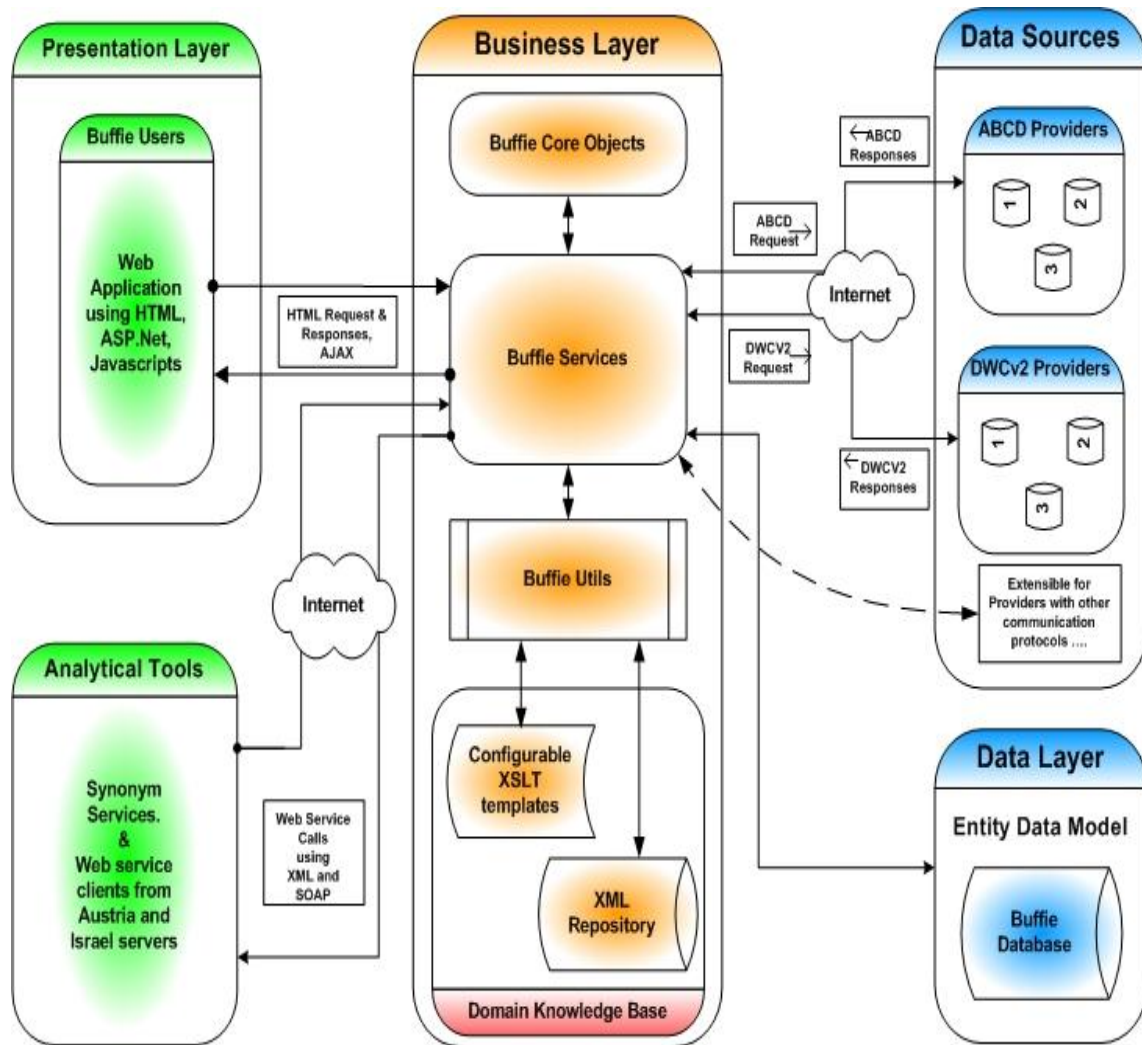


Figure 5.1: Multi Layered, Web based Service oriented architecture.

The architecture provides a complete insulation for the biodiversity data user from the biodiversity data resources, for example the user need not know about the likes of the data format and accessing methods of the data providers. The BUFFIE Common Access System (BCAS) takes care of soliciting the user query and validating the data entered through its user interface. This presentation layer is designed as a web application accessed through internet browser. Complementing this layer, this architecture provides another provision for the client programs to access the *BuffieServices* directly through a web service interface. This feature of BUFFIE enables biodiversity data harvesting [21] and query enriching. Figure 5.1 shows the overall architecture for the BUFFIE 2.0 framework, relationships and data flow of the various components needed to develop the common access application for the biodiversity data interoperability problem. The business layer components consist of *BuffieCore Objects*, *BuffieServices*, *BuffieUtils* and *DomainKnowledgeBase*. These

components are used in combination to perform multiple business operations. The whole process of the system can be described in three main sub-divisions (based on functionalities) as follows:

1. **Common Access Framework:** This deals with the orchestration of the set of components and interfaces that are spread across object model, functional model and Domain model. The object model is a software engineering technique that follows the system design and architecture and is intended to provide a structural view of the whole system, including a functional description of the entities, their relationships, and their responsibilities. The object model provides the advantages of data abstraction, encapsulation and inheritance while developing the system. The Domain model is an object model designed by looking at a particular problem's domain and tries to abstract processes and data flows in terms of those entities and relationships. Here the problem domain is the interoperability framework which is different from the research domain, which is biodiversity informatics. Hence the domain model refers to the classes that are needed to perform the interoperability requirement. The functional model represents the methods of the system from the perspective of data flow and transformation of data.

BuffieCore Objects (BCO) is intended to provide a structural view of the middleware system that is independent of the biodiversity data domain. It describes the functional description of the entities, their relationships and their responsibilities [149]. These are mainly interrelated classes that are abstracted from the data domain and are used for the operation of the framework in general for example to pass data between the components.

BuffieServices and *BuffieUtils* are based on a functional model which is designed to accomplish a specific task for a given set of arguments. The functional objects exposed in these components do not rely on any external state and emphasizes the application of functions on the objects. *BuffieEngine* performs the orchestrated workflow of combining these functions and objects to perform query processing.

Domain Knowledge Base (DKB) consists of *XML Repository* and *XSLT templates* which are based on a domain model. These components are designed to meet a given set of requirements to address a particular problem's domain. In our research, these are the domain entities like biodiversity concepts captured in the configurable XSLT templates. The XML repository stores the metadata for the data resources and the schemas for the communication protocol. The configuration files fetches value for the parameters that govern the business workflow of the *BuffieServices*.

2. **Query Processing:** When the user or the client program submits a query, The BUFFIE application triggers the query generation process. The communication between the presentation layer, analytical tools and data providers with the *BuffieServices* is based on the HTTP and SOAP protocol over the internet. The query process involves identifying the concept for which the query is submitted and then uses AJAX and web services technology [150] to communicate with synonymy servers for enriching the query. Sections 5.4 and 5.5 explain more about query enriching process. The *Domain Knowledge Base (DKB)* provides the knowledge for the query generation as:

- how many queries are to be generated
- what format these queries are to be structured
- Destination of data providers (example Access points, connection details)

Given the knowledge about the data providers and the user requests the BUFFIE system generates the tailor made queries for each data provider and sends them as request messages asynchronously across the internet. The *BuffieEngine* object of *BuffieServices* component uses multithreaded event-based asynchronous pattern [149] for sending the request to various data provider. The responses from the providers are received and transformed as per the requirement of the initial query by the *BuffieServices*. In BUFFIE system design and architecture, the Business logic layer (BLL) and the data resources are very loosely coupled and communicate using XML based data structure through the Internet using HTTP protocol. The BLL has to send and receive query request to the data resources at the same time. Hence Multithreaded event based asynchronous pattern is used here. This improves

the performance and provides concurrent communication with the data sources. For example, if one data resource is failing in communication it should not affect the communication between BLL and other data sources.

3. **Data Persistence and Mapping:** The data providers of the BUFFIE system are independent, autonomous and remotely distributed. The remote data providers and the local *Buffie database* together comprise the data layer of the architecture. The Buffie database uses Microsoft's Entity Data Model (EDM) [151] for data transfer between the data layer and Business layer. EDM provides a conceptual model that accurately reflects common business objects from the physical structure of relational tables. This allows the developers to define flexible mapping to relational data. This mapping helps to isolate BUFFIE from changes in the underlying storage schema. The Entity Framework also contains support for Language Integrated Query (LINQ) to Entities, which provides LINQ support for business objects exposed through the Entity Framework. The response sent by the providers of biodiversity data are in a heterogeneous format and this architecture provides the feature to persist the entire user query, request messages and providers responses in *BuffieDatabase* using the EDM. Figure 5.2 shows the architecture of XML document mapping process.

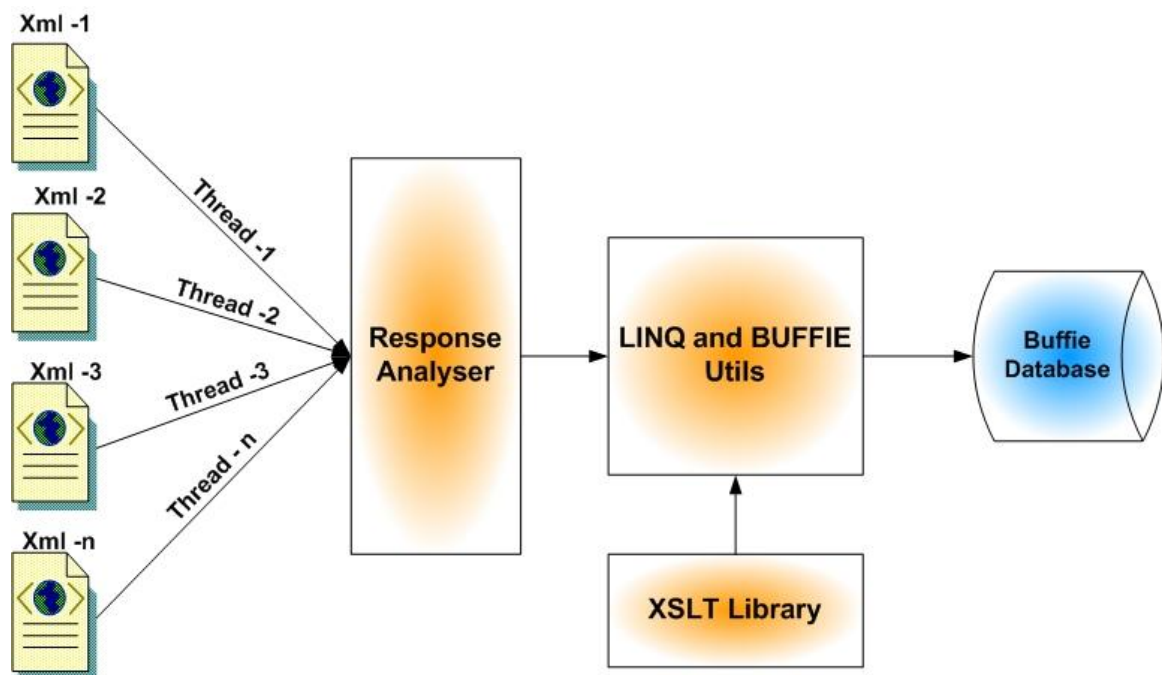


Figure 5.2: Architecture of XML data mapping process.

This deals with integrating the data schemas of the response from the providers as required by the Client/User. This mapping strategy is formulated by analysing the communication patterns and the data schemas of the individual data providers of the system by *Response Analyser*. This performs a meta-data analysis and validates the response XML messages and passes the result to the *BuffieServices*. The LINQ components and *BuffieUtils* loop through the response and perform the data transformation. The XSLT templates library from the *Domain Knowledge Base* provides the correct templates for each XML document to carry out the data transformation. The result of all the transformations are gathered and stored in the *BuffieDatabase*.

This BUFFIE system architecture is very much optimised for the biodiversity data domain.

5.3 Generating the Queries for Heterogeneous Providers

The conceptual system design and architecture of the query generation process in BUFFIE is explained in this section. To answer the users' query effectively it is important to identify the users requirement and also to find the source of information to satisfy that user requirement. BUFFIE framework is built with the biodiversity domain knowledge about the data resources to accomplish this task. The *Query Analyzer* (QA) functions are analogous to a prism which produces multiple outputs from a single input.

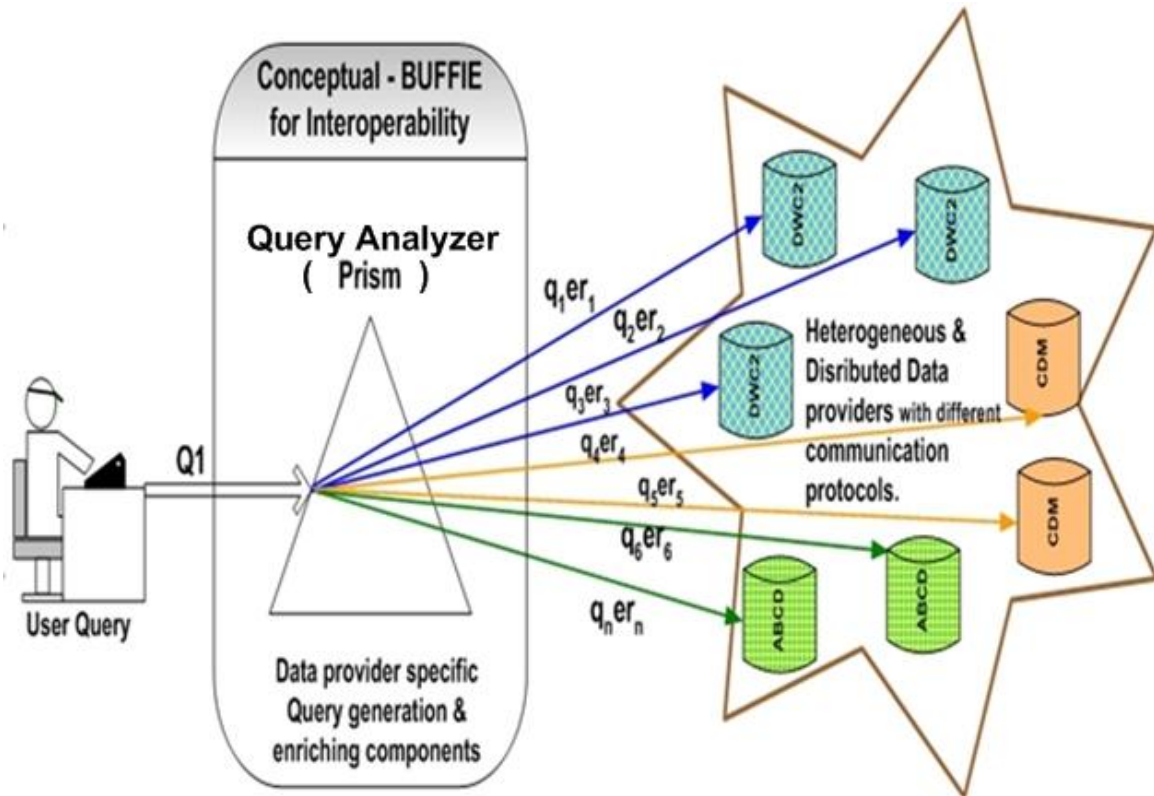


Figure 5.3: Conceptual view of Query Generation in BUFFIE.

It receives the query information through a web page or web-service Interface. This query will be in a text format and contains information such as, type of communication protocol, concept names and values to form the query and other security parameters. BUFFIE validates the user's security parameters and when the authentication is successful all user queries and schema definitions pass through to *Query Analyzer* (QA) module for onward processing. The QA records the query information in the local *BuffieDatabase* and invokes the Query enriching service and also collects the information on Providers from the Domain Knowledge Base (DKB). BUFFIE creates multiple XML request messages for the user query as follows:

If DKB returns “n” data providers, each having a specific XML schema for data communication then:

Q is the result of the following loop:
 For each provider (p) in providers ($P_{[n]}$)
 {
 $Q_{[p]}$ is the result of transformation given by $UI * S_{[p]}$
 }
 }
 Where Q is the set of queries generated in Buffie,

p is a specific provider,
 $P_{[n]}$ is a list of providers in DKB,
 $S_{[p]}$ is an XML schema for a provider p ,
 ui is the user input in Buffie system,
 $Q_{[p]}$ is the resultant XML request message produced by transformation for a given provider p .

The following figure 5.4 shows an example of request query generated in XML message for a Darwin Core type data provider.

```

<request xmlns="http://digir.net/schema/protocol/2003/1.0"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:dwc="http://digir.net/schema/conceptual/darwin/2003/1.0"
xmlns:darwin="http://digir.net/schema/conceptual/darwin/2003/1.0"
xsi:schemaLocation="http://digir.sourceforge.net/schema/conceptual/darwin/2003/1.0/darwin2.xsd">
  <header>
    <version>1.0.0</version>
    <sendTime>06/07/2009 09:17:57</sendTime>
    <source>BuffieV2.0</source>
    <destination resource="SNAILS">http://ip**.***.uva.nl:8000/digir/DiGIR.php</destination>
    <type>search</type>
  </header>
  <search>
    <filter>
      <equals>
        <darwin:ScientificName>Acicula Lineata</darwin:ScientificName>
      </equals>
    </filter>
    <records limit="100" start="0">
      <structure schemaLocation="http://digir.sourceforge.net/schema/darwin2full.xsd" />
    </records>
    <count>true</count>
  </search>
</request>

```

Figure 5.4: Sample XML request message for data provider.

5.4 Enriching the User Query

All Common access system has to manage the problem of matching the query parameters with the information in heterogeneous data resources. For example the BUFFIE system has to retrieve the correct information for a query like, *Find species info which is commonly known as "Breadcrumb Sponge"?* Usually this problem is managed in the system architecture as follows [152]:

- At the data resources side, by indexing the data - known as index enrichment.
- At the moment of processing of the specific query - known as query enrichment.

Earlier researches in query enrichment have mainly focused on semantic query optimization, which uses the semantics or conceptual basis in database queries to reformulate a query more efficiently into a different but semantically equivalent form that returns correct answers [153]. In our research, BUFFIE the query enrichment is supported by looking at the concept values of the user query. In this context, we define the *query enrichment* as that, the query concept value is augmented with its extensions like synonyms thereby improving the quality of response from the heterogeneous and distributed biodiversity data resources. One of the unavoidable problems with taxonomy data is that different people will know the same species by different names [154]. This may be due to valid changes in the taxonomy through re-classification or simply that one biologist/database developer records a species by a different name for example its common name as opposed to its scientific name. In biodiversity domain species with multiple names like these are classified as accepted name, common name and synonyms. A classic example for a multiple species name for a same species is *Halichondria panicea* commonly known as the breadcrumb sponge which has been given 56 names in the scientific literature since it was first named in 1766, according to researchers compiling the census [155]. Among them:

- *Alcyonium manusdiaboli* (1794),
- *Spongia compacta* (1806),
- *Halichondria albescens* (1818) and
- *Seriatula seriata* (1826).

For example, if the Common access system is sending out a query request to find species info on “*Halichondria panacea*” and if the data providers have stored that species information, indexed on one of its other names as shown above like “*Seriatula seriata*” then there will be no valid response for the request query. To overcome this data invisibility problem BUFFIE framework is providing the query enrichment feature in its architecture, by looking at the value of the concepts and using the publicly accessible domain tools like SPICE checklists [156]. Figure 5.5 shows the search concept is semantically enriched with a generalization of the information provided in its value. The belief here is that the values associated with a concept possess a knowledge source and using that, the querying power of the concept should be augmented with its value’s extensions [157].

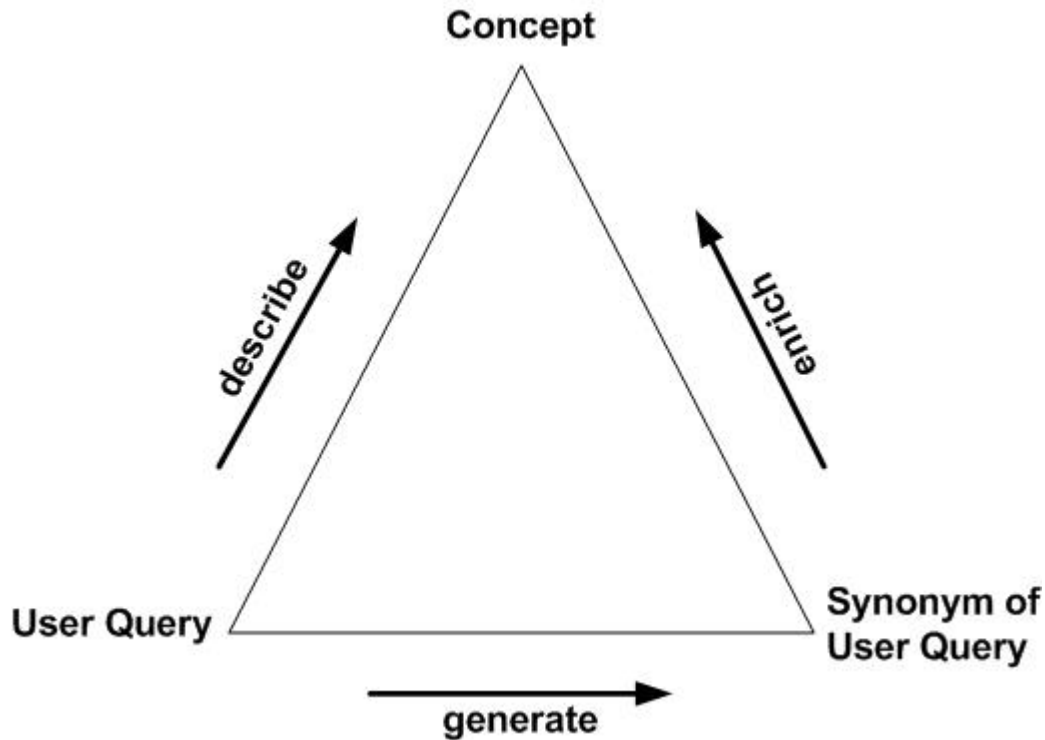


Figure 5.5: Enrich a concept by the generalization of its values.

Query Enriching Logic:-

Let Q is the main query.

E is a set of query concepts. (E refers to all searchable concepts that can be used in a user query)

$E(x)$ is concept x in E . ($E(x)$ could refer to a particular search concept used in a user query namely “ScientificName”.

$X[i]$ is the value of the concept x . ($X[i]$ could refer to a particular value of the search concept for example “Seriata seriata”)

$\{S\}$ set of a name value collection of enrichment services. (Synonym services)

For Each Synonym Service in $\{S\}$

```
{
  Process  $X[i]$  and return  $\{Ev\}$  the enriched values for this service
  (for example, get the synonym values for “Seriata seriata”)
}
```

Let $\{Ev\} = [Ev1, Ev2, \dots, Evn]$ (enriched values from all enriching service)

Generate the main query Q to $\{Ev\} \Rightarrow (QEv)^x = \{QEv1, QEv2, \dots, QEv_n\}$.

QEv is the set of queries that will be used by BUFFIE for sending request messages to all the data resources in the system.

5.5 Architecture for User Query Enrichment

User query enrichment approach takes the initial user query submitted through the web page interface and validates the search concept and its values as the input from the presentation layer. The general architecture of the user query enrichment in BUFFIE is depicted in figure 5.6. It uses the Microsoft AJAX framework known as ASP.NET AJAX Extensions [150].

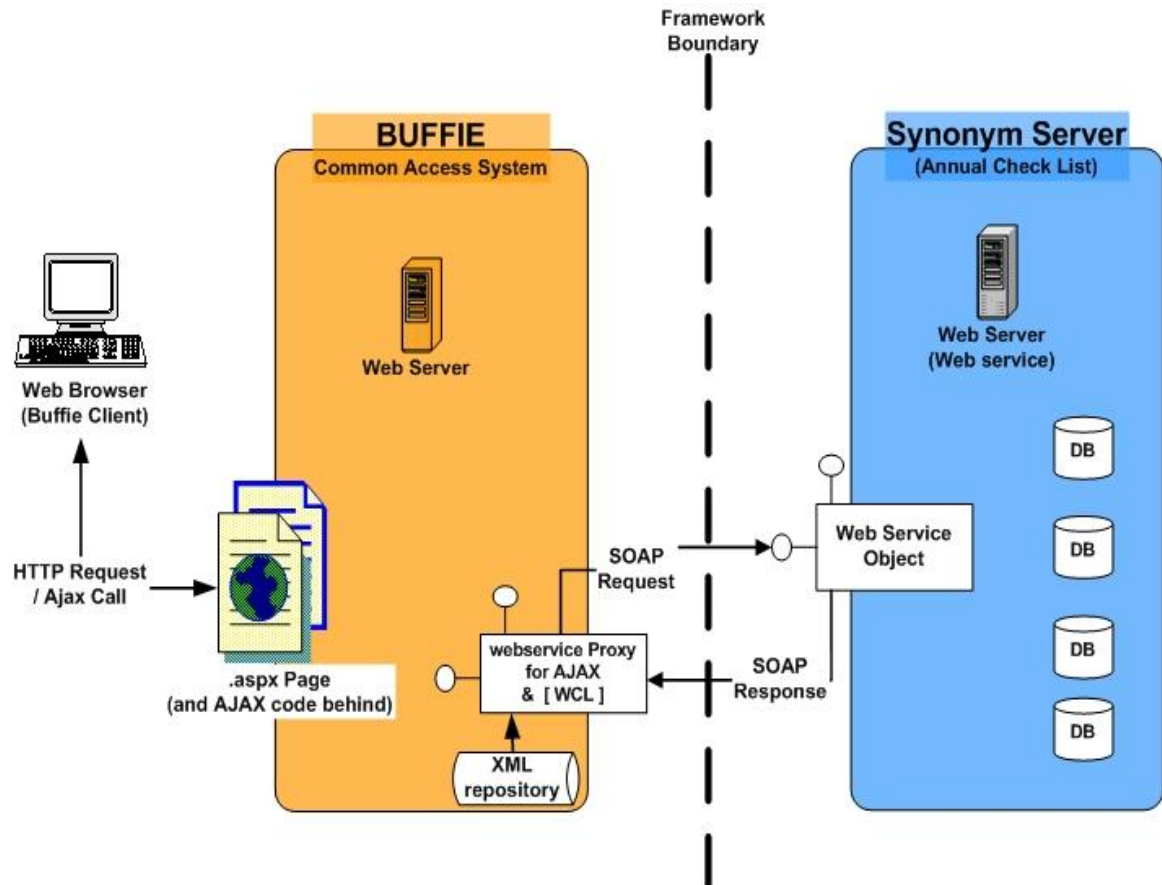


Figure 5.6: Architecture for Query Enrichment.

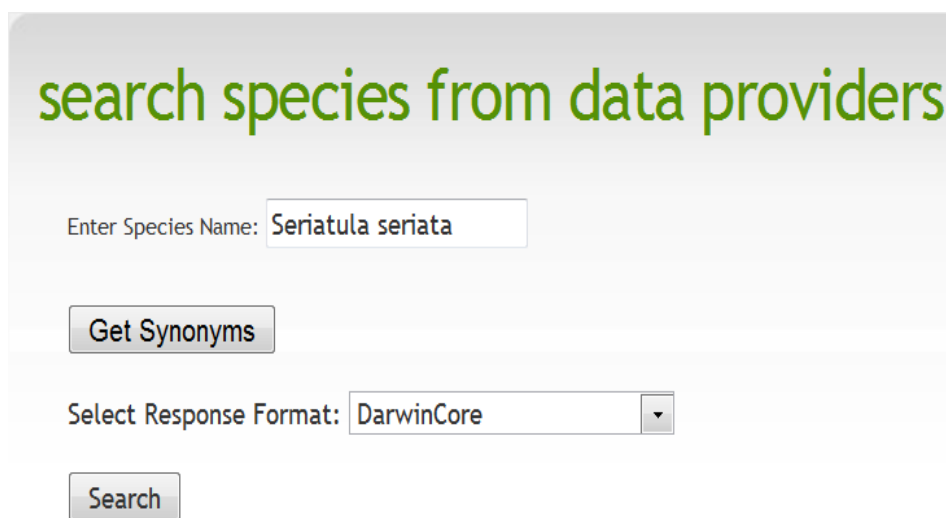
Using ASP.NET AJAX in this scenario is a best fit as it allows calls to different data resources through web service from the client browser without affecting the performance of the web page in the browser [158]. The server side code page (.aspx) renders the Html page with the necessary JavaScript codes. This AJAX-enabled Web page makes an initial request to the server's Web service communication layer (WCL). These web services are in the form of ASP.NET Web services (.asmx services) and running in the same domain as the BUFFIE application. The web services in the WCL then makes subsequent asynchronous requests to other web services for data. WCL acts like a bridging service between the AJAX pages and the

external web service used for query enrichment. To enable BUFFIE application to call ASP.NET Web services by using client script, the server's WCL automatically generates JavaScript proxy classes. A proxy class is generated for each web service that is referenced by a ServiceReference element in the ScriptManager control in the page. Data is exchanged asynchronously between client and server, typically in a text based JavaScript Object Notation (JSON) format [159].

The configuration files from the XML repository provide the information about the external web service interface methods and data format. Using this knowledge WCL creates the request messages to the external synonym server web service for e.g. SPECIES 2000 dynamic check list. The architecture also provides secure web service communication using the Microsoft cryptography framework [160] using digital certificates to authenticate the request and response messages.

5.5.1 Example of Query Enrichment in BUFFIE

This section explains the query enrichment process in BUFFIE web application. It shows a real example of a biodiversity data with the query concept as Species “scientific name” and its value as “Seriatula seriata”. The synonymy server used in this example for testing is a publicly available dynamic check list from Species 2000. The following figure 5.7 shows the AJAX form that submits the query details to the



The image shows a web interface for searching species. At the top, the text "search species from data providers" is displayed in green. Below this, there is a form with the following elements: a label "Enter Species Name:" followed by a text input field containing "Seriatula seriata"; a button labeled "Get Synonyms"; a label "Select Response Format:" followed by a dropdown menu showing "DarwinCore"; and a button labeled "Search".

Figure 5.7: AJAX web page with species scientific name and data.

web service proxy of the server generated by Web service Communication Layer. The XML config file in BUFFIE application provides the following information about the synonym services to the WCL like; web service, destination URL and its accessing parameters as follows:

Table5.1: Synonym service Providers information

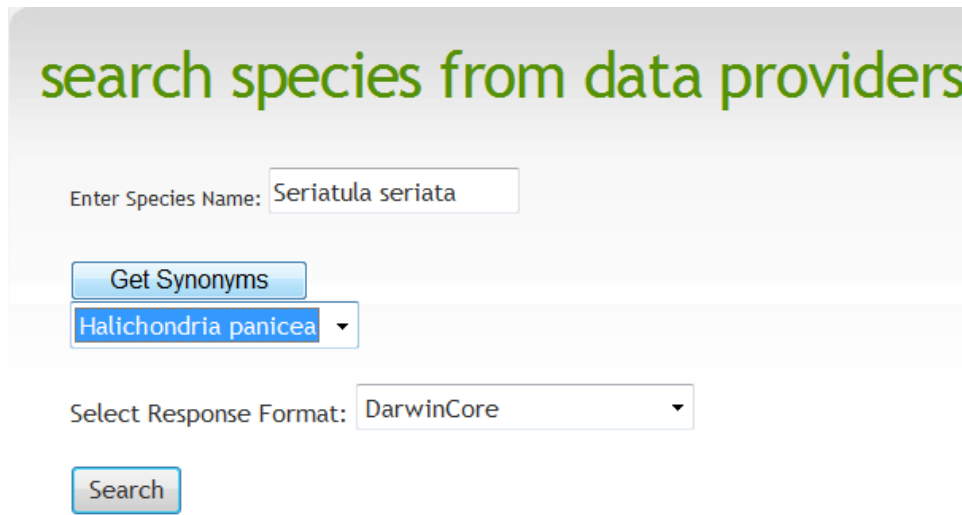
webservice name:	MsgDestn.Url.Species2000
web service location:	http://webservice.catalogueoflife.org/annual-checklist/2009/search.php
Web service Query Parameter	?name=
Web service Request Message	Xml Schema location to create the request message for this service
AuthenticodeX.509 certificate	Certificate file location used for secure communication.

Using the values as shown in the table the web service in the WCL creates a request message and sends the requests asynchronously to the synonymy servers. The “Authenticode” security feature is provided as an optional feature based on the communication type of external service provider.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<results id="" name="Seriatula seriata" total_number_of_results="1" start="0" number_of_results_returned="1">
  <result>
    <id>569363</id>
    <name>Seriatula seriata</name>
    <rank>Species</rank>
    <name_status>synonym</name_status>
    <name_html>Seriatula seriata (Grant, 1826)</name_html>
    <url>http://www.catalogueoflife.org/annual-checklist/2009/show_species_details.php?record_id=569363</url>
    <source_database></source_database>
    <online_resource/>
    <accepted_name>
      <id>1579928</id>
      <name>Halichondria panicea</name>
      <rank>Species</rank>
      <name_status>accepted name</name_status>
      <name_html>Halichondria panicea (Pallas, 1766)</name_html>
      <url>http://www.catalogueoflife.org/annual-checklist/2009/show_species_details.php?record_id=1579928</url>
      <source_database></source_database>
      <online_resource></online_resource>
      <source>
    </accepted_name>
  </result>
</results>
```

Figure 5.8: XML result from the synonym web service for species name.

The result for this query from the Species2000 dynamic check list service is an XML document as shown figure 5.7. The WCL component uses LINQ to XML which is a LINQ-enabled, in-memory XML programming interface to parse the result XML and sends the synonym values as delimited string to the client page. The JavaScript functions on the client page dynamically parses the result string and updates the contents of the page as shown in figure 5.9



The screenshot shows a web application interface with the following elements:

- Header: "search species from data providers" in green text.
- Form: "Enter Species Name: Seriatula seriata" with a text input field.
- Button: "Get Synonyms" in a blue button.
- Dropdown: A dropdown menu showing "Halichondria panicea" with a downward arrow.
- Form: "Select Response Format: DarwinCore" with a dropdown menu.
- Button: "Search" in a blue button.

Figure 5.9: results of synonym web service call in the application.

Now the initial user query concept of scientific name with a query value “Seriatula seriata” is enriched with other values namely the accepted name “Halichondria Panicea”. These query values are passed to the other components in Buffie Query analyzer for onward processing that will generate request query for both these scientific name values and send it to all the available providers of the system.

CHAPTER 6

The Query Response Retrieval and Transformation Process

6.1 Introduction

In this chapter we deal with another important stage of the research which is the query response retrieval and translation process. It is the second stage of the BUFFIE framework's common access system. Here we start with a discussion about the responses for the query sent to the data providers and the process of integrating these heterogeneous responses to the required format specified by the user. The previous chapter described about how to enhance the chance of finding the right and required information even if the data resource is in heterogeneous formats. This chapter deals with how the information found on these heterogeneous resources are integrated to make the results more

meaningful for the user. Then the knowledgebase is described using XSLT templates.

6.2 Response Data Integration Strategy from Heterogeneous Providers

Apart from enriching the query to increase the visibility of the data sources, in order to augment the level of interoperability achieved, the BUFFIE system also employs a data integration strategy to address the heterogeneous responses to concur with the requirement of the initial user query. This research limits its boundary to those biodiversity data sources that can provide response for the query in XML structured documents across the Internet (HTTP Protocol). We identify the following requirements (or assumptions) for a meaningful data integration of different responses.

- All the responses from data providers for a specific query must describe the same species identified in the real-world.
- Methods to eliminate the differences in the name, structure and representation of the data models used to describe the species.
- All the responses received for the query will be in XML formatted messages with the same or different schemas as long as the schema is identified in the *Domain Knowledgebase* of the BUFFIE system.

The schema elements or concepts described in XML serve as meta-data for the actual biodiversity data it contains. Though the XML schemas define the structure, typing and naming about data, there is still a great deal of semantic knowledge which cannot be properly expressed within the schemas used for data exchange. For example a certain specification like the dimension of the species is stored as feet in one schema and as meters in another, and then the software component cannot integrate them unless it has the knowledge about the imperial to metric conversion details. If only all the biodiversity standard XML schemas can accurately describe the data structure and if fully automatic schema translation and integration were possible then several important interoperability problems could be resolved purely at a syntactical level. To

tackle the semantic interoperability problem, use of ontology has been researched as a solution, but the limitation of ontology is that it cannot capture real world semantics and describes only the logical relationships between the concepts of a domain [[161], [162]. Also ontologies in a domain have lots of limitations due to ambiguity and incompleteness in describing the data and this only proves that human involvement cannot be entirely eliminated and data integration requires devising logical programming components in such a way that semantics is followed throughout the entire data integration stack rather than at a particular instance [163]. Establishing a semantic match for data describing concepts and managing the representational differences is a knowledge management problem: How to arrange the right system to have the right knowledge about what the data means? [164].

In our research we demonstrate the “data integration on demand” [165] where the BUFFIE prototype system acts as a middleware layer and performs dynamic integration of data based on the user query as opposed to the data warehouse approach or “data integration in advance” Figure 6.1 shows the conceptual view of the heterogeneous response integration in the BUFFIE system.

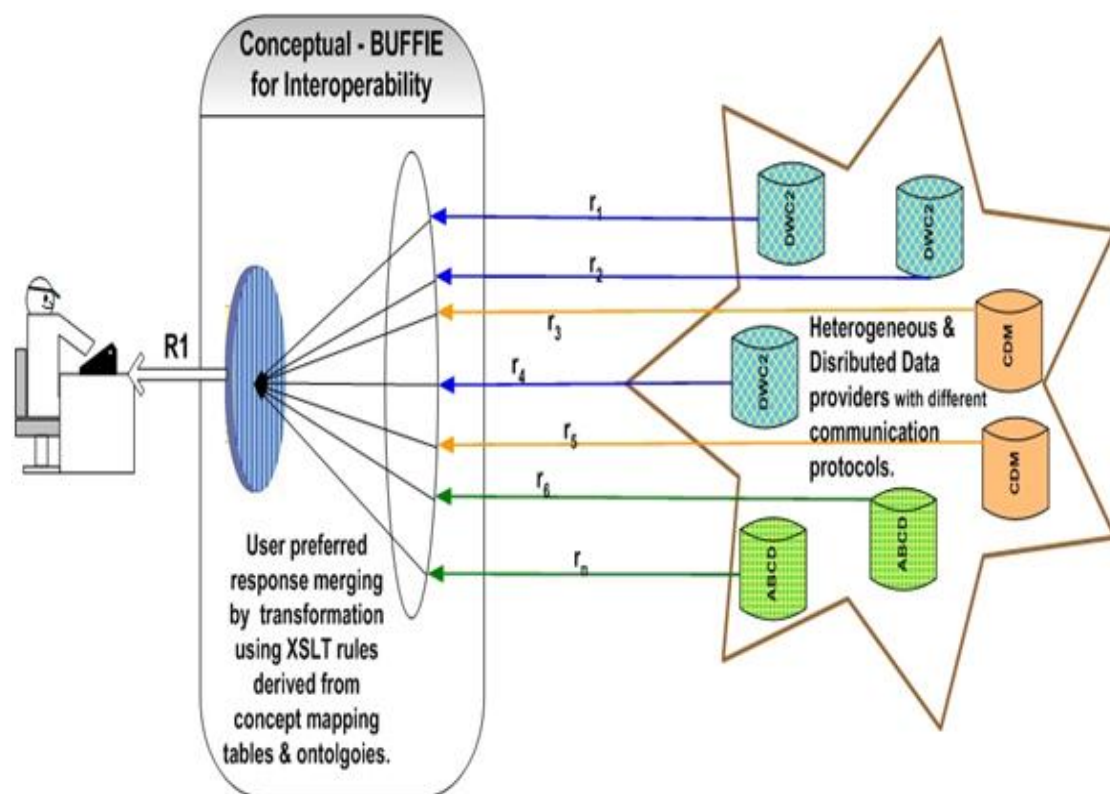


Figure 6.1: Conceptual view of heterogeneous response integration in BUFFIE.

Response Integration:-

Let **R** is the resultant response

X is the transformation functions

Then For Each Provider in the BUFFIE system

{

Receive the response for the query in XML format.

Each response may consist of one or more data records
and follows a specific XML schema standard.

$$\mathbf{R}_i = \mathbf{S}_i [\mathbf{r}^1 + \mathbf{r}^2 + \mathbf{r}^3 + \dots + \mathbf{r}^n]$$

Where, **R_i** is the response from a particular data provider (**i**).

S_i is the data schema (e.g. Darwin Core) for the provider (**i**).

r¹ is record one (data unit) and **r²** is record two and so on.

Validate the response records (**r¹ + r² + r³ + + rⁿ**) using the
schema **S_i** and save the response **R_i** from each provider in local
database of Buffie.

}

Apply transformation functions for the response records based on
provider schema and the required resultant format.

Merge the responses from all the providers:

$$\mathbf{R} = \mathbf{X} [\mathbf{R}_1 + \mathbf{R}_2 + \mathbf{R}_3 + \dots + \mathbf{R}_n]$$

When the variance in the syntax and semantics of the response schema to the required
schema is zero then the transformation function (**X**) just propagates the source format
to the resultant format otherwise various schema matching process are applied on the
responses received from data providers before merging them to the result (**R**).

6.3 Schema Matching Model

Building a metadata knowledge base layer for semantic and syntactic matching of the
heterogeneous data has been proposed as a solution to the problem of data integration,
when the data model is described as a structured XML [166]. Schema matching in
our research involves the design-time analysis of biodiversity data-communication
schemas like Darwin Core, ABCD and Spice CDM to produce mappings logic. The

run-time scenario allows the user to specify the required output format along with the query. The criteria used to match elements from different standards are based on heuristics and published mapping tables (Appendix A) of the different elements by the domain experts. Discussions with biologists and knowledge sharing with other experts of the ENBI forum have augmented the development of mapping biodiversity concepts in BUFFIE.

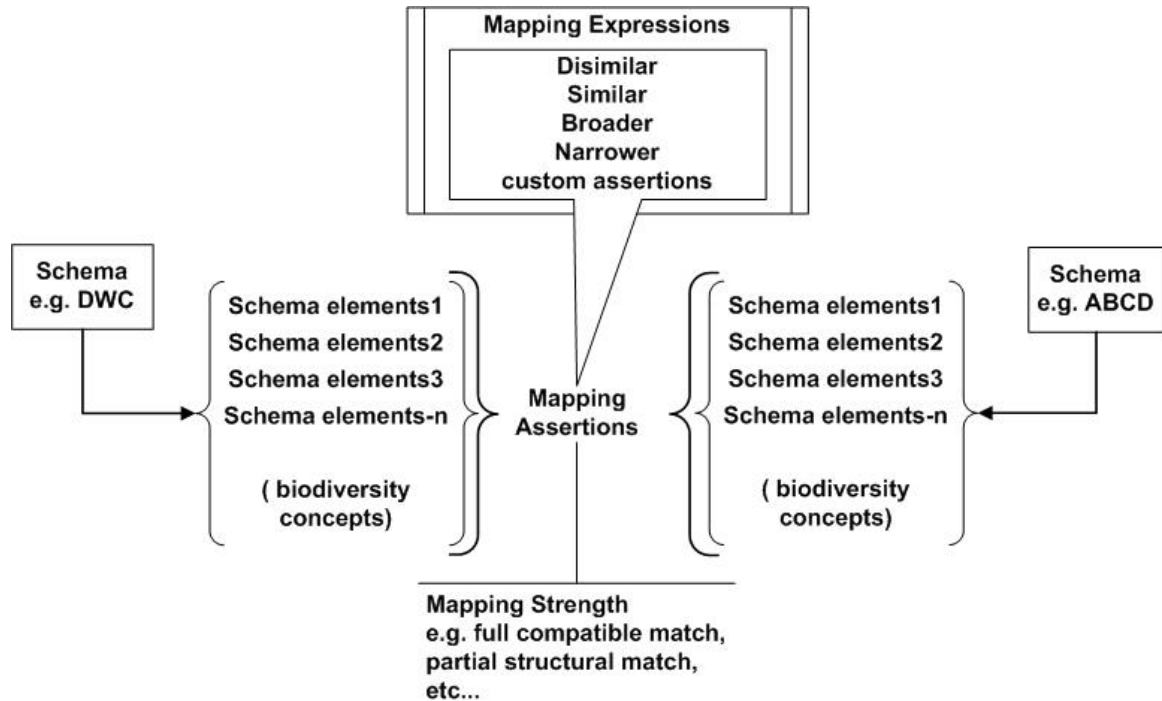


Figure 6.2: Schema Mapping Assertion Model.

The mapping relationships between concepts are not captured in a precise mathematical way; instead we followed a more pragmatic approach in the goal of producing a mapping that is consistent with heuristics (experience-based) that approximate our understanding of what biodiversity data users would consider to be a good match. Figure 6.2 shows a schematic representation of schema assertion model for describing the mappings in a systematic way in BUFFIE.

A mapping assertion is a defined relationship between two schema elements which are biodiversity concepts from a different schema. A mapping expression is attached to a mapping assertion that specifies how the schema elements are related. The mapping expressions are either:

- Directional in which case an element from one schema refers to an element in another schema e.g., similar reference.
- (or) non-directional that is a relation between the elements of different schemas are defined using
 - scalars (e.g. =, ≤, ≥)
 - functions like addition or concatenation, customized functions
 - relationship like is-part-of, is-a, contains

The mapping expression logic for all the schema elements are built into the Domain knowledge Base (DKB) component of the BUFFIE.

6.4 Biodiversity Data Transformation Architecture

Building a metadata knowledge base layer for semantic and syntactic matching of XML formatted data has been followed in this approach [167]. Figure 6.3 shows the architecture for data matching and transformation process in BUFFIE framework.

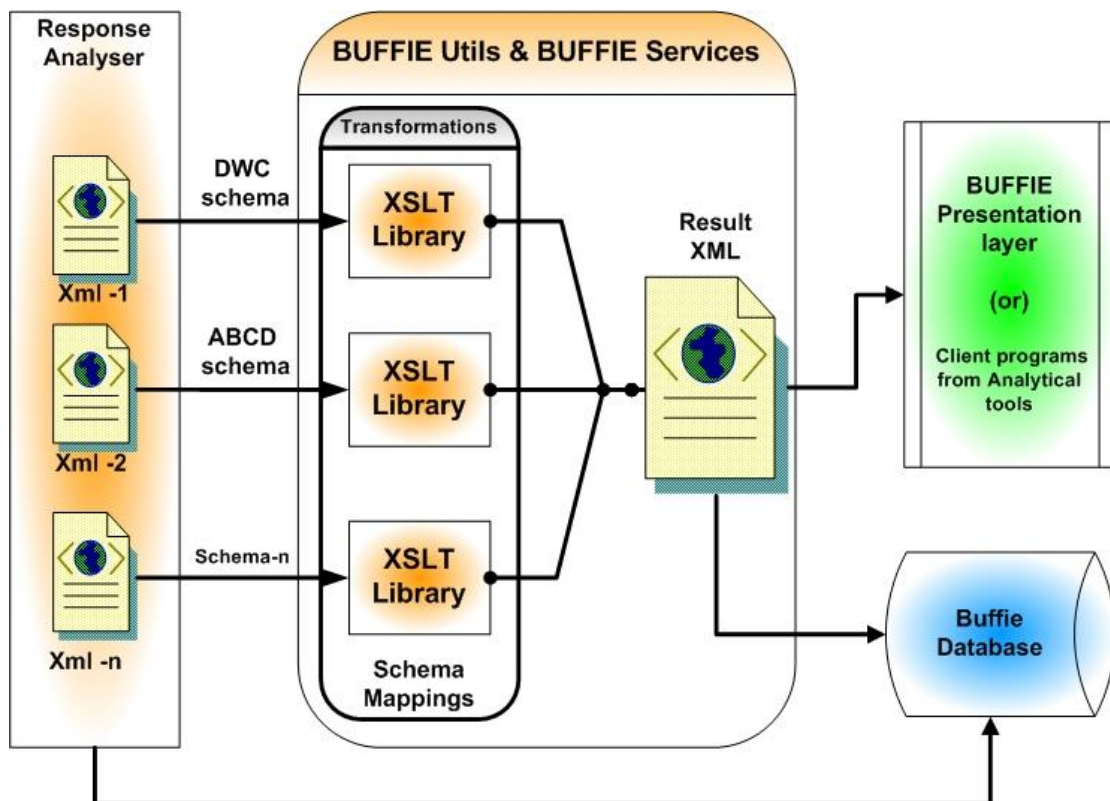


Figure 6.3: Biodiversity Data Transformation using Schema matching.

The response analyzer receives the heterogeneous responses from the providers and validates the XML messages. All the response messages from the providers are stored in local BUFFIE database that helps in debugging and future analysis. The *BuffieUtils* and *BuffieServices* components process these XML messages by using the information about the provider and the schema from the DKB. *BuffieUtils* components employ the right XSLT library modules that contain the schema transformation functionalities for a given input XML data to produce the resultant output XML message. The XSLT library module contains the templates that are designed to resolve the semantic and logical heterogeneities of the data in the schema elements. Most XML schemas' elements in biodiversity domain have some semantics that affect the matching criteria but are not formally expressed or documented. So we followed a semi- automatic schema mapping approach between the elements of the different schemas, i.e. the biologists/developers of the XSLT templates define the mapping logic during design time and for some elements the program automatically applies the mapping logic as suitable. More about the XSLT templates and about its design are discussed in the next section. The output of all these transformations is merged into one single result.xml that holds the result for the initial user query. This result is stored in the local Buffie database and sent to the Buffie presentation layer over HTTP protocol or to the client programs through the web-service messages. Please refer to appendix D for the XSLT templates used in the DKB.

6.5 Functional Approach for Schema Integration

In the previous section, we described the biodiversity schema matching model and the process of matching the data in the BUFFIE system. This section we discuss the approach followed in our research and explain the techniques and operation process of the schema integration. Schema matching, through transformation of XML documents have to resolve the scalability and semantic relationship discovery problems [168]. These problems are increased with the richness of the XML data model of the participating schemas. The scalability problem was addressed with the extensible architecture of the BUFFIE system and the design of the Domain Knowledgebase (DKB) component. Semantic relationship between the elements of the schema and data is the interpretations by domain experts according to the knowledge of the real

world. Explicit and formal meanings of the participating schema elements were developed using lambda functions of C# and XSLT in a functional style components. Each schema element (concept) is analyzed from the semantic viewpoint of the data that it holds and its logical structure. The concept mapping algorithm is developed as an integrated “hybrid matcher” that identifies both the individual element to element match and the use of multiple matching criteria like name and type of data held by the element in the schema. The idea here is to derive transformation algorithms that enable automated restructuring of the data elements of the provider’s responses without manual intervention at the run-time of BUFFIE system. This is possible because the transformation algorithms are enabled using LINQ and XSLT that implements the functional programming logic which facilitates the semi-automatic approach. Please refer to the codes in appendix B for the component codes that demonstrates this logic.

6.5.1 XSLT Library for Schema Mapping

Based on the BUFFIE framework, we identify the main causes of heterogeneity between biodiversity communications protocols defined as XML schema entities are at different levels. They are classified into two main categories namely semantic heterogeneities and logical heterogeneities. The flexibility of XML schema languages gives rise to a larger variety of possibilities to model the same biodiversity concepts than text files. For example a species collection date may be represented as “strings” in one schema or in another schema as instances of a primitive data type “Date”. These conflicts are difficult to resolve and generally requires human intervention due to the tacit knowledge needed from another domain for e.g. computer data types. In this research the mapping process provides a predefined library of logical transformation operations produced as XSLT templates generated based on the mapping table and auxiliary information produced by the biodiversity domain experts. We analyzed both published, standardized biodiversity schemas [70] like Darwin Core, ABCD and the data instances received from the providers, who use these standards for data exchange. The matching logic was created from individual schema elements or attributes or for combinations of elements like complex structures from a single schema to all the other schemas involved in the research.

6.5.1.1 Matching on Schema Information

The initial development of the concept mappings were based on the schema information of the communications protocols such as name, description, data type, relationship, constraints and schema structure. The unit of matching is defined at the atomic-level or at a structure-level (multiple elements). For each element or attribute of the first schema (e.g. Darwin Core), atomic-level matching determines the matching elements or attributes from the other schemas (e.g. ABCD, MANIS, etc...) Table 6.1 shows some sample atomic-level mappings used in the XSLT templates. E.g. “Darwin.InstitutionCode \cong ABCD.SourceInstitutionCode”

Table 6.1: Sample atomic-level match.

Schema 1(elements)	Schema 2(elements)
<i>Darwin Core</i>	<i>ABCD</i>
GlobalUniqueIdentifier	UnitGUID
InstitutionCode	SourceInstitutionCode

Structure-level matching refers to matching combinations of elements that appear together in a structure. This matching is decided by the factors like how complete and how precise a match of the structures are required. It could be a fully compatible match, where all the components of the structures in two are more schemas match exactly or a partial structural match, where only some of the components in a structure are required to match. The effectiveness of this structure matching can be increased by using auxiliary information and known equivalence patterns from the biodiversity domain. For example, two structures in an “is-a” hierarchy is merged to a single structure in the transformed output. For e.g. the sub element (child node or attribute) of the first schema is represented as a Boolean attribute in the second schema. Another pattern is that two different structures of the source schema with a referential relationship are joined as a single structure in the output schema. Table 6.2 shows some examples of a full and partial structure-level match.

Table 6.2: Full and Partial structural match

Schema 1(elements)	Schema 2(elements)	
<i>Darwin Core</i>	<i>ABCD</i>	
Longitude	CoordinatesLatLong	Full Structural match
Latitude	LongitudeDecimal	
CoordinatePrecision	LatitudeDecimal	
MinimumElevation	CoordinateErrorDistanceInMeters	
MaximumElevation	SpatialDatum	
Country	GatheringSite	Partial structure match
StateProvince	LocalityText	
County	Country	
Locality	NamedAreas	

All the transformation result may relate to one or more elements of one schema, to one or more elements of the other, this is described as transformation match cardinality. An element from schema1 or schema2 can participate in zero, one or many mapping elements of another schema. There are four types of relationship identified from the schema mapping as shown in the table 6.3. The transformation match cardinality of one-to-one element-level matching is typically restricted to individual mapping elements of the participating schema. One-to-many, and many-to-one match cardinality may have either individual mapping elements or structure-level matching. Many-to-many mapping elements usually involves the structural embedding of the schema elements requiring the structure-level matching. In the first row of table 6.4 the match is one-to-one like the value of the “UnitGUID” element from ABCD schema is assigned to the element named “GlobalUniqueIdentifier” of the Darwin Core schema. When matching multiple elements from these schemas, expressions or functions are used to specify how these elements are related. For example row 2 explains a many-to-one element-level or structure-level matching, if the element ABCD.NameAuthorYearString has a valid value, then it is assigned to Darwincore.ScientificName otherwise the child elements (structure) of ABCD.ScientificNameAtomized are concatenated using a function and assigned to Darwincore.ScientificName.

Table 6.3: Transformation Match Cardinalities

Transformation cardinalities	Schema 1(elements)	Schema 2(elements)	Matching Expression
	<i>ABCD</i>	<i>Darwin Core</i>	
1. One-to-One (element-level)	UnitGUID	GlobalUniqueId entifier	Equal to (=)
2. Many-to-One (element-level)	NameAuthorYearString ScientificNameAtomized <i>Zoological</i> <i>Genus</i> <i>SpeciesEpithet</i> <i>AuthorTeamOriginal</i> <i>AndYear</i>	ScientificName	Either NameAuthorYear string (or) Function child nodes of (ScientificName Atomized)
3. One-to-Many (element-level)	GatheringDateTime ISODateTimeBegin	YearCollected MonthCollected DayCollected JulianDay TimeOfDay	Function StringSplit
4. Many-to One (Structure-level) Many-to-Many (element-level)	LocalityText Country ISO3Letter NamedAreas NamedArea NamedAreaClass NamedAreaName	Country StateProvince County Locality	Country, stateprovince = localityText,ISO3Letter County, Locality = Named Area, NamedAreaName

In row 4 of the table 6.4 the values for the Darwin Core structure are derived from two different structures of ABCD schema. This is an example for many-to-one at the structure-level matching but at the element-level it is a many-to-many relation. In our approach to schema matching in the BUFFIE, we had discovered that most of the biodiversity protocols expressed as XML schemas primarily have hierarchical structures of biodiversity concepts based on some form of containment relationship with a parent and child nodes. Hence to perform a transformation match we used an XPath Navigator cursor model [169] that allows forward and backwards movement of the hierarchical structures.

6.5.1.2 Matching on Data Instance

In our research the data validation of the responses are carried out, only to the extent of the data model (schemas) used for the communication. It is assumed that if the data response sent by all the providers for a request query, conforms to a particular schema, then the biodiversity data contained in the response is valid and then accepted for data integration in the BUFFIE system.

```

<record>
  <darwin:DateLastModified>2005-08-17</darwin:DateLastModified>
  <darwin:InstitutionCode>BioGIS</darwin:InstitutionCode>
  <darwin:CollectionCode>Land snails - HUU collections</darwin:CollectionCode>
  <darwin:CatalogNumber>18</darwin:CatalogNumber>
  <darwin:ScientificName>Buliminus labrosus</darwin:ScientificName>
  <darwin:BasisOfRecord xsi:nil="true"/>
  <darwin:Kingdom/>
  <darwin:Phylum/>
  <darwin:Class/>
  <darwin:Order xsi:nil="true"/>
  <darwin:Family>Enidae</darwin:Family>
  <darwin:Genus>Buliminus</darwin:Genus>
  <darwin:Species>labrosus</darwin:Species>
  <darwin:Subspecies xsi:nil="true"/>
  <darwin:Country>Israel</darwin:Country>
  <darwin:StateProvince xsi:nil="true"/>
  <darwin:County xsi:nil="true"/>
  <darwin:Locality>Khirbet Sa'ida</darwin:Locality>
  <darwin:Longitude>35.204148</darwin:Longitude>
  <darwin:Latitude>31.757835</darwin:Latitude>
  <darwin:CoordinatePrecision>0.7</darwin:CoordinatePrecision>
  <darwin:MinimumElevation>742.3</darwin:MinimumElevation>
  <darwin:MaximumElevation>742.3</darwin:MaximumElevation>
  <darwin:MinimumDepth xsi:nil="true"/>
  <darwin:MaximumDepth xsi:nil="true"/>
  <darwin:Sex xsi:nil="true"/>
  <darwin:PreparationType xsi:nil="true"/>
  <darwin:IndividualCount xsi:nil="true"/>
  <darwin:PreviousCatalogNumber xsi:nil="true"/>
  <darwin:RelationshipType xsi:nil="true"/>
  <darwin:RelatedCatalogItem xsi:nil="true"/>
  <darwin:Notes xsi:nil="true"/>
</record>

```

Figure 6.4: Sample Response from a Darwin Core Provider.

For example figure 6.4 shows a valid response message from a resource named “snails”, we could tell as long as all the elements in the XML message confirms to a Darwin Core schema, then the data is valid, but there is no way of confirming the integrity of the data contained within these XML elements as this requires knowledge of multiple domains. For example the structural integrity with respect to a standard is fully verified using schema validation technique, whereas the semantic integrity with respect to the related concepts in the standard cannot be verified completely. Having

stated that, to improve the overall interoperability, we analyzed the data contents of the response messages (XML documents) from the providers for a range of queries. This process of analysing, instance-level data information brings out some important semantics with respect to the schema elements and which are applied for the data matching. The transformation templates designed in the domain knowledge base takes into consideration about the constraints such as data types, value ranges, relationship types and cardinalities of the data instances received from the providers. The main benefit of evaluating the data instances is to get a more accurate transformation of the actual contents of the schema elements. For example a schema-level matching could equate the following concept name country from the Darwin Core and ABCD schema as shown in the first row of table 6.4. But then analysis of the data-instance shows, the value for the concept “country” is stored as “Israel” in one and as “ISR” in another. To negotiate these types of differences in the data, the domain knowledge base provides a special country code table from which the transformation functions derives the equivalent value.

Table 6.4: Full and Partial structural match

	Schema 1(elements)	Schema 2(elements)
	<i>Darwin Core</i>	<i>ABCD</i>
1.	<code><darwin:Country>Israel</code> <code></darwin:Country></code>	<code><Country></code> <code><ISO3Letter>ISR</ISO3Letter></code> <code></Country></code>
2.	<code><darwin:Longitude>35.204148</code> <code></darwin:Longitude></code> <code><darwin:Latitude>31.757835</code> <code></darwin:Latitude></code>	<code><LongitudeDecimal>11.5</code> <code></LongitudeDecimal></code> <code><LatitudeDecimal>47.25</code> <code></LatitudeDecimal></code>

Another application of data-instance matching is based on the constraints of the data such as numerical value ranges and the precision as shown in row 2 of table 6.4. Instance-level matching is also performed by using the auxiliary information provided by the biologists like previous published mapping information. The transformation component uses a hybrid approach of both schema-level and data-instance-level match to increase the effectiveness of the matching between the participating schemas.

6.5.2 Transformation Functions in DKB

This section describes a set of primitive transformation operations or functions used by the schema matching process of the XSLT templates in the Domain Knowledge Base (DKB). These primitive operations are composed together to form larger transformation functions that convert the heterogeneous responses to the required format specified in the query. We explain operations based on two schemas i.e. the responses from the data providers are known as source schema for the transformation and the required schema by user is called as target schema. The operations are as follows:

- **Add:** adds a schema element or an entity (biodiversity concept) to the target XML message. Entities can be concepts and attributes that are based on the source schema or it can be a new element introduced by the transformation operation.
- **Delete:** Removing an entity from the target schema. This operation carries out the opposite transformation of Add.
- **Merge:** Two distinct entities from the source schema are merged into one entity in the target schema. This is carried out using functions like aggregation and concatenation. For e.g. when transforming the concepts like `ScientificName` the resultant values in the target schema (Darwin Core) is a concatenation of values from the child nodes of `ScientificNameAtomized` from the source schema (ABCD) concepts.
- **Split:** The value of the source schema is decomposed to form different concepts in the target schema. This is the reverse operation of merge. For e.g. the value of `GatheringDateTime` concept from ABCD schema is split to form different schema elements like `YearCollected`, `MonthCollected`, `DayCollected`, `TimeOfDay` in the darwincore schema
- **Rename:** This operation changes the concept and properties names on to the target schema.

- **Connect:** This just substitutes the source schema value as a one to one mapping to the target schema without any transformation. For e.g. most of the schema elements of the header part of the message from the source is directly copied on to the target schema of the transformation.

Apart from these primitive operations, mathematical and logical functions were implemented to effect the transformations. These functions perform the logical operations that also use the extra information provided by the domain knowledgeable users (like biologists). The framework of BUFFIE is designed such that the configuration of the XML files and XSLT library within the domain knowledge base can be easily modify or extended by the developer.

6.6 Example of Data Transformation in BUFFIE

In this section we introduce an example of a data transformation of the response from an ABCD provider to a Darwin Core type request. We discussed in the previous sections of this chapter about the architecture for data transformation and the approaches for schema mapping using the XSLT libraries of the domain knowledge base. To illustrate this process a response from a provider is investigated and it is shown how it goes through the transformation process to form the resultant message. All of these transformations are implemented as part of the middleware in the BUFFIE system and hence there is no presentation layer representation for the data. The example messages shown here are from the extract of the BUFFIE local database. Consider the user query: Find the information for species with “scientificname” as “*Buliminus labrosus*”. This query is made through the BUFFIE common access system and one of the Darwin Core data source named as “Snails” has sent a response message in the XML format as shown in figure 6.5. The response message is composed of three parts namely header, content and diagnostics. The header and diagnostics part of the message contains the information about the data providers and the schema format used for data communication from source to destination. These are part of the meta-data that helps BUFFIE system to identify the type of the response message. The Buffie engine components check, if the response message has got a valid “content” structure with records. These records are the elements that hold the

data required by the query. In the example shown below there are 265 records returned for the query from BUFFIE for a species named “Buliminus labrosus”. Now suppose the user wants the data to be in the ABCD format then the BUFFIE system uses its domain knowledge base to transform the Darwin Core format data into ABCD format.

```

<?xml version="1.0" encoding="utf-8"?>
<response xmlns="http://digir.net/schema/protocol/2003/1.0">
  <header>
    <version>$Revision: 1.14 $</version>
    <sendTime>2007-10-01T18:46:24+0200</sendTime>
    <source resource="SNAILS">http://ip62.eti.uva.nl:8000/digir/DiGIR.php</source>|
    <destination>81.79.102.40</destination>
    <type>search</type>
  </header>
  <content xmlns:dwc="http://digir.net/schema/conceptual/darwin/2003/1.0" xmlns:darwin=
  <record>
    <darwin:DateLastModified>2005-08-17</darwin:DateLastModified>
    <darwin:InstitutionCode>BioGIS</darwin:InstitutionCode>
    <darwin:CollectionCode>Land snails - HUJ collections</darwin:CollectionCode>
    <darwin:CatalogNumber>18</darwin:CatalogNumber>
    <darwin:ScientificName>Buliminus labrosus</darwin:ScientificName>
    <darwin:BasisOfRecord xsi:nil="true"/>
    <darwin:Kingdom/>
    <darwin:Phylum/>
    <darwin:Class/>
    <darwin:Order xsi:nil="true"/>
    <darwin:Family>Enidae</darwin:Family>
    <darwin:Genus>Buliminus</darwin:Genus>
    <darwin:Species>labrosus</darwin:Species>
    <darwin:Subspecies xsi:nil="true"/>
    <darwin:ScientificNameAuthor xsi:nil="true"/>
    <darwin:IdentifiedBy>Heller J.</darwin:IdentifiedBy>
    <darwin:YearIdentified xsi:nil="true"/>
    <darwin:MonthIdentified xsi:nil="true"/>
    <darwin:DayIdentified xsi:nil="true"/>
    <darwin:TypeStatus xsi:nil="true"/>
    <darwin:CollectorNumber xsi:nil="true"/>
    <darwin:FieldNumber xsi:nil="true"/>
    <darwin:Collector>Heller J.</darwin:Collector>
    <darwin:YearCollected>1946</darwin:YearCollected>
    <darwin:MonthCollected>0</darwin:MonthCollected>
    <darwin:DayCollected>0</darwin:DayCollected>
    <darwin:JulianDay xsi:nil="true"/>
    <darwin:TimeOfDay xsi:nil="true"/>
    <darwin:ContinentOcean xsi:nil="true"/>
    <darwin:Country>Israel</darwin:Country>
    <darwin:StateProvince xsi:nil="true"/>
    <darwin:County xsi:nil="true"/>
    <darwin:Locality>Khirbet Sa'ida</darwin:Locality>
    <darwin:Longitude>35.204148</darwin:Longitude>
    <darwin:Latitude>31.757835</darwin:Latitude>
    <darwin:CoordinatePrecision>0.7</darwin:CoordinatePrecision>
    <darwin:MinimumElevation>742.3</darwin:MinimumElevation>
    <darwin:MaximumElevation>742.3</darwin:MaximumElevation>
    <darwin:MinimumDepth xsi:nil="true"/>
    <darwin:MaximumDepth xsi:nil="true"/>
    <darwin:Sex xsi:nil="true"/>
    <darwin:PreparationType xsi:nil="true"/>
    <darwin:IndividualCount xsi:nil="true"/>
    <darwin:PreviousCatalogNumber xsi:nil="true"/>
    <darwin:RelationshipType xsi:nil="true"/>
    <darwin:RelatedCatalogItem xsi:nil="true"/>
    <darwin:Notes xsi:nil="true"/>
  </record>
  <record>...</record>
  <record>...</record>
  <record>...</record>
  <record>...</record>
  <record>...</record>
  <record>...</record>
  </content>
  <diagnostics>
    <diagnostic code="STATUS_INTERVAL" severity="info">600</diagnostic>
    <diagnostic code="STATUS_DATA" severity="info">1,1,1</diagnostic>
    <diagnostic code="Unknown PHP Error [8]" severity="DIAG_WARNING">Undefined index:
    <diagnostic code="Unknown PHP Error [8]" severity="DIAG_WARNING">Undefined index:
    <diagnostic code="MATCH_COUNT" severity="info">265</diagnostic>
    <diagnostic code="RECORD_COUNT" severity="info">265</diagnostic>
    <diagnostic code="END_OF_RECORDS" severity="info">true</diagnostic>
  </diagnostics>
</response>

```

265 records are returned for this query

Figure 6.5: Example of a source response message from Darwin core provider.

```

<response xmlns="http://www.biocase.org/schemas/protocol/1.3"
<header>...</header>
<content >
<DataSets xmlns="http://www.tdwg.org/schemas/abcd/1.2">
<DataSet>
<OriginalSource>...</OriginalSource>
<DatasetDerivations>...</DatasetDerivations>
<Units>
<Unit>
<UnitID></UnitID>
<Identifications>
<Identification>
<TaxonIdentified>
<NameAuthorYearString>Buliminus labrosus Heller J</NameAuthorYearString>
<ScientificNameAtomized>
<Zoological>
<Genus>Buliminus</Genus>
<SpeciesEpithet>labrosus</SpeciesEpithet>
<AuthorTeamOriginalAndYear>Heller J.</AuthorTeamOriginalAndYear>
</Zoological>
</ScientificNameAtomized>
</TaxonIdentified>
</Identification>
</Identifications>
<Gathering>
<GatheringDateTime>
<ISODateTimeBegin>1946-01-01</ISODateTimeBegin>
</GatheringDateTime>
<GatheringSite>
<LocalityText>Khirbet Sa'ida</LocalityText>
<Country>
<ISO3Letter>ISR</ISO3Letter>
</Country>
<NamedAreas>
<NamedArea>
<NamedAreaClass>Bundesland</NamedAreaClass>
<NamedAreaName>Ti</NamedAreaName>
</NamedArea>
</NamedAreas>
<SiteCoordinateSets>
<SiteCoordinates>
<CoordinatesLatLong>
<LongitudeDecimal>35.204148</LongitudeDecimal>
<LatitudeDecimal>31.757835</LatitudeDecimal>
<SpatialDatum></SpatialDatum>
<CoordinateErrorDistanceInMeters>500</CoordinateErrorDistanceInMeters>
</CoordinatesLatLong>
</SiteCoordinates>
</SiteCoordinateSets>
<Altitude>
<MeasurementAtomized>
<MeasurementLowerValue>742.3</MeasurementLowerValue>
</MeasurementAtomized>
</Altitude>
</GatheringSite>
</Gathering>
</Unit>
<Unit>...</Unit>
<Unit>...</Unit>
<Unit>...</Unit>
<Unit>...</Unit>
<Unit>...</Unit>
<Unit>...</Unit>
<Unit>...</Unit>
<Unit>...</Unit>
</Units>
</DataSet>
</DataSets>
</content>
<diagnostics>
<diagnostic>OK</diagnostic>
</diagnostics>
</response>

```

Figure 6.6: Example of the transformed xml message in ABCD format.

CHAPTER 7

The BUFFIE Implementation

7.1 Introduction

In this chapter, we present the implementation details of the BUFFIE v2.0 framework from the designed architecture and the tools that were used to develop and deploy the various components of the system. The architecture of this prototype system is shown in figure 5.6. The prime objective of developing this prototype is to demonstrate, as to how the BUFFIE framework improves the interoperability between the biodiversity networks composed of heterogeneous and distributed data providers. The implementation of the prototype also shows how the structural and semantic interoperability of biodiversity data can be accomplished. Unlike the previous version of BUFFIE v1.0, which was implemented on Java platform on Apache Tomcat web server, this one is implemented using Microsoft.Net3.5 platform. The main tool used for developing the prototype is Microsoft Visual Studio 2008 professional [170], an

integrated development environment that helps the developers to code, debug, test and deploys the system. Microsoft Internet Information Server 6.0 [171] is used as the application Web server for hosting the BUFFIE web application and web services and SQL 2008 server [172] is used for storing the local BUFFIE database. The choices of using these latest tools were made, based on maintaining the objectives of the BUFFIE architecture in the implementation process as well:

- Extensibility and Scalability – this allows adding or removing a communication protocol and XSLT templates to the domain knowledge base without affecting the application. This type of implementation allows controlling the number of users or providers in the BUFFIE system.
- Code Compactness and Reuse: The coding of the system is followed based on the Microsoft coding standards. The components and modules are designed such that same implementation can be used for a different data domain by developing and adding the corresponding domain knowledge base component into the framework.
- Security and Performance: the system is organized into separate set of assemblies under appropriate namespaces. Basic forms authentication and provision for web service security were included in the design.

The BUFFIE architecture has four main components, Query designer user interface, Query enrichment, Query processor and Domain knowledge Base (DKB). These components are designed as modules in several layers as shown in Figure 7.1 that shows the multi-layered implementation of the BUFFIE v2.0 system comprising of six projects arranged in three layers namely the Presentation Layer, Business Logic Layer and Data Access Layer. BUFFIE common access system is a middleware system aimed for interoperation of XML messages and hence the user interface or the presentation layer is very light and only used for query submission and for the display of the response. The main part of the query processing lies in the business logic layer. The programming approach followed here is a combination of both object-oriented-design and functional programming. The classes and libraries used in the projects were designed with high cohesion (grouping a set of responsibilities together that are strongly related) and low coupling (less dependency between software modules) as this favours easy maintenance and reusability [173].

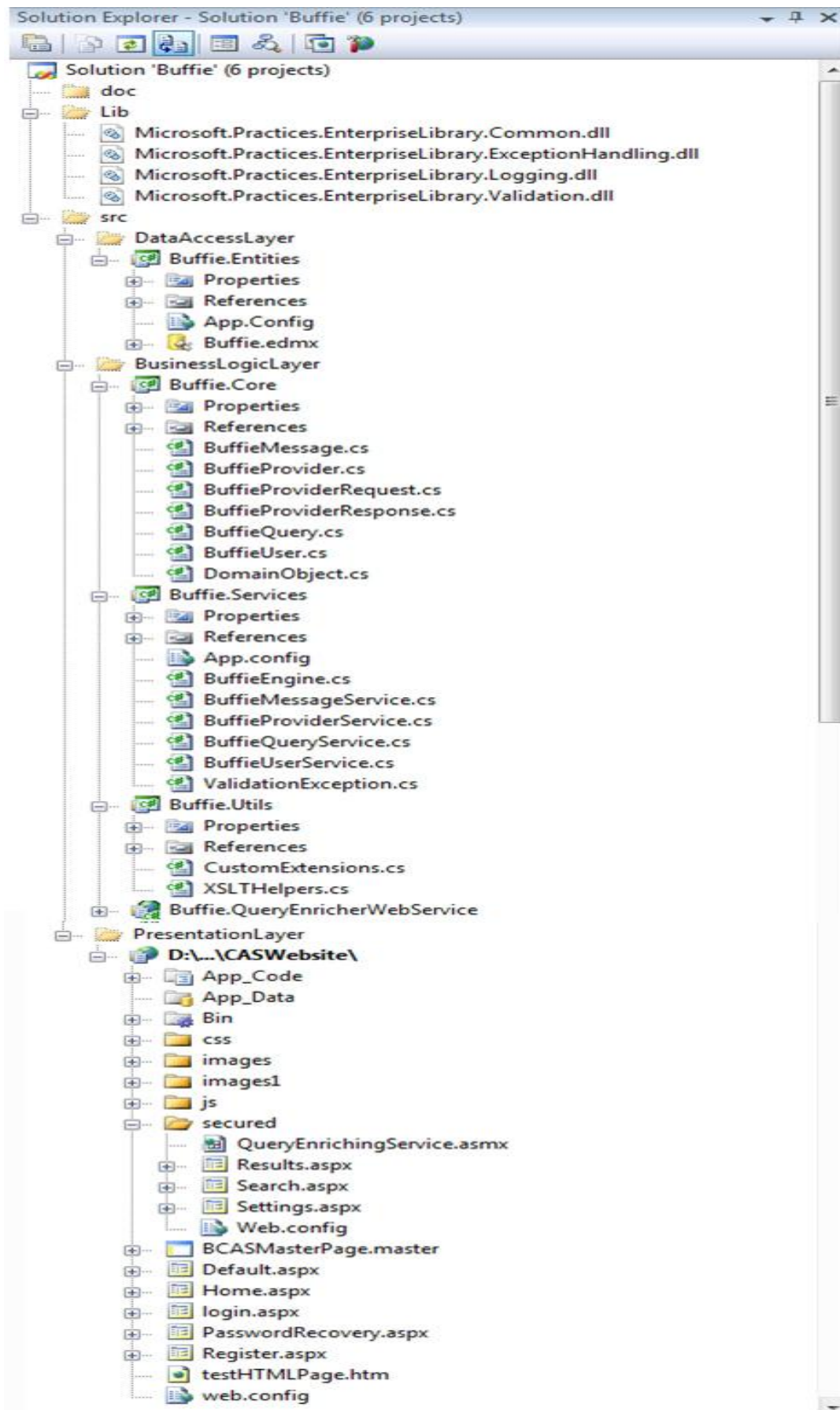


Figure 7.1: Layered Implementation of BUFFIE Architecture.

7.2 Implementation Principles in BUFFIE System

Converting the multi-layered system architecture of the BUFFIE system as described in chapter five into an implementation involves breaking the system into distinct and possibly without any overlapping features. To achieve the objectives of the architecture, we followed some established principles while developing the prototype system. Using the “Separation of Concerns” principle [174] we identified the following concerns to be developed as separate modules:

- Query Generation
- Query Enrichment
- Query Sending to Provider
- Response Receiving
- Biodiversity data transformation
- Results presentation

These are the significant features of BUFFIE framework that are important for the query processing and we used “Gang of Four” Object-oriented-design principles [175] to develop objects and factor them into classes at the right granularity for these modules. These modules need the knowledge about users, query, providers and data model used for communication which is obtained from the Domain Knowledge base component (DKB). This DKB is implemented using the functional programming principles in C# 3.0 language features like lambda functions, LINQ to XML and XSLT. Due to the nature of complexity in BUFFIE the software of the system is organized in layers and each layer represents a logical section of the framework.

7.3 Query Processing in the Business Logic Layer

This layer is the main nerve centre of BUFFIE framework, the modules developed in this layer include all the business objects, functional algorithms and calculations that makes the query processing to work and interact with the other layers. This section deals with all the steps involved in query processing right from generating to the response integration. This layer is implemented as three main assemblies namely “*BuffieCore*”, “*BuffieServices*”, “*BuffieUtils*” and a repository of “*config-files*”. We

had implemented this layer using Microsoft.Net3.5 framework, C# 3.0 language, LINQ to XML, XSLT templates and XPath Functions.

7.3.1 Buffie Core

BuffieCore objects represent the generic and abstract entities of the framework business rules. In our prototype system, we designed them to be purely from the business domain that assists in the middleware operations. It is completely independent of the data domain, for example these objects will have no dependency with the biodiversity data models. We followed the design of the BUFFIE architecture and implemented these components focusing on the required system operations at a generic level, rather than taking a data-centric approach. Figure 7.2 shows the core classes of the framework that uses the “information hiding” principle there by the other components are programmed to the interface exposed by these classes in properties and methods. The common features are defined in a base class with all the plumbing and used as a base class for this domain model.

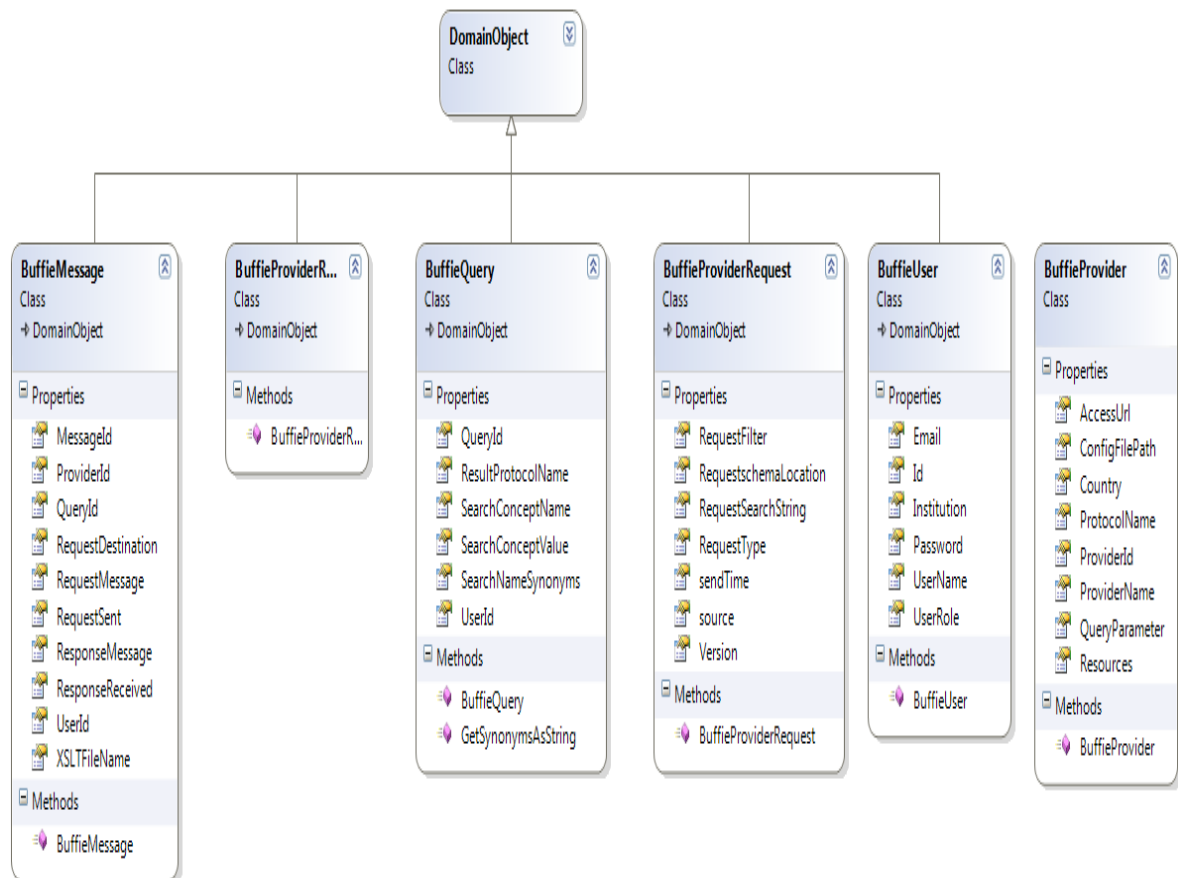


Figure 7.2: BuffieCore classes from framework Business domain.

7.3.2 Buffie Services and Utils

The modules in the *BuffieServices* and *BuffieUtils* components use the *BuffieCore* objects and orchestrate the whole query process of the common access system. Figure 7.3 shows the modules of the *BuffieServices* component that bridges the presentation layer, web services and data access layer.

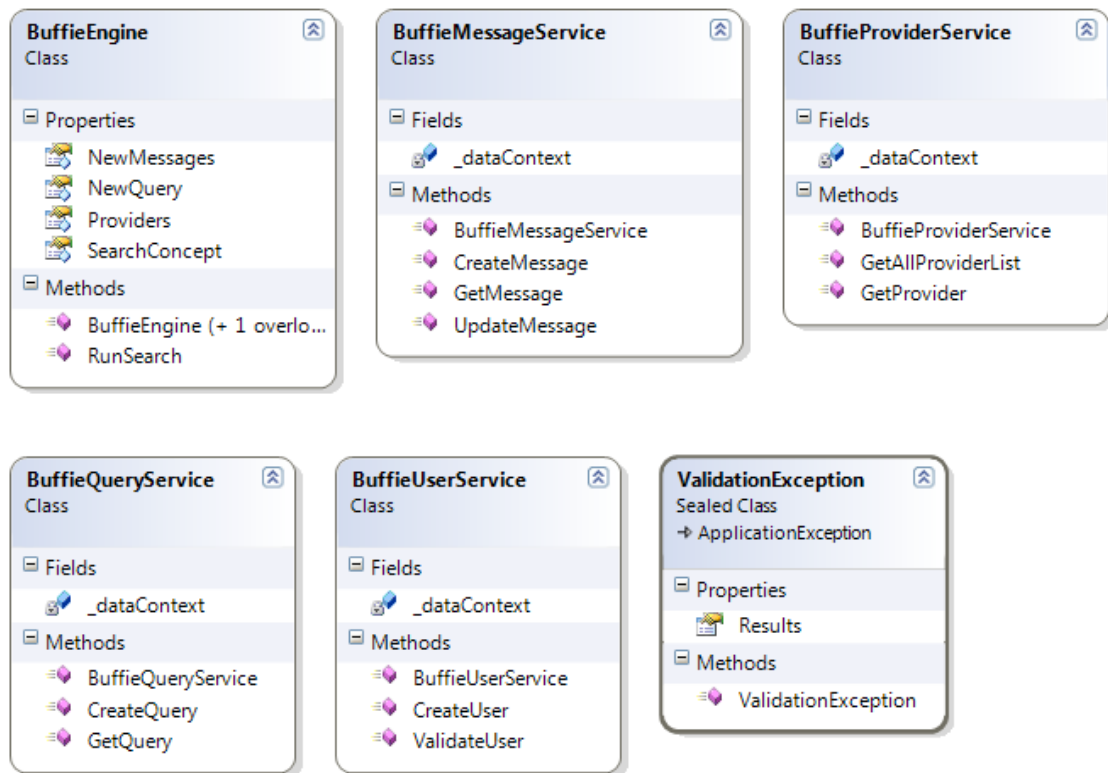


Figure 7.3: BuffieServices classes.

Figure 7.4 shows the main public scoped modules of the *BuffieUtils* component which is based on the functional approach that facilitates the *BuffieServices* to perform the functions. The workflow of the query process is explained with an example as follows:

The User Query is a search concept name on “**ScientificName**” and search concept value is “**Asthenargus helveticus**” these values are entered from the user screen and clicking the get synonyms button (as shown in figure 7.xx) would invoke the AJAX [176] codes of the Buffie system that communicates to the web service layer and get the synonyms for the scientific name as “aaaaaa”, “bbbbbb”. (A new example has to be introduced in this paragraph to demonstrate the service.)

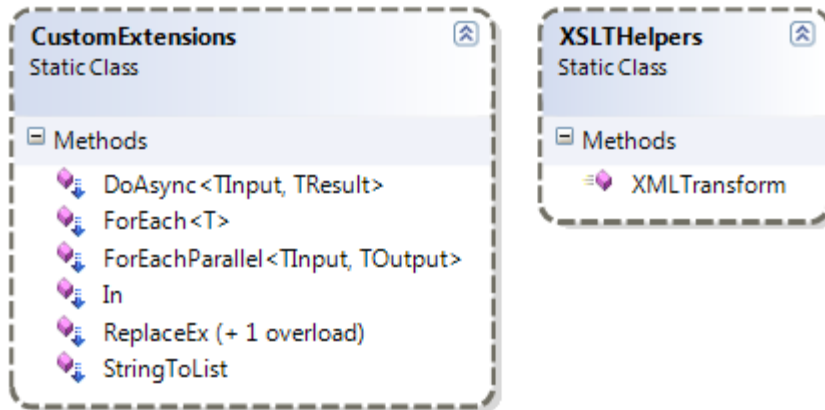


Figure 7.4: Buffie Utils Classes.

The *BuffieEngine* module receives the search concept name, concept value, synonyms and the required format (e.g. Darwin Core, ABCD) of the response. The module now knows what to search for and the next part is to find, where and how to search for answers to these queries. *DomainKnowledgeBase* (DKB) components provides the knowledge for where and how to search. Domain Knowledge Base is implemented as a set of XML and XSLT files stored in a *Config* file repository. This repository is an independent file structure which can be altered without affecting the Buffie common access application. The *BuffieProviderService* module provides the information about the provider like "accessurl", "resource", "xslt template" using this information *Buffie engine* uses the following algorithm to create request messages.

Algorithm:

- Step 1: Get the search *concept name, concept value, synonyms*
 Step 2: Get a list of providers from the *BuffieProvider Service*
 Step 3: For Each Provider
 {
 Loop Step 1: Get the Provider information accessurl, resourcename, xslt template.
 Loop Step 2: Create Request XML message by using *BuffieUtils* components
 Loop Step 3: Create a *BuffieMessage* for the current provider
 Loop Step 4: Save the new message to database and add the same to the *NewMessages* collection
 } end loop
 Step 4: Pass the *NewMessages* collection for asynchronous communication.

The *NewMessages* is a collection of *BuffieMessage* object and is stored in the local buffie database, where the request and response properties are implemented as XML

documents. Figure 7.5 shows a sample of the request XML created for the Darwin Core provider.

```
<request xmlns="http://digir.net/schema/protocol/2003/1.0" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:digir="http://digir.net/schema/protocol/2003/1.0"
  xmlns:dwc="http://digir.net/schema/conceptual/darwin/2003/1.0" xmlns:darwin="http://digir.net/schema/conceptual/darwin/2003/1.0"
  xsi:schemaLocation="http://digir.net/schema/protocol/2003/1.0">
  <header>
    <version>1.0.0</version>
    <sendTime>06/07/2009 09:17:57</sendTime>
    <source>BuffieV2.0</source>
    <destination resource="SNAILS">http://ip62.eti.uva.nl:8000/digir/DiGIR.php</destination>
    <type>search</type>
  </header>
  <search>
    <filter>
      <equals>
        <darwin:ScientificName>Asthenargus helveticus</darwin:ScientificName>
      </equals>
    </filter>
    <records limit="100" start="0">
      <structure schemaLocation="http://digir.sourceforge.net/schema/conceptual/darwin/full/2003/1.0/darwin2full.xsd" />
    </records>
    <count>true</count>
  </search>
</request>
```

Figure 7.5: Sample Request XML schema created by BuffieServices.

The *BuffieEngine* module implements the asynchronous communication with all the providers using the .Net Framework 3.5 system delegate `Func<Of <(T, TResult)>>` and assigning a lambda function (statements) to it. The lambda expressions use the `System.Net.WebResponse` object to send the request to the destination URL. The responses from the providers are updated in the corresponding *BuffieMessage* objects. The following block of the code segment shows how it works.

```
//for each message submit the request string asynchronously to
provider
Func<BuffieMessage, BuffieMessage> fl = uri =>
{
  WebResponse response = WebRequest.Create(uri.RequestDestination
+
uri.RequestMessage).GetResponse();
  uri.ResponseMessage = new StreamReader
(response.GetResponseStream()).ReadToEnd().ToString();
  uri.ResponseReceived = DateTime.Now;
  var Res = BMS.UpdateMessage(uri);
  return uri;
};
```

This module calls a method in the *BuffieMessageService* (`BMS.UpdateMessage(uri)`) module and updates the local database with the

response received from the provider. All the received responses are checked for validity and the next process is to transform the response data and merge them into a results.xml to be presented to the user. Each *BuffieMessage* object created in this module has got the information about the transformation details and the response XML messages and the procedure that invokes the transformation is shown as follows:

```

StringBuilder ResultsForQuery = new StringBuilder("<results>");
NewMessages.ForEachParallel(f1, result => result.ForEach(val =>
{
    if(val.XSLTFileName != "None")
    {
        ResultsForQuery.Append(XSLTHelpers.XMLTransform
            (val.ResponseMessage, val.XSLTFileName,
            null, null, null));
    }else
    {
        ResultsForQuery.Append("");
    }
}));
ResultsForQuery.Append("</results>");

```

The C# 3.0 extension methods were created and attached to the *NewMessages* object to perform recursive function calls. The XSLT transformation is performed on the results using .Net framework's *XslCompiledTransform* class. The results XML is loaded into an *XpathDocument* object [177] which provides a fast, read-only, in-memory representation of an XML document using the XPath data model. The Domain Knowledge Base (DKB) is implemented as XSLT templates and XML files under the "config" folder of the BUFFIE file system. Separate folders are used for each specific communication protocols as shown in the figure 7.6. The DKB provides the required XSLT template through the *XmlReader* class. When the Load method is called in *XslCompiledTransform*, it reads the data-transformation template through the *XmlReader* and creates an abstract syntax tree (AST) of the template including all its imports and includes. Once the data-transformation template is fully loaded, *XslCompiledTransform* can transform the input XML document.

Transformation of the input XML document to output involves the following steps:

1. Parsing the input document and building an in-memory XML tree representation.
2. Transforming the input XML tree to the output tree.

3. Serialisation of the output tree.

The transformation is applied to all the responses and appended to the results XML. These final results are returned to the presentation layer and to the clients of the web-service as the response to the initial query. The sample of the XSLT templates and the result XML is shown in Appendix D.

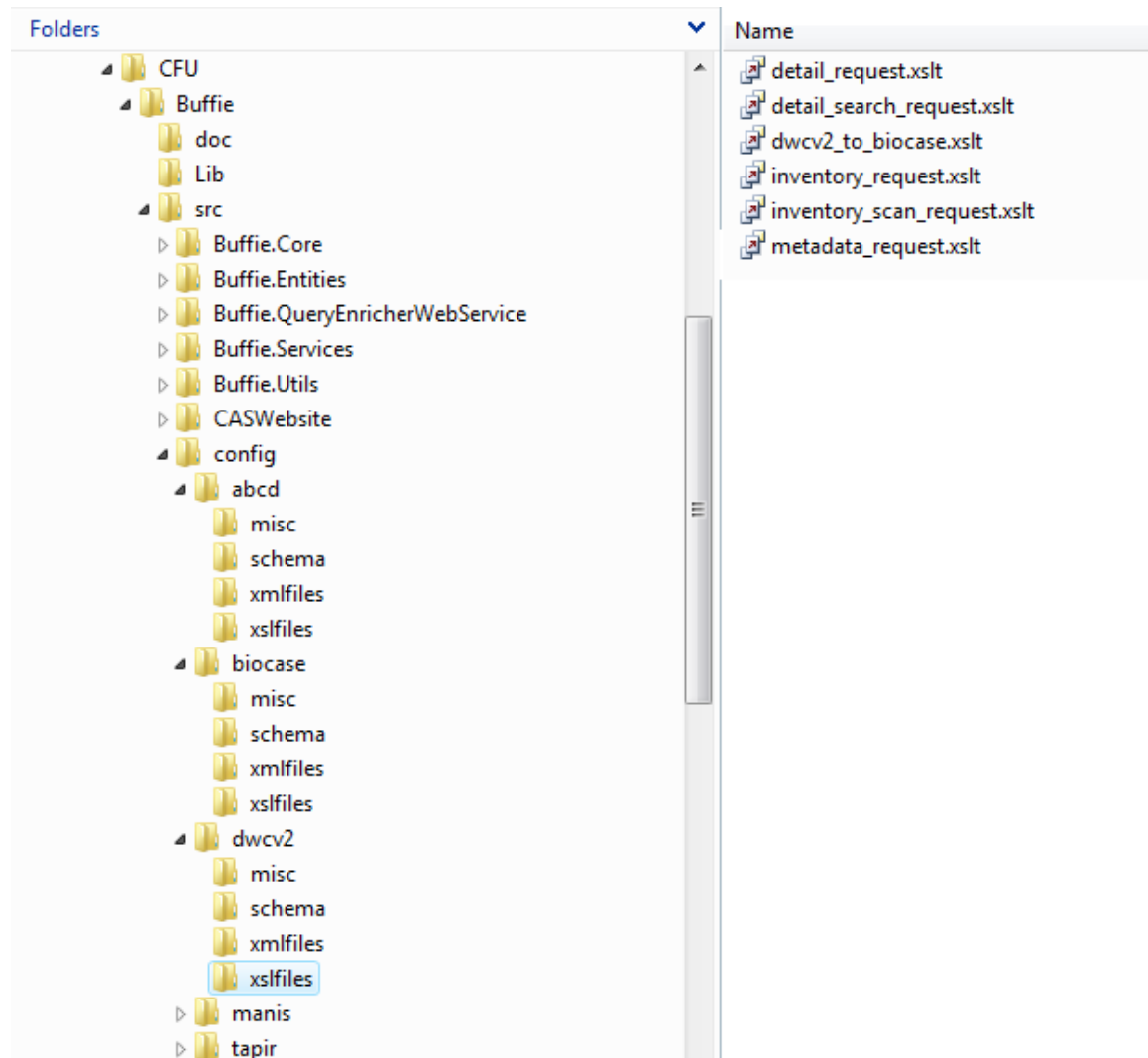


Figure 7.6: DomainKnowledgeBase Implemented as XSLT files in config folders.

7.4 The Data Access Layer of the Prototype

The Data layer of the BUFFIE system is classified into two categories namely the local *BuffieDatabase* and the independent heterogeneous and distributed data providers. The role of the data providers in the BUFFIE system is to provide a

response to the query in an XML format. They are located completely outside the boundary of the BUFFIE common access system. The local *BuffieDatabase* forms the main part of the data access layer and plays an important role in the functioning of the BUFFIE framework. The local *BuffieDatabase* is accessed by the Business layer using the Entity Data Model (EDM) framework. Figure 7.7 shows the Entity Data model created for the BUFFIE framework. This EDM is a conceptual model defining the entities and relationships used in the BUFFIE framework, and acts as a logical model that represents the underlying relational model which is implemented in Microsoft SQL server 2008. This provides a programmable interface using LINQ to Entities [178]. The user entity deals with the secured authentication of the application and for every new query a unique record is created with the query table and each query can have multiple messages. Each message is created for a specific query and a provider. The messages are stored as XML strings in the SQL database. All the data communications are recorded in this data store for debugging and for future analysis. The advantage of using this EDM gives the flexibility of changing the SQL storage model without affecting the modules in the business layer.

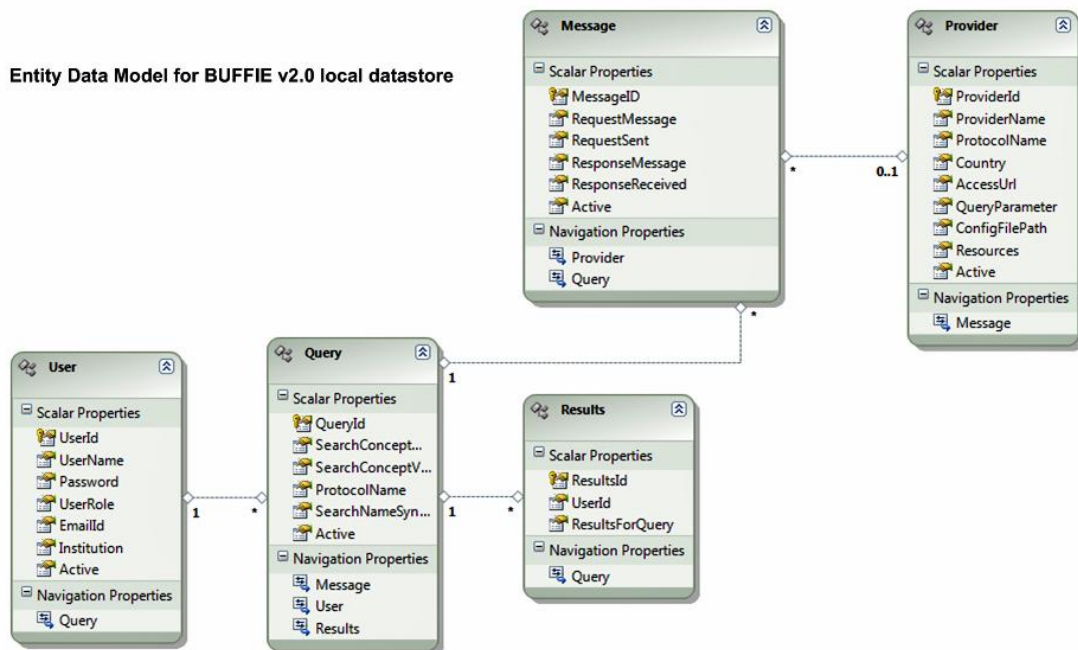


Figure 7.7; Entity Data Model for BuffieDatabase.

7.5 The Presentation Layer Prototype

BUFFIE framework is primarily a web based middleware system and hence there is not much user intervention in the process of the dataflow apart from the initial user query design and for the presentation of the results. Figure 7.8 shows the Query design page of the Buffie web application.

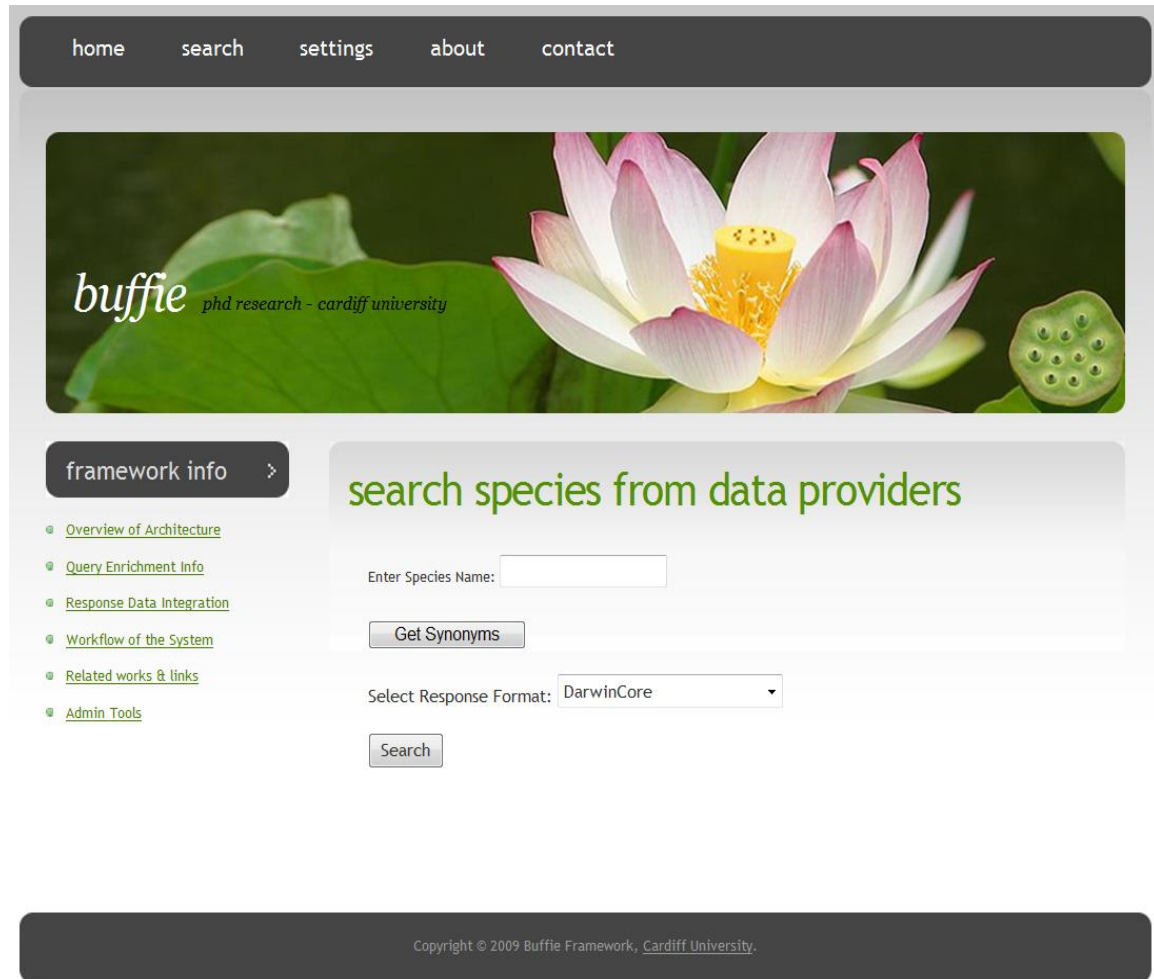
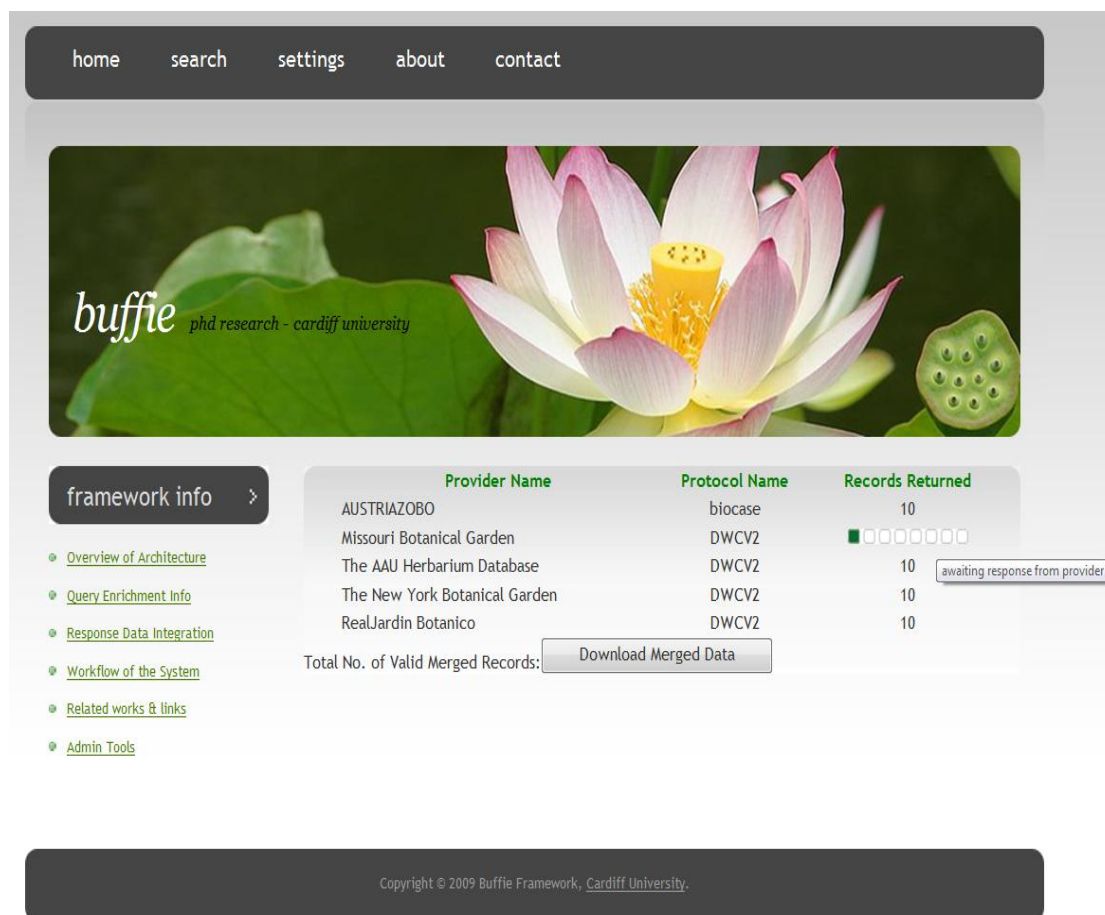


Figure 7.8: Query Design Page.

The following figure 7.9 shows the results received from the various data providers for the requested query. It displays the provider name and the data standard used for the communication and the number of valid records returned for the query. The display of the progress bar image in the column named “Records Returned” shows the outstanding status for response from the provider. The button “download merged data” presents the merged response from all the data providers.



home search settings about contact

buffie phd research - cardiff university

framework info >

- Overview of Architecture
- Query Enrichment Info
- Response Data Integration
- Workflow of the System
- Related works & links
- Admin Tools

Provider Name	Protocol Name	Records Returned
AUSTRIAZOBO	biocase	10
Missouri Botanical Garden	DWCV2	10 ■ □ □ □ □ □ □ □ □ □
The AAU Herbarium Database	DWCV2	10 awaiting response from provider
The New York Botanical Garden	DWCV2	10
RealJardin Botanico	DWCV2	10

Total No. of Valid Merged Records:

Copyright © 2009 Buffie Framework, Cardiff University.

Figure 7.9: Query Results Page.

Please refer to the examples, shown in the following sections 7.6 and 7.7 in conjunction with the outlook of the screenshots and description discussed in this section, which illustrates the working of the BUFFIE system in achieving interoperability of heterogeneous and distributed biodiversity data resources.

7.6 BUFFIE System Tested with Data Providers

This section describes the query process of the Buffie system with a real data example that is shown as an interoperability demonstration of the BUFFIE system. It involves the generation of a heterogeneous query based on the protocol and data standards used by five different biodiversity data providers spread across Europe and USA. The received responses from the heterogeneous data providers are integrated and displayed to the user. This was a data demonstration performed to test the interoperability of biodiversity data networks in the ENBI community, in a real-world

scenario using our Cardiff server communicating with the data providers, and the same server was being used by the clients' programs for harvesting the data from multiple data providers. The following test used a species search on a specimen commonly called as "*Fig Fruit*".



*Figure 7.10: Common Name: "Fruit-Fig"
ScientificName: "Guarea grandifolia DC."
Source of Image: Smithsonian Tropical Research Institute (ESP)*

When a user enters the search query in the Common Access web application, the middleware layer of the BUFFIE system generates the query using the scientificname for the Fruit fig as *Guarea grandifolia* DC. This source query from the user is saved in the Buffie database – dbo.Query table with a QueryID 5, as shown in the next figure 7.11.

QueryId	UserId	SearchConceptName	SearchConceptValue	ProtocolName	SearchNameSynonyms	Active
1	1	ScientificName	Guarea grandifolia DC.	DarwinCore	;	True
2	1	ScientificName	Guarea grandifolia DC.	DarwinCore	;	True
3	1	ScientificName	Guarea grandifolia DC.	DarwinCore	;	True
4	1	ScientificName	Guarea grandifolia DC.	DarwinCore	;	True
5	1	ScientificName	Guarea grandifolia DC.	DarwinCore	;	True
*	NULL	NULL	NULL	NULL	NULL	NULL

Figure 7.11: UserQuery stored in the Bufferie database.

In this demonstration five Data Providers from two different community networks are used as shown in the following figure 7.12. This information about the data providers consists of the biodiversity standards used by them and the access url for the resources. Other parameters required to make successful web service communication over the internet are also stored in the Bufferie database in the `dbo.providers` table.

ProviderId	ProviderName	ProtocolName	Country	AccessUrl	QueryParameter	ConfigFilePath	Resources	Active
1	AUSTRIAZOBO	biocase	Austria	http://81.10.184.26:9000/biocase/datab...	query=	D:\Sandbox\CFU\bufferie\... AUSTRIAZOBO	True	
3	Missouri Botanical Garden	DWCV2	USA	http://digir.mobot.org/digir/DIGIR.php?	doc=	D:\Sandbox\CFU\bufferie\... MOBOT	True	
5	The AAU Herbarium Database	DWCV2	Australia	http://192.38.46.42:80/digir/DIGIR.php?	doc=	D:\Sandbox\CFU\bufferie\... AAU-herbarium	True	
6	The New York Botanical Garden	DWCV2	USA	http://digir.nybg.org:1234/digir/DIGIR.php?	doc=	D:\Sandbox\CFU\bufferie\... NY	True	
10	RealJardin Botánico	DWCV2	Spain	http://taray.csic.es:6000/digir/DIGIR.php?	doc=	D:\Sandbox\CFU\bufferie\... MA	True	
**	NULL	NULL	NULL	NULL	NULL	NULL	NULL	

Figure 7.12: Heterogeneous data-providers information.

The Bufferie system uses this knowledge about the data providers along with the knowledge of the data derived from the query enriching process and generates the

request XML queries. In this demonstration, for the query with QueryID as 5, five data provider-specific request messages are created for the five different providers and the responses received from them are also stored in the database in the dbo.Message table as shown below.

MessageID	QueryId	ProviderID	RequestMessage	RequestSent	ResponseMessage	ResponseReceived	Active
21	5	1	<request xmlns...	2010-01-02 13:...	<?xml version='1.0' encoding...	2010-01-02 13:51:56.613	True
22	5	3	<request xmlns...	2010-01-02 13:...	The operation has timed out	NULL	True
23	5	5	<request xmlns...	2010-01-02 13:...	<?xml version='1.0' encoding...	2010-01-02 13:52:03.003	True
24	5	6	<request xmlns...	2010-01-02 13:...	<?xml version='1.0' encoding...	2010-01-02 13:52:05.770	True
25	5	10	<request xmlns...	2010-01-02 13:...	<?xml version='1.0' encoding...	2010-01-02 13:52:09.523	True
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure 7.13: XML Request and Response messages in Buffie system.

The expansion of the XML responses returned from the various data providers are shown using the XML Spy tool in the following figures 7.14 to 7.17.

XML generated by BioCASE wrapper software written by Markus Doering, Botanic Garden and Botanical Museum, Berlin.

response

- xmlns: http://www.biocase.org/schemas/protocol/1.3
- xmlns:x: http://www.w3.org/2001/XMLSchema-instance
- xmlns:sc: http://www.biocase.org/schemas/protocol/1.3 http://www.bgbm.org/biodivint/schema/protocol_1_3.xsd

header

content

- recordDropped: 0
- recordCount: 3
- recordStart: 0
- totalSearchHits: 3

DataSets

- DataSet (1)
 - OriginalSource: http://www.tdwg.org/schemas/abcd/1.2
 - DatasetDerivations: (empty)
 - Units (3)

UnitID	NamedCollections	Identifications	Gathering
1 100082971	Biologiezentrum, Linz-Dornach, A	<ul style="list-style-type: none"> Identification <ul style="list-style-type: none"> TaxonIdentified <ul style="list-style-type: none"> NameAuthorYea...: Guarea grandifolia DC. ScientificNameAtomized Identifier 	<ul style="list-style-type: none"> Gathering <ul style="list-style-type: none"> GatheringDateT... GatheringAgents GatheringSite
2 100082972			
3 100082973			

Figure 7.14: AustrianZobo data provider. (returns 3 records in a BioCASE data format)

{} dar...	{} {} darw...	{} da...	{} darwin:ScientificName	{} {} dar...	{} {} darwin:Phylo...	{} {} {} darwin: {} darw...	{} darwin: {} {} d...					
1	2007-10-19T18:54:12.000EST	NY	Herbarium	201444	Guarea grandifolia DC.	S	Plantae	Magnoliophyta (flowering plants)	Meliaceae	Guarea	grandifolia	DC.
2	2009-3-1 2T11:42:29.000EST	NY	Herbarium	754984	Guarea grandifolia DC.	S	Plantae	Magnoliophyta (flowering plants)	Meliaceae	Guarea	grandifolia	DC.
3	2009-3-1 2T12:01:06.000EST	NY	Herbarium	754985	Guarea grandifolia DC.	S	Plantae	Magnoliophyta (flowering plants)	Meliaceae	Guarea	grandifolia	DC.
4	2009-3-1 2T11:40:15.000EST	NY	Herbarium	865837	Guarea grandifolia DC.	S	Plantae	Magnoliophyta (flowering plants)	Meliaceae	Guarea	grandifolia	DC.
5	2009-3-1 2T12:14:0	NY	Herbarium	865939	Guarea grandifolia DC.	S	Plantae	Magnoliophyta (flowering plants)	Meliaceae	Guarea	grandifolia	DC.

Figure 7.15: New York Botanical Garden from USA, Herbarium data provider. (returns 10 records in a DWCV2 data format).

{} dar...	{} {} {} dar...	{} {} {} darwin:ScientificName	{} {} d...	{} {} {} {} {} darwin:Family	{} dar...	{} darwin: {} d... {} {} {} d...					
1	2008-01-0 1T00:00:00.0Z	MA	MA	639193-1	Guarea grandifolia DC.	S	Plantae	Meliaceae	Guarea	grandifolia	DC.
2	2008-01-0 1T00:00:00.0Z	MA	MA	635685-1	Guarea grandifolia DC.	S	Plantae	Meliaceae	Guarea	grandifolia	DC.
3	2008-01-0 1T00:00:00.0Z	MA	MA	637805-1	Guarea grandifolia DC.	S	Plantae	Meliaceae	Guarea	grandifolia	DC.
4	2008-01-0 1T00:00:00.0Z	MA	MA	637809-1	Guarea grandifolia DC.	S	Plantae	Meliaceae	Guarea	grandifolia	DC.
5	2008-01-0 1T00:00:00.0Z	MA	MA	637847-1	Guarea grandifolia DC.	S	Plantae	Meliaceae	Guarea	grandifolia	DC.

Figure 7.16: RealJardin Botanico data provider from Spain. (returns 9 records in a DWCV2 data format).

Of the five data providers used in testing of Buffie system four have responded with suitable and successful response and one has timed out during the query request process as shown in figure 7.13. The various XML responses were integrated using XSLT templates and the integrated results are stored as a XML in the BUFFIE database as shown in figure 7.17 below. This merged data is sent to the client web application in the XML format requested by the user as a response to their initial query.

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left displays the database structure for WINVISTAPC (SQL Server 10.0.2531), including Databases, System Databases, Database Snapshots, Buffie, Database Diagrams, Tables, System Tables, and user tables (dbo.Message, dbo.Provider, dbo.Query, dbo.Results, dbo.User). The main window displays the 'WINVISTAPC.Buffie - dbo.Results' table with the following data:

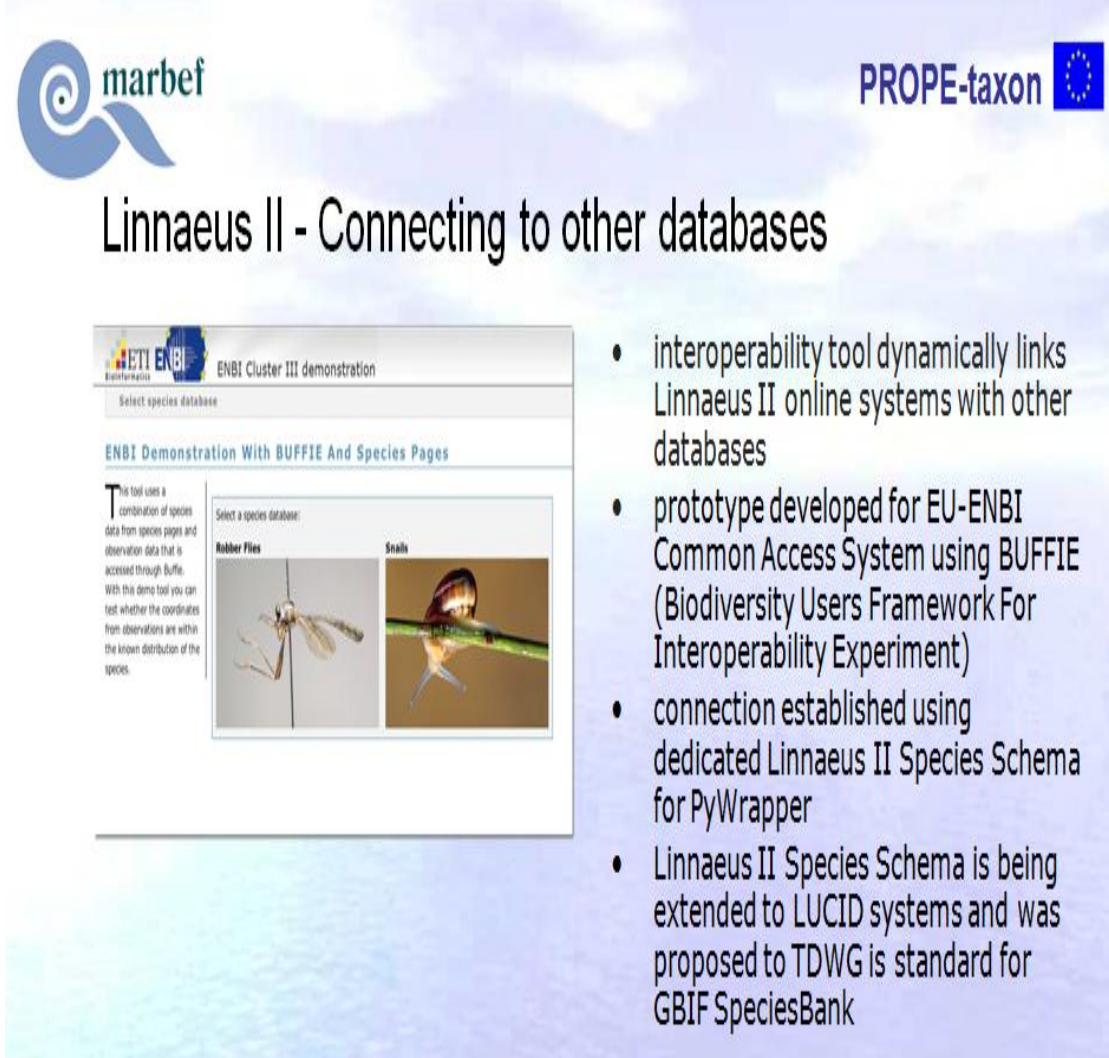
ResultsId	UserId	QueryId	ResultsForQuery
1	1	1	<results>
2	1	2	<results>
3	1	3	<results>
4	1	4	<results>
5	1	5	<results>
*	NULL	NULL	NULL

Figure 7.17: Merged results stored in Buffie system.

7.7 BUFFIE System's Interoperation with Linnaeus II

The previous section showed how BUFFIE can request and receive data simultaneously from distributed heterogeneous data providers. This section describes another demonstration of the use of BUFFIE to meet the objective of achieving interoperability in the biodiversity domain among three types of distributed components in three different countries: data providers, a data mediator and a data user. As shown in figure 7.18, the demonstration, which was part of the ENBI Cluster III project activities, involves a user using a web-based client program called Linnaeus II hosted on a server at ETI in Amsterdam. The user's query originates from the Linnaeus II program and is sent to the BUFFIE middleware framework's common access system, which acts as a mediator, hosted on a Cardiff University server.

The BUFFIE system has knowledge about the data protocols used by Linnaeus II and by the participating data providers which can provide the answers to the user's query, which concerns data from Israel. Hence the BUFFIE system requests the required species information from the data providers, receives the responses and transforms them to the required format. The transformed result is displayed in Linnaeus II along with other information.



Linnaeus II - Connecting to other databases

- interoperability tool dynamically links Linnaeus II online systems with other databases
- prototype developed for EU-ENBI Common Access System using BUFFIE (Biodiversity Users Framework For Interoperability Experiment)
- connection established using dedicated Linnaeus II Species Schema for PyWrapper
- Linnaeus II Species Schema is being extended to LUCID systems and was proposed to TDWG is standard for GBIF SpeciesBank

Figure 7.18: BUFFIE used by Linnaeus II to connect to providers databases.

This is further explained in the discussion section of Chapter 8 and in section 8.3.1. All the images used in this section are sourced from the published documents of Marbef [179] and other European projects presented in international seminars and biodiversity meetings [180].

The figure consists of two side-by-side screenshots of web pages. The left screenshot shows the 'Linnaeus II web' interface for the species *Buliminus labrosus*. It features a navigation menu on the left with options like 'Index', 'Species', 'Higher groups', 'Atlas', 'Glossary', 'Literature', 'Help', and 'WBD index'. The main content area displays the species name, author '(Olivier, 1804)', a shell image, and various attributes: Type (Shell with shell), Description of Shell (Height 35 mm, Diameter 17 mm, Dentition none, Surface smooth), Faunal Element (Asian), and Origin (Native). The right screenshot shows the 'ETI Bioinformatics' website interface for the same species. It includes a search bar, a 'Description' section with a shell image, and a 'Nomenclature' section with fields for 'Accepted name' (Buliminus labrosus), 'English name', and 'Synonyms'. A map of Israel is also visible, indicating the species' distribution.

Figure 7.19: BUFFIE demonstration with species data.

The above images show species data that was received from the BUFFIE middleware as a response to the query, being displayed using Linnaeus II web pages and the same species data are being displayed in an external website of ETI Bioinformatics, in Amsterdam.

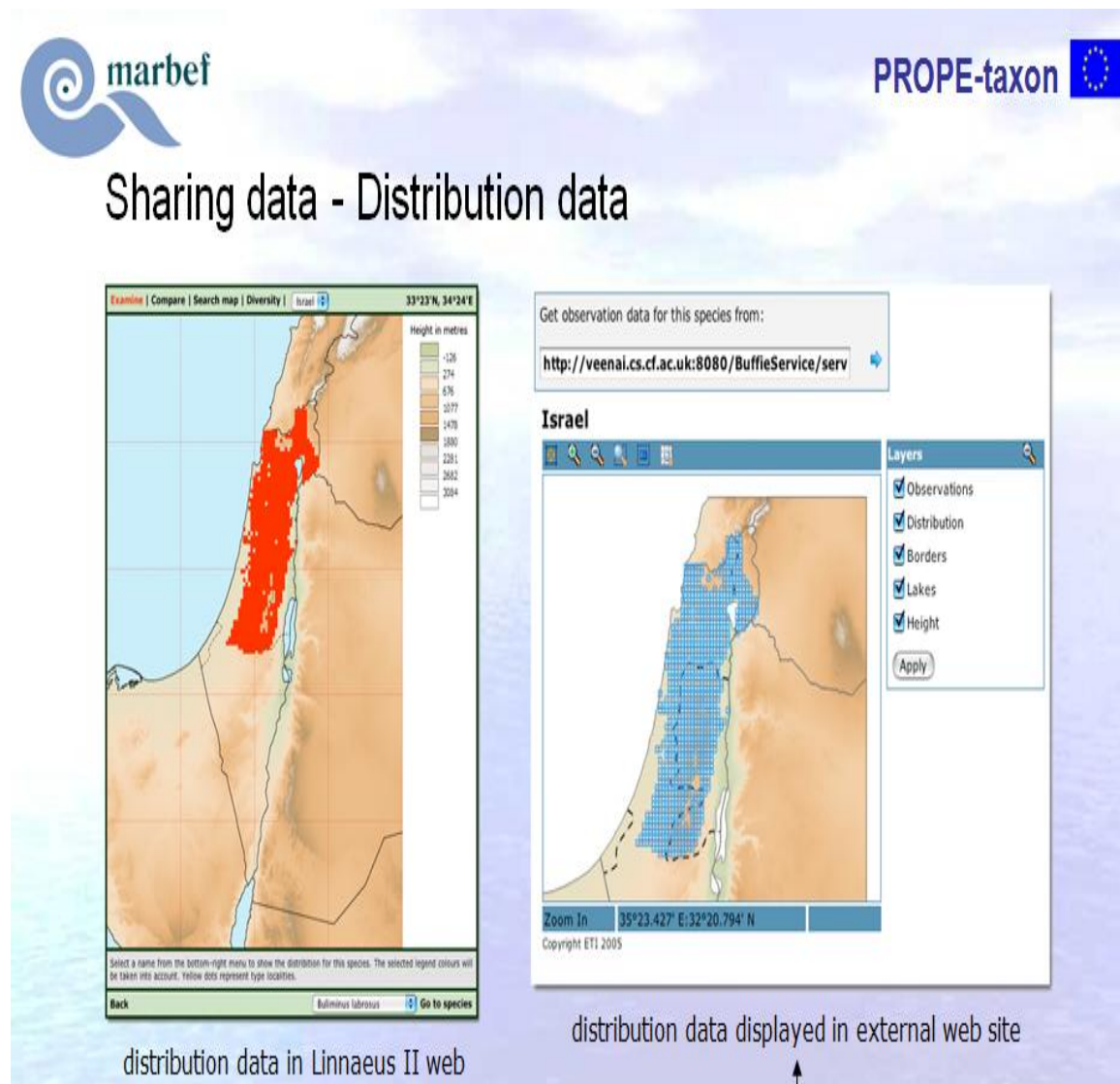


Figure 7.20: BUFFIE demonstration with Linnaeus II.

The images in figure 7.20 and 7.21 shows the species observation data collected using the BUFFIE middleware framework which was hosted on a server at Cardiff University. The BUFFIE system merged the heterogeneous data responses from the species query and the co-ordinated information about the species is plotted against maps of Israel to create a species distribution map of that country. The client program displayed the distribution data using Linnaeus II web page and also using an external website.

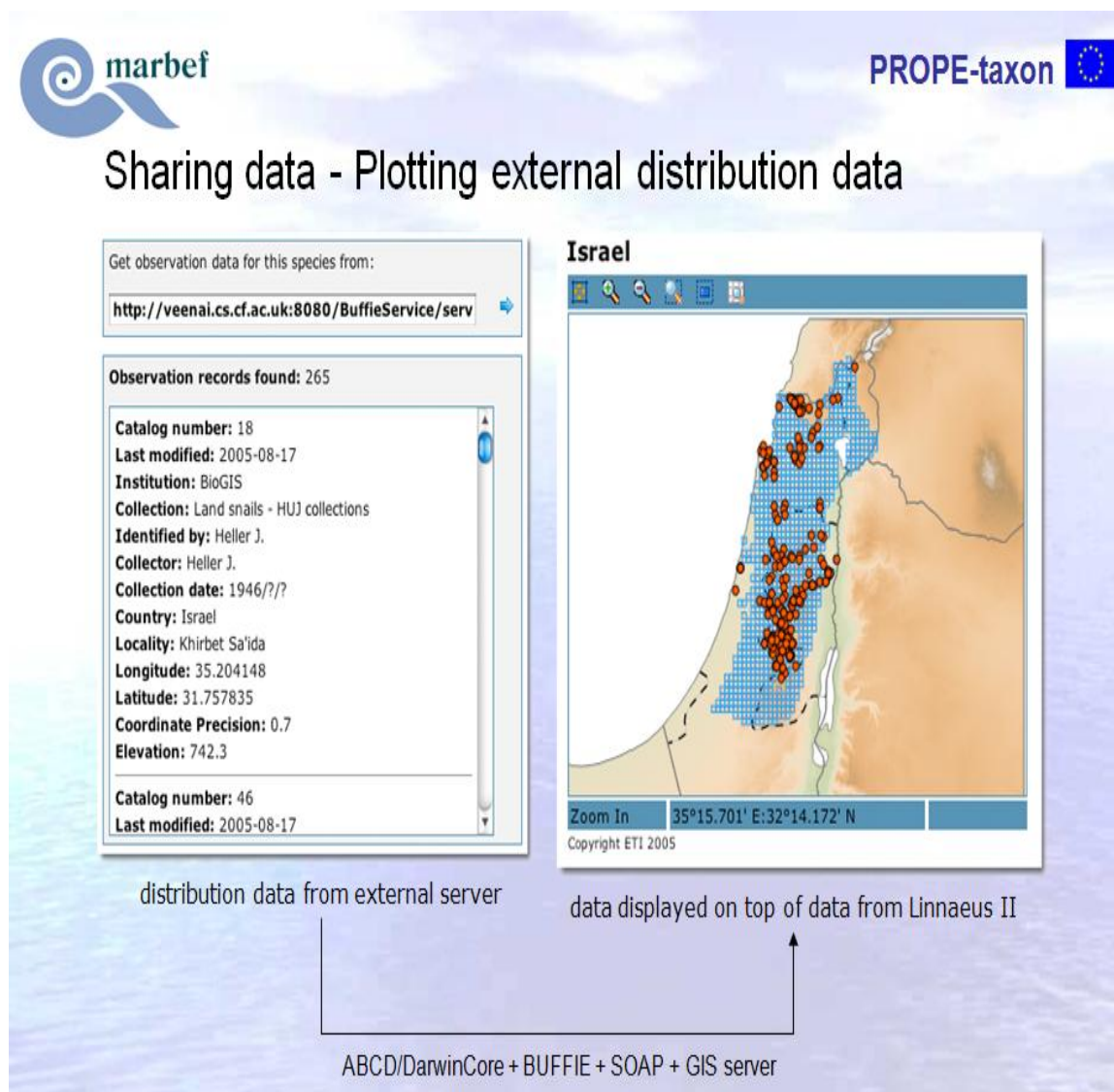


Figure 7.21: Heterogeneous data merged using BUFFIE system used by client application.

The preceding images shown in this section (from Figure 7.18 through to 7.21) are discussed here as evidences of real world client applications communicating to the BUFFIE system to achieve interoperability across the heterogeneous biodiversity data providers. The client systems such as Linnaeus II and other applications used the BUFFIE system that was hosted on the Veenai server at Cardiff University <http://veenai.cs.cf.ac.uk:8080/BufieService/services>. The client applications communicate with BUFFIE using a web service interface.

CHAPTER 8

Evaluation & Discussion

8.1 Introduction

This chapter is used to assess the research project. We evaluate the functionality and the extensibility of the BUFFIE system and then discuss the suitability of the architecture and design of the framework for interoperation of biodiversity data communication. The system was implemented with a three-tier architecture and some flavours of web service orientation. The prototype components of BUFFIE v1.0 were developed on the Java platform and the most recent version BUFFIE v2.0 was developed on the Windows platform using the Microsoft .Net3.5 framework,

following a hybrid of object-oriented and functional design. The domain knowledgebase was developed as XSLT templates and XML files in repositories. The complex biodiversity data modelled using different XML data standards used by the providers are required to interoperate in the BUFFIE system. Whenever disparate systems are required to exchange information there will be a need for a test programme to evaluate the extent of interoperability that can be achieved [22]. This section is to analyse and discuss the research on various aspects and in particular whether the set objectives of the research are met based on the evaluation criteria.

8.2 Evaluation

The end result of the system was measured to prove that the objective of the system has been achieved. As stated, interoperability can only be achieved by designing and building systems against a defined interoperability requirement, and then maintaining that interoperability throughout the system changes and upgrades [22]. This evaluation is performed against the hypothesis and objectives shown in chapter 1 which were to show that interoperability among heterogeneous biodiversity databases can be achieved, by developing a new framework using a service oriented system architecture with domain knowledge expressed in a knowledgebase, and could be demonstrated by:

1. designing, developing and implementing a suitable framework,
2. designing the components and integrating the services required to perform the interoperation process, and
3. developing a Web-based prototype application to verify the hypothesis using test datasets.

The prototype system was deployed on a Windows platform and SQL server 2008 was used for the database. The efficiency of the BUFFIE system and the effectiveness of the results from our research are measured in terms of following:

- Functionality of the BUFFIE framework with regard to its objective of achieving the interoperability of biodiversity data and helping users to make queries and receive the responses in the required format.
- Extensibility of the BUFFIE system with regard to its suitability to a dynamic environment, where the data providers can be added or removed and the data communication protocols are changed.
- The architecture of the BUFFIE system with regard to its design, performance, maintainability and the role as query enrichment and response integration tool.
- Construction of the Domain Knowledgebase (DKB) with regard to its structure and role as a repository which stores the schema mapping information.
- Choice of XML as the data model used in our interoperable system, and the protocols used to communicate with the data providers.
- Implementation of the BUFFIE system and demonstration of the BUFFIE system being used to access real time biodiversity data by the users.
- Various applications of the BUFFIE system and the type of users who can use it.

8.2.1 Functionality of the BUFFIE Framework

The BUFFIE system is a valuable common access tool for the users who want to search biodiversity information based on the species name without having any knowledge about the data providers and their communication protocol. It can also be used as an Application Programming Interface to access or retrieve biodiversity data from the distributed and heterogeneous data providers. The BUFFIE prototype system provides a user interface for designing the initial query through a web application. Another interface is provided for the client programs through a web-service. The main functionalities of the BUFFIE system are:

- Query Enrichment
- Query Generation
- Query Messaging
- Response Schema Integration
- Presentation of the results

This query enrichment in the context of the search concept's data is very important to find the correct answer for the queries from the data providers. The nature of biodiversity data is that it may have multiple names for the same species across different regions of the world. The user might search a scientific name and the data provider might have indexed the required data against a synonym name. This functionality enables a successful query result and it provides increased visibility of the data in the data providers. Query generation function allows the user to submit one query to the BUFFIE common access system and using that information it generates multiple heterogeneous queries to the providers. Query generation consults the Domain Knowledge Base to create the request messages. Query messaging functionality provides the asynchronous sending and receiving of the request and response messages respectively. This functionality uses the recursive functions and multithreading which increases the performance of the communication between the BUFFIE system and the data providers.

Response schema integration is responsible for making the semi-automatic structural and semantic transformation of the heterogeneous XML responses received from the data providers. This process is configurable through the application "config" files and the schema matching templates are produced at the design time using the auxiliary knowledge provided by the domain experts. The transformed messages are merged continuously in the run-time and presented to the user as an XML file in the required schema. All these functionalities are performed as a middleware operation of the query processing, so that the user need not have any knowledge about the heterogeneous data providers and the method or data standards used for communication.

8.2.2 Extensibility of the Framework Model

The BUFFIE system was created using the following design principles like:

- High cohesion: building the classes such that all the related functionalities like query enrichment, query generation and schema matching templates are grouped together.
- Low coupling: each module is encapsulated and the public interfaces are clearly defined so that the dependency is minimised when any one component needs to be updated.
- Separation of concerns and modularity: the components are separated into modules that can be reused based on the functionality.

The design patterns followed in BUFFIE were a hybrid of object-oriented and functional patterns. These principles and patterns allow BUFFIE to achieve the requirements and make the system maintainable and extensible. The framework is configurable by using the settings of the system variables in the XML files. It allows the adding or removing of the data providers without affecting the query processing functionality of the system. Due to the dynamic nature of the biodiversity data, the corresponding XML representation in the standards could change. This problem is resolved in BUFFIE system because of the extensible feature, because a new request XML schema format and schema matching template can be introduced into the “config” folders of the system replacing an old schema. The BUFFIE system does not follow a universal data model approach for representing the data. Instead it uses the XML data structure of the data providers and transforms them dynamically to the required format of the user. The BUFFIE core system has the operational logic of the framework and the domain knowledge is fetched from a separate knowledge base. This makes the BUFFIE framework flexible and reusable to other data domains with minimal changes and adding a new knowledgebase to the framework.

8.2.3 Architecture of BUFFIE Framework

Designing and developing a common access system to resolve the interoperability problem in biodiversity data networks presents numerous entities to work with and also involves complex business rules to be defined. A structured guidance is needed to create the components using architectural patterns. The architecture of the BUFFIE system describes the organisational structure of the system and it specifies the responsibilities of all the components. The architectural design for the BUFFIE system is shown on Figure 5.1 which is a “Multi-layered Web based Service Oriented Architecture”. The BUFFIE architecture includes two main subsystems:

- Query Enriching: Figure 5.5 shows the architecture diagram for the user query enrichment and section 5.5 describes its functionalities.
- XML Schema Matching: Figure 5.2 shows the architecture diagram for the schema matching process for the responses from the data providers.

Though the BUFFIE framework architecture shows all the components involved in the framework, the Common Access System functions like a middleware which does all of the query processing and is designed in the Business logic layer.

- The presentation layer is a very thin component which has two types of interface to make a query and receive the responses: The web page for the end users to access the Buffie Common Access System and a Web service for the client programs such as analytical tools to communicate with BUFFIE.
- Buffie Core Components: includes the business rules and is responsible for the main middleware framework which is independent of the data domain.
- Buffie Services: exposes the Buffie Core objects and orchestrates the workflow of the query processing, right from query enriching, query generation, response integration and results presentation. Buffie Utils provide the helper functions to the services of the framework.

- Domain Knowledge Base: consists of XML repository and XSLT templates which are based on a data domain model. These are functional modules that take the input and transform them based on the rules and provide the output result.
- Data Providers: the data providers of the BUFFIE system are independent, heterogeneous and distributed. They provide response for the query in the form of XML messages over the internet.
- Data layer and BUFFIE local database: Buffie database supports the operation of the main framework and is used for data persistence. The data layer was designed using the .NET3.5 Entity framework model

8.2.4 Domain Knowledge Base (DKB)

The Domain Knowledgebase (DKB) has been developed as a set of XML and XSLT files and is included in the BUFFIE system with a specific folder structure under the config folders. Biodiversity schema matching information of the participating schemas are built into these XSLT templates as functions. This schema matching logic is based on the mapping tables of the various biodiversity XML standards as shown in appendix A and the auxiliary information provided by the domain experts. The XSLT templates were built during design time but the transformation of data during runtime is continuous as the DKB fetches the right transformation template based on the providers' response format. The provider information from the DKB is used by the query generation to produce multiple heterogeneous queries from the user query. The user enters the detail for What to Query? DKB provides the knowledge of:

- How to send query
- Where to send the queries and
- How to transform and merge the responses

Domain Knowledge Base is extensible in design and functionality, for example if a new provider with a proprietary XML data standard needs to be included in the BUFFIE system, then the access details should be added to the providers list and a

new XSLT transformation template with the mappings logic for the schema should be added to the config folders of the BUFFIE system. The common access system will automatically pick up these details during query generation and the response transformation. Similarly existing templates can be updated and replaced into the DKB component without any knock-on effect on the other components of the BUFFIE framework.

8.2.5 Applications of the BUFFIE Common Access System

Colossal amounts of biodiversity data are captured and stored in digital databases. These databases are distributed, with different data representation and they use different data standards for data exchange. End users may not have enough knowledge to access these data or about the data format of the data providers. The use of existing applications allows them to query the data from providers that are participating in a homogenous data networks like set of Darwin Core providers or set of ABCD providers. Our BUFFIE common access system provides the flexibility of allowing the users to query various data providers who use heterogeneous data standards for communication. It also allows them to specify the format of the response so that the heterogeneous responses from the providers are transformed and integrated as required. The system can be used in two different ways:

1. User search for species information: An end user after successful authentication can log in to the Common Access system using the web application forms. In this approach the user can design the query by fetching search concept and search value and the required format of response. The user can invoke the “GetSynonyms” button to enable the query enriching process to his query. Then the Business logic of the BUFFIE system performs all the required process and returns the response in an XML format in the web application.
2. Data Harvesting and Analysis by client tools: The second approach is a programmatic interface, in which the *BuffieServices* component can be accessed through a client program using the published web-services. The web

service methods receive the query in the format of name/value string type parameters and return the result as XML string to the calling client programs.

8.2.6 Implementation and Verification of the BUFFIE System

The first version of the common access system prototype BUFFIE v1.0 system was implemented in a java platform using the tool Borland Jbuilder enterprise 2005. The main application components were created as Java classes and the web application using JSP. The system does not persist the state of the query process and rather it acts like a middleware system that presents all the responses to the user. The transformation components were built using XSLT, Xpath and JDOM parser. This web application and web service were deployed on the Apache Tomcat server on windows 2003 server. The latest version of the common access system prototype BUFFIE v2.0 is implemented in Microsoft .Net3.5 framework, and developed using Visual studio 2008. The BUFFIE framework components are developed using C#3.0, ASP.NET3.5 and the domain knowledge base is implemented using XSLT, XML and XPath. The advancement in this version is that it provides a better system with much newer functionalities like:

- Better design and architecture using hybrid patterns of object-oriented and functional design.
- Data persistence using an Entity Data Model and LINQ to XML technologies and local Buffie Database using SQL server2008.
- Flexible and pragmatic approach that use knowledge base to achieve interoperability like
 - Query enriching using the search concept value.
 - Response schema matching with functions.
- Better extensibility of components, performance due to multithreaded programming and lesser codes.

This system is deployed on a server using Internet Information Services (IIS7.0) and SQL Server2008.

8.3 Discussion

Interoperability has been the most challenging and most important requirement when it comes to querying information from heterogeneous and distributed data resources. This problem is further magnified when the data resources are autonomous and the volume of data is increased. Many research projects are following different approaches to resolve this issue, such as a data warehouse approach, a data standards approach, or building a universal schema for the data domain. But none of these approaches has succeeded in solving the interoperability problem fully or sustaining the level of success achieved, over a period of time. This is certainly due to the dynamic nature of the data and the representation of it in the data providers.

Unlike many other research projects that typically apply one specific approach or technique, BUFFIE applies a hybrid of software engineering technologies and a comprehensive approach including enriching request queries and integration of heterogeneous responses for achieving interoperability among biodiversity data networks. In the context of heterogeneous data resources interoperability standards are the main and primary step to accomplish data exchange. The downside of standards is that they have a tendency to quickly evolve away from the initially perfected norm, where the modifications are conditioned by participating systems capabilities, workflows and changing business requirements of the data providers. As the data providers are autonomous and independent, Buffie provides a middleware approach to solve interoperability issue. Earlier interoperability projects in the biodiversity domain were either a provider-centric approach or user-centric approach in which all the participating data providers agree to use a particular common standard that has given rise to community of networks.

The flexible architecture used in Buffie is a middleware approach where the main advantage is the extreme independence for the data providers and the users. One downside of using XSLT templates in the domain knowledge base is that the number of templates required increases rapidly with an increase in the number of protocols used in the Buffie system. In this research we aimed to resolve the interoperability issues using our framework, between communities of data providers that already

adopted one of the established data standards of the domain. This network of communities and their established data standards, determines the number of XSLT templates (rule-sets) required for interoperation. If there are n input formats and r output groups of standards, then the number of XSLT combinations required is usually determined by the formula $C(n, r) = n! / r! (n - r)!$. In this case, where pairs of formats require conversion templates, $r = 2$ and $C(n) = n(n-1) / 2$.

For example in the biodiversity domain there are about 4 or 5 data standards as described in Section 2.5. Assuming 1 transformation template for each pair then there needs to be $C(5) = 5 * 4 / 2 = 10$ XSLT templates. This assumes that a single template can be designed to perform two-way transformations between the data standards, but such reverse transformations may not be possible in all cases, so twice this number of one-way conversion templates may be needed. Where possible the transformations can be routed using a central schema and in other cases direct transformation templates are to be created. Combining these two transformation possibilities the requirement of the XSLT templates can be optimised. In our example of 5 data standards this can vary from a minimum of 4 (if the central schema is one of the providers' standards) to a maximum of 10 templates. The advantage of using a central schema for transformation within the Buffie architecture between the input and output schemas helps to significantly reduce the number of translations required for interoperability. BUFFIE exploits the most prominently used biodiversity data communication schemas like Darwin Core and ABCD and the domain experts published knowledge for creating the mapping relationships between the various concepts.

ABCD is used as a default universal data standard or central schema for routing transformations between protocols in the system. We conducted experiments with data from the providers and compared the result, since this is a new framework and we have no other similar approach available for comparison, we evaluated the overall interoperability results using examples. Please refer to the examples shown in the Chapter 7 sections 7.6 & 7.7 that demonstrates the interoperability of biodiversity data.

8.3.1 Verifications of Goals Achieved

Based on the objectives of this research identified in chapter 1 (section 1.5, 1.6) we have achieved the following goals and verified them using tests:

- Designed and developed a Flexible Framework for Interoperability between heterogeneous and distributed biodiversity data resources, which were using various XML data standards for communication.
- Tested the BUFFIE common access system, Query Enriching part using the SPICE Species 2000 web service. The Domain Knowledge base was implemented using XML and XSLT templates and using LINQ technologies the data schema matching were performed in the BUFFIE v2.0 system.
- Demonstrated the frameworks architecture and working of the application at the international biodiversity seminars and meetings [136].
- The BUFFIE system was used to support a network of research teams in three countries:
 - Researchers using the Israel Biodiversity Information system (BioGIS) used the BUFFIE system to harvest data from other data resources irrespective of the standards of the data providers [181].
 - Researchers using the Linnaeus II online system from ETI Bioinformatics in Amsterdam, which used the BUFFIE system to access species data from Darwin Core and ABCD data providers [180].
 - The BUFFIE application was hosted on an Apache Tomcat web server running on a Windows Server 2003 platform at the Cardiff University URL: <http://veenai.cs.cf.ac.uk:8080/BufferService> and was accessed by the client programs described above [183]. The results are shown in Chapter 7 sections 7.7.

The data providers for these client programs through the BUFFIE common access system were “IsraelSnails” from Amsterdam server in a Darwin Core format and “AustrianZobo” from Austria server in a ABCD format.

8.4 Applicability and Limitations

Although the BUFFIE system has successfully demonstrated the possibility of both structural and semantic interoperability between heterogeneous and distributed biodiversity data bases that use XML data standards for communication, it has some limitations with its applicability. This approach would be best suited in domains where variety of schemas exists and for which the solution of moving rapidly to a common schema is unlikely. The limitations are primarily due to practical difficulties and also due to the nature of the data in the biodiversity domain.

- The BUFFIE framework currently allows only those data providers who use XML format for data communication in the biodiversity data domain. This might exclude legacy systems that use flat file structures or objects for data exchange. This can be mitigated by using wrappers at the data providers' end that could convert their proprietary data structure into an XML data structure.
- The current implementation of the BUFFIE framework uses XSLT and LINQ technologies in its flexible architecture to achieve structural and semantic interoperability. The semantic interoperability can be enhanced by replacing XSLT transformations rule-set by well defined and fit for purpose ontologies that can mediate between the related concepts defined in the heterogeneous data standards.
- The BUFFIE prototype system can only interoperate between the data providers whose XML communication knowledge is provided to the knowledgebase component of BUFFIE during the design time. It uses a core schema as reference for example ABCD format to create new mapping rules used by the domain knowledge base component of the system.
- The developer of the knowledgebase modules needs to be aware of the relevant biodiversity data concepts and will have to update their systems to accommodate new data structure schemas as biodiversity standards progress.

- Though data modelling using ontologies can be very useful for semantic data integration, this was not included in the design of the BUFFIE architecture as there were no established biodiversity domain-specific ontologies that will fit our purpose for the interoperability. If any ontology model is to be used with BUFFIE then it would need to be converted to RDF XML and XSLT using tools like Protégé.
- The research aim is to develop a prototype as a “proof-of-concept” for the objectives of this research and hence will use the only “species scientific name” concept for query enrichment and query search. This can be extended to other concepts in the biodiversity domain.
- This interoperable approach can be implemented in other data domains such as health-care and, astronomical data where a variety of data standards exist but the number of well established standards in practical use are limited. For example in the health care domain the most used clinical data codes are 'Systematized Nomenclature of Medicine Clinical Terms' (SNOMED-CT) and NHS-Read Codes [182]. The maintenance problem of this approach could increase with the scale of the different XML standards available for interoperation. However this can be leveraged by dynamically choosing one of the best possible schemas from the available set of schemas as the core schema for data transformation.

CHAPTER 9

Summary, Conclusion and Future Work

This chapter concludes the thesis by briefly summarizing the research work, presenting the conclusions of the thesis and addressing the future scope for further work and development.

9.1 Thesis Summary

We have presented a flexible framework (BUFFIE) to interoperate between the distributed and heterogeneous biodiversity data resources that communicate using XML data standards. A general overview of the interoperability problem was discussed and how the various levels of interoperability like technical, structural and semantic interoperability were described. This research is specifically concerned with the interoperability problem in biodiversity XML standards. The solution proposed was providing a flexible framework that would allow structural and semantic interoperation of biodiversity protocols using software engineering technologies. The causes of the interoperability and different approaches to solve them and the related

projects in the biodiversity domain were presented. After that we explained the different technologies of software engineering that can be applied to resolve the interoperability issues. This approach involves a new framework that is flexible and is based on the multi-layered web based service oriented architecture. Based on the architecture a prototype system called BUFFIE was developed that interoperates between the XML data standards used by the data providers. This is achieved using the following functionalities of BUFFIE:

- User query enriching based on the search concept value
- Multiple query generation for heterogeneous data providers
- Asynchronous messaging service between BUFFIE and data access point
- Response schema integration.

Query enriching is performed using the AJAX technique calling the synonym web-service. *BuffieCore* objects, *BuffieServices* and *BuffieUtils* orchestrate the complete query processing and they use *BuffieDatabase* for data persistence during the different stages of processing of BUFFIE system. *BuffieCore* objects represent the main framework and define the business logic rules for query processing and are independent of biodiversity data domain.

Query generation and the schema integration use the *Domain Knowledge Base* (DKB). The DKB is specific to the biodiversity data domain and is created using XML configuration files, XSLT mapping templates that were generated in the design time of the system. The published mapping details for the data standards and the auxiliary information provided by the experts were used by the developers to generate the mapping logic. LINQ to XML and XPathdocument components from .Net framework 3.5 were used for transforming and integrating the heterogeneous responses from the providers. The first version of the prototype BUFFIE v1.0 was developed on a Java platform and deployed on Apache Tomcat server. JDOM parser was used for implementing XML and XSLT transformation. A more stable version of this system is deployed on a production server at Cardiff University (Veenai) [183], which was used by other client programs for accessing species data from different

data providers. BUFFIE v2.0 is an advanced version and is implemented using C#3.0, LINQ to XML components and functional programming using extension methods, lambda functions and asynchronous threading model. The results were tested and found to have achieved the objectives of interoperation between the heterogeneous and distributed biodiversity data resources that were using different XML data structures for communication.

Thus this Buffie framework has served to demonstrate the hypothesis originally formulated in section 1.4, namely that

“Interoperability among distributed, autonomous and heterogeneous biodiversity databases can be achieved by developing a new framework that exploits the synergies of multi-layered service-oriented system architecture, domain knowledge expressed in a knowledgebase designed using XML, XSLT and object-oriented functional design of components.”

9.1.1 Publications from Thesis

During the course of this research work, communication with the computer science and biodiversity communities was maintained through seminar meetings, conference presentations and writing documentation and manuscripts. This has resulted in the following peer reviewed publications:

- *“The BUFFIE Architecture”* was presented to the domain experts at the Biodiversity conference meeting in Stockholm, Sweden in 2005. This publication was made after an analysis of the related literature review, interoperable technologies and has contributed to validate the proposed system design and framework model to the experts of the community. Please refer to [136] in bibliography that relates to the work reported in chapters 4, 5 and 6.
- *“Web Based Middleware Framework for Interoperability between Heterogeneous, Distributed Biodiversity Data Resources”* was presented at the International Software Engineering conference at Innsbruck, Austria in 2007. This presentation and publication of the paper relates to our prototype

implementation based on the proposed BUFFIE framework architecture. The material contributes to validate the achievements of our research work towards the objectives of our research in interoperability among the participating data providers. Please refer to [181] in the bibliography that relates to the contents of this thesis from chapters 2 to 8.

- “*A Service Oriented Architecture with Domain Knowledge Base for Interoperability of Heterogeneous Distributed Biodiversity Resources*” was presented at the International Software Engineering conference at Cambridge-MIT, Massachusetts, USA in 2009. This presentation and publication of the paper contributes to the overall research work. This material is used in the thesis right from stating the objectives of the research, proposed architecture of BUFFIE framework, querying process and integration of response data. The outcome of this presentation was used in the discussion and verification sections of this thesis work. Please refer to [149] in the bibliography; elements of this paper are to be found in chapters 2 to 9 of this thesis.

9.2 Conclusions

The novelty of this research work is in that, interoperation in biodiversity databases are achieved at three different levels using a new flexible framework. At the biodiversity domain level this is the new framework for achieving interoperability between heterogeneous data providers, using XML based communication protocol. At the software engineering and implementation viewpoint the research shows the novel uses of service oriented architecture with Functional programming that uses Lambda functions, LINQ to XML and XSLT technologies to achieve interoperability. At the data level using the expert knowledge of biologists and matching tables for XML standards are used to produce the mappings logic that can be used for interoperation by integration. The previous approaches either use a global universal schema to accommodate all the available standards, or demand the data resources to accept query and return responses in a specific standardized format.

In this approach there is no centralized global schema, instead the framework is made flexible, that the conversion or transformation logic is updated with any required changes. This feature is best suited with the dynamic nature of the biodiversity data and also helps to preserve the autonomy of the data providers. The framework design and architecture clearly separates the business logic rules of the query processing system from the data domain knowledge base. This makes the system more generic and gives the flexibility of plugging in a new knowledge base to work in a different data domain. Another advantage of the framework is that the data units used in the different components of the BUFFIE system are either objects with serialization capability or plain XML document object. The *BuffieDatabase* persists the query messages and the results of all the response transformations in XML data types.

9.3 Future work

The research work presented in this thesis can lead to a number of exciting possibilities for future work in many ways. There are both biodiversity domain based data interoperability issues and software engineering based framework interoperability issues. These issues can be addressed to overcome the limitation described in the section 8.5 and to increase the effectiveness of the framework to achieve better structural and semantic interoperability between heterogeneous and distributed biodiversity data resources that use XML standards. The biodiversity research community is already working towards the open framework and conducting workshops and special interest group discussions to work in the area of biodiversity data interoperability using standards, protocols and open architectures. For example the GIGAS project promotes the coherent and interoperable development [184], [185]. We suggest the following lists for future work:

- The scope of query enriching can be expanded by including more third party web services that can enhance the value of the search concepts. For example the search concept can be enhanced by adding some value using the attributes about the user to choose the data providers accordingly that suit his requirement.

- Although we have exploited the fact that the data providers have implemented specific mechanisms to query their data through wrappers, using SQL and SPARQL to directly query the biodiversity data from the participating data providers could be considered for future work.
- The semi-automatic process of schema matching can be improved so that the mapping logic is formed based on the data by using multiple knowledge bases spanning across many interrelated domains.
- The scope of the biodiversity XML data standards covered can be increased. Biodiversity domain ontologies can be constructed and they can be used into the BUFFIE architecture in place of the domain knowledge base. Using biodiversity ontologies could improve the level of semantic interoperability achieved by the BUFFIE system and reduce the number of transformation templates required for interoperation.
- The Buffie v2.0 prototype can be improved by adding more data providers who can fetch the data for the users query through BUFFIE system. This can also be hosted by an appropriate organisation to provide a production service with high availability of data to users.

Bibliography

- [1]. **James, L. E., Meredith, A.L. and Ebbe, S. N.** Interoperability of biodiversity databases: Biodiversity Information on Every desktop. *Science Magazine*. 2000, Vol. 289, pp. 2312-2314.
- [2]. **Conn, B.J.** Information standards in botanical databases-the limits to data interchange. *Telopea*. 2003, Vol. 10, 1, pp. 53-60. http://www.rbgsyd.nsw.gov.au/__data/assets/pdf_file/0005/72707/Tel10Con053.pdf.
- [3]. **W3C.** Extensible Markup Language (XML) 1.0 W3C. *World Wide Web Consortium*. [Online] 26 November 2008. <http://www.w3.org/TR/2008/REC-xml-20081126/>.
- [4]. **Jones, A. C., White, R.J., X, Xu., Pittas, N., Gray, W.A., Fiddian, N.J.** SPICE: A Flexible Architecture for Integrating Autonomous Databases to Comprise a distributed catalogue of Life. s.l. : Springer-Verlag, 2000. pp. 981-992.
- [5]. **Wieczorek, John.** The Mammal Networked Information System. *MaNIS Portals*. [Online] University of California, Berkeley, CA 94720., 18 Mar 2009. [Cited: 10 Nov 2009.] <http://manisnet.org/portals.html>.
- [6]. **Holetschek, J. and Döring, M.** Biological Collection Access Services. *BioCASE Portal*. [Online] Botanischer Garten und Botanisches Museum, Berlin, Germany, 2005. [Cited: 10 Nov 2009.] <http://www.biocase.org/>.
- [7]. **He, Hao.** What Is Service-Oriented Architecture. [Online] 30 September 2003. [Cited: 11 02 2009.] <http://www.xml.com/lpt/a/1292>.
- [8]. **Gruber, T R.** Toward principles for the design of ontologies used for knowledge sharing. Formal Ontology in Conceptual Analysis and Knowledge Representation. [ed.] Nicola Guarino and Roberto Poli. *International Workshop on Formal Ontology*, March 1993. http://www-ksl.stanford.edu/KSL_Abstracts/KSL-93-04.html.
- [9]. **Ram, Sudha and Park, Jinsoo.** Semantic Conflict Resolution Ontology (SCROL) An Ontology for Detecting and resolving Data and Schema-Level Semantic Conflicts. *IEEE Transactions on Knowledge and Data Engineering*. 2004, Vol. 16, 2.

-
- [10]. **Preece, Alun., Gray, Alex., Bench-Capon, Trevor., Cui, Zhan.** The KRAFT Architecture for Knowledge Fusion and Transformation. *Knowledge-Based Systems*. April 2000, Vol. 13, 2-3, pp. 113-120. <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.30.7912>.
- [11]. **California Biodiversity Council.** Scientific Definitions of Biodiversity. *California Biodiversity Council*. [Online] California Biodiversity Council, 2008. [Cited: 31 August 2009.] http://biodiversity.ca.gov/Biodiversity/biodiv_def2.html.
- [12]. **Heidorn, Bryan P.** Biodiversity and Biocomplexity Informatics: Policy and Implementation Science versus Citizen Science. *2nd ACM/IEEE-CS joint conference on Digital libraries*. 2002, pp. 362 - 364.
- [13]. **Paterson, T and Kennedy, J.** Approaches to Storing and Querying Structural Information in Botanical Specimen Descriptions. *21st Annual British National Conference on Databases (BNCOD21)*. 2004, pp. 80-91.
- [14]. **J, Kennedy., Hyam, R., Kukla, R., Patterson, T.** A Standard Data Model representation for Taxonomic Information. *A Journal of Integrative Biology -OMICS*. June 2006, Vol. 10, 2, pp. 220-230.
- [15]. **Wiederhold, Gio.** Intelligent integration of information. *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*. ACM New York, NY, USA, 1993, pp. 434 - 437.
- [16]. **Bisby, Frank A.** The Quiet Revolution: Biodiversity Informatics on the Internet. *Science*, 289. Science/AAAS - Cambridge, September 2000, Vol. 289, 5488, pp. 2309 - 2312.
- [17]. **European Topic Centre on Biological Diversity.** *Eionet*. [Online] Eionet: European Information and Observation Network, 26 08 2009. [Cited: 10 Nov 2009.] <http://biodiversity.eionet.europa.eu/>.
- [18]. **Wegner, Peter.** Interoperability. *ACM Computing Surveys*. Rhode Island, ACM New York, NY, USA, March 1996, Vol. 28, 1, pp. 285 - 287. <http://doi.acm.org/10.1145/234313.234424>.
- [19]. **Saarenmaa, Hannu.** Sharing and Accessing Biodiversity Data Globally. *ESRI User Conference*. San Diego, March 2006. <http://www.esri.com/news/arcuser/0206/biodiversity1of2.html>.
- [20]. **GBIF Secretariat.** GBIF Portal. *GBIF Portal*. [Online] GBIF Secretariat, Universitetsparken 15, DK-2100 Copenhagen, Denmark, 2009. [Cited: 31 August 2009.] <http://www.gbif.org/>.
- [21]. **Beller, Aaron.** Local GIS Biodiversity Portals. *ENBI - ENBI community*. [Online] ENBI, 15 September 2005. [Cited: 31 August 2009.] http://circa.gbif.net/irc/Download/kqerA1J_moGCXrhcFjRxPI1q6sjeQGdU/FkPuSj6eLHj2p04_NH_UWx7uLXPq7T16/g24pYxtvF37u/GIS%20portals%20at%20ENBI_Sp2Keu.pdf.

- [22]. **Pridmore, J. and Rumens, D.J.** Interoperability-how do we know when we have achieved it? *Command, Control, Communications and Management Information Systems, 1989*. IEEE explore Digital Library, May 1989, pp. 192-205. Software Sciences Ltd,.
- [23]. **Bassman, Mitchell J., Dahlke, Carl and Russell, Lucian.** Development of an interoperability tool for software engineering environments. *Proceedings of the fifth Washington Ada symposium on Ada*. ACM New York, NY, USA, 1988, pp. 49 - 57.
- [24]. **Park, Jinsoo and Ram, Sudha.** Information systems interoperability: What lies beneath? *ACM Transactions on Information Systems*. ACM New York, NY, USA, October 2004, Vol. 22, 4, pp. 595 - 632.
- [25]. **Berendsohn, Walter G.** Access to Biological Collections Data. *Biodiversity Informations Standard TDWG*. [Online] ABCD Task Group , 23 August 2007 . [Cited: 10 Nov 2009.] <http://www.tdwg.org/activities/abcd/>.
- [26]. **Wieczorek, John.** DarwinCore Group - DwC. *Biodiversity Data Standards TDWG* . [Online] TDWG Task Group, 19 Feb 2007 . [Cited: 10 Nov 2009.] <http://www.tdwg.org/activities/darwincore/charter/>.
- [27]. **Lakshmanan, Laks V. S. and al., et.** Languages for multi-database interoperability. *Proceedings of the 1997 ACM SIGMOD international conference on Management of data*. ACM New York, NY, USA, 1997, pp. 536 - 538. ISBN:0-89791-911-4.
- [28]. **Microsoft.** Microsoft BizTalk Server . *Microsoft* . [Online] Microsoft , 2009. [Cited: 10 Nov 2009.] <http://www.microsoft.com/biztalk/en/us/overview.aspx>.
- [29]. **Altova.** Altova MissionKit – Suite of XML, Database & UML Tools. *Altova XML Editor*. [Online] Altova, 2009. [Cited: 10 Nov 2009.] <http://www.altova.com/>.
- [30]. **Davis, Leigh A., Payton, Jamie and Gamble, Rose.** How system architectures impede interoperability. *Proceedings of the 2nd international workshop on Software and performance*. Ottawa, Ontario, Canada: ACM New York, NY, USA, 2000, pp. 145 - 146. ISBN:1-58113-195-X.
- [31]. **Weber, Darcy Wiborg.** Data Topology and Process Patterns for Distributed Development. *LNCS - Software Configuration Management*. Springer Berlin / Heidelberg, January 2003, Vol. 2649/2003, pp. 206-216,. 978-3-540-14036-8.
- [32]. **Kelkar, A and Gamble, R. F.** Understanding the Architectural Characteristics behind Middleware Choices. *1st International Conference in Information Reuse and Integration*. November 1999. <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.24.2654>.
- [33]. **Jakobovits, Rex.** *Integrating Autonomous Heterogeneous Information Sources*. Dept. of Computer Science & Engineering, University of Washington. 1997. <ftp://ftp.cs.washington.edu/tr/1997/12/UW-CSE-97-1> .

-
- [34]. **Object Management Group.** *CORBA Architecture and Specification, July Revision 2.0.* s.l. : Object Management Group, 1995.
- [35]. **Brockschmidt, Kraig.** *Inside Ole (Microsoft Programming Series).* s.l. : Microsoft Press, May 1995. Vol. 2nd. ISBN-13: 978-1556158438.
- [36]. **Wiederhold, Gio.** Interoperation, Mediation, and Ontologies. *International Symposium of Fifth Generation Computer Systems (FGCS94).* Tokyo, Japan, 1994, pp. 33-48.
- [37]. **Wiederhold, Gio.** Mediation in Information Systems. *ACM Computing Surveys.* ACM, NY, USA, June 1995, Vol. 27, 2.
- [38]. **Duwairi, R. M., Fiddian, N. J. and Gray, W. A.** A Multiple View Definition System for Supporting Interoperability among Heterogeneous and Autonomous Databases. *10th ERCIM Database Research Group Workshop on Heterogeneous Information Management.* Prague, ERCIM, 1996. <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.53.1309&rep=rep1&type=pdf>.
- [39]. **Karunaratna, D.D, Gray, W.A. and Fiddian, N.J.** Organising Knowledge of a Federated Database System to Support Multiple View Generation. *5th KRDB Workshop (Knowledge Representation meets Data Bases).* Seattle, Citeseer, May 1998. <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.51.5829&rep=rep1&type=url&i=0>.
- [40]. **Litwin, Witold, Mark, Leo and Roussopoulos, Nick.** Interoperability of Multiple Autonomous Databases. *ACM Computing Surveys.* ACM, NY, USA, September 1990, Vol. 22, 3.
- [41]. **Research Collections.** *COMPONENT DATABASE SYSTEMS.* [ed.] Klaus Dittrich and Andreas Geppert. s.l. : MORGAN KAUFFMAN, 2000. p. 294. http://www.elsevier.com/wps/find/bookdescription.cws_home/677928/description?vopenmenu=4. ISBN-13: 978-1-55860-642-5.
- [42]. **Bontempo, Charles and Zagelow, George.** The IBM Data Warehouse Architecture. *Communications of the ACM.* ACM, NY, USA, September 1998, Vol. 41, 9.
- [43]. **Terrasse, Marie-Noelle, et al.** Do We need metamodels and Ontologies for Engineering Platforms? *Proceedings of the 2006 international workshop on Global integrated model management.* ACM New York, NY, USA, Shangai, 2006, pp. 21 - 28. ISBN:1-59593-410-3.
- [44]. **Malucelli, Andreia, Palzer, Daniel and Oliveira, Eugénio.** Ontology-based Services to help solving the heterogeneity problem in e-commerce negotiations. *Electronic Commerce Research and Applications.* 4 November 2005, Vol. 5, 1, pp. 29-43 .

- [45]. **Su, Xiaomeng and Gulla, Jon Atle.** An information retrieval approach to ontology mapping. *Data & Knowledge engineering* . 2006, pp. 47-69.
- [46]. **Australian Government, DEWHA.** Biodiversity and its value. *Biodiversity Publications*. [Online] Paper no. 1, 1993. <http://www.environment.gov.au/biodiversity/publications/series/paper1/what.html>.
- [47]. **The Linnean Society of London.** Carl Linnaeus - The father of modern plant and animal classification . *The Linnean Society of London*. [Online] The Linnean Society of London, 2009. [Cited: 31 August 2009.] <http://www.linnean.org/index.php?id=51>.
- [48]. **Butz, Stephen D.** *Science of Earth Systems*. 2. s.l. : Delmar Learning , 2007. p. 655. ISBN-13: 978-0766833913 .
- [49]. **Giri, Chandra Prasad., Shrestha, Surendra., Foresman, Timotthy W., Singh, Ashbindu.** GLOBAL BIODIVERSITY DATA AND INFORMATION. *United Nations Economic and Social Council*. [Online] 2009. [Cited: 10 Nov 2009.] <http://www.unescap.org/stat/envstat/stwes-26.pdf>.
- [50]. **CBD Secretariat** . Convention on Biological Diversity. *Convention on Biological Diversity*. [Online] Secretariat of the Convention on Biological Diversity , 16 Nov 2009. [Cited: 16 Nov 2009.] <http://www.cbd.int/convention/>.
- [51]. **Royal Botanic Garden Edinburgh** . Biodiversity Collections Index. *biodiversitycollectionsindex.org*. [Online] Royal Botanic Garden Edinburgh , 2008. [Cited: 31 August 2009.] <http://www.biodiversitycollectionsindex.org/static/index.html>.
- [52]. **Lindström, Jan.** Database model for taxonomic and observation data. *Proceedings of the 2nd IASTED international conference on Advances in computer science and technology*. ACTA Press Anaheim, CA, USA, 2006, pp. 316 - 321. ISBN ~ ISSN:1482-7905 , 0-88986-545-0.
- [53]. **SABIF.** Standard and Protocols . *SouthAfricanBiodiversityInformationFacility*. [Online] Department of Science&Technology - , 2009. [Cited: 1 September 2009.] http://www.sabif.ac.za/index.php?option=com_content&view=article&id=17&Itemid=49&showall=1.
- [54]. **ISO Central Secretariat** . About ISO . *International Standards for Business, Government and Society*. [Online] ISO Central Secretariat , 2009. [Cited: 20 Nov 2009.] <http://www.iso.org/iso/home.htm>.
- [55]. **DCMI.** Dublin Core Metadata Initiative. *DublinCore*. [Online] 1995. <http://www.dublincore.org/>.
- [56]. **Botanic Garden Community.** International Transfer Format for Botanic Garden Plant Records . *TDWG- Biodiversity Information Standards*. [Online] Botanic Garden Community, 01 Oct 1987. [Cited: 20 Nov 2009.] <http://www.tdwg.org/standards/102/>.

- [57]. **Conn, Barry J and Croft, J.R.** HISPID3. *Biodiversity Information Standards TDWG*. [Online] 1989. [Cited: 20 Nov 2009.] <http://plantnet.rbgsyd.nsw.gov.au/HISCOM/HISPID/HISPID3/H3.html>.
- [58]. **White, Richard J and Allkin, Robert.** A Language for the definition and exchange of biological data sets. *Mathematical and Computer Modelling*. Published by Elsevier Ltd, June-July 1992, Vol. 16, 6-7, pp. 199–233. doi:10.1016/0895-7177(92)90163-F.
- [59]. **School of Biological Sciences at the University of Reading, UK.** ILDIS. *International Legume Database & Information Service*. [Online] 2008. [Cited: 20 Nov 2009.] hosted by Cardiff School of Computer Science. . <http://www.ildis.org/>.
- [60]. **John, Wieczorek; Taxonomic Databases Working Group.** TDWG Wiki - DarwinCore. *Biodiversity Information Standards TDWG*. [Online] TDWG, June 2009. [Cited: 31 August 2009.] <http://wiki.tdwg.org/twiki/bin/view/DarwinCore/DesignAndPurpose>.
- [61]. **USGS.** Biological Informatics Program Standards for Data and Metadata. *U.S. Geological Survey*. [Online] U.S. Department of the Interior, 30 November 2007. [Cited: 1 September 2009.] <http://biology.usgs.gov/bio/standards.html>.
- [62]. **Berendsohn, Walter; ABCD Schema - Task Group .** ABCD Objectives. *Access to Biological Collection Data*. [Online] A joint CODATA and TDWG initiative supported by GBIF, 06 March 2005 . [Cited: 1 September 2009.] <http://www.bgbm.org/TDWG/CODATA/default.htm>.
- [63]. **TDWG and CODATA Task Group .** ABCD - Access to Biological Collection Data. *Biodiversity Information Standards - TDWG* . [Online] 10 July 2007 . [Cited: 1 September 2009.] <http://wiki.tdwg.org/ABCD/>.
- [64]. **TAPIR Task Group.** TAPIR. *TDWG Standards*. [Online] TDWG, 20 February 2009. <http://www.tdwg.org/activities/tapir/>.
- [65]. **Renato De Giovanni et. al.** TAPIR - TDWG Access Protocol for Information Retrieval. *TSWG Standards*. [Online] 05 February 2009. http://www.tdwg.org/dav/subgroups/tapir/1.0/docs/tdwg_tapir_specification_2009-02-05.htm.
- [66]. **Jones, AC, White, Richard J, et al.** SPICE. *Species 2000*. [Online] Species 2000 Project, September 2000. [Cited: 1 September 2009.] http://www.sp2000.org/index.php?option=com_content&task=view&id=38&Itemid=49.
- [67]. **Kennedy, Jessie and Kukla, Robert.** TDWGOntology. *Biodiversity Information Standards TDWG*. [Online] TDWG, 18 October 2006. [Cited: 20 Nov 2009.] <http://wiki.tdwg.org/twiki/bin/view/TAG/TDWGOntology>.
- [68]. **Partnership for Biodiversity Informatics.** Science Environment for Ecological Knowledge . *SEEK*. [Online] University of New Mexico, The Regents of the

University of California, and University of Kansas, 21 Jan 2005. [Cited: 20 Nov 2009.] <http://seek.ecoinformatics.org/Wiki.jsp?page=SEEKComponents>.

[69]. **ENBI**. Information Page. *European Network for Biodiversity Information*. [Online] ENBI, 2003. [Cited: 31 August 2009.] <http://www.enbi.info/forums/enbi/index.php>.

[70]. **TDWG**. Biodiversity Information Standards. *TDWG*. [Online] TDWG, 27 May 2009. [Cited: 1 September 2009.] <http://www.tdwg.org/standards/>.

[71]. —. Biodiversity Information Projects of the World. *Biodiversity Information Standards TDWG*. [Online] TDWG, 10 Nov 2009. [Cited: 20 Nov 2009.] <http://www.tdwg.org/biodiv-projects/>.

[72]. **LifeWatch**. *LifeWatch Supporting Project*. [Online] February 2008. [Cited: 20 Nov 2009.] <http://www.lifewatch.eu/>.

[73]. **Goldfarbr, Charles F.** Current Text of ISO 8879 (SGML). *SGMLSource*. [Online] ISO SGML committee, 6 December 1998. [Cited: 31 August 2009.] <http://www.sgmlsource.com/8879/index.htm>.

[74]. **Brown, Alex.** XML in serial publishing: past, present and future. *OCLC Systems & Services*. 2003, Vol. 19, 4, pp. 149-154,.

[75]. **WorkingGroup, HTML**. W3C HTML. W3C. [Online] 7 March 2007. <http://www.w3.org/html/wg/>.

[76]. **W3C-XML Core Working Group**. Extensible Markup Language (XML). *W3C - XML Activity*. [Online] W3C, 2003. [Cited: 31 August 2009.] <http://www.w3.org/XML/>.

[77]. **Walsh, Norman.** A Technical Introduction to XML. *O'Reilly XML.com*. [Online] 03 October 1998. <http://xml.com/pub/a/98/10/guide0.html>.

[78]. **David C. Fallside (IBM)**. XML Schema Part 0: Primer. *W3C Recommendation*. [Online] W3C, 2 May 2001. [Cited: 31 August 2009.] <http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/>.

[79]. **Jelliffe, Rick**. The Current State of the Art of Schema Languages for XML. *XML Asia Pacific 2001 Conference*. Sydney, Australia, 2001. <http://www.planetpublish.com/pdfs/RickJelliffe.pdf>.

[80]. **W3Schools**. XML DOM Parser. *W3schools.com*. [Online] W3c Working Group - specification, 2009. [Cited: 10 Nov 2009.] <http://www.w3schools.com/Dom/default.asp>.

[81]. **David Megginson -Open Source**. About SAX. *SAX*. [Online] SourceForge project, 27 April 2004. [Cited: 10 Nov 2009.] <http://www.saxproject.org/>.

- [82]. **W3C Web Applications Working Group** . Document Object Model (DOM). *W3C Architecture Domain*. [Online] W3C DOM IG. , 19 January 2005. [Cited: 31 August 2009.] <http://www.w3.org/DOM/>.
- [83]. **The Apache Software Foundation**. AXis Object Model. *ws.apache.org*. [Online] The Apache Software Foundation, 15 January 2009. [Cited: 10 Nov 2009.] <http://ws.apache.org/commons/axiom/>.
- [84]. **Burke, Eric M.** *Java and XSLT*. [ed.] Mike Loukides. Sabastapol : O'Reilly Associates & Inc, 2001. ISBN: 0-596-00143-6.
- [85]. **Hunter, Jason and McLaughlin, Brett**. JDOMTM Documentation . *JDOM.org*. [Online] JDOM, 2002. [Cited: 31 August 2009.] <http://www.jdom.org/downloads/docs.html>.
- [86]. **JavaCommunityProcess**. JSR-000173 Streaming API for XML . <http://jcp.org/>. [Online] [Cited: 10 Nov 2009.] <http://jcp.org/aboutJava/communityprocess/first/jsr173/>.
- [87]. **Clark, James; W3C**. XSL Transformations (XSLT) Version 1.0. *W3C Recommendation*. [Online] W3C, 16 November 1999. [Cited: 31 August 2009.] <http://www.w3.org/TR/xslt>.
- [88]. **Novatchev, Dimitre**. The Functional Programming Language XSLT. *Sourceofrge.Net*. [Online] Open Source, November 2001. [Cited: 10 Nov 2009.] <http://fxsl.sourceforge.net/articles/FuncProg/9.html#Summary>.
- [89]. **Hughes, John**. Why functional programming matters. [ed.] D. A. Turner. *In Research Topics in Functional Programming*. 1990, pp. 17-42.
- [90]. **Clark, James and DeRose, Steve**. XML Path Language (XPath) Version 1.0. *W3C Recommendation*. [Online] W3C, 16 November 1999. [Cited: 31 August 2009.] <http://www.w3.org/TR/xpath>.
- [91]. **W3C,**. Web Services Activity. *W3C.org*. [Online] 2002. <http://www.w3.org/2002/ws/>.
- [92]. **Liu, David., Peng, Jun., Law, Kincho H., Wiederhold, Gio** Efficient integration of web services with distributed data flow and active mediation. *Proceedings of the 6th international conference on Electronic commerce*. Delft, The Netherlands. ACM New York, NY, USA, 2004, pp. 11 - 20. ISBN:1-58113-930-6.
- [93]. **W3C**. SOAP Version 1.2. *W3C SOAP Recommendation*. [Online] 27 April 2007. <http://www.w3.org/TR/2007/REC-soap12-part0-20070427/>.
- [94]. **Feuerlicht, George and Meesathit, Sooksathit**. Design framework for interoperable service interfaces. *2nd international conference on Service oriented computing*. ACM New York, NY, USA, 2004, pp. 299 - 307. ISBN:1-58113-871-7.

- [95]. **IBM, Microsoft, Oracle, SAP, Intel.** UDDI Spec Technical Committee Specification. *UDDI.org*. [Online] 19 July 2002. <http://www.uddi.org/pubs/uddi-v3.00-published-20020719.htm>.
- [96]. **W3C.** Web Services Description Working Group. *W3C*. [Online] 2002. <http://www.w3.org/2002/ws/desc/>.
- [97]. **OASIS.** OASIS Standards. *OASIS* . [Online] April 2006. <http://www.oasis-open.org/specs/>.
- [98]. **GBIF.** Global Biodiversity Information Facility UDDI Registry . *GBIF*. [Online] 2004. <http://registry.gbif.net/uddi/web>.
- [99]. **Wong, A.K.Y., Ray, P., Parameswaran, N., Strassner, J.** Ontology mapping for the interoperability problem in network management. *IEEE Journal on Selected Areas in Communications*. IEEE, October 2005, Vol. 23, 10, pp. 2058- 2068. ISSN: 0733-8716, DOI: 10.1109/JSAC.2005.854130.
- [100]. **Gruber, Thomas R.** In Formal Ontology in Conceptual Analysis and Knowledge Representation. *Stanford Knowledge Systems Laboratory*. Kluwer Academic Publishers, 1993.
- [101]. **McGuinness, Natalya F. Noy and Deborah L.** *Ontology Development 101: A Guide to Creating Your First Ontology*. Stanford : Stanford Knowledge Systems Laboratory Technical Report KSL-01-05, 2001.
- [102]. **Zhang, Junte and Olango, Proscovia.** Populating an Ontology . *University of Groningen*. [Online] February 2005. [Cited: 31 August 2009.] http://semweb weblog.ub.rug.nl/sites/semweb weblog.ub.rug.nl/files/report_Wiki_taxonomy_Junte_Prossy.pdf.
- [103]. **Goguen, Joseph A.** Ontology, Society, and Ontotheology. *International Conference on Formal Ontologies in Informatin Systems*. Torino, Italy, 2004.
- [104]. **Perez, Gomez, Garcia, Corcho and Lopez, Fernandez.** Ontological Engineering:.. *Springer-Verlag New York, Inc.* Nov 2003.
- [105]. **W3C.** OWL Web Ontology Language Overview. *W3C Recommendation* . [Online] 10 February 2004 . <http://www.w3.org/TR/owl-features/>.
- [106]. **Youn, Seongwook and McLeod, Dennis.** Ontology Development Tools for Ontology-Based Knowledge Management. *Encyclopedia of E-Commerce, E-Government and Mobile Commerce*. Los Angles : Idea Group Inc, 2006 .
- [107]. **Protege Community.** welcome to protégé. *Stanford Center for Biomedical Informatics Research at Stanford*. [Online] January 2009. [Cited: 31 August 2009.] <http://protege.stanford.edu/>.

- [108]. **Youn, Seongwook and McLeod, Dennis.** Ontology Development Tools for Ontology-Based Knowledge Management. *Encyclopedia of E-Commerce, E-Government and Mobile Commerce*. s.l. : Idea Group Inc, 2006.
- [109]. **Esposito, Dino and Saltarello, Andrea.** *Microsoft .NET architecting applicaitons for the enterprise*. Redmond, Washington : Microsoft Press, 2009.
- [110]. **Young, Paul., Berzins, Valdis., Ge, Jun., Luqi.** Using an object oriented model for resolving representational differences between heterogeneous systems. *Proceedings of the 2002 ACM symposium on Applied computing*. Madrid, Spain. ACM New York, NY, USA, 2002, pp. 976 - 983. ISBN:1-58113-445-2.
- [111]. **Paepcke, Andreas, Cousins, Steve B.,** Using Distributed Objects for Digital Library Interoperability. *IEEE Computer archive*. IEEE Computer Society Press, May 1996, Vol. 29, 5, pp. 61 - 68. ISSN:0018-9162.
- [112]. **Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides.** *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading: . s.l. : Addison-Wesley, 1994.
- [113]. **Microsoft.** .NET Framework Developer's Guide. *MSDN Framework development center*. [Online] Microsoft, 2009. [Cited: 31 August 2009.] <http://msdn.microsoft.com/en-us/library/w0x726c2.aspx>.
- [114]. **Fisher, Marina., Lai, Ray., Sharma, Sonu., Moroney, L.** *Java EE and .Net Interoperability: Integration Strategies, Patterns, and Best Practices*. s.l. : Prentice Hall, 2006.
- [115]. **Sun Microsystems, Inc.** Java Technology Reference. *Sun Developer Network*. [Online] Sun Microsystems, Inc, 2009. [Cited: 31 August 2009.] <http://java.sun.com/reference/index.jsp#documentation>.
- [116]. **Hudak, Paul.** Conception, evolution, and application of functional programming languages. *ACM Computing Surveys (CSUR) archive*. ACM New York, NY, USA, September 1989, Vol. 21, 3, pp. 359 - 411. ISSN:0360-0300.
- [117]. **Microsoft .** Introduction to Pure Functional Transformations. *MSDN Developer Library*. [Online] 2008. [Cited: 31 August 2009.] <http://msdn.microsoft.com/en-us/library/bb943915.aspx>.
- [118]. **Marguerie, Fabrice, Eichert, Steve and Wooley, Jim.** *LINQ in Action*. s.l. : Manning Publications, 4 Feb 2008. ISBN-13: 978-1933988160.
- [119]. **Calvert, Charlie and Kulkarni, Dinesh.** *Essential LINQ*. Boston, USA : Addison-Wesley Professional; 1 edition, March 22, 2009. ISBN-13: 978-0321564160 .
- [120]. **Box, Don and Hejlsberg, Anders.** LINQ: .NET Language-Integrated Query. *MSDN Developer Centre*. [Online] February 2007. [Cited: 31 August 2009.] http://msdn.microsoft.com/en-gb/library/bb308959.aspx#linqoverview_topic7.

[121]. **Champion, Michael**. .NET Language-Integrated Query for XML Data. *MSDN Developer Centre*. [Online] February 2007. [Cited: 31 August 2009.] <http://msdn.microsoft.com/en-us/library/bb308960.aspx>.

[122]. **Microsoft Corporation**. The ADO.NET Entity Framework Overview. *Visual Studio 2005 Technical Articles*. [Online] Microsoft, June 2006. [Cited: 31 August 2009.] [http://msdn.microsoft.com/en-us/library/aa697427\(VS.80\).aspx](http://msdn.microsoft.com/en-us/library/aa697427(VS.80).aspx).

[123]. **Connolly, Thomas M and Begg, Carolyn E**. *Database Systems: A Practical Approach to Design, Implementation, and Management*. Edition: 4, illustrated. s.l. : Pearson Education, 2005. p. 1374.

[124]. **Ramakrishnan, Raghu and Gehrke, Johannes**. *Database Management Systems*. s.l. : McGraw-Hill Professional, 2003.

[125]. **Litwin, Witold, Leo, Mark and Roussopoulos, Nick**. Interoperability of multiple autonomous databases. *ACM Computing Surveys*. September 1990, Vol. 22, 3, pp. 267-293. DOI= <http://doi.acm.org/10.1145/96602.96608>.

[126]. **Litwin, Witold and Abdellatif, Abdelaziz**. Multidatabase Interoperability. *Computer archive*. IEEE Computer Society Press Los Alamitos, CA, USA, 1986, Vol. 19, 12, pp. 10-18. ISSN:0018-9162.

[127]. **White, Richard J and Allkin, R**. A Language for the definition and exchange of biological data sets. *Mathematical and Computer Modelling*. TDWG, 1992, Vol. 16, pp. 199–233.

[128]. **Vieglais, Dave**. *The Species Analyst Project*. University of Kansas Natural History Museum. s.l. : OASIS, 2001. <http://xml.coverpages.org/speciesAnalyst.html>.

[129]. **The Library of Congress - USA**. Information Retrieval (Z39.50):. *Z39.50 Resources*. [Online] The Library of Congress - ANSI/NISO Z39.50-2003, 2003. [Cited: 10 Nov 2009.] <http://www.loc.gov/z3950/agency/Z39-50-2003.pdf>. ISSN: 1041-5653.

[130]. **Jones, Andrew C**. Applying Computer Science Research to Biodiversity Informatics: Some Experiences and Lessons. *Transactions on Computational Systems Biology IV*. Springer Berlin / Heidelberg, March 2006, Vol. 3939/2006, pp. 44-57. ISBN 978-3-540-33245-9.

[131]. **Species 2000 Secretariat**. Species 2000 Web Services. *Species 2000*. [Online] Species 2000 Secretariat, 2009. [Cited: 31 August 2009.] http://www.sp2000.org/index.php?option=com_content&task=view&id=40&Itemid=49.

[132]. **Jones, Andrew C., White, Richard J., Sutherland, Iain., Embury, Suzanne M., Gray, Alex W., Robinson, John S., Bisby, Frank A., Brandt, Sue M**. Techniques for Effective Integration, Maintenance and Evolution of Species Databases. *12th International Conference on Scientific and Statistical Database*

Management. IEEE Computer Society Washington, DC, USA, 2000, p. 3. ISBN ~ ISSN:1099-3371 , 0-7695-0686-0.

[133]. **BioCASE**. The Biological Collection Access Service for Europe. [Online] Botanischer Garten und Botanisches Museum Berlin-Dahlem (BGBM),, 2006. [Cited: 10 Dec 2009.] <http://search.biocase.org/europe/>.

[134]. **Pahwa, Jaspreet S., Brewer, P; Sutton, T., Jones, Andrew C., White, Richard J., Gray, Alex W., Fiddian, Nick J., Bisby, Frank A., Scoble, M., Bhagwat, S.** Biodiversity World: A Problem-Solving Environment for Analysing Biodiversity Patterns. *Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2006)*. IEEE,xplore, May 2006, p. 8. DOI: 10.1109/CCGRID.2006.23.

[135]. **Wieczorek, John**. Mammal Networked Information System. *manisnet*. [Online] University of California, Berkeley, 18 March 2009. [Cited: 1 September 2009.] <http://manisnet.org/>.

[136]. **Sundaravadivelu, R.** The BUFFIE Architecture. *Biodiversity Conference meeting, Stockholm - Sweeden*. ENBI, 2005. http://circa.gbif.net/irc/Download/kheyA5JSmUGsG36eHl0pkTqSyIqd9RmC/YxhrUiUxPDF3WMVcOagIe47UxVqIIDf3/WM4sNMitck6/BUFFIE%20ENBI_Sp2Keu.pdf.

[137]. **GBIF**. GBIF Infrastructure. *GBIF Integrated Publishing Toolkit*. [Online] GBIF, 2009. [Cited: 10 Dec 2009.] <http://www.gbif.org/informatics/infrastructure/publishing/>.

[138]. **Nativia, Stefano., Mazzettia, Paolo., Saarenmaab, Hannu., Kerrc, Jeremy., Tuamad, Éamonn Ó.** Biodiversity and climate change use scenarios framework for the GEOSS interoperability pilot process. *Ecological Informatics*. Elseiver, January 2009, Vol. 4, 1, pp. 23-33.

[139]. **4D4Life**. *Distributed Dynamic Diversity Databases for Life*. [Online] Cardiff University, Reading University and other partners, 2009. [Cited: 10 Dec 2009.] <http://www.4d4life.eu/index.php>.

[140]. **Salvatore Salamone**. LSID: An Informatics Lifesaver . *Bio-ITWorld.com*. [Online] Cambridge Healthtech Institute , 12 January 2002. [Cited: 10 Nov 2009.] <http://www.bio-itworld.com/archive/011204/lsid.html>.

[141]. **TDWG, GBIF, and SourceForge**. Life Science Identifiers . *Life Sciences Identifiers -Resolution Project*. [Online] Opensource. [Cited: 10 Nov 2009.] <http://lsids.sourceforge.net/>.

[142]. **Jones, Andrew C.** Applying Computer Science Research to Biodiversity Informatics: Some Experiences and Lessons . [book auth.] Lecture Notes in Computer Science. *Transactions on Computational Systems Biology IV*. s.l. : Springer Berlin / Heidelberg, 2006, Vol. 3939/2006, pp. 44-57.

- [143]. **Stan, Blum; Dave, Vieglais; P.J. Schwartz; et.al;**. Distributed Generic Information Retrieval . *Sourceforge.net*. [Online] open source, December 2005. [Cited: 1 September 2009.] <http://digir.sourceforge.net/>.
- [144]. **Stonebraker, Michael**. Too much middleware. *ACM SIGMOD Record archive*. ACM New York, NY, USA, 2002, Vol. 31, 1, pp. 97 - 106. ISSN:0163-5808.
- [145]. **Shaw, M and Clements, P**. A field guide to boxology: preliminary classification of architectural styles for software systems. *Computer Software and Applications Conference, 1997. COMPSAC '97. Proceedings, Washington, DC, USA*. IEEE Xplore, Aug 1997, pp. 6-13. ISBN: 0-8186-8105-5.
- [146]. **Fielding, Roy Thomas**. Architectural Styles and the Design of Network-based Software Architectures. *Doctoral dissertation*. Irvine : University of California, 2000.
- [147]. *Software interoperability: principles and practice*. **Wileden, Jack C. and Kaplan, Alan**. Los Angeles, California, United States : ACM New York, NY, USA , 1999 . Proceedings of the 21st international conference on Software engineering . pp. 675 - 676. ISBN:1-58113-074-0.
- [148]. **Meier, J.D.; Homer, Alex; et al;**. *Application Architecture Guide 2.0 - patterns & practices*. Redmond, USA : Microsoft, 2008.
- [149]. **Sundaravadivelu, R., White, R.J., Jones, A.C.** A Service-Oriented Architecture with Domain Knowledge Base for Interoperability of Heterogeneous Distributed Biodiversity Resources. *Software Engineering and Applications (SEA 2009), Cambridge - MIT, Massachusetts, USA*. ACTA Press, November 2009. <http://www.actapress.com/Abstract.aspx?paperId=36753>.
- [150]. **Esposito, Dino**. Cutting Edge: AJAX Application Architecture. *MSDN magazine*. September 2007. <http://msdn.microsoft.com/en-us/magazine/cc163363.aspx>.
- [151]. **Papa, John**. Designing an Entity Data Model. *MSDN Technical Articles*. [Online] February 2008. [Cited: 10 Dec 2009.] MSDN Magazine. <http://msdn.microsoft.com/en-us/magazine/cc163286.aspx#S1>.
- [152]. **Gelbukh, Alexander F**. Lazy Query Enrichment: A Method for Indexing Large Specialized Document Bases with Morphology and Concept Hierarchy. *Book Series Lecture Notes in Computer Science*. 01 January 2000, Vol. 1873/2000, pp. 526-535.
- [153]. **Owei, Vesper and Navathe, Shamkant B**. Enriching the conceptual basis for query formulation through relationship semantics in databases. *Publisher Elsevier Science Ltd. Oxford, UK*. September 2001, Vol. 26, 6, pp. 445 - 475. ISSN:0306-4379.
- [154]. **Paterson, Trevor, et al**. A Universal Character Model and Ontology of Defined Terms for Taxonomic Description. [book auth.] *Lecture Notes in Computer Science. Book Data Integration in the Life Sciences* . s.l. : Springer Berlin / Heidelberg, 2004, Vol. 2994/2004 .

- [155]. **Schmid, Randolph E.** Marine species entangled in extra names. *msnbc - Technology & science*. [Online] 25 June 2008. [Cited: 1 September 2009.] <http://www.msnbc.msn.com/id/25369944/>.
- [156]. **White, Richard ; Cardiff Biodiversity Informatics team .** Spice software home page. *Species 2000*. [Online] November 2007. [Cited: 1 September 2009.] <http://biodiversity.cs.cf.ac.uk/spice/>.
- [157]. **Su, Xiaomeng, Hakkarainen, Sari and Brasethvik, Terje.** Semantic enrichment for improving systems interoperability. *ACM Symposium on Applied Computing*. March 2004, pp. 14-17.
- [158]. **MacDonald, Matthew.** *Pro ASP.NET 3.5 in C# 2008*. 2nd Edition. s.l. : APRESS, 15 Nov 2007. p. 1498. ISBN-13: 978-1590598931.
- [159]. **Open source.** Introducing JSON. *JSON*. [Online] Opensource. [Cited: 1 September 2009.] <http://www.json.org/>.
- [160]. **Microsoft Technet.** Microsoft CryptoAPI and Cryptographic Service Providers . *Microsoft Technet Technet library*. [Online] Microsoft Technet, 2009. [Cited: 1 September 2009.] <http://technet.microsoft.com/en-gb/library/cc962093.aspx>.
- [161]. **Goguen, Joseph A.** Data, Schema, Ontology and Logic Integration. *Logic Journal of the IGPL*. Oxford University Press, June 2005, Vol. 13, 6, pp. 685-715.
- [162]. **Obrst, Leo.** Ontologies for semantically interoperable systems. *Proceedings of the twelfth international conference on Information and knowledge management, New Orleans, LA, USA*. ACM New York, USA, 2003, pp. 366 - 369. ISBN:1-58113-723-0.
- [163]. **Goguen, Joseph.** Ontology, society, and ontotheology. *Formal Ontology in Information Systems*. Torino, Italy, IOS Press, 2004, pp. 95-103.
- [164]. **Renner, Scott A.** *A "Community of Interest" Approach to Data Interoperability*. San Deigo : The MITRE Corporation, 2001. http://www.mitre.org/work/tech_papers/tech_papers_01/renner_community/index.html.
- [165]. **Jovellanos, Chito.** Semantic and syntactic interoperability: in transactional systems. *Proceedings of the 4th ACM conference on Electronic commerce, San Diego, CA, USA*. ACM New York, NY, USA, 2003, pp. 266 - 267. ISBN:1-58113-679-X.
- [166]. **Al-Wasil, Fahad M, Gray, W.A. and Fiddian, N.J.** Establishing an XML Metadata Knowledge Base to Assist Integration of Structured and Semi-structured Databases. *Australasian Database Conference*. Australasian Computer Society, 2006.
- [167]. **Hunter, Jane and Lagoze, Carl.** Combining RDF and XML schemas to enhance interoperability between metadata application profiles. *Proceedings of the 10th international conference on World Wide Web*. Hong Kong. ACM New York, NY, USA, 2001, pp. 457 - 466. ISBN:1-58113-348-0.

- [168]. **Boukottaya, A, Vanoirbeek, C and et.al.** Automating XML documents Transformations: A Conceptual modelling based approach. *Proceedings of the First Asia-Pacific Conference on Conceptual Modelling*. Dunedin, New Zealand, January 2004, pp. 81-90. ISBN 1-920682-13-9.
- [169]. **Microsoft.** XPathNavigator in the .NET Framework. *MSDN developer*. [Online] Microsoft, 2009. [Cited: 1 September 2009.] [http://msdn.microsoft.com/en-us/library/aa735770\(VS.71\).aspx](http://msdn.microsoft.com/en-us/library/aa735770(VS.71).aspx).
- [170]. **Microsoft Corporation.** Microsoft Visual Studio 2008. *Microsoft Visual Studio 2008*. [Online] [Cited: 18 August 2009.] <http://www.microsoft.com/visualstudio/en-gb/default.aspx>.
- [171]. **Microsoft.** Internet Information Services. *Internet Information Services*. [Online] 2003. [Cited: 18 August 2009.] <http://www.microsoft.com/windowsserver2003/iis/default.aspx>.
- [172]. **Microsoft..** SQL Server 2008. *SQL Server 2008*. [Online] 2008. [Cited: 18 August 2009.] <http://www.microsoft.com/sqlserver/2008/en/us/default.aspx>.
- [173]. **McEwan, A.A.; Schneider, S; Ifill, W.; Welch, P.H. ;.** *Communicating Process Architectures 2007: WoTUG-30*. illustrated edition. s.l. : IOS Press,US, 2007. p. 524 . ISBN-13: 978-1586037673 .
- [174]. **Dijkstra, Edsger W.** On the role of scientific thought. *Selected Writings on Computing: A Personal Perspective*. 1982 , pp. 60–66.
- [175]. **Gamma, Erich, et al.** *Design Patterns: Elements of Reusable Object-Oriented Software*. s.l. : Addison-Wesley Professional, 1994. ISBN-13: 978-0201633610.
- [176]. **Esposito, Dino.** Cutting Edge AJAX Application Architecture. *MSDN Magazine*. September 2007. <http://msdn.microsoft.com/en-us/magazine/cc135414.aspx>.
- [177]. **Microsoft.** Process XML Data Using the XPath Data Model. *.NET Framework Developer's Guide*. [Online] 2009. <http://msdn.microsoft.com/en-us/library/87274khy.aspx>.
- [178]. **Microsoft .** LINQ to Entities Overview. *MSDN .Net Framework*. [Online] 2009. [Cited: 25 August 2009.] <http://msdn.microsoft.com/en-us/library/bb386992.aspx>.
- [179]. Marine Biodiversity and Ecosystem Functioning. *MarBEF site*. [Online] 10 12 2010. <http://www.marbef.org/>.
- [180]. **Schalk, Peter H and Altenburg, Ruud.** Technologies and facilities of ETI BioInformatics electronic information sharing mechanism for MARBEF. *Linnaeus II software for taxonomic data management*. [Online] 22 October 2005. [Cited: 31 August 2009.] www.medobis.org/prope/presentations/Altenburg.ppt.

- [181]. **Beller, Aaron, et al.** *Design document for mapping tools to work with common access system*. European Network of Biodiversity Information. Jerusalem - Israel, Linz-Austria :ENBI,2005.
http://circa.gbif.net/irc/Download/kte_AiJZmtGFH37I2VCZKTcR5R978bRs/pjTrA4Rb1O0SpTfpQEdCbWKF/WP10_D10_2a%20TR.pdf.
- [182]. **CFH - UK.** Data Standards across the NHS. *NHS Connecting for Health*. [Online] Department of Health Informatics Directorate., 10 Dec 2010.
<http://www.connectingforhealth.nhs.uk/systemsandservices/data/uktc/snomed>.
- [183]. **Sundaravadivelu, R., White, R.J., Jones, A.C., Gray, W.A.** Web based Middleware Framework for Interoperability between Heterogeneous, Distributed Biodiversity Data Resources. *Proceedings of the 25th conference on IASTED International Multi-Conference, Innsbruck, Austria*. ACTA Press, Anaheim, CA, USA., 2007, pp. 142--147.
<http://portal.acm.org/citation.cfm?id=1332068&CFID=82124900>.
- [184]. **GIGAS Project Office.** Interoperability between INSPIRE, GEOSS, GMES, SEIS, SISE: opportunities for convergence and innovation. *GIGAS project*. [Online] 7th Framework Programme of the European Commission, 22 June 2010. [Cited: 4 August 2010.] <http://www.thegigasforum.eu/project/project.html>.
- [185]. *A standards set to share biodiversity data related to fisheries*. **Bardie, Julian, Cauquil, Pascal and Cury, Philippe.** Paris, France : IMDIS, March 2010. International Conference on Marine Data and Information Systems.
<http://wwz.ifremer.fr/imdis2010/content/download/69501/487582/version/1/file/IMD>.
- [186]. **Kennedy, Jessie, Kukla, Robert and Paterson, Trevor.** Taxonomic Concept Transfer Schema. *Taxonomic Concept Transfer Schema*. [Online] 16 September 2005.
<http://www.tdwg.org/standards/117/>.
- [187]. **Jovanovic, Jelena and Gasevic, Dragan.** Achieving Knowledge interoperability: An XML/XSLT approach. *Expert Systems with Applications - Elsevier*. 2005, pp. 535-553.
- [188]. **White, Richard J, Jones, Andrew C and Bisby, Frank A.** Federating Taxonomic Databases: Progress With the Catalogue of Life Dynamic Checklist. *Proceedings of TDWG, 2006*. [Online] 2006.
<http://www.tdwg.org/proceedings/article/view/85>.
- [189]. **Java Community Process.** Apache Tomcat. *Apache Software Foundation*. [Online] 2008. <http://tomcat.apache.org/>.
- [190]. **Conn, B.J.** *HISPID3. Herbarium Information Standards and Protocols for Interchange of Data.Version 3*. Royal Botanic Gardens Sydney. Sydney : TDWG standard, 1996. <http://plantnet.rbgsyd.nsw.gov.au/HISCOM/default.htm>.
- [191]. **Rahm, Erhard and Bernstein, Philip A.** A survey of approaches to automatic schema matching. *The VLDB Journal*. 2001, Vol. 10, pp. 334-350.

Appendix A

Mapping between Darwin Core (DWCV2) and ABCD (BioCASE) Concepts

DwC 1.4 Record-level Element	ABCD 2.06b X-Path	DwC to ABCD	ABCD to DwC
	Datasets/Dataset/Units/Unit /...		
GlobalUniqueIdentifier	UnitGUID	Fully compatible.	Fully compatible.
DateLastModified	DateLastEdited	Fully compatible.	Fully compatible.
BasisOfRecord	RecordBasis	Fully compatible DwC gives only recommendations for content. The examples given are the same as the restriction for ABCD, except that "StillImage" is used instead of "DrawingOrPhotograph", and that "MovingImage" and "SoundRecording" are listed, which should be mapped to "MultimediaObject" in ABCD	Fully compatible ABCD is restricting content to values representing: "PreservedSpecimen", "LivingSpecimen", "FossileSpecimen", "OtherSpecimen", "HumanObservation", "MachineObservation", "DrawingOrPhotograph", "MultimediaObject" and "AbsenceObservation".
InstitutionCode	SourceInstitutionID	Fully compatible	Fully compatible
CollectionCode	SourceID	Fully compatible	Fully compatible
CatalogNumber	UnitID	Fully compatible	Fully compatible
InformationWithheld	InformationWithheld	Fully compatible	Fully compatible ABCD allows the language to be stated.
Remarks	Notes	Fully compatible	Fully compatible
Taxonomic Elements	Datasets/Dataset/Units/Unit / Identifications/Identification/ TaxonIdentified/...		

Appendix A. Mapping of DarwinCore (DWCV2) & ABCD (BioCase) Concepts

ScientificName	ScientificName/FullScientificNameString	Fully compatible (but ABCD able to support multiple identifications and identification history)	Fully compatible preferred identification must be used
HigherTaxon	HigherTaxa/HigherTaxon/HigherTaxonName	+/- compatible. Unbounded ABCD element can be parsed from DwC text string.	+/- compatible. List can be compiled from unbounded ABCD element.
Kingdom	HigherTaxa/HigherTaxon/HigherTaxonName with HigherTaxa/HigherTaxon/HigherTaxonRank = regnum	Fully compatible. "regnum" as constant	Fully compatible if complete ABCD data (incl. rank) are provided.
Phylum	HigherTaxa/HigherTaxon/HigherTaxonName with HigherTaxa/HigherTaxon/HigherTaxonRank = phylum	Fully compatible. "phylum" as constant	Fully compatible if complete ABCD data (incl. rank) are provided.
Class	HigherTaxa/HigherTaxon/HigherTaxonName with HigherTaxa/HigherTaxon/HigherTaxonRank = classis	Fully compatible. "classis" as constant	Fully compatible if complete ABCD data (incl. rank) are provided.
Order	HigherTaxa/HigherTaxon/HigherTaxonName with HigherTaxa/HigherTaxon/HigherTaxonRank = ordo	Fully compatible. "ordo" as constant	Fully compatible if complete ABCD data (incl. rank) are provided.
Family	HigherTaxa/HigherTaxon/HigherTaxonName with HigherTaxa/HigherTaxon/HigherTaxonRank = familia	Fully compatible. "familia" as constant	Fully compatible if complete ABCD data (incl. rank) are provided.
Genus	ScientificName/NameAtomised/Bacterial/GenusOrMonomial ScientificName/NameAtomised/Botanical/GenusOrMonomial ScientificName/NameAtomised/Viral/GenusOrMonomial ScientificName/NameAtomised/Zoological/GenusOrMonomial	Compatible if taxonomic context (Code of Nomenclature) is known, which may also be deduced from value for Regnum in most cases.	Fully compatible for Genus as part of name.

Appendix A. Mapping of DarwinCore (DWCV2) & ABCD (BioCase) Concepts

SpecificEpithet	<p>ScientificName/NameAtomised/Bacterial/SpeciesEpithet</p> <p>ScientificName/NameAtomised/Botanical/FirstEpithet</p> <p>ScientificName/NameAtomised/Zoological/SpeciesEpithet</p>	Compatible if taxonomic context (Code of Nomenclature) is known, which may also be deduced from value for Regnum in most cases.	Fully compatible for zoological and bacteriological names, in Botany subdivisions of genera may be included. ABCD additionally supports viral names.
InfraspecificRank	ScientificName/NameAtomised/Botanical/Rank	Compatible if taxonomic context (Code of Nomenclature) is known, which may also be deduced from value for Regnum in most cases. It defaults to subspecies in zoology and bacteriology.	Fully compatible. Subspecies as constant for zoological and bacterial names.
InfraspecificEpithet	<p>ScientificName/NameAtomised/Bacterial/SubspeciesEpithet</p> <p>ScientificName/NameAtomised/Botanical/SecondEpithet</p> <p>ScientificName/NameAtomised/Zoological/SubspeciesEpithet</p>	Compatible if taxonomic context (NomenclaturalCode) is known, which may also be deduced from value for Regnum in most cases.	Fully compatible ABCD additionally supports viral names, breeds and named individuals, and cultivar groups, names, and trade designations.
AuthorYearOfScientificName	<p>ScientificName/NameAtomised/Bacterial/ParaneticalAuthorTeamAndYear + ScientificName/NameAtomised/Bacterial/AuthorTeamAndYear</p> <p>ScientificName/NameAtomised/Botanical/AuthorTeamParenthesis + ScientificName/NameAtomised/Botanical/AuthorTeam</p> <p>ScientificName/NameAtomised/Zoological/AuthorTeamOriginalAndYear + [= or] ScientificName/NameAtomised/Zoological/AuthorTeamParenthesisAndYear</p>	Content compatible, but needs parsing to classify paranetical author(s).	Compatible when concatenated
NomenclaturalCode	Code	Fully compatible	Fully compatible

Appendix A. Mapping of DarwinCore (DWCV2) & ABCD (BioCase) Concepts

IdentificationQualifier	IdentificationQualifier	Fully compatible	Fully compatible In addition ABCD provides an attribute to define the insertion point in a string concatenated from atomised data.
Locality Elements	Datasets/Dataset/Units/Unit/Gathering/		
HigherGeography	LocalityText or NamedAreas/NamedArea/AreaName	DwC element is part of the ABCD Element. May be parsed	Compatible for the purpose stated for DwC ('like' queries) Compatible. List can be compiled from unbounded ABCD element. ABCD allows the language to be stated.
Continent	NamedAreas/NamedArea/AreaName with NamedAreas/NamedArea/AreaClass = Continent	Fully compatible "continent" as constant	Fully compatible if complete ABCD data are provided. ABCD allows the language to be stated.
WaterBody	NamedAreas/NamedArea/AreaName with NamedAreas/NamedArea/AreaClass = Water body	Fully compatible "Water body" as constant	Fully compatible if complete ABCD data are provided. ABCD allows the language to be stated.
IslandGroup	NamedAreas/NamedArea/AreaName with NamedAreas/NamedArea/AreaClass = IslandGroup	Fully compatible "Island group" as constant	Fully compatible if complete ABCD data are provided. ABCD allows the language to be stated.
Island	NamedAreas/NamedArea/AreaName with NamedAreas/NamedArea/AreaClass = Island	Fully compatible "island" as constant	Fully compatible if complete ABCD data are provided. ABCD allows the language to be stated.
Country	Country/CountryName	Fully compatible	Fully compatible ABCD allows the language to be stated.

Appendix A. Mapping of DarwinCore (DWCV2) & ABCD (BioCase) Concepts

StateProvince	NamedAreas/NamedArea/ AreaName with NamedAreas/NamedArea/ AreaClass = State or = Province (etc.)	+/- compatible "State or Province" as constant	Fully compatible if complete ABCD data are provided. ABCD allows the language to be stated.
County	NamedAreas/NamedArea/ AreaName with NamedAreas/NamedArea/ AreaClass = County	+/- compatible "county" as constant	+/- compatible if complete ABCD data are provided. ABCD allows the language to be stated.
Locality	AreaDetail	Fully compatible	Fully compatible ABCD allows the language to be stated.
MinimumElevationIn Meters	Altitude/MeasurementOrFactAtomised/ LowerValue	Fully compatible. "m" as constant	Fully compatible for metric values, otherwise conversion is necessary.
MaximumElevationIn Meters	Altitude/MeasurementOrFactAtomised/ UpperValue	Fully compatible "m" as constant	Fully compatible for metric values, otherwise conversion is necessary.
MinimumDepthIn Meters	Depth/MeasurementOrFactAtomised/ LowerValue	Fully compatible "m" as constant	Fully compatible for metric values, otherwise conversion is necessary.
MaximumDepthIn Meters	Depth/MeasurementAtomised/ UpperValue	Fully compatible "m" as constant	Fully compatible for metric values, otherwise conversion is necessary.
Collecting Event Elements	Datasets/Dataset/Units/Unit/Gathering/		
CollectingMethod	Method	Fully compatible	Fully compatible
ValidDistributionFlag (under discussion)	ValidDistributionFlag	Fully compatible	Fully compatible
EstablishmentMeans (under discussion)	EstablishmentMeans	Fully compatible	Fully compatible
EarliestDateCollected	DateTime/ISODateTimeBegin	Fully compatible (Note that some versions of DwC use three fields, namely YearCollected, MonthCollected, and DayCollected, which may be concatenated to ISO date.)	Fully compatible (The three fields used in some DwC versions may be extracted from the ISO datetime in ABCD.)

Appendix A. Mapping of DarwinCore (DWCV2) & ABCD (BioCase) Concepts

LatestDateCollected	DateTime/ISODateTimeEnd	Fully compatible	Fully compatible
(TimeCollected) (deprecated element in v.1.4, covered by EarliestDateCollected)	DateTime/TimeOfDayBegin	Compatible	Compatible Time maintained as separate element in ABCD for cases where no date is given. ABCD provides end of time period
DayOfYear (JulianDay)	DateTime/DayNumberBegin	Fully compatible	Compatible (should not be given if ABCD's DayNumberEnd is given, because in DwC this does not refer to time periods)
Collector	GatheringAgents/GatheringAgentsText	Fully compatible	Fully compatible ABCD provides also atomised version.
Biological Elements	Datasets/Dataset/Units/Unit /		
Sex	Sex	Fully compatible	Fully compatible
LifeStage	ZoologicalUnit/PhasesOrStages/PhaseOrStage or MycologicalUnit/MycologicalLifeStages/ MycologicalLifeStage or MycologicalUnit/MycologicalSexualStage	May be compatible where taxonomic domain is known	Partly compatible, but left to community to define ABCD allows the language to be stated.
Attributes	MeasurementsOrFacts (alternatively: Notes)	A well-formed string may be parsed into character-character state pairs that fit into an ABCD MeasurementOrFact element. Otherwise put into Notes.	MeasurementsOrFacts can be concatenated and accommodated in this DwC element.
References Elements	Datasets/Dataset/Units/Unit /		

Appendix A. Mapping of DarwinCore (DWCV2) & ABCD (BioCase) Concepts

ImageURL	MultimediaObjects/MultimediaObject/FileURI or MultimediaObjects/MultimediaObject/ProductURI	+/- compatible (needs clearer definition of DwC item)	Fully compatible but unbound in ABCD
RelatedInformation	Notes	Fully compatible	Fully compatible

Source of Information for the above mapping table is from TDWG and CoDATA website the reference is:

<http://www.bgbm.org/tdwg/codata/Schema/Mappings/DwCAndExtensions.htm>

C # code for BUFFIE Framework's Core Components and Services

Buffie.Core.Message

```
using System;
using Microsoft.Practices.EnterpriseLibrary.Validation.Validators;

namespace Buffie.Core
{
    /// <summary>
    /// Defines a Buffie Message From Query
    /// </summary>
    public class BuffieMessage: DomainObject
    {
        /// <summary>
        /// Default constructor
        /// </summary>
        public BuffieMessage()
        {
            //inititalize object if needed
        }

        /// <summary>
        /// Gets or sets the Id
        /// </summary>
        [NotNullValidator(MessageTemplate = "The MessageId cannot be
null")]
        public virtual int MessageId { get; set; }

        /// <summary>
        /// Gets or sets the QueryId
        /// </summary>
        [NotNullValidator(MessageTemplate = "The QueryID cannot be
null")]
        public virtual int QueryId { get; set; }

        /// <summary>
        /// Gets or sets the UserId
        /// </summary>
        [NotNullValidator(MessageTemplate = "The UserId cannot be
null")]
        public virtual int UserId { get; set; }

        /// <summary>
        /// Gets or sets the ProviderId
        /// </summary>
        [NotNullValidator(MessageTemplate = "The ProviderId cannot be
null")]
        public virtual int ProviderId { get; set; }
    }
}
```

```

    /// <summary>
    /// Gets or sets the RequestDestination
    /// </summary>
    [NotNullValidator(MessageTemplate = "The RequestDestination
cannot be null")]
    public virtual string RequestDestination { get; set; }

    /// <summary>
    /// Gets or sets the RequestMessage
    /// </summary>
    [NotNullValidator(MessageTemplate = "The RequestMessage
cannot be null")]
    public virtual string RequestMessage { get; set; }

    /// <summary>
    /// Gets or sets the ResponseMessage
    /// </summary>
    [NotNullValidator(MessageTemplate = "The ResponseMessage
cannot be null")]
    public virtual string ResponseMessage { get; set; }

    /// <summary>
    /// Gets or sets the RequestSent
    /// </summary>
    [NotNullValidator(MessageTemplate = "The RequestSent cannot
be null")]
    public virtual DateTime? RequestSent { get; set; }

    /// <summary>
    /// Gets or sets the ResponseReceived
    /// </summary>
    [NotNullValidator(MessageTemplate = "The ResponseReceived
cannot be null")]
    public virtual DateTime? ResponseReceived { get; set; }

    /// <summary>
    /// Gets or sets the XSLTName
    /// </summary>
    public virtual string XSLTFileName { get; set; }
}
}

```

Buffie.Message Services

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data.Objects;
using System.Data.Objects.DataClasses;
using Buffie.Core;
using Buffie.Entities;
using Microsoft.Practices.EnterpriseLibrary.Validation;
using Buffie.Utils;
using System.Diagnostics;

namespace Buffie.Services

```

```

{
    /// <summary>
    /// Defines the interface for Messages
    /// </summary>
    public class BuffieMessageService
    {
        BuffieEntities _dataContext = new BuffieEntities();
        /// <summary>
        /// default constructor
        /// </summary>
        public BuffieMessageService()
        {
        }

        public string CreateMessage(Buffie.Core.BuffieMessage
NewMessage, int CurQueryId)
        {
            try
            {
                //select the reference to Query that creates this
message.
                var _mQuery = (from query in _dataContext.Query
                               where query.QueryId == CurQueryId
                               select query).First();

                var _mProvider = (from prd in _dataContext.Provider
NewMessage.ProviderId
                               where prd.ProviderId ==
                               select prd).First();

                var data = new Buffie.Entities.Message {
Query=_mQuery,
Provider =
_mProvider,
RequestMessage = NewMessage.RequestMessage,
RequestSent
= NewMessage.RequestSent,
ResponseMessage = NewMessage.ResponseMessage,
ResponseReceived = NewMessage.ResponseReceived,
Active =
true };

                _dataContext.AddToMessage(data);
                _dataContext.SaveChanges();

                //return the newquery object with new id value.
                NewMessage.MessageId = data.MessageID;
                return "Success";
            }
            catch (Exception ex)
            {
                return "Error";
            }
        }

        public Buffie.Core.BuffieMessage GetMessage(int MessageID)
        {

```

```

        try
        {
            var Message = _dataContext.Message.First(m =>
m.MessageID == MessageID);
            return new Buffie.Core.BuffieMessage
            {
                UserId = Message.Query.User.UserId,
                QueryId = Message.Query.QueryId,
                MessageId = Message.MessageID,
                ProviderId = Message.Provider.ProviderId,
                RequestSent = Message.RequestSent ,
                RequestMessage = Message.RequestMessage,
                ResponseMessage = Message.ResponseMessage,
                ResponseReceived = Message.ResponseReceived
            };
        }
        catch (Exception ex)
        {
            return null;
        }
    }

    public string UpdateMessage(Buffie.Core.BuffieMessage
curMessage)
    {
        try
        {
            var updMessage = (from Msg in _dataContext.Message
                             where Msg.MessageID ==
curMessage.MessageId
                             select Msg).First();

            updMessage.RequestMessage =
curMessage.RequestMessage;
            updMessage.RequestSent = curMessage.RequestSent;
            updMessage.ResponseMessage =
curMessage.ResponseMessage.ReplaceEx("UTF-8", "UTF-16", true);

            updMessage.ResponseMessage =
curMessage.ResponseMessage; ;
            updMessage.ResponseReceived =
curMessage.ResponseReceived;

            _dataContext.SaveChanges();

            return "Success";
        }
        catch (Exception ex)
        {
            Debug.Print(ex.Message + "*****/n"+
ex.InnerException);
            return "Error";
        }
    }
}

```

Buffie.Core.Query

```

using System.Collections.Generic;
using System.Text;
using Microsoft.Practices.EnterpriseLibrary.Validation.Validators;

namespace Buffie.Core
{
    /// <summary>
    /// Defines a Buffie Query
    /// </summary>
    public class BuffieQuery: DomainObject
    {
        /// <summary>
        /// Default constructor of a BuffieQuery
        /// </summary>
        public BuffieQuery()
        {
        }

        /// <summary>
        /// Gets or sets the Id
        /// </summary>
        [NotNullValidator(MessageTemplate = "The User ID cannot be
null")]
        public virtual int UserId { get; set; }

        /// <summary>
        /// Gets or sets the Id
        /// </summary>
        [NotNullValidator(MessageTemplate = "The QueryID cannot be
null")]
        public virtual int QueryId { get; set; }

        /// <summary>
        /// Gets or sets the SearchConceptName
        /// </summary>
        [NotNullValidator(MessageTemplate = "The SearchConceptName
cannot be null")]
        public virtual string SearchConceptName { get; set; }

        /// <summary>
        /// Gets or sets the SearchConceptValue
        /// </summary>
        [NotNullValidator(MessageTemplate = "The SearchConceptValue
cannot be null")]
        public virtual string SearchConceptValue { get; set; }

        /// <summary>
        /// Gets or sets the ResultProtocolName
        /// </summary>
        [NotNullValidator(MessageTemplate = "The ProtocolName cannot
be null")]
        public virtual string ResultProtocolName { get; set; }

        /// <summary>
        /// Gets or sets the SearchNameSynonyms
    }
}

```



```

    /// </summary>
    [NotNullValidator(MessageTemplate = "The SearchNameSynonyms
cannot be null")]
    public virtual IList<string> SearchNameSynonyms { get; set; }

    public string GetSynonymsAsString()
    {
        StringBuilder tmpS = new StringBuilder();
        foreach (string SNS in this.SearchNameSynonyms)
        {
            tmpS.Append(SNS + ";");
        }
        return tmpS.ToString();
    }
}

```

Buffie.Query Service

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data.Objects;
using System.Data.Objects.DataClasses;
using Buffie.Core;
using Buffie.Entities;
using Microsoft.Practices.EnterpriseLibrary.Validation;
using Buffie.Utils;

namespace Buffie.Services
{
    /// <summary>
    /// class to provide the DT service for query
    /// </summary>
    public class BuffieQueryService
    {
        BuffieEntities _dataContext = new BuffieEntities();
        /// <summary>
        /// default constructor
        /// </summary>
        public BuffieQueryService()
        {
        }

        /// <summary>
        /// method to create a query record in the database and
        /// </summary>
        /// <param name="NewQuery"></param>
        /// <returns>The QueryID generated from the database if
success otherwise return 0</returns>
        public string CreateQuery(Buffie.Core.BuffieQuery NewQuery,
int CurUserId)
        {
            CurUserId = 1;
            try
            {
                if (NewQuery == null)
                {

```

```

        throw new ArgumentNullException("NewQuery", "The
specified NewQuery cannot be null");
    }
    else
    {
        ValidationResult results = NewQuery.Validate();
        if (!results.IsValid)
        {
            throw new ValidationException(results);
        }
        else
        {
            //select the reference to user who creates
this Query.
            var _muser = (from user in _dataContext.User
                where user.UserId == CurUserId
                select user).First();

            var data = new Buffie.Entities.Query {
User=_muser, SearchConceptName=NewQuery.SearchConceptName,
SearchConceptValue = NewQuery.SearchConceptValue,
ProtocolName = NewQuery.ResultProtocolName,
SearchNameSynonyms = NewQuery.GetSynonymsAsString(),
Active
= true };

            _dataContext.AddToQuery(data);
            _dataContext.SaveChanges();
            //return the newquery object with new id
value.
            NewQuery.QueryId = data.QueryId;
            return "Success";
        }
    }
}
catch (Exception ex)
{
    return "Error";
}
}

/// <summary>
/// this method returns buffiequery object for a given
queryid from the database.
/// </summary>
/// <param name="QID"></param>
/// <returns></returns>
public Buffie.Core.BuffieQuery GetQuery(int QID)
{
    try
    {
        var Query = _dataContext.Query.First(q => q.QueryId
== QID);
        return new Buffie.Core.BuffieQuery { QueryId =
Query.QueryId, UserId= Query.User.UserId,

```

```
        ResultProtocolName = Query.ProtocolName,
SearchConceptName= Query.SearchConceptName,
        SearchConceptValue=Query.SearchConceptValue,

SearchNameSynonyms=Query.SearchNameSynonyms.StringToList(';')    } ;

    }
    catch (Exception ex)
    {
        return null;
    }

}

}
}
```

Buffie.Core.Provider

```
using Microsoft.Practices.EnterpriseLibrary.Validation.Validators;

namespace Buffie.Core
{
    /// <summary>
    /// class that defines the provider of data for Buffie system
    /// </summary>
    public class BuffieProvider
    {
        /// <summary>
        /// Default constructor
        /// </summary>
        public BuffieProvider()
        {
        }

        /// <summary>
        /// unique provider Id
        /// </summary>
        [NotNullValidator(MessageTemplate = "The MessageId cannot be
null")]
        public virtual int ProviderId { get; set; }

        /// <summary>
        /// Gets or sets the ProviderName
        /// </summary>
        [NotNullValidator(MessageTemplate = "The ProviderName cannot
be null")]
        public virtual string ProviderName { get; set; }

        /// <summary>
        /// Gets or sets the ProtocolName
        /// </summary>
        [NotNullValidator(MessageTemplate = "The ProtocolName cannot
be null")]
        public virtual string ProtocolName { get; set; }
    }
}
```

```

    /// <summary>
    /// Gets or sets the Country
    /// </summary>
    [NotNullValidator(MessageTemplate = "The Country cannot be
null")]
    public virtual string Country { get; set; }

    /// <summary>
    /// Gets or sets the AccessUrl
    /// </summary>
    [NotNullValidator(MessageTemplate = "The AccessUrl cannot be
null")]
    public virtual string AccessUrl { get; set; }

    /// <summary>
    /// Gets or sets the QueryParameter
    /// </summary>
    [NotNullValidator(MessageTemplate = "The QueryParameter
cannot be null")]
    public virtual string QueryParameter { get; set; }

    /// <summary>
    /// Gets or sets the ConfigFilePath
    /// </summary>
    [NotNullValidator(MessageTemplate = "The ConfigFilePath
cannot be null")]
    public virtual string ConfigFilePath { get; set; }

    /// <summary>
    /// Gets or sets the Resources
    /// </summary>
    [NotNullValidator(MessageTemplate = "The Resources cannot be
null")]
    public virtual string Resources { get; set; }
}
}

```

Buffie.Provider Services

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data.Objects;
using System.Data.Objects.DataClasses;
using Buffie.Core;
using Buffie.Entities;
using Microsoft.Practices.EnterpriseLibrary.Validation;
using Buffie.Utils;

namespace Buffie.Services
{
    /// <summary>
    /// defines the service interface for provider class.
    /// </summary>
    public class BuffieProviderService
    {
        BuffieEntities _dataContext = new BuffieEntities();
        /// <summary>

```

```

    /// Default Constructor
    /// </summary>
    public BuffieProviderService()
    {
    }

    public Buffie.Core.BuffieProvider GetProvider(int ProviderID)
    {
        try
        {
            var Provider = _dataContext.Provider.First(p =>
p.ProviderId == ProviderID);
            return new Buffie.Core.BuffieProvider
            {
                ProviderId = Provider.ProviderId,
                ProviderName=Provider.ProviderName, AccessUrl=Provider.AccessUrl,
                ProtocolName=Provider.ProtocolName, Country=
                Provider.Country, ConfigFilePath=Provider.ConfigFilePath,
                QueryParameter=Provider.QueryParameter,
                Resources=Provider.Resources
            };
        }
        catch (Exception ex)
        {
            return null;
        }
    }

    /// <summary>
    /// this method returns all the active providers as list of
    BuffieProvider object collection
    /// </summary>
    /// <returns></returns>
    public List<Buffie.Core.BuffieProvider> GetAllProviderList()
    {
        List<Buffie.Core.BuffieProvider> Results = new
List<Buffie.Core.BuffieProvider>();

        try
        {
            var ProviderQuery = _dataContext.Provider.ToList();

            foreach (var PQ in ProviderQuery)
            {
                if (PQ.Active)
                {
                    Results.Add(new BuffieProvider { AccessUrl =
PQ.AccessUrl, ConfigFilePath = PQ.ConfigFilePath,
                    Country = PQ.Country, ProtocolName =
PQ.ProtocolName, ProviderId = PQ.ProviderId,
                    ProviderName = PQ.ProviderName,
                    QueryParameter = PQ.QueryParameter, Resources = PQ.Resources });
                }
            }
            return Results;
        }
        catch (Exception ex)
        {
            return null;
        }
    }

```

```
    }
}
```

Buffie.Utils Services

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.Practices.EnterpriseLibrary.Validation;

namespace Buffie.Utils
{
    /// <summary>
    /// Defines the extension methods used for Buffie framework.
    /// </summary>
    public static class CustomExtensions
    {
        /// <summary>
        /// method to split the input string based on the delimiter
        and return the array of strings as "IList<string>" collection
        /// </summary>
        /// <param name="InputString"></param>
        /// <param name="Delimiter"></param>
        /// <returns></returns>
        public static IList<string> StringToList(this String
        InputString, Char Delimiter)
        {
            IList<string> Result = null;
            try
            {
                foreach( var S in InputString.Split(Delimiter))
                {
                    Result.Add(S);
                }
                return Result;
            }
            catch (Exception ex)
            {
                return Result;
            }
        }

        /// <summary>
        ///
        /// </summary>
        /// <param name="o"></param>
        /// <param name="c"></param>
        /// <returns></returns>
        public static bool In(this object o, IEnumerable c)
        {
            foreach (object i in c)
            {
                if (i.Equals(o)) return true;
            }
            return false;
        }
    }
}
```

```

    public static void ForEach<T>(this IEnumerable<T> items,
Action<T> act)
    {
        foreach (T item in items)
            act(item);
    }

    /// <summary>
    ///
    /// </summary>
    /// <typeparam name="TInput"></typeparam>
    /// <typeparam name="TOutput"></typeparam>
    /// <param name="items"></param>
    /// <param name="f"></param>
    /// <param name="finalMethod"></param>
    public static void ForEachParallel<TInput, TOutput>(this
IEnumerable<TInput> items, Func<TInput, TOutput> f,
Action<IEnumerable<TOutput>> finalMethod)
    {
        Int32 count = items.Count();
        List<TOutput> results = new List<TOutput>(count);

        items.ForEach(x =>
            f.DoAsync(x, y =>
                {
                    results.Add(y);

                    if (results.Count == count)
                        finalMethod(results);
                }
            ));
    }

    /// <summary>
    ///
    /// </summary>
    /// <typeparam name="TInput"></typeparam>
    /// <typeparam name="TResult"></typeparam>
    /// <param name="f"></param>
    /// <param name="arg"></param>
    /// <param name="callback"></param>
    public static void DoAsync<TInput, TResult>(this Func<TInput,
TResult> f, TInput arg, Action<TResult> callback)
    {
        f.BeginInvoke(arg, x => callback(f.EndInvoke(x)), null);
    }

    /// <summary>
    /// String replace function that support
    /// </summary>
    /// <param name="OrigString">Original input string</param>
    /// <param name="FindString">The string that is to be
replaced</param>
    /// <param name="ReplaceWith">The replacement string</param>
    /// <param name="Instance">Instance of the FindString that is
to be found. if Instance = -1 all are replaced</param>
    /// <param name="CaseInsensitive">Case insensitivity
flag</param>

```

```

    /// <returns>updated string or original string if no
matches</returns>
    public static string ReplaceEx(this string OrigString, string
FindString, string ReplaceWith, int Instance, bool CaseInsensitive)
    {
        int at1 = 0;
        for (int x = 0; x < Instance; x++)
        {
            if (CaseInsensitive)
                at1 = OrigString.IndexOf(FindString, at1,
OrigString.Length - at1, StringComparison.OrdinalIgnoreCase);
            else
                at1 = OrigString.IndexOf(FindString, at1);

            if (at1 == -1)                return OrigString;

            if (x < Instance-1)          at1 +=
FindString.Length;
        }
        return OrigString.Substring(0, at1) + ReplaceWith +
OrigString.Substring(at1 + FindString.Length);
        //StringBuilder sb = new StringBuilder(OrigString);
        //sb.Replace(FindString, ReplaceString, at1,
FindString.Length);
        //return sb.ToString();
    }

    /// <summary>
    /// Replaces a substring within a string with another
substring with optional case sensitivity turned off.
    /// </summary>
    /// <param name="OrigString">String to do replacements
on</param>
    /// <param name="FindString">The string to find</param>
    /// <param name="ReplaceString">The string to replace
found string wiht</param>
    /// <param name="CaseInsensitive">If true case
insensitive search is performed</param>
    /// <returns>updated string or original string if no
matches</returns>
    public static string ReplaceEx(this string OrigString, string
FindString, string ReplaceString, bool CaseInsensitive)
    {
        int at1 = 0;
        while(true)
        {
            if (CaseInsensitive)
            {
                at1 = OrigString.IndexOf(FindString, at1,
OrigString.Length - at1, StringComparison.OrdinalIgnoreCase);
            }
            else
            {
                //at1 = OrigString.IndexOf(FindString, at1);
                return OrigString.Replace(FindString,
ReplaceString);
            }

            if (at1 == -1)
                return OrigString;
        }
    }

```



```

        OrigString = OrigString.Substring(0, at1) +
ReplaceString + OrigString.Substring(at1 + FindString.Length);
        at1 += ReplaceString.Length;
    }
}
}
}

```

Buffie.XML Transform Services

```

using System;
using System.IO;
using System.Text;
using System.Collections.Generic;
using System.Linq;
using System.Xml;
using System.Xml.XPath;
using System.Xml.Xsl;

namespace Buffie.Utils
{
    public static class XSLTHelpers
    {
        /// <summary>
        /// this method provides the interface for xslt
transformation
        /// </summary>
        /// <param name="InputXML"></param>
        /// <param name="InputXSL"></param>
        /// <param name="settings"></param>
        /// <param name="resolver"></param>
        /// <param name="argList"></param>
        /// <returns></returns>
        public static string XMLTransform(string InputXML, string
InputXSL, XsltSettings settings, XmlUrlResolver resolver,
XsltArgumentList argList)
        {
            XmlReader xmlReader = null;
            XmlReader xslReader = null;
            StringBuilder Result = new StringBuilder();

            try
            {
                if (InputXML.EndsWith(".xml"))
                {
                    xmlReader = XmlReader.Create(InputXML);
                }
                else
                {
                    //xmlReader = XmlReader.Create(new
MemoryStream(Encoding.UTF8.GetBytes(InputXML)));
                    xmlReader = XmlReader.Create(new
StringReader(InputXML));
                }
                if (InputXSL.EndsWith(".xslt"))
                {
                    xslReader = XmlReader.Create(InputXSL);
                }
            }
        }
    }
}

```

```

        else
        {
            xmlReader = XmlReader.Create(new
MemoryStream(Encoding.UTF8.GetBytes(InputXSL)));
        }

        // Open input xml as an XPathDocument.
        XPathDocument doc = new XPathDocument(xmlReader);

        // Create an XmlWriter to write the output.
        XmlWriter writer = XmlWriter.Create(Result);

        // Create the XslCompiledTransform and load the style
sheet.
        XslCompiledTransform xslt = new
XslCompiledTransform();
        xslt.Load(xmlReader, settings, resolver);

        // Execute the transformation.
        xslt.Transform(doc, argList, writer);

        return Result.ToString();
    }
    catch (Exception ex)
    {
        return "Error";
    }
}
}
}

```

Buffie.Engine Services

```

using System;
using System.IO;
using System.Text;
using System.Collections.Generic;
using System.Linq;
using System.Xml;
using System.Xml.XPath;
using System.Xml.Xsl;
using Buffie.Utills;
using Buffie.Core;
using System.Reflection;
using System.Net;
using System.Diagnostics;

namespace Buffie.Services
{
    public class BuffieEngine
    {
        //properties for this Buffie Engine class
        protected internal Buffie.Core.BuffieQuery NewQuery { get;
set; }
    }
}

```

```

        protected internal List<Buffie.Core.BuffieMessage>
NewMessages { get; set; }
        protected internal List<Buffie.Core.BuffieProvider> Providers
{ get; set; }
        protected internal Buffie.Core.BuffieResult Result { get;
set; }
        protected internal string SearchConcept { get; set; }

public BuffieEngine()
{
    //default constructor
}

public BuffieEngine(string SearchString, IList<string>
SearchSynonyms, String ResponseFormat)
{
    try
    {
        this.NewMessages = new List<BuffieMessage>();
        this.Providers = new List<BuffieProvider>();
        this.Result = new BuffieResult();

        IList<string> Syns = null;

        this.NewQuery = new BuffieQuery { SearchConceptName =
"ScientificName",
                                SearchConceptValue = SearchString,
ResultProtocolName = ResponseFormat,
                                SearchNameSynonyms = SearchSynonyms,
UserId = 0, QueryId = 0 };
    }
    catch (Exception ex)
    {
    }
}

/// <summary>
///
/// </summary>
/// <param name="SearchString"></param>
/// <param name="ResponseFormat"></param>
public bool RunSearch()
{
    if (this.NewQuery.Equals(null)) return false;
    //start processing when a newquery is set
    try
    {
        int CurUserId = 1;
        //create Query and messages.
        var res = new
BuffieQueryService().CreateQuery(this.NewQuery, CurUserId );
        //if new query created succsses fully then
        if (res.Equals("Success"))
        {

```

```

        //create collection of Provider Objs
        BuffieProviderService BPS = new
BuffieProviderService ();
        BuffieMessageService BMS = new
BuffieMessageService ();
        BuffieResultService BRS = new
BuffieResultService ();
        var PLs = BPS.GetAllProviderList ();
        foreach (var P in PLs)
        {
            // create a request object that returns the
requestxml string
            var argList = new
System.Xml.Xsl.XsltArgumentList ();
            argList.AddParam("accessurl", "",
P.AccessUrl);
            argList.AddParam("source", "",
"192.168.1.105");
            argList.AddParam("resource", "",
P.Resources);
            argList.AddParam("conceptname", "",
this.NewQuery.SearchConceptName);
            argList.AddParam("conceptvalue", "",
this.NewQuery.SearchConceptValue);
            argList.AddParam("currenttime", "",
DateTime.Now.ToString());

            string InputXml = @"<?xml version=""1.0""
encoding=""UTF-8""?><request/>";
            string InputXsl = P.ConfigFilePath +
"detail_search_request.xslt" ;

            string path =
Path.GetDirectoryName (Assembly.GetAssembly (typeof (BuffieEngine)).Code
Base);

            string requestXML =
XSLTHelpers.XMLTransform (InputXml,
                                InputXsl, new
System.Xml.Xsl.XsltSettings { EnableScript = true }, null, argList);

            //string requestXML = "test" ;
            Debug.Print (requestXML);

            string TransformFilepath = "None";
            if (P.ProtocolName !=
this.NewQuery.ResultProtocolName)
                TransformFilepath = P.ConfigFilePath +
P.ProtocolName + "_to_" + this.NewQuery.ResultProtocolName + ".xslt";

            if (this.NewQuery.SearchNameSynonyms.Count >
0)
            {
                // change request xml,
            }

            // create a new message for each provider.
            Buffie.Core.BuffieMessage NewMessage = new
Buffie.Core.BuffieMessage { ProviderId = P.ProviderId, QueryId =
NewQuery.QueryId,

```

```

RequestMessage = requestXML,

RequestDestination = P.AccessUrl + P.QueryParameter,

RequestSent = DateTime.Now, ResponseMessage = "",

ResponseReceived = null, UserId = NewQuery.UserId,

XSLTFileName = TransformFilepath };
    var Result = BMS.CreateMessage(NewMessage,
NewQuery.QueryId);
    if (Result.Equals("Success"))
    {
        NewMessages.Add(NewMessage);
        //run a loop for each synonyms
        // change request xml replace scientific
name with synonym name and create message
    }

    }

    Func<BuffieMessage, BuffieMessage> f1 = uri =>
    {
        try
        {
            WebRequest request =
WebRequest.Create(uri.RequestDestination + uri.RequestMessage);
            request.Timeout = 30000;
            request.Credentials =
CredentialCache.DefaultCredentials;
            WebResponse response =
(HttpWebResponse)request.GetResponse();
            uri.ResponseMessage = new
StreamReader(response.GetResponseStream()).ReadToEnd().ToString();
            uri.ResponseReceived = DateTime.Now;
        }
        catch (Exception ex)
        {
            uri.ResponseMessage = ex.Message;
        }
        finally
        {
            var Res = BMS.UpdateMessage(uri);
        }
        return uri;
    };

    StringBuilder ResultsForQuery = new
StringBuilder("<results>");
    NewMessages.ForEachParallel(f1, result =>
result.ForEach(val =>
        {

            if(val.XSLTFileName != "None")
                {

ResultsForQuery.Append(XSLTHelpers.XMLTransform(val.ResponseMessage,

```

```

val.XSLTFileName, null,null,null));
                                }else
                                {

ResultsForQuery.Append("");
                                }
                                }));

                                string QueryRes =
BRS.CreateResults(ResultsForQuery.ToString(), this.NewQuery.QueryId,
CurUserId);

                                }
                                return true;
                                }
                                catch (Exception ex)
                                {
                                return false;
                                }

                                }
                                }
}

```

AJAX for Enriching Query in JavaScript:

```

var QueryEnrichingServiceProxy;

// Initializes global and proxy default variables.
function pageLoad() {

    // Instantiate the service proxy.
    QueryEnrichingServiceProxy = new QueryEnrichingService();

    // Set the default call back functions.

QueryEnrichingServiceProxy.set_defaultSucceededCallback(SucceededCall
back);

QueryEnrichingServiceProxy.set_defaultFailedCallback(FailedCallback);
}

function GetSynonyms(sn) {
    var x = document.getElementById(sn);
    var sname = x.getAttribute("value").toString();
    var val =
QueryEnrichingServiceProxy.GetSpeciesNameSynonyms('Species2000',
sname);
}

// Callback function that processes the service return value.
function SucceededCallback(result) {

```

```
alert("I am in SucceededCallback" + result.toString());
var str1 = result.toString().split(";", 5);
var RsltElem = document.getElementById("Results");

var relem = document.getElementById(hdnResult).value = result;

// var options = RsltElem.getElementsByTagName("option");

if (str1 != null) {
    var elem = "<SELECT>";
    for (var x in str1) {
        if (str1[x].length > 0) {
            elem = elem + '<OPTION value="' + str1[x] + '>' +
str1[x] + "</OPTION> ";
        }
    }
    RsltElem.innerHTML = "</SELECT>" + elem;
}

}

function FailedCallback(error, userContext, methodName) {
    alert("I am in FailedCallback");
    if (error != null) {
        var RsltElem = document.getElementById("Results");

        RsltElem.innerHTML = "An error occurred: " +
            error.get_message();
    }
}

if (typeof (Sys) !== "undefined")
Sys.Application.notifyScriptLoaded();
```

Enriching Query - WebService Call Layer:

```
using System;
using System.Collections.Specialized;
using System.Collections.Generic;
using System.Linq;
using System.Web.Services;
using System.Net;
using System.Text;
using System.Xml.Linq;
using System.Web.Configuration;
using System.IO;

/// <summary>
/// Summary description for QueryEnrichingService
/// </summary>
[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
// To allow this Web Service to be called from script, using ASP.NET
AJAX, uncomment the following line.
[System.Web.Script.Services.ScriptService]
public class QueryEnrichingService : System.Web.Services.WebService
{
```

```

public QueryEnrichingService()
{
    //Uncomment the following line if using designed components
    //InitializeComponent();
    string targetUrl = Cfg["MsgDestn.Url."];
}

public static NameValueCollection Cfg { get { return
(NameValueCollection)WebConfigurationManager.GetSection("appSettings"
); } }

[WebMethod]
public string GetSpeciesNameSynonyms(string providerCode, string
data)
{
    //call the species 2000 webservice are process the return
xml
    string MsgResponse = null;
    string RetVal = null;
    string targetUrl = Cfg["MsgDestn.Url." + providerCode];
    string contentType = Cfg["MsgDestn.ContentType." +
providerCode];

    try
    {
        string RequestUrl = targetUrl + data;
        WebRequest request =
(WebRequest)WebRequest.Create(RequestUrl);
        request.Method = "POST";
        if (contentType != null && contentType.Length >
0)
            request.ContentType = contentType;
        else
            request.ContentType = "application/x-www-
form-urlencoded";

        string certificateFile = Cfg["MsgDestn.CertFile."
+ providerCode];

        if (certificateFile != null)
        {
            System.Security.Cryptography.X509Certificates.X509Certificate cert =
System.Security.Cryptography.X509Certificates.X509Certificate.CreateF
romCertFile(@certificateFile);

            ((HttpWebRequest)request).ClientCertificates.Add(cert);
        }
        //Stream requestStream =
request.GetRequestStream();
        //StreamWriter requestWriter = new
StreamWriter(requestStream);
        //string urlEncode = Cfg["MsgDestn.UrlEncode." +
providerCode];
        //string xmlPrefix = Cfg["MsgDestn.XmlPrefix." +
providerCode];
        //if (urlEncode != null && bool.Parse(urlEncode))
        //    requestWriter.Write(xmlPrefix +
HttpUtility.UrlEncode(data));
        //else
        //    requestWriter.Write(xmlPrefix + data);

```



```
        //requestWriter.Close();
        Stream responseStream =
request.GetResponse().GetResponseStream();
        StreamReader responseReader = new
StreamReader(responseStream, Encoding.GetEncoding("utf-8"));
        MsgResponse = responseReader.ReadToEnd();
        responseReader.Close();

        // parse the xml and return the synonyms
        XElement root = XElement.Parse(MsgResponse);

        IEnumerable<XElement> results = from el in
root.Elements("result")
                                        select el;

       RetVal = (from e2 in
results.Elements("accepted_name").Elements("name")
                select (string)e2).Aggregate(new
StringBuilder(),
                                        (sb, i)
=> sb.Append(i + ";"),
                                        sb =>
sb.ToString());
    }
    catch (Exception exp)
    {
        //log.Error("Error getting quote from " + targetUrl,
exp);
        return exp.Message;
    }
    return RetVal;
}
}
```

Appendix C

SQL code for BUFFIE Database and Entity Data Model Services

```
USE [Buffie]
GO
/***** Object: User [buffie] *****/
CREATE USER [buffie] WITHOUT LOGIN WITH DEFAULT_SCHEMA=[dbo]
GO
/***** Object: Table [dbo].[User] *****/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[User] (
    [UserId] [int] IDENTITY(1,1) NOT NULL,
    [UserName] [nvarchar] (50) NOT NULL,
    [Password] [nvarchar] (50) NOT NULL,
    [UserRole] [nvarchar] (50) NULL,
    [EmailId] [nvarchar] (150) NULL,
    [Institution] [nvarchar] (50) NULL,
    [Active] [bit] NOT NULL,
    CONSTRAINT [PK_User] PRIMARY KEY CLUSTERED
(
    [UserId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
ON [PRIMARY]
) ON [PRIMARY]
GO
EXEC sys.sp_addextendedproperty @name=N'MS_Description',
@value=N'Holds the user login information for BUFFIE.' ,
@level0type=N'SHEMA',@level0name=N'dbo',
@level1type=N'TABLE',@level1name=N'User'
GO

/***** Object: Table [dbo].[Provider] *****/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Provider] (
    [ProviderId] [int] IDENTITY(1,1) NOT NULL,
    [ProviderName] [nvarchar] (50) NOT NULL,
    [ProtocolName] [nvarchar] (50) NOT NULL,
    [Country] [nvarchar] (50) NULL,
    [AccessUrl] [nvarchar] (500) NULL,
    [QueryParameter] [nvarchar] (50) NULL,
    [ConfigFilePath] [nvarchar] (150) NULL,
    [Resources] [nvarchar] (50) NULL,
```

```

        [Active] [bit] NOT NULL,
    CONSTRAINT [PK_Provider] PRIMARY KEY CLUSTERED
    (
        [ProviderId] ASC
    )WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
    IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
    ON [PRIMARY]
    ) ON [PRIMARY]
GO

/***** Object: Table [dbo].[Query] *****/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Query] (
    [QueryId] [int] IDENTITY(1,1) NOT NULL,
    [UserId] [int] NOT NULL,
    [SearchConceptName] [nvarchar](200) NULL,
    [SearchConceptValue] [nvarchar](200) NULL,
    [ProtocolName] [nvarchar](50) NULL,
    [SearchNameSynonyms] [nvarchar](550) NULL,
    [Active] [bit] NOT NULL,
    CONSTRAINT [PK_Query] PRIMARY KEY CLUSTERED
    (
        [QueryId] ASC
    )WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
    IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
    ON [PRIMARY]
    ) ON [PRIMARY]
GO

/***** Object: Table [dbo].[Message] *****/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Message] (
    [MessageID] [int] IDENTITY(1,1) NOT NULL,
    [QueryId] [int] NOT NULL,
    [ProviderID] [int] NULL,
    [RequestMessage] [xml] NULL,
    [RequestSent] [datetime] NULL,
    [ResponseMessage] [nvarchar](max) NULL,
    [ResponseReceived] [datetime] NULL,
    [Active] [bit] NOT NULL,
    CONSTRAINT [PK_Message] PRIMARY KEY CLUSTERED
    (
        [MessageID] ASC
    )WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
    IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
    ON [PRIMARY]
    ) ON [PRIMARY]
GO
EXEC sys.sp_addextendedproperty @name=N'MS_Description',
@value=N'This table holds the enriched request messages and responses
received for

```

```

each request message from the data provider' ,
@level0type=N'SHEMA',@level0name=N'dbo',
@level1type=N'TABLE',@level1name=N'Message'
GO

/***** Object: Table [dbo].[Results] *****/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Results](
    [ResultsId] [int] IDENTITY(1,1) NOT NULL,
    [UserId] [int] NOT NULL,
    [QueryId] [int] NOT NULL,
    [ResultsForQuery] [nvarchar](max) NOT NULL,
    CONSTRAINT [PK_Results] PRIMARY KEY CLUSTERED
(
    [ResultsId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
ON [PRIMARY]
) ON [PRIMARY]
GO

/***** Object: StoredProcedure [dbo].[GetQueryResults] *****/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =====
-- Author:          <Author,,Name>
-- Create date:    <Create Date,,>
-- Description:    <Description,,>
-- =====
CREATE PROCEDURE [dbo].[GetQueryResults]
    -- Add the parameters for the stored procedure here
    @QId int

AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    -- Insert statements for procedure here
    SELECT Provider.ProviderName, Provider.ProtocolName,
[Message].ResponseMessage from Provider, [Message]
    WHERE [Message].ProviderID = Provider.ProviderId and
[Message].QueryId = @QId
END
GO
/***** Object: Default [DF_User_Active] *****/
ALTER TABLE [dbo].[User] ADD CONSTRAINT [DF_User_Active] DEFAULT
((1)) FOR [Active]
GO
/***** Object: Default [DF_Query_Active] *****/
ALTER TABLE [dbo].[Query] ADD CONSTRAINT [DF_Query_Active] DEFAULT
((1)) FOR [Active]
GO
/***** Object: ForeignKey [FK_Query_User] *****/

```

```

ALTER TABLE [dbo].[Query] WITH NOCHECK ADD CONSTRAINT
[FK_Query_User] FOREIGN KEY([UserId])
REFERENCES [dbo].[User] ([UserId])
GO
ALTER TABLE [dbo].[Query] CHECK CONSTRAINT [FK_Query_User]
GO
EXEC sys.sp_addextendedproperty @name=N'MS_Description',
@value=N'Each user can have many queires' ,
@level0type=N'SHEMA',@level0name=N'dbo',
@level1type=N'TABLE',@level1name=N'Query',
@level2type=N'CONSTRAINT',@level2name=N'FK_Query_User'
GO
/***** Object: ForeignKey [FK_Message_Provider] *****/
ALTER TABLE [dbo].[Message] WITH NOCHECK ADD CONSTRAINT
[FK_Message_Provider] FOREIGN KEY([ProviderID])
REFERENCES [dbo].[Provider] ([ProviderId])
GO
ALTER TABLE [dbo].[Message] CHECK CONSTRAINT [FK_Message_Provider]
GO
/***** Object: ForeignKey [FK_Message_Query] *****/
ALTER TABLE [dbo].[Message] WITH NOCHECK ADD CONSTRAINT
[FK_Message_Query] FOREIGN KEY([QueryId])
REFERENCES [dbo].[Query] ([QueryId])
GO
ALTER TABLE [dbo].[Message] CHECK CONSTRAINT [FK_Message_Query]
GO
EXEC sys.sp_addextendedproperty @name=N'MS_Description',
@value=N'Each Query generates multiple messages. ' ,
@level0type=N'SHEMA',@level0name=N'dbo',
@level1type=N'TABLE',@level1name=N'Message',
@level2type=N'CONSTRAINT',@level2name=N'FK_Message_Query'
GO
/***** Object: ForeignKey [FK_Results_Query] *****/
ALTER TABLE [dbo].[Results] WITH NOCHECK ADD CONSTRAINT
[FK_Results_Query] FOREIGN KEY([QueryId])
REFERENCES [dbo].[Query] ([QueryId])
NOT FOR REPLICATION
GO
ALTER TABLE [dbo].[Results] CHECK CONSTRAINT [FK_Results_Query]
GO

```

BUFFIE EntityDataModel

```

//-----
//-----
// <auto-generated>
//     This code was generated by a tool.
//     Runtime Version:2.0.50727.3074
//
//     Changes to this file may cause incorrect behavior and will be
lost if
//     the code is regenerated.
// </auto-generated>
//-----
//-----

[assembly:
global::System.Data.Objects.DataClasses.EdmSchemaAttribute()]

```

```
[assembly:
global::System.Data.Objects.DataClasses.EdmRelationshipAttribute("BuffieModel", "FK_Message_Provider", "Provider",
global::System.Data.Metadata.Edm.RelationshipMultiplicity.ZeroOrOne,
typeof(Buffie.Entities.Provider), "Message",
global::System.Data.Metadata.Edm.RelationshipMultiplicity.Many,
typeof(Buffie.Entities.Message))]
[assembly:
global::System.Data.Objects.DataClasses.EdmRelationshipAttribute("BuffieModel", "FK_Message_Query", "Query",
global::System.Data.Metadata.Edm.RelationshipMultiplicity.One,
typeof(Buffie.Entities.Query), "Message",
global::System.Data.Metadata.Edm.RelationshipMultiplicity.Many,
typeof(Buffie.Entities.Message))]
[assembly:
global::System.Data.Objects.DataClasses.EdmRelationshipAttribute("BuffieModel", "FK_Query_User", "User",
global::System.Data.Metadata.Edm.RelationshipMultiplicity.One,
typeof(Buffie.Entities.User), "Query",
global::System.Data.Metadata.Edm.RelationshipMultiplicity.Many,
typeof(Buffie.Entities.Query))]
[assembly:
global::System.Data.Objects.DataClasses.EdmRelationshipAttribute("BuffieModel", "FK_Results_Query", "Query",
global::System.Data.Metadata.Edm.RelationshipMultiplicity.One,
typeof(Buffie.Entities.Query), "Results",
global::System.Data.Metadata.Edm.RelationshipMultiplicity.Many,
typeof(Buffie.Entities.Results))]

// Original file name:
// Generation date: 27/05/2009 23:09:12
namespace Buffie.Entities
{
    /// <summary>
    /// There are no comments for BuffieEntities in the schema.
    /// </summary>
    public partial class BuffieEntities :
global::System.Data.Objects.ObjectContext
    {
        /// <summary>
        /// Initializes a new BuffieEntities object using the
        connection string found in the 'BuffieEntities' section of the
        application configuration file.
        /// </summary>
        public BuffieEntities() :
            base("name=BuffieEntities", "BuffieEntities")
        {
            this.OnContextCreated();
        }
        /// <summary>
        /// Initialize a new BuffieEntities object.
        /// </summary>
        public BuffieEntities(string connectionString) :
            base(connectionString, "BuffieEntities")
        {
            this.OnContextCreated();
        }
        /// <summary>
        /// Initialize a new BuffieEntities object.
        /// </summary>

```

```

    public
BuffieEntities(global::System.Data.EntityClient.EntityConnection
connection) :
    base(connection, "BuffieEntities")
    {
        this.OnContextCreated();
    }
    partial void OnContextCreated();
    /// <summary>
    /// There are no comments for Message in the schema.
    /// </summary>
    public global::System.Data.Objects.ObjectQuery<Message>
Message
    {
        get
        {
            if ((this._Message == null))
            {
                this._Message =
base.CreateQuery<Message>("[Message]");
            }
            return this._Message;
        }
    }
    private global::System.Data.Objects.ObjectQuery<Message>
_Message;
    /// <summary>
    /// There are no comments for Provider in the schema.
    /// </summary>
    public global::System.Data.Objects.ObjectQuery<Provider>
Provider
    {
        get
        {
            if ((this._Provider == null))
            {
                this._Provider =
base.CreateQuery<Provider>("[Provider]");
            }
            return this._Provider;
        }
    }
    private global::System.Data.Objects.ObjectQuery<Provider>
_Provider;
    /// <summary>
    /// There are no comments for Query in the schema.
    /// </summary>
    public global::System.Data.Objects.ObjectQuery<Query> Query
    {
        get
        {
            if ((this._Query == null))
            {
                this._Query = base.CreateQuery<Query>("[Query]");
            }
            return this._Query;
        }
    }
    private global::System.Data.Objects.ObjectQuery<Query>
_Query;
    /// <summary>

```

```

    /// There are no comments for Results in the schema.
    /// </summary>
    public global::System.Data.Objects.ObjectQuery<Results>
Results
    {
        get
        {
            if ((this._Results == null))
            {
                this._Results =
base.CreateQuery<Results>("[Results]");
            }
            return this._Results;
        }
        private global::System.Data.Objects.ObjectQuery<Results>
_Results;
    /// <summary>
    /// There are no comments for User in the schema.
    /// </summary>
    public global::System.Data.Objects.ObjectQuery<User> User
    {
        get
        {
            if ((this._User == null))
            {
                this._User = base.CreateQuery<User>("[User]");
            }
            return this._User;
        }
        private global::System.Data.Objects.ObjectQuery<User> _User;
    /// <summary>
    /// There are no comments for Message in the schema.
    /// </summary>
    public void AddToMessage(Message message)
    {
        base.AddObject("Message", message);
    }
    /// <summary>
    /// There are no comments for Provider in the schema.
    /// </summary>
    public void AddToProvider(Provider provider)
    {
        base.AddObject("Provider", provider);
    }
    /// <summary>
    /// There are no comments for Query in the schema.
    /// </summary>
    public void AddToQuery(Query query)
    {
        base.AddObject("Query", query);
    }
    /// <summary>
    /// There are no comments for Results in the schema.
    /// </summary>
    public void AddToResults(Results results)
    {
        base.AddObject("Results", results);
    }
    /// <summary>

```



```

    /// There are no comments for User in the schema.
    /// </summary>
    public void AddToUser(User user)
    {
        base.AddObject("User", user);
    }
}
/// <summary>
/// There are no comments for BuffieModel.Message in the schema.
/// </summary>
/// <KeyProperties>
/// MessageID
/// </KeyProperties>

[global::System.Data.Objects.DataClasses.EdmEntityTypeAttribute (Names
paceName="BuffieModel", Name="Message")]

[global::System.Runtime.Serialization.DataContractAttribute (IsReferen
ce=true)]
    [global::System.Serializable()]
    public partial class Message :
global::System.Data.Objects.DataClasses.EntityObject
    {
        /// <summary>
        /// Create a new Message object.
        /// </summary>
        /// <param name="messageID">Initial value of
MessageID.</param>
        /// <param name="active">Initial value of Active.</param>
        public static Message CreateMessage(int messageID, bool
active)
        {
            Message message = new Message();
            message.MessageID = messageID;
            message.Active = active;
            return message;
        }
        /// <summary>
        /// There are no comments for Property MessageID in the
schema.
        /// </summary>

[global::System.Data.Objects.DataClasses.EdmScalarPropertyAttribute (E
ntityKeyProperty=true, IsNullable=false)]
        [global::System.Runtime.Serialization.DataMemberAttribute()]
        public int MessageID
        {
            get
            {
                return this._MessageID;
            }
            set
            {
                this.OnMessageIDChanging(value);
                this.ReportPropertyChanging("MessageID");
                this._MessageID =
global::System.Data.Objects.DataClasses.StructuralObject.SetValidValu
e(value);
                this.ReportPropertyChanged("MessageID");
                this.OnMessageIDChanged();
            }
        }
    }

```

```

    }
    private int _MessageID;
    partial void OnMessageIDChanging(int value);
    partial void OnMessageIDChanged();
    /// <summary>
    /// There are no comments for Property RequestMessage in the
schema.
    /// </summary>

[global::System.Data.Objects.DataClasses.EdmScalarPropertyAttribute()]
]
    [global::System.Runtime.Serialization.DataMemberAttribute()]
    public string RequestMessage
    {
        get
        {
            return this._RequestMessage;
        }
        set
        {
            this.OnRequestMessageChanging(value);
            this.ReportPropertyChanging("RequestMessage");
            this._RequestMessage =
global::System.Data.Objects.DataClasses.StructuralObject.SetValidValu
e(value, true);
            this.ReportPropertyChanging("RequestMessage");
            this.OnRequestMessageChanged();
        }
    }
    private string _RequestMessage;
    partial void OnRequestMessageChanging(string value);
    partial void OnRequestMessageChanged();
    /// <summary>
    /// There are no comments for Property RequestSent in the
schema.
    /// </summary>

[global::System.Data.Objects.DataClasses.EdmScalarPropertyAttribute()]
]
    [global::System.Runtime.Serialization.DataMemberAttribute()]
    public global::System.Nullable<global::System.DateTime>
RequestSent
    {
        get
        {
            return this._RequestSent;
        }
        set
        {
            this.OnRequestSentChanging(value);
            this.ReportPropertyChanging("RequestSent");
            this._RequestSent =
global::System.Data.Objects.DataClasses.StructuralObject.SetValidValu
e(value);
            this.ReportPropertyChanging("RequestSent");
            this.OnRequestSentChanged();
        }
    }
    private global::System.Nullable<global::System.DateTime>
_RequestSent;

```

```

        partial void
OnRequestSentChanging(global::System.Nullable<global::System.DateTime
> value);
        partial void OnRequestSentChanged();
        /// <summary>
        /// There are no comments for Property ResponseMessage in the
schema.
        /// </summary>

[global::System.Data.Objects.DataClasses.EdmScalarPropertyAttribute()]
    [global::System.Runtime.Serialization.DataMemberAttribute()]
    public string ResponseMessage
    {
        get
        {
            return this._ResponseMessage;
        }
        set
        {
            this.OnResponseMessageChanging(value);
            this.ReportPropertyChanging("ResponseMessage");
            this._ResponseMessage =
global::System.Data.Objects.DataClasses.StructuralObject.SetValidValu
e(value, true);
            this.ReportPropertyChanging("ResponseMessage");
            this.OnResponseMessageChanged();
        }
        private string _ResponseMessage;
        partial void OnResponseMessageChanging(string value);
        partial void OnResponseMessageChanged();
        /// <summary>
        /// There are no comments for Property ResponseReceived in
the schema.
        /// </summary>

[global::System.Data.Objects.DataClasses.EdmScalarPropertyAttribute()]
    [global::System.Runtime.Serialization.DataMemberAttribute()]
    public global::System.Nullable<global::System.DateTime>
ResponseReceived
    {
        get
        {
            return this._ResponseReceived;
        }
        set
        {
            this.OnResponseReceivedChanging(value);
            this.ReportPropertyChanging("ResponseReceived");
            this._ResponseReceived =
global::System.Data.Objects.DataClasses.StructuralObject.SetValidValu
e(value);
            this.ReportPropertyChanging("ResponseReceived");
            this.OnResponseReceivedChanged();
        }
        private global::System.Nullable<global::System.DateTime>
_ResponseReceived;

```

```

        partial void
OnResponseReceivedChanging(global::System.Nullable<global::System.Date
Time> value);
        partial void OnResponseReceivedChanged();
        /// <summary>
        /// There are no comments for Property Active in the schema.
        /// </summary>

[global::System.Data.Objects.DataClasses.EdmScalarPropertyAttribute(Is
Nullable=false)]
        [global::System.Runtime.Serialization.DataMemberAttribute()]
        public bool Active
        {
            get
            {
                return this._Active;
            }
            set
            {
                this.OnActiveChanging(value);
                this.ReportPropertyChanging("Active");
                this._Active =
global::System.Data.Objects.DataClasses.StructuralObject.SetValidValu
e(value);
                this.ReportPropertyChanging("Active");
                this.OnActiveChanged();
            }
        }
        private bool _Active;
        partial void OnActiveChanging(bool value);
        partial void OnActiveChanged();
        /// <summary>
        /// There are no comments for Provider in the schema.
        /// </summary>

[global::System.Data.Objects.DataClasses.EdmRelationshipNavigationPro
pertyAttribute("BuffieModel", "FK_Message_Provider", "Provider")]
        [global::System.Xml.Serialization.XmlIgnoreAttribute()]
        [global::System.Xml.Serialization.SoapIgnoreAttribute()]
        [global::System.Runtime.Serialization.DataMemberAttribute()]
        public Provider Provider
        {
            get
            {
                return
((global::System.Data.Objects.DataClasses.IEntityWithRelationships) (t
his)).RelationshipManager.GetRelatedReference<Provider>("BuffieModel.
FK_Message_Provider", "Provider").Value;
            }
            set
            {
                ((global::System.Data.Objects.DataClasses.IEntityWithRelationships) (t
his)).RelationshipManager.GetRelatedReference<Provider>("BuffieModel.
FK_Message_Provider", "Provider").Value = value;
            }
        }
        /// <summary>
        /// There are no comments for Provider in the schema.
        /// </summary>
        [global::System.ComponentModel.BrowsableAttribute(false)]

```

```

        [global::System.Runtime.Serialization.DataMemberAttribute()]
        public
global::System.Data.Objects.DataClasses.EntityReference<Provider>
ProviderReference
    {
        get
        {
            return
((global::System.Data.Objects.DataClasses.IEntityWithRelationships) (t
his)).RelationshipManager.GetRelatedReference<Provider>("BuffieModel.
FK_Message_Provider", "Provider");
        }
        set
        {
            if ((value != null))
            {
                ((global::System.Data.Objects.DataClasses.IEntityWithRelationships) (t
his)).RelationshipManager.InitializeRelatedReference<Provider>("Buffie
Model.FK_Message_Provider", "Provider", value);
            }
        }
    }
    /// <summary>
    /// There are no comments for Query in the schema.
    /// </summary>

[global::System.Data.Objects.DataClasses.EdmRelationshipNavigationPro
pertyAttribute("BuffieModel", "FK_Message_Query", "Query")]
[global::System.Xml.Serialization.XmlIgnoreAttribute()]
[global::System.Xml.Serialization.SoapIgnoreAttribute()]
[global::System.Runtime.Serialization.DataMemberAttribute()]
public Query Query
    {
        get
        {
            return
((global::System.Data.Objects.DataClasses.IEntityWithRelationships) (t
his)).RelationshipManager.GetRelatedReference<Query>("BuffieModel.FK_
Message_Query", "Query").Value;
        }
        set
        {
            ((global::System.Data.Objects.DataClasses.IEntityWithRelationships) (t
his)).RelationshipManager.GetRelatedReference<Query>("BuffieModel.FK_
Message_Query", "Query").Value = value;
        }
    }
    /// <summary>
    /// There are no comments for Query in the schema.
    /// </summary>
[global::System.ComponentModel.BrowsableAttribute(false)]
[global::System.Runtime.Serialization.DataMemberAttribute()]
public
global::System.Data.Objects.DataClasses.EntityReference<Query>
QueryReference
    {
        get
        {

```

```

        return
        ((global::System.Data.Objects.DataClasses.IEntityWithRelationships) (this)).RelationshipManager.GetRelatedReference<Query>("BuffieModel.FK_Message_Query", "Query");
    }
    set
    {
        if ((value != null))
        {
            ((global::System.Data.Objects.DataClasses.IEntityWithRelationships) (this)).RelationshipManager.InitializeRelatedReference<Query>("BuffieModel.FK_Message_Query", "Query", value);
        }
    }
}
}
}
/// <summary>
/// There are no comments for BuffieModel.Provider in the schema.
/// </summary>
/// <KeyProperties>
/// ProviderId
/// </KeyProperties>

[global::System.Data.Objects.DataClasses.EdmEntityTypeAttribute (NamespaceName="BuffieModel", Name="Provider")]

[global::System.Runtime.Serialization.DataContractAttribute (IsReference=true)]
[global::System.Serializable()]
public partial class Provider :
global::System.Data.Objects.DataClasses.EntityObject
{
    /// <summary>
    /// Create a new Provider object.
    /// </summary>
    /// <param name="providerId">Initial value of ProviderId.</param>
    /// <param name="providerName">Initial value of ProviderName.</param>
    /// <param name="protocolName">Initial value of ProtocolName.</param>
    /// <param name="active">Initial value of Active.</param>
    public static Provider CreateProvider(int providerId, string providerName, string protocolName, bool active)
    {
        Provider provider = new Provider();
        provider.ProviderId = providerId;
        provider.ProviderName = providerName;
        provider.ProtocolName = protocolName;
        provider.Active = active;
        return provider;
    }
    /// <summary>
    /// There are no comments for Property ProviderId in the schema.
    /// </summary>

[global::System.Data.Objects.DataClasses.EdmScalarPropertyAttribute (EntityTypeKeyProperty=true, IsNullable=false)]
[global::System.Runtime.Serialization.DataMemberAttribute()]

```

```

public int ProviderId
{
    get
    {
        return this._ProviderId;
    }
    set
    {
        this.OnProviderIdChanging(value);
        this.ReportPropertyChanging("ProviderId");
        this._ProviderId =
global::System.Data.Objects.DataClasses.StructuralObject.SetValidValu
e(value);
        this.ReportPropertyChanging("ProviderId");
        this.OnProviderIdChanged();
    }
}
private int _ProviderId;
partial void OnProviderIdChanging(int value);
partial void OnProviderIdChanged();
/// <summary>
/// There are no comments for Property ProviderName in the
schema.
/// </summary>

[global::System.Data.Objects.DataClasses.EdmScalarPropertyAttribute(Is
Nullable=false)]
[global::System.Runtime.Serialization.DataMemberAttribute()]
public string ProviderName
{
    get
    {
        return this._ProviderName;
    }
    set
    {
        this.OnProviderNameChanging(value);
        this.ReportPropertyChanging("ProviderName");
        this._ProviderName =
global::System.Data.Objects.DataClasses.StructuralObject.SetValidValu
e(value, false);
        this.ReportPropertyChanging("ProviderName");
        this.OnProviderNameChanged();
    }
}
private string _ProviderName;
partial void OnProviderNameChanging(string value);
partial void OnProviderNameChanged();
/// <summary>
/// There are no comments for Property ProtocolName in the
schema.
/// </summary>

[global::System.Data.Objects.DataClasses.EdmScalarPropertyAttribute(Is
Nullable=false)]
[global::System.Runtime.Serialization.DataMemberAttribute()]
public string ProtocolName
{
    get
    {
        return this._ProtocolName;
    }
}

```

```

    }
    set
    {
        this.OnProtocolNameChanging(value);
        this.ReportPropertyChanging("ProtocolName");
        this._ProtocolName =
global::System.Data.Objects.DataClasses.StructuralObject.SetValidValue(value, false);
        this.ReportPropertyChanging("ProtocolName");
        this.OnProtocolNameChanged();
    }
}
private string _ProtocolName;
partial void OnProtocolNameChanging(string value);
partial void OnProtocolNameChanged();
/// <summary>
/// There are no comments for Property Country in the schema.
/// </summary>

[global::System.Data.Objects.DataClasses.EdmScalarPropertyAttribute()]
]
[global::System.Runtime.Serialization.DataMemberAttribute()]
public string Country
{
    get
    {
        return this._Country;
    }
    set
    {
        this.OnCountryChanging(value);
        this.ReportPropertyChanging("Country");
        this._Country =
global::System.Data.Objects.DataClasses.StructuralObject.SetValidValue(value, true);
        this.ReportPropertyChanging("Country");
        this.OnCountryChanged();
    }
}
private string _Country;
partial void OnCountryChanging(string value);
partial void OnCountryChanged();
/// <summary>
/// There are no comments for Property AccessUrl in the
schema.
/// </summary>

[global::System.Data.Objects.DataClasses.EdmScalarPropertyAttribute()]
]
[global::System.Runtime.Serialization.DataMemberAttribute()]
public string AccessUrl
{
    get
    {
        return this._AccessUrl;
    }
    set
    {
        this.OnAccessUrlChanging(value);
        this.ReportPropertyChanging("AccessUrl");
    }
}

```



```

        this._AccessUrl =
global::System.Data.Objects.DataClasses.StructuralObject.SetValidValue(value, true);
        this.ReportPropertyChanged("AccessUrl");
        this.OnAccessUrlChanged();
    }
}
private string _AccessUrl;
partial void OnAccessUrlChanging(string value);
partial void OnAccessUrlChanged();
/// <summary>
/// There are no comments for Property QueryParameter in the
schema.
/// </summary>

[global::System.Data.Objects.DataClasses.EdmScalarPropertyAttribute()]
]
[global::System.Runtime.Serialization.DataMemberAttribute()]
public string QueryParameter
{
    get
    {
        return this._QueryParameter;
    }
    set
    {
        this.OnQueryParameterChanging(value);
        this.ReportPropertyChanging("QueryParameter");
        this._QueryParameter =
global::System.Data.Objects.DataClasses.StructuralObject.SetValidValue(value, true);
        this.ReportPropertyChanged("QueryParameter");
        this.OnQueryParameterChanged();
    }
}
private string _QueryParameter;
partial void OnQueryParameterChanging(string value);
partial void OnQueryParameterChanged();
/// <summary>
/// There are no comments for Property ConfigFilePath in the
schema.
/// </summary>

[global::System.Data.Objects.DataClasses.EdmScalarPropertyAttribute()]
]
[global::System.Runtime.Serialization.DataMemberAttribute()]
public string ConfigFilePath
{
    get
    {
        return this._ConfigFilePath;
    }
    set
    {
        this.OnConfigFilePathChanging(value);
        this.ReportPropertyChanging("ConfigFilePath");
        this._ConfigFilePath =
global::System.Data.Objects.DataClasses.StructuralObject.SetValidValue(value, true);
        this.ReportPropertyChanged("ConfigFilePath");
        this.OnConfigFilePathChanged();
    }
}

```

```

    }
}
private string _ConfigFilePath;
partial void OnConfigFilePathChanging(string value);
partial void OnConfigFilePathChanged();
/// <summary>
/// There are no comments for Property Resources in the
schema.
/// </summary>

[global::System.Data.Objects.DataClasses.EdmScalarPropertyAttribute()]
[global::System.Runtime.Serialization.DataMemberAttribute()]
public string Resources
{
    get
    {
        return this._Resources;
    }
    set
    {
        this.OnResourcesChanging(value);
        this.ReportPropertyChanging("Resources");
        this._Resources =
global::System.Data.Objects.DataClasses.StructuralObject.SetValidValue(value, true);
        this.ReportPropertyChanged("Resources");
        this.OnResourcesChanged();
    }
}
private string _Resources;
partial void OnResourcesChanging(string value);
partial void OnResourcesChanged();
/// <summary>
/// There are no comments for Property Active in the schema.
/// </summary>

[global::System.Data.Objects.DataClasses.EdmScalarPropertyAttribute(IsNullable=false)]
[global::System.Runtime.Serialization.DataMemberAttribute()]
public bool Active
{
    get
    {
        return this._Active;
    }
    set
    {
        this.OnActiveChanging(value);
        this.ReportPropertyChanging("Active");
        this._Active =
global::System.Data.Objects.DataClasses.StructuralObject.SetValidValue(value);
        this.ReportPropertyChanged("Active");
        this.OnActiveChanged();
    }
}
private bool _Active;
partial void OnActiveChanging(bool value);
partial void OnActiveChanged();
/// <summary>

```

```

    /// There are no comments for Message in the schema.
    /// </summary>

[global::System.Data.Objects.DataClasses.EdmRelationshipNavigationPropertyAttribute("BuffieModel", "FK_Message_Provider", "Message")]
[global::System.Xml.Serialization.XmlIgnoreAttribute()]
[global::System.Xml.Serialization.SoapIgnoreAttribute()]
[global::System.Runtime.Serialization.DataMemberAttribute()]
public
global::System.Data.Objects.DataClasses.EntityCollection<Message>
Message
{
    get
    {
        return
((global::System.Data.Objects.DataClasses.IEntityWithRelationships) (this)).RelationshipManager.GetRelatedCollection<Message>("BuffieModel.FK_Message_Provider", "Message");
    }
    set
    {
        if ((value != null))
        {
            ((global::System.Data.Objects.DataClasses.IEntityWithRelationships) (this)).RelationshipManager.InitializeRelatedCollection<Message>("BuffieModel.FK_Message_Provider", "Message", value);
        }
    }
}
    /// <summary>
    /// There are no comments for BuffieModel.Query in the schema.
    /// </summary>
    /// <KeyProperties>
    /// QueryId
    /// </KeyProperties>

[global::System.Data.Objects.DataClasses.EdmEntityTypeAttribute(NameSpaceName="BuffieModel", Name="Query")]

[global::System.Runtime.Serialization.DataContractAttribute(IsReference=true)]
[global::System.Serializable()]
public partial class Query :
global::System.Data.Objects.DataClasses.EntityObject
{
    /// <summary>
    /// Create a new Query object.
    /// </summary>
    /// <param name="queryId">Initial value of QueryId.</param>
    /// <param name="active">Initial value of Active.</param>
    public static Query CreateQuery(int queryId, bool active)
    {
        Query query = new Query();
        query.QueryId = queryId;
        query.Active = active;
        return query;
    }
    /// <summary>
    /// There are no comments for Property QueryId in the schema.

```

```

    /// </summary>

[global::System.Data.Objects.DataClasses.EdmScalarPropertyAttribute (E
ntityKeyProperty=true, IsNullable=false)]
    [global::System.Runtime.Serialization.DataMemberAttribute()]
    public int QueryId
    {
        get
        {
            return this._QueryId;
        }
        set
        {
            this.OnQueryIdChanging (value);
            this.ReportPropertyChanging ("QueryId");
            this._QueryId =
global::System.Data.Objects.DataClasses.StructuralObject.SetValidValu
e (value);

            this.ReportPropertyChanged ("QueryId");
            this.OnQueryIdChanged ();
        }
    }
    private int _QueryId;
    partial void OnQueryIdChanging (int value);
    partial void OnQueryIdChanged ();
    /// <summary>
    /// There are no comments for Property SearchConceptName in
the schema.
    /// </summary>

[global::System.Data.Objects.DataClasses.EdmScalarPropertyAttribute ()
]
    [global::System.Runtime.Serialization.DataMemberAttribute()]
    public string SearchConceptName
    {
        get
        {
            return this._SearchConceptName;
        }
        set
        {
            this.OnSearchConceptNameChanging (value);
            this.ReportPropertyChanging ("SearchConceptName");
            this._SearchConceptName =
global::System.Data.Objects.DataClasses.StructuralObject.SetValidValu
e (value, true);

            this.ReportPropertyChanged ("SearchConceptName");
            this.OnSearchConceptNameChanged ();
        }
    }
    private string _SearchConceptName;
    partial void OnSearchConceptNameChanging (string value);
    partial void OnSearchConceptNameChanged ();
    /// <summary>
    /// There are no comments for Property SearchConceptValue in
the schema.
    /// </summary>

[global::System.Data.Objects.DataClasses.EdmScalarPropertyAttribute ()
]
    [global::System.Runtime.Serialization.DataMemberAttribute()]

```

```

public string SearchConceptValue
{
    get
    {
        return this._SearchConceptValue;
    }
    set
    {
        this.OnSearchConceptValueChanging(value);
        this.ReportPropertyChanging("SearchConceptValue");
        this._SearchConceptValue =
global::System.Data.Objects.DataClasses.StructuralObject.SetValidValu
e(value, true);
        this.ReportPropertyChanging("SearchConceptValue");
        this.OnSearchConceptValueChanged();
    }
}
private string _SearchConceptValue;
partial void OnSearchConceptValueChanging(string value);
partial void OnSearchConceptValueChanged();
/// <summary>
/// There are no comments for Property ProtocolName in the
schema.
/// </summary>

[global::System.Data.Objects.DataClasses.EdmScalarPropertyAttribute()]
[global::System.Runtime.Serialization.DataMemberAttribute()]
public string ProtocolName
{
    get
    {
        return this._ProtocolName;
    }
    set
    {
        this.OnProtocolNameChanging(value);
        this.ReportPropertyChanging("ProtocolName");
        this._ProtocolName =
global::System.Data.Objects.DataClasses.StructuralObject.SetValidValu
e(value, true);
        this.ReportPropertyChanging("ProtocolName");
        this.OnProtocolNameChanged();
    }
}
private string _ProtocolName;
partial void OnProtocolNameChanging(string value);
partial void OnProtocolNameChanged();
/// <summary>
/// There are no comments for Property SearchNameSynonyms in
the schema.
/// </summary>

[global::System.Data.Objects.DataClasses.EdmScalarPropertyAttribute()]
[global::System.Runtime.Serialization.DataMemberAttribute()]
public string SearchNameSynonyms
{
    get
    {
        return this._SearchNameSynonyms;
    }
}

```

```

    }
    set
    {
        this.OnSearchNameSynonymsChanging(value);
        this.ReportPropertyChanging("SearchNameSynonyms");
        this._SearchNameSynonyms =
global::System.Data.Objects.DataClasses.StructuralObject.SetValidValu
e(value, true);
        this.ReportPropertyChanging("SearchNameSynonyms");
        this.OnSearchNameSynonymsChanged();
    }
}
private string _SearchNameSynonyms;
partial void OnSearchNameSynonymsChanging(string value);
partial void OnSearchNameSynonymsChanged();
/// <summary>
/// There are no comments for Property Active in the schema.
/// </summary>

[global::System.Data.Objects.DataClasses.EdmScalarPropertyAttribute(I
sNullable=false)]
[global::System.Runtime.Serialization.DataMemberAttribute()]
public bool Active
{
    get
    {
        return this._Active;
    }
    set
    {
        this.OnActiveChanging(value);
        this.ReportPropertyChanging("Active");
        this._Active =
global::System.Data.Objects.DataClasses.StructuralObject.SetValidValu
e(value);
        this.ReportPropertyChanging("Active");
        this.OnActiveChanged();
    }
}
private bool _Active;
partial void OnActiveChanging(bool value);
partial void OnActiveChanged();
/// <summary>
/// There are no comments for Message in the schema.
/// </summary>

[global::System.Data.Objects.DataClasses.EdmRelationshipNavigationPro
pertyAttribute("BuffieModel", "FK_Message_Query", "Message")]
[global::System.Xml.Serialization.XmlIgnoreAttribute()]
[global::System.Xml.Serialization.SoapIgnoreAttribute()]
[global::System.Runtime.Serialization.DataMemberAttribute()]
public
global::System.Data.Objects.DataClasses.EntityCollection<Message>
Message
{
    get
    {
        return
((global::System.Data.Objects.DataClasses.IEntityWithRelationships)(t
his)).RelationshipManager.GetRelatedCollection<Message>("BuffieModel.
FK_Message_Query", "Message");
    }
}

```

```

    }
    set
    {
        if ((value != null))
        {
            ((global::System.Data.Objects.DataClasses.IEntityWithRelationships) (t
his)).RelationshipManager.InitializeRelatedCollection<Message>("Buffie
Model.FK_Message_Query", "Message", value);
        }
    }
}
/// <summary>
/// There are no comments for User in the schema.
/// </summary>

[global::System.Data.Objects.DataClasses.EdmRelationshipNavigationPro
pertyAttribute("BuffieModel", "FK_Query_User", "User")]
[global::System.Xml.Serialization.XmlIgnoreAttribute()]
[global::System.Xml.Serialization.SoapIgnoreAttribute()]
[global::System.Runtime.Serialization.DataMemberAttribute()]
public User User
{
    get
    {
        return
            ((global::System.Data.Objects.DataClasses.IEntityWithRelationships) (t
his)).RelationshipManager.GetRelatedReference<User>("BuffieModel.FK_Q
uery_User", "User").Value;
    }
    set
    {
        ((global::System.Data.Objects.DataClasses.IEntityWithRelationships) (t
his)).RelationshipManager.GetRelatedReference<User>("BuffieModel.FK_Q
uery_User", "User").Value = value;
    }
}
/// <summary>
/// There are no comments for User in the schema.
/// </summary>
[global::System.ComponentModel.BrowsableAttribute(false)]
[global::System.Runtime.Serialization.DataMemberAttribute()]
public
global::System.Data.Objects.DataClasses.EntityReference<User>
UserReference
{
    get
    {
        return
            ((global::System.Data.Objects.DataClasses.IEntityWithRelationships) (t
his)).RelationshipManager.GetRelatedReference<User>("BuffieModel.FK_Q
uery_User", "User");
    }
    set
    {
        if ((value != null))
        {
            ((global::System.Data.Objects.DataClasses.IEntityWithRelationships) (t

```

```

his)).RelationshipManager.InitializeRelatedReference<User>("BuffieModel.FK_Query_User", "User", value);
    }
}
}
/// <summary>
/// There are no comments for Results in the schema.
/// </summary>

[global::System.Data.Objects.DataClasses.EdmRelationshipNavigationPropertyAttribute("BuffieModel", "FK_Results_Query", "Results")]
[global::System.Xml.Serialization.XmlIgnoreAttribute()]
[global::System.Xml.Serialization.SoapIgnoreAttribute()]
[global::System.Runtime.Serialization.DataMemberAttribute()]
public
global::System.Data.Objects.DataClasses.EntityCollection<Results>
Results
{
    get
    {
        return
((global::System.Data.Objects.DataClasses.IEntityWithRelationships)(this)).RelationshipManager.GetRelatedCollection<Results>("BuffieModel.FK_Results_Query", "Results");
    }
    set
    {
        if ((value != null))
        {
            ((global::System.Data.Objects.DataClasses.IEntityWithRelationships)(this)).RelationshipManager.InitializeRelatedCollection<Results>("BuffieModel.FK_Results_Query", "Results", value);
        }
    }
}
}
/// <summary>
/// There are no comments for BuffieModel.Results in the schema.
/// </summary>
/// <KeyProperties>
/// ResultsId
/// </KeyProperties>

[global::System.Data.Objects.DataClasses.EdmEntityTypeAttribute(NameSpaceName="BuffieModel", Name="Results")]

[global::System.Runtime.Serialization.DataContractAttribute(IsReference=true)]
[global::System.Serializable()]
public partial class Results :
global::System.Data.Objects.DataClasses.EntityObject
{
    /// <summary>
    /// Create a new Results object.
    /// </summary>
    /// <param name="resultsId">Initial value of ResultsId.</param>
    /// <param name="userId">Initial value of UserId.</param>
    /// <param name="resultsForQuery">Initial value of ResultsForQuery.</param>

```



```

        public static Results CreateResults(int resultsId, int
userId, string resultsForQuery)
    {
        Results results = new Results();
        results.ResultsId = resultsId;
        results.UserId = userId;
        results.ResultsForQuery = resultsForQuery;
        return results;
    }
    /// <summary>
    /// There are no comments for Property ResultsId in the
schema.
    /// </summary>

[global::System.Data.Objects.DataClasses.EdmScalarPropertyAttribute(E
ntityKeyProperty=true, IsNullable=false)]
    [global::System.Runtime.Serialization.DataMemberAttribute()]
    public int ResultsId
    {
        get
        {
            return this._ResultsId;
        }
        set
        {
            this.OnResultsIdChanging(value);
            this.ReportPropertyChanging("ResultsId");
            this._ResultsId =
global::System.Data.Objects.DataClasses.StructuralObject.SetValidValu
e(value);

            this.ReportPropertyChanged("ResultsId");
            this.OnResultsIdChanged();
        }
    }
    private int _ResultsId;
    partial void OnResultsIdChanging(int value);
    partial void OnResultsIdChanged();
    /// <summary>
    /// There are no comments for Property UserId in the schema.
    /// </summary>

[global::System.Data.Objects.DataClasses.EdmScalarPropertyAttribute(I
sNullable=false)]
    [global::System.Runtime.Serialization.DataMemberAttribute()]
    public int UserId
    {
        get
        {
            return this._UserId;
        }
        set
        {
            this.OnUserIdChanging(value);
            this.ReportPropertyChanging("UserId");
            this._UserId =
global::System.Data.Objects.DataClasses.StructuralObject.SetValidValu
e(value);

            this.ReportPropertyChanged("UserId");
            this.OnUserIdChanged();
        }
    }

```

```

private int _UserId;
partial void OnUserIdChanging(int value);
partial void OnUserIdChanged();
/// <summary>
/// There are no comments for Property ResultsForQuery in the
schema.
/// </summary>

[global::System.Data.Objects.DataClasses.EdmScalarPropertyAttribute(IsNullable=false)]
[global::System.Runtime.Serialization.DataMemberAttribute()]
public string ResultsForQuery
{
    get
    {
        return this._ResultsForQuery;
    }
    set
    {
        this.OnResultsForQueryChanging(value);
        this.ReportPropertyChanging("ResultsForQuery");
        this._ResultsForQuery =
global::System.Data.Objects.DataClasses.StructuralObject.SetValidValue(value, false);
        this.ReportPropertyChanging("ResultsForQuery");
        this.OnResultsForQueryChanged();
    }
}
private string _ResultsForQuery;
partial void OnResultsForQueryChanging(string value);
partial void OnResultsForQueryChanged();
/// <summary>
/// There are no comments for Query in the schema.
/// </summary>

[global::System.Data.Objects.DataClasses.EdmRelationshipNavigationPropertyAttribute("BuffieModel", "FK_Results_Query", "Query")]
[global::System.Xml.Serialization.XmlIgnoreAttribute()]
[global::System.Xml.Serialization.SoapIgnoreAttribute()]
[global::System.Runtime.Serialization.DataMemberAttribute()]
public Query Query
{
    get
    {
        return
((global::System.Data.Objects.DataClasses.IEntityWithRelationships) (this)).RelationshipManager.GetRelatedReference<Query>("BuffieModel.FK_Results_Query", "Query").Value;
    }
    set
    {
        ((global::System.Data.Objects.DataClasses.IEntityWithRelationships) (this)).RelationshipManager.GetRelatedReference<Query>("BuffieModel.FK_Results_Query", "Query").Value = value;
    }
}
/// <summary>
/// There are no comments for Query in the schema.
/// </summary>
[global::System.ComponentModel.BrowsableAttribute(false)]

```

```

        [global::System.Runtime.Serialization.DataMemberAttribute()]
        public
global::System.Data.Objects.DataClasses.EntityReference<Query>
QueryReference
    {
        get
        {
            return
((global::System.Data.Objects.DataClasses.IEntityWithRelationships) (t
his)).RelationshipManager.GetRelatedReference<Query>("BuffieModel.FK_
Results_Query", "Query");
        }
        set
        {
            if ((value != null))
            {
                ((global::System.Data.Objects.DataClasses.IEntityWithRelationships) (t
his)).RelationshipManager.InitializeRelatedReference<Query>("BuffieMo
del.FK_Results_Query", "Query", value);
            }
        }
    }
}
/// <summary>
/// There are no comments for BuffieModel.User in the schema.
/// </summary>
/// <KeyProperties>
/// UserId
/// </KeyProperties>

[global::System.Data.Objects.DataClasses.EdmEntityTypeAttribute (Names
paceName="BuffieModel", Name="User")]

[global::System.Runtime.Serialization.DataContractAttribute (IsReferen
ce=true)]
[global::System.Serializable()]
public partial class User :
global::System.Data.Objects.DataClasses.EntityObject
{
    /// <summary>
    /// Create a new User object.
    /// </summary>
    /// <param name="userId">Initial value of UserId.</param>
    /// <param name="userName">Initial value of UserName.</param>
    /// <param name="password">Initial value of Password.</param>
    /// <param name="active">Initial value of Active.</param>
    public static User CreateUser(int userId, string userName,
string password, bool active)
    {
        User user = new User();
        user.UserId = userId;
        user.UserName = userName;
        user.Password = password;
        user.Active = active;
        return user;
    }
    /// <summary>
    /// There are no comments for Property UserId in the schema.
    /// </summary>

```

```
[global::System.Data.Objects.DataClasses.EdmScalarPropertyAttribute(EntityKeyProperty=true, IsNullable=false)]
    [global::System.Runtime.Serialization.DataMemberAttribute()]
    public int UserId
    {
        get
        {
            return this._UserId;
        }
        set
        {
            this.OnUserIdChanging(value);
            this.ReportPropertyChanging("UserId");
            this._UserId =
global::System.Data.Objects.DataClasses.StructuralObject.SetValidValue(value);
                this.ReportPropertyChanged("UserId");
                this.OnUserIdChanged();
            }
        }
        private int _UserId;
        partial void OnUserIdChanging(int value);
        partial void OnUserIdChanged();
        /// <summary>
        /// There are no comments for Property UserName in the
schema.
        /// </summary>

[global::System.Data.Objects.DataClasses.EdmScalarPropertyAttribute(IsNullable=false)]
    [global::System.Runtime.Serialization.DataMemberAttribute()]
    public string UserName
    {
        get
        {
            return this._UserName;
        }
        set
        {
            this.OnUserNameChanging(value);
            this.ReportPropertyChanging("UserName");
            this._UserName =
global::System.Data.Objects.DataClasses.StructuralObject.SetValidValue(value, false);
                this.ReportPropertyChanged("UserName");
                this.OnUserNameChanged();
            }
        }
        private string _UserName;
        partial void OnUserNameChanging(string value);
        partial void OnUserNameChanged();
        /// <summary>
        /// There are no comments for Property Password in the
schema.
        /// </summary>

[global::System.Data.Objects.DataClasses.EdmScalarPropertyAttribute(IsNullable=false)]
    [global::System.Runtime.Serialization.DataMemberAttribute()]
    public string Password
```

```

    {
        get
        {
            return this._Password;
        }
        set
        {
            this.OnPasswordChanging(value);
            this.ReportPropertyChanging("Password");
            this._Password =
global::System.Data.Objects.DataClasses.StructuralObject.SetValidValu
e(value, false);
            this.ReportPropertyChanged("Password");
            this.OnPasswordChanged();
        }
    }
    private string _Password;
    partial void OnPasswordChanging(string value);
    partial void OnPasswordChanged();
    /// <summary>
    /// There are no comments for Property UserRole in the
schema.
    /// </summary>

[global::System.Data.Objects.DataClasses.EdmScalarPropertyAttribute()]
]
    [global::System.Runtime.Serialization.DataMemberAttribute()]
    public string UserRole
    {
        get
        {
            return this._UserRole;
        }
        set
        {
            this.OnUserRoleChanging(value);
            this.ReportPropertyChanging("UserRole");
            this._UserRole =
global::System.Data.Objects.DataClasses.StructuralObject.SetValidValu
e(value, true);
            this.ReportPropertyChanged("UserRole");
            this.OnUserRoleChanged();
        }
    }
    private string _UserRole;
    partial void OnUserRoleChanging(string value);
    partial void OnUserRoleChanged();
    /// <summary>
    /// There are no comments for Property EmailId in the schema.
    /// </summary>

[global::System.Data.Objects.DataClasses.EdmScalarPropertyAttribute()]
]
    [global::System.Runtime.Serialization.DataMemberAttribute()]
    public string EmailId
    {
        get
        {
            return this._EmailId;
        }
        set
    }

```

```

        {
            this.OnEmailIdChanging(value);
            this.ReportPropertyChanging("EmailId");
            this._EmailId =
global::System.Data.Objects.DataClasses.StructuralObject.SetValidValu
e(value, true);
            this.ReportPropertyChanging("EmailId");
            this.OnEmailIdChanged();
        }
    }
    private string _EmailId;
    partial void OnEmailIdChanging(string value);
    partial void OnEmailIdChanged();
    /// <summary>
    /// There are no comments for Property Institution in the
schema.
    /// </summary>

[global::System.Data.Objects.DataClasses.EdmScalarPropertyAttribute()]
]
    [global::System.Runtime.Serialization.DataMemberAttribute()]
    public string Institution
    {
        get
        {
            return this._Institution;
        }
        set
        {
            this.OnInstitutionChanging(value);
            this.ReportPropertyChanging("Institution");
            this._Institution =
global::System.Data.Objects.DataClasses.StructuralObject.SetValidValu
e(value, true);
            this.ReportPropertyChanging("Institution");
            this.OnInstitutionChanged();
        }
    }
    private string _Institution;
    partial void OnInstitutionChanging(string value);
    partial void OnInstitutionChanged();
    /// <summary>
    /// There are no comments for Property Active in the schema.
    /// </summary>

[global::System.Data.Objects.DataClasses.EdmScalarPropertyAttribute(I
sNullable=false)]
    [global::System.Runtime.Serialization.DataMemberAttribute()]
    public bool Active
    {
        get
        {
            return this._Active;
        }
        set
        {
            this.OnActiveChanging(value);
            this.ReportPropertyChanging("Active");
            this._Active =
global::System.Data.Objects.DataClasses.StructuralObject.SetValidValu
e(value);

```

```

        this.ReportPropertyChanged("Active");
        this.OnActiveChanged();
    }
}
private bool _Active;
partial void OnActiveChanging(bool value);
partial void OnActiveChanged();
/// <summary>
/// There are no comments for Query in the schema.
/// </summary>

[global::System.Data.Objects.DataClasses.EdmRelationshipNavigationPropertyAttribute("BuffieModel", "FK_Query_User", "Query")]
[global::System.Xml.Serialization.XmlIgnoreAttribute()]
[global::System.Xml.Serialization.SoapIgnoreAttribute()]
[global::System.Runtime.Serialization.DataMemberAttribute()]
public
global::System.Data.Objects.DataClasses.EntityCollection<Query> Query
{
    get
    {
        return
        ((global::System.Data.Objects.DataClasses.IEntityWithRelationships)(this)).RelationshipManager.GetRelatedCollection<Query>("BuffieModel.FK_Query_User", "Query");
    }
    set
    {
        if ((value != null))
        {
            ((global::System.Data.Objects.DataClasses.IEntityWithRelationships)(this)).RelationshipManager.InitializeRelatedCollection<Query>("BuffieModel.FK_Query_User", "Query", value);
        }
    }
}
}
}

```

Appendix D

XSLT Templates from Domain Knowledge Base

DetailSearchRequest.xslt (biocase)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" version="1.0" encoding="UTF-8"
indent="yes"/>
  <xsl:param name="accessurl" select="'default'"/>
  <xsl:param name="source" select="'buffiecas'"/>
  <xsl:param name="currenttime" select="'20051010T090000+0100'"/>
  <xsl:param name="resource" select="'default'"/>
  <xsl:param name="conceptname" select="'ScientificName'"/>
  <xsl:param name="conceptvalue" select="'default'"/>
  <xsl:param name="Parameter1" select="'default'"/>
  <xsl:template match="/">
    <request xmlns="http://digir.net/schema/protocol/2003/1.0"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:digir="http://digir.net/schema/protocol/2003/1.0"
xmlns:dwc="http://digir.net/schema/conceptual/darwin/2003/1.0"
xmlns:darwin="http://digir.net/schema/conceptual/darwin/2003/1.0"
xsi:schemaLocation="http://digir.net/schema/protocol/2003/1.0
http://digir.sourceforge.net/schema/protocol/2003/1.0/digir.xsd
http://digir.net/schema/conceptual/darwin/2003/1.0
http://digir.sourceforge.net/schema/conceptual/darwin/2003/1.0/darwin
2.xsd">
      <header>
        <version>1.0.0</version>
        <sendTime>
          <xsl:value-of select="$currenttime"/>
        </sendTime>
        <source>
          <xsl:value-of select="$source"/>
        </source>
        <destination>
          <xsl:attribute name="resource">
            <xsl:value-of select="$resource"/>
          </xsl:attribute>
          <xsl:value-of select="$accessurl"/>
        </destination>
        <type>search</type>
      </header>
      <search>
        <filter>
          <equals>
            <xsl:choose>
              <xsl:when test="$conceptname= 'Country'">
                <xsl:element name="darwin:Country">
                  <xsl:value-of select="$conceptvalue"/>
                </xsl:element>
              </xsl:when>
            </xsl:choose>
          </equals>
        </filter>
      </search>
    </request>
  </template>
</xsl:stylesheet>
```



```

        <xsl:when test="$conceptname= 'InstitutionCode'">
            <xsl:element name="darwin:InstitutionCode">
                <xsl:value-of select="$conceptvalue"/>
            </xsl:element>
        </xsl:when>
        <xsl:otherwise>
            <xsl:element name="darwin:ScientificName">
                <xsl:value-of select="$conceptvalue"/>
            </xsl:element>
        </xsl:otherwise>
    </xsl:choose>
</equals>
</filter>
<records limit="10" start="0">
    <structure
schemaLocation="http://digir.sourceforge.net/schema/conceptual/darwin
/full/2003/1.0/darwin2full.xsd"/>
    </records>
    <count>true</count>
</search>
</request>
</xsl:template>
</xsl:stylesheet>

```

Detail Search Request.xslt (DWCV2)

```

<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="xml" version="1.0" encoding="UTF-8"
indent="yes"/>
    <xsl:param name="accessurl" select="'default'"/>
    <xsl:param name="source" select="'buffiecas'"/>
    <xsl:param name="currenttime" select="'20091010T090000+0100'"/>
    <xsl:param name="resource" select="'default'"/>
    <xsl:param name="conceptname" select="'ScientificName'"/>
    <xsl:param name="conceptvalue" select="'default'"/>
    <xsl:param name="Parameter1" select="'default'"/>
    <xsl:template match="/">
        <request xmlns="http://digir.net/schema/protocol/2003/1.0"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:digir="http://digir.net/schema/protocol/2003/1.0"

xmlns:dwc="http://digir.net/schema/conceptual/darwin/2003/1.0"
xmlns:darwin="http://digir.net/schema/conceptual/darwin/2003/1.0"

xsi:schemaLocation="http://digir.net/schema/protocol/2003/1.0
http://digir.sourceforge.net/schema/protocol/2003/1.0/digir.xsd
http://digir.net/schema/conceptual/darwin/2003/1.0
http://digir.sourceforge.net/schema/conceptual/darwin/2003/1.0/darwin
2.xsd">
            <header>
                <version>1.0.0</version>
                <sendTime>
                    <xsl:value-of select="$currenttime"/>
                </sendTime>
                <source>
                    <xsl:value-of select="$source"/>
                </source>
            </header>
        </request>
    </template>
</xsl:stylesheet>

```

```

    <destination>
      <xsl:attribute name="resource">
        <xsl:value-of select="$resource"/>
      </xsl:attribute>
      <xsl:value-of select="$accessurl"/>
    </destination>
  </type>search</type>
</header>
<search>
  <filter>
    <equals>
      <xsl:choose>
        <xsl:when test="$conceptname= 'Country'">
          <xsl:element name="darwin:Country">
            <xsl:value-of select="$conceptvalue"/>
          </xsl:element>
        </xsl:when>
        <xsl:when test="$conceptname= 'InstitutionCode'">
          <xsl:element name="darwin:InstitutionCode">
            <xsl:value-of select="$conceptvalue"/>
          </xsl:element>
        </xsl:when>
        <xsl:otherwise>
          <xsl:element name="darwin:ScientificName">
            <xsl:value-of select="$conceptvalue"/>
          </xsl:element>
        </xsl:otherwise>
      </xsl:choose>
    </equals>
  </filter>
  <records limit="10" start="0">
    <structure
schemaLocation="http://digir.sourceforge.net/schema/conceptual/darwin
/full/2003/1.0/darwin2full.xsd"/>
  </records>
  <count>true</count>
</search>
</request>
</xsl:template>
</xsl:stylesheet>

```

Biocase to DarwinCoreV2 Format

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2005 sp2 (http://www.altova.com) by R
Sundar-->
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:darwin="http://digir.net/schema/conceptual/darwin/2003/1.0"
xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance">
  <xsl:output method="xml" version="1.0" encoding="UTF-8"
indent="yes"/>
  <xsl:variable
name="lcletters">abcdefghijklmnopqrstuvwxy</xsl:variable>
  <xsl:variable
name="ucletters">ABCDEFGHIJKLMNPOQRSTUVWXYZ</xsl:variable>
  <xsl:template match="/">
    <xsl:variable name="SrcInsCode"
select="response/content/DataSets/DataSet/OriginalSource/SourceInsti
tutionCode"> </xsl:variable>

```

```

<response>
  <!--template to call header information -->
  <xsl:apply-templates select="response/header"/>
  <content>
    <xsl:for-each
select="response/content/DataSets/DataSet/Units/Unit">
      <record>
        <darwin:GlobalUniqueIdentifier>
          <xsl:value-of select="UnitGUID"/>
        </darwin:GlobalUniqueIdentifier>
        <darwin:DateLastModified>
          <xsl:value-of
select="./Gathering/GatheringDateTime/ISODateTimeBegin"/>
        </darwin:DateLastModified>
        <darwin:BasisOfRecord>
          <xsl:attribute name="xsi:nil">
            <xsl:value-of select="'true'"/>
          </xsl:attribute>
          <xsl:value-of select="RecordBasis"/>
        </darwin:BasisOfRecord>
        <darwin:InstitutionCode>
          <xsl:value-of select="$SrcInsCode"/>
        </darwin:InstitutionCode>
        <darwin:CollectionCode>
          <xsl:value-of select="SourceID"/>
        </darwin:CollectionCode>
        <darwin:CatalogNumber>
          <xsl:value-of select="UnitID"/>
        </darwin:CatalogNumber>
        <!--template to call taxonomic information -->
        <xsl:apply-templates
select="Identifications/Identification"/>
        <darwin:IdentifiedBy>
          <xsl:value-of select="''"/>
        </darwin:IdentifiedBy>
        <!--template to call Collecting Event information -->
        <xsl:apply-templates select="Gathering"/>
        <!--template to call Locality information -->
        <xsl:apply-templates select="Gathering/GatheringSite"/>
        <!--template to call Geospatial information -->
        <darwin:Longitude>
          <xsl:value-of
select="Gathering/GatheringSite/SiteCoordinateSets/SiteCoordinates/Co
ordinatesLatLong/LongitudeDecimal"/>
        </darwin:Longitude>
        <darwin:Latitude>
          <xsl:value-of
select="Gathering/GatheringSite/SiteCoordinateSets/SiteCoordinates/Co
ordinatesLatLong/LatitudeDecimal"/>
        </darwin:Latitude>
        <darwin:CoordinatePrecision>
          <xsl:value-of
select="Gathering/GatheringSite/SiteCoordinateSets/SiteCoordinates/Co
ordinatesLatLong/LatitudeDecimal/CoordinateErrorDistanceInMeters"/>
        </darwin:CoordinatePrecision>
        <!--template to call Biological information -->
        <darwin:Sex>
          <xsl:choose>
            <xsl:when test="//ZoologySex !='' ">
              <xsl:value-of select="'//ZoologySex'"/>
            </xsl:when>
          </xsl:choose>
        </darwin:Sex>
      </record>
    </xsl:for-each>
  </content>
</response>

```

```

        <xsl:when test="//MycologicalSexualStage !='' ">
            <xsl:value-of select="//MycologicalSexualStage"/>
        </xsl:when>
        <xsl:otherwise>
            <xsl:attribute name="xsi:nil">
                <xsl:value-of select="'true'"/>
            </xsl:attribute>
        </xsl:otherwise>
    </xsl:choose>
</darwin:Sex>
<darwin:LifeStage>
    <xsl:choose>
        <xsl:when test="//ZoologyPhase !='' ">
            <xsl:value-of select="//ZoologyPhase"/>
        </xsl:when>
        <xsl:when test="//MycologicalLiveStage !='' ">
            <xsl:value-of select="//MycologicalLiveStage"/>
        </xsl:when>
        <xsl:otherwise>
            <xsl:attribute name="xsi:nil">
                <xsl:value-of select="'true'"/>
            </xsl:attribute>
        </xsl:otherwise>
    </xsl:choose>
</darwin:LifeStage>
<!--template to call reference information -->
<darwin:ImageURL>
    <xsl:value-of
select="UnitDigitalImages/UnitDigitalImage/ImageURI"/>
</darwin:ImageURL>
<darwin:PreparationType>
    <xsl:attribute name="xsi:nil">
        <xsl:value-of select="'true'"/>
    </xsl:attribute>
</darwin:PreparationType>
<darwin:IndividualCount>
    <xsl:attribute name="xsi:nil">
        <xsl:value-of select="'true'"/>
    </xsl:attribute>
</darwin:IndividualCount>
<darwin:PreviousCatalogNumber>
    <xsl:attribute name="xsi:nil">
        <xsl:value-of select="'true'"/>
    </xsl:attribute>
</darwin:PreviousCatalogNumber>
<darwin:RelationshipType>
    <xsl:attribute name="xsi:nil">
        <xsl:value-of select="'true'"/>
    </xsl:attribute>
</darwin:RelationshipType>
<darwin:RelatedCatalogItem>
    <xsl:attribute name="xsi:nil">
        <xsl:value-of select="'true'"/>
    </xsl:attribute>
</darwin:RelatedCatalogItem>
<darwin:Notes>
    <xsl:value-of select="UnitNotes"/>
</darwin:Notes>
</record>
</xsl:for-each>
</content>

```

```

    </response>
</xsl:template>
<!-- template match for header information -->
<!-- template match for header information -->
<!-- template match for header information -->
<xsl:template match="header">
  <header>
    <version>
      <xsl:value-of select="'DWCV2-to-ABCD-BioCASE-V1.0'"/>
    </version>
    <sendTime>
      <xsl:value-of select="sendTime"/>
    </sendTime>
    <source>
      <xsl:attribute name="resource">
        <xsl:value-of select="source"/>
      </xsl:attribute>
    </source>
    <destination>
      <xsl:value-of select="destination"/>
    </destination>
    <type>
      <xsl:value-of select="type"/>
    </type>
  </header>
</xsl:template>
<!-- template match for taxonomic information -->
<!-- template match for taxonomic information -->
<!-- template match for taxonomic information -->
<xsl:template match="Identifications/Identification">
  <darwin:ScientificName>
    <xsl:choose>
      <xsl:when test="//TaxonIdentified/NameAuthorYearString !=''
">
        <xsl:value-of
select="//TaxonIdentified/NameAuthorYearString"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:for-each
select="//TaxonIdentified/ScientificNameAtomized/child::node() ">
          <xsl:value-of select="."/>
          <xsl:text> </xsl:text>
        </xsl:for-each>
      </xsl:otherwise>
    </xsl:choose>
  </darwin:ScientificName>
  <xsl:call-template name="highertaxon">
  </xsl:call-template>
  <darwin:Genus>
    <xsl:choose>
      <xsl:when
test="//TaxonIdentified/ScientificNameAtomized/Bacterial/Genus !=''
">
        <xsl:value-of
select="//TaxonIdentified/ScientificNameAtomized/Bacterial/Genus"/>
      </xsl:when>
      <xsl:when
test="//TaxonIdentified/ScientificNameAtomized/Botanical/Genus !=''
">
        <xsl:value-of
select="//TaxonIdentified/ScientificNameAtomized/Botanical/Genus"/>

```

```

        </xsl:when>
        <xsl:when
test="//TaxonIdentified/ScientificNameAtomized/Viral/Genus !='' ">
            <xsl:value-of
select="//TaxonIdentified/ScientificNameAtomized/Viral/Genus"/>
        </xsl:when>
        <xsl:when
test="//TaxonIdentified/ScientificNameAtomized/Zoological/Genus !=''
">
            <xsl:value-of
select="//TaxonIdentified/ScientificNameAtomized/Zoological/Genus"/>
        </xsl:when>
        <xsl:otherwise>
            <xsl:value-of
select="//TaxonIdentified/NameAuthorYearString"/>
        </xsl:otherwise>
    </xsl:choose>
</darwin:Genus>
<darwin:Species>
    <xsl:choose>
        <xsl:when
test="//TaxonIdentified/ScientificNameAtomized/Bacterial/SpeciesEpith
et !='' ">
            <xsl:value-of
select="//TaxonIdentified/ScientificNameAtomized/Bacterial/SpeciesEpi
thet"/>
        </xsl:when>
        <xsl:when
test="//TaxonIdentified/ScientificNameAtomized/Botanical/FirstEpithet
!='' ">
            <xsl:value-of
select="//TaxonIdentified/ScientificNameAtomized/Botanical/FirstEpith
et"/>
        </xsl:when>
        <xsl:when
test="//TaxonIdentified/ScientificNameAtomized/Zoological/SpeciesEpi
thet !='' ">
            <xsl:value-of
select="//TaxonIdentified/ScientificNameAtomized/Zoological/SpeciesEpi
thet"/>
        </xsl:when>
        <xsl:otherwise>
            <xsl:value-of select="''"/>
        </xsl:otherwise>
    </xsl:choose>
</darwin:Species>
<darwin:Subspecies>
    <xsl:choose>
        <xsl:when
test="//TaxonIdentified/ScientificNameAtomized/Bacterial/SubspeciesEp
ithet !='' ">
            <xsl:value-of
select="//TaxonIdentified/ScientificNameAtomized/Bacterial/Subspecies
Epithet"/>
        </xsl:when>
        <xsl:when
test="//TaxonIdentified/ScientificNameAtomized/Botanical/SecondEpithe
t !='' ">
            <xsl:value-of
select="//TaxonIdentified/ScientificNameAtomized/Botanical/SecondEpit
het"/>

```

```

        </xsl:when>
        <xsl:when
test="//TaxonIdentified/ScientificNameAtomized/Zoological/SubspeciesE
pithet !='' ">
            <xsl:value-of
select="//TaxonIdentified/ScientificNameAtomized/Zoological/Subspecie
sEpithet"/>
        </xsl:when>
        <xsl:otherwise>
            <xsl:value-of select="''"/>
        </xsl:otherwise>
    </xsl:choose>
</darwin:Subspecies>
<darwin:ScientificNameAuthor>
    <xsl:choose>
        <xsl:when
test="//TaxonIdentified/ScientificNameAtomized/Botanical/AuthorTeam
!='' ">
            <xsl:value-of
select="//TaxonIdentified/ScientificNameAtomized/Botanical/AuthorTeam
"/>
        </xsl:when>
        <xsl:when
test="//TaxonIdentified/ScientificNameAtomized/Bacterial/AuthorTeam
!='' ">
            <xsl:value-of
select="//TaxonIdentified/ScientificNameAtomized/Bacterial/AuthorTeam
"/>
        </xsl:when>
        <xsl:when
test="//TaxonIdentified/ScientificNameAtomized/Zoological/AuthorTeam
!='' ">
            <xsl:value-of
select="//TaxonIdentified/ScientificNameAtomized/Zoological/AuthorTea
m"/>
        </xsl:when>
        <xsl:otherwise>
            <xsl:value-of select="''"/>
        </xsl:otherwise>
    </xsl:choose>
</darwin:ScientificNameAuthor>
</xsl:template>
<!-- template match for Gathering information -->
<!-- template match for Gathering information -->
<!-- template match for Gathering information -->
<xsl:template match="Gathering">
    <xsl:variable name="ISODate" select="normalize-
space(GatheringDateTime/ISODateTimeBegin)"/>
    <darwin:YearIdentified>
        <xsl:value-of select="substring-before($ISODate, '-' )"/>
    </darwin:YearIdentified>
    <darwin:MonthIdentified>
        <xsl:value-of select="        substring-before( substring-
after($ISODate, '-'), '-')"/>
    </darwin:MonthIdentified>
    <darwin:DayIdentified>
        <xsl:value-of select="substring-after( substring-
after($ISODate, '-'), '-')"/>
    </darwin:DayIdentified>
    <darwin:TypeStatus>
        <xsl:attribute name="xsi:nil">

```

```

        <xsl:value-of select="'true'"/>
    </xsl:attribute>
</darwin:TypeStatus>
<darwin:CollectorNumber>
    <xsl:attribute name="xsi:nil">
        <xsl:value-of select="'true'"/>
    </xsl:attribute>
</darwin:CollectorNumber>
<darwin:FieldNumber>
    <xsl:attribute name="xsi:nil">
        <xsl:value-of select="'true'"/>
    </xsl:attribute>
</darwin:FieldNumber>
<darwin:Collector>
    <xsl:value-of select="GatheringAgent/GatheringAgentsText"/>
</darwin:Collector>
<darwin:YearCollected>
    <!-- <xsl:value-of
select="GatheringDateTime/ISODateTimeBegin"/>-->
    <xsl:value-of select="substring-before($ISODate, '-' )"/>
</darwin:YearCollected>
<darwin:MonthCollected>
    <xsl:value-of select="        substring-before( substring-
after($ISODate, '-'), '-')"/>
</darwin:MonthCollected>
<darwin:DayCollected>
    <xsl:value-of select="substring-after( substring-
after($ISODate, '-'), '-')"/>
</darwin:DayCollected>
<darwin:JulianDay>
    <xsl:value-of select="//DayNumberBegin"/>
</darwin:JulianDay>
<darwin:TimeOfDay>
    <xsl:value-of select="//TimeOfDayBegin"/>
</darwin:TimeOfDay>
</xsl:template>
<xsl:template match="Gathering/GatheringSite">
    <xsl:call-template name="areaname"/>
    <darwin:Country>
        <xsl:choose>
            <xsl:when test="Country/CountryName !='' ">
                <xsl:value-of select="Country/CountryName"/>
            </xsl:when>
            <xsl:otherwise>
                <xsl:for-each select="Country/child::node() ">
                    <xsl:value-of select="."/>
                    <xsl:text> </xsl:text>
                </xsl:for-each>
            </xsl:otherwise>
        </xsl:choose>
    </darwin:Country>
    <darwin:Locality>
        <xsl:choose>
            <xsl:when test="LocalityText != ''">
                <xsl:value-of select="LocalityText"/>
            </xsl:when>
            <xsl:otherwise>
                <xsl:value-of select="AreaDetail"/>
            </xsl:otherwise>
        </xsl:choose>
    </darwin:Locality>

```



```

    <darwin:MinimumElevation>
      <xsl:value-of
select="Altitude/MeasurementAtomized/MeasurementLowerValue"/>
    </darwin:MinimumElevation>
    <darwin:MaximumElevation>
      <xsl:value-of
select="Altitude/MeasurementAtomized/MeasurementUpperValue"/>
    </darwin:MaximumElevation>
    <darwin:MinimumDepth>
      <xsl:value-of
select="Depth/MeasurementAtomized/MeasurementLowerValue"/>
    </darwin:MinimumDepth>
    <darwin:MaximumDepth>
      <xsl:value-of
select="Depth/MeasurementAtomized/MeasurementUpperValue"/>
    </darwin:MaximumDepth>
  </xsl:template>
  <!-- Named templates for Taxonomic details -->
  <xsl:template name="hightertaxon">
    <xsl:choose>
      <xsl:when test="*/HigherTaxa/HigherTaxon !='' ">
        <xsl:for-each select="*/HigherTaxa/HigherTaxon">
          <xsl:variable name="taxonrank" select="@TaxonRank"/>
          <xsl:choose>
            <xsl:when
test="translate($taxonrank,$ucletters,$lcletters)='kingdom'">
              <darwin:Kingdom>
                <xsl:value-of select="HigherTaxonName"/>
              </darwin:Kingdom>
            </xsl:when>
            <xsl:when
test="translate($taxonrank,$ucletters,$lcletters)='phylum'">
              <darwin:Phylum>
                <xsl:value-of select="HigherTaxonName"/>
              </darwin:Phylum>
            </xsl:when>
            <xsl:when
test="translate($taxonrank,$ucletters,$lcletters)='class'">
              <darwin:Class>
                <xsl:value-of select="HigherTaxonName"/>
              </darwin:Class>
            </xsl:when>
            <xsl:when
test="translate($taxonrank,$ucletters,$lcletters) = 'order'">
              <darwin:Order>
                <xsl:value-of select="HigherTaxonName"/>
              </darwin:Order>
            </xsl:when>
            <xsl:when
test="translate($taxonrank,$ucletters,$lcletters) = 'family'">
              <darwin:Family>
                <xsl:value-of select="HigherTaxonName"/>
              </darwin:Family>
            </xsl:when>
            <xsl:otherwise/>
          </xsl:choose>
        </xsl:for-each>
      </xsl:when>
      <xsl:otherwise>
        <darwin:Kingdom/>
        <darwin:Phylum/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>

```

```

        <darwin:Class/>
        <darwin:Order/>
        <darwin:Family/>
    </xsl:otherwise>
</xsl:choose>
</xsl:template>
<!-- Named templates for Location details -->
<xsl:template name="areaname">
    <xsl:choose>
        <xsl:when test="*/NamedAreas/NamedArea !=''">
            <xsl:for-each select="*/NamedAreas/NamedArea">
                <xsl:variable name="area" select="@NamedAreaClass"/>
                <xsl:choose>
                    <xsl:when test="translate($area,$ucletters,$lcletters) =
'continent'">
                        <darwin:ContinentOcean>
                            <xsl:value-of select="NamedAreaName"/>
                        </darwin:ContinentOcean>
                    </xsl:when>
                    <xsl:when test="translate($area,$ucletters,$lcletters) =
('state' or 'province')">
                        <darwin:StateProvince>
                            <xsl:value-of select="NamedAreaName"/>
                        </darwin:StateProvince>
                    </xsl:when>
                    <xsl:when test="translate($area,$ucletters,$lcletters) =
'county'">
                        <darwin:County>
                            <xsl:value-of select="NamedAreaName"/>
                        </darwin:County>
                    </xsl:when>
                    <xsl:otherwise/>
                </xsl:choose>
            </xsl:for-each>
        </xsl:when>
        <xsl:otherwise>
            <darwin:ContinentOcean/>
            <darwin:StateProvince/>
            <darwin:County/>
        </xsl:otherwise>
    </xsl:choose>
</xsl:template>
</xsl:stylesheet>

```