TOOLS FOR INDUSTRIAL KNOWLEDGE MODELING AND MANAGEMENT

A Thesis

Submitted to the Faculty of Graduate Studies and Research

In Partial Fulfillment of the Requirements

for the Degree of

Master of Applied Science

In Electronic Systems Engineering

University of Regina

By

Robert Harrison

Regina, Saskatchewan, Canada

July 2008

© 2008: R. Harrison

Library and
Archives Canada

Bibliothèque et
Archives Canada

Published Heritage
Branch

Direction du
Patrimoine de l'édition

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

ISBN: 978-0-494-42396-7

Our file Notre référence

ISBN: 978-0-494-42396-7

NOTICE:

The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protègent cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

UNIVERSITY OF REGINA
FACULTY OF GRADUATE STUDIES AND RESEARCH
SUPERVISORY AND EXAMINING COMMITTEE

Robert William Harrison, candidate for the degree of Master of Applied Science in Electronic Systems Engineering, has presented a thesis titled, ***Tools for Industrial Knowledge Modeling and Management***, in an oral examination held on May 5, 2008. The following committee members have found the thesis acceptable in form and content, and that the candidate demonstrated satisfactory knowledge of the subject material.

External Examiner: Dr. Samira Sadaoui-Mouhoub,
Department of Computer Science

Supervisor: Dr. Christine Chan, Software Systems Engineering

Committee Member: Dr. Luigi Benedicenti, Software Systems Engineering

Chair of Defense: Dr. Harold Weger, Department of Biology

ABSTRACT

This thesis presents the development and application of a suite of software tools for constructing knowledge to be used in knowledge based systems or the Semantic Web. The Inferential Modeling Technique, which is a technique for modeling the static and dynamic knowledge elements of a problem domain, provided the basis for the tools. A survey of existing knowledge modeling tools revealed weaknesses in the following four areas: 1. support for an ontological engineering methodology or technique, 2. support for dynamic knowledge modeling, 3. support for dynamic knowledge testing, and 4. support for ontology management. Two new tools were developed to address these four areas of weakness. Both tools were based on the Inferential Modeling Technique. The Class Editor was created to address the area of ontology management. The Class Editor enables users to model static knowledge using a UML class diagram like visualization and provides ontology management support through the back-end ontology management system Distributed Framework for Knowledge Evolution (DFKE). To address the areas of ontological engineering methodology, dynamic ontology modeling, and ontology testing, a Protégé plug-in called Dyna was created. Dyna helps users define task behaviour, a component of dynamic knowledge, in a formalized language, and create test cases, which enable verification of the knowledge model. Dyna also supports knowledge sharing and reuse because it can convert the dynamic knowledge models into the XML and OWL file formats, enabling the models to be shared and re-used. Both the Class Editor and Dyna are applied for constructing an ontological model in the domain of selecting a remediation technology for petroleum contaminated sites.

Acknowledgements

I would like to thank my supervisor, Dr. Christine Chan. I am very grateful for all her support, guidance, and the opportunities she has given me. I collaborated with Dan Obst and I thank him for his contribution to the work in ontology management. I would also like to thank Zhiying Hu for her help in clarifying knowledge in the ontology model of the domain of selection of a remediation technology for a petroleum contaminated site. I would like to thank Yuxiang Wu for his work on a Java event delivery technique. I would like to thank the members of the thesis defence committee. I would also like to thank my father, David Harrison, for proof reading my thesis. Additionally, I am grateful for the generous support of Research Grants from the Canada Research Chair Program and Natural Sciences and Engineering Research Council (NSERC).

Post Defence Acknowledgements

I would like to thank Dr. Benedicenti and the external examiner, Dr. Sadaoui-Mouhoub for all their questions and comments. I would also like to thank the chair of the defence, Dr. Weger.

Dedication

I would like to thank my parents, David and Inna Harrison. I could not have completed this work without their support and encouragement to do my best. I dedicate this thesis to them.

Table of Contents

Abstract	ii
Acknowledgements	iii
Post Defense Acknowledgements	iv
Dedication	v
List of Tables	x
List of Figures	xi
List of Abbreviations	xii
List of Appendices	xiii
1. Introduction	1
1.1 Overview	1
1.2 Contribution of Thesis	2
1.3 Organization of Thesis	3
2. Background Literature	5
2.1 Knowledge Based Systems, the Semantic Web, and Ontologies	5
2.2 Ontology Components & Characteristics	8
2.3 Inferential Modeling Technique	16
2.4 Knowledge Modeling System	17
2.5 Ontology Construction Support Tools	17
2.5.1 Ontological Engineering Methodology	19
2.5.2 Ontology Modeling	21
2.5.3 Ontology Testing	32
2.5.4 Ontology Management	34

2.6 Application Problem Domain	40
2.6.1 Overview of Petroleum Contamination Remediation Selection	40
2.6.2 Details of Petroleum Contamination Remediation Selection ...	42
2.7 Conclusions	43
3. Design and Implementation of a Tool for Knowledge Management	44
3.1 Overview	44
3.2 Knowledge Representation	47
3.3 Static Knowledge Modeling	52
3.4 Ontology Management	54
3.5 Conclusions	60
4. Design and Implementation of a Tool of Dynamic Knowledge Modeling	61
4.1 Overview	61
4.2 Knowledge Representation	63
4.3 Dynamic Knowledge Modeling	64
4.3.1 Objective Modeling	64
4.3.2 Task Modeling	65
4.4 Testing Dynamic Knowledge	69
4.5 Knowledge Storage and Interoperability	72
4.5.1 XML	72
4.5.2 OWL	74
4.6 Conclusions	75
5. Application of the Tools	77
5.1 Overview	77

5.2 Application of Class Editor	78
5.3 Application of Dyna	82
5.4 Conclusions	91
6. Discussion	92
7. Conclusions	99
8. Future Work	102
8.1 Knowledge Evolution in Dyna	102
8.2 Semi-Automatic Ontology Generation	102
8.3 Support Arrays in TBL	103
8.4 Visualizing Flow or Sequence of Tasks and Objectives	104
8.5 Code Coverage Metrics	105
8.6 Top-Level Ontology	106
8.7 Evaluation of Dyna	106
8.8 Miscellaneous Ideas	106
8.8.1 Uncertainty Modeling	107
8.8.2 Semi-Automatic Application Generation	109
8.8.3 Time and Parallel Processes	110
8.9 Conclusions	110
References	112
Appendices	117
Appendix I: Petroleum Contamination Remediation Selection Ontology: Static Knowledge created with Protégé	117
Appendix II : Petroleum Contamination Remediation Selection Ontology: Dynamic Knowledge created with Dyna	122

Appendix III : Petroleum Contamination Remediation Selection Ontology: Static Knowledge Created with Protégé (OWL File Format)	150
Appendix IV : Petroleum Contamination Remediation Selection Ontology: Dynamic Knowledge created with Dyna (XML File Format)	152
Appendix V : Petroleum Contamination Remediation Selection Ontology: Dynamic Knowledge created with Dyna (OWL File Format)	155
Appendix VI : Dynamic Knowledge Elements (OWL File Format)	159

List of Tables

1	Summary of Objectives for Class Editor and Dyna	3
2	Paths for task behaviour “SetSiteSize”	87
3	Select Remediation Method Objective	90

List of Figures

1	Person represented as a semantic network	11
2	Ontology Modeler	19
3	Creating a class for representing Task	25
4	Create a class for representing TaskPriority	26
5	Create a class for representing Objective	26
6	Create task for Calculate Weighted Normalized Benzene Concentration	28
7	Task Priority for Calculate Weighted Normalized Benzene Concentration	29
8	Determine Contamination Level Objective	30
9	Determine Contamination Level Objective with multiple task priorities	31
10	Overview of contaminant problem from petroleum production activities (Chen, 2001)	41
11	High-level architecture of Class Editor	46
12	Classes in the Common Package	48
13	Classes in the Knowledge Package	49
14	Monotonic Properties Package	50
15	Classes in the Ontology package	51
16	Server View of Ontology Projects	56
17	Unified Ontology View	56
18	P2P Architecture	57
19	High-level Architecture of Dyna	62
20	Dyna Knowledge Representation	64
21	Architecture of TBL Interpreter	68
22	Test Suite for BillPayment Task	72
23	Result of Running Test Case testBillPayment	72
24	Class Editor: acquiring user information	78
25	Class Editor	79
26	Editing documentation for the Media class in Class Editor	80
27	Access information for the Media class in Class Editor	81
28	Project Browser	81
29	Class Hierarchy in Protege	84
30	Dyna – Task Behaviour	86
31	Dyna – Testing Task Behaviour	88
32	Dyna – DetermineSiteSize Objective	89
33	Objective : DetermineSiteHydraulicConductivity	97
34	Task/Objective Flow Diagram	105

ABBREVIATIONS

BN	Bayesian Network
CORBA	Common Object Request Broker Architecture
DAML+OIL	DARPA Agent Markup Language & Ontology Inference Layer
DFKE	Distributed Framework for Knowledge Evolution
DL	Descriptive Logic
DSS	Decision Support System
EAI	Enterprise Application Integration
ES	Expert System
FOAF	Friend-of-a-Friend
GUI	Graphical User Interface
HTML	Hyper Text Markup Language
IMT	Inferential Modeling Technique
KB	Knowledge Base
KBS	Knowledge Based System
KMS	Knowledge Modeling System
MDA	Model Driven Architecture
MVC	Model View Controller
ODM	Ontology Definition Metamodel
OWL	Web Ontology Language
P2P	Peer-to-Peer
RDF	Resource Description Format
RDFS	Resource Description Format Schema
SW	Semantic Web
SWRL	Semantic Web Rule Language
TBL	Task Behaviour Language
TDD	Test Driven Development
UML	Unified Modeling Language
UOV	Unified Ontology View
URI	Uniform Resource Identifier
Web	World Wide Web
WWW	World Wide Web
XML	Extensible Markup Language

List of Appendices

I	Petroleum Contamination Remediation Selection Ontology : Static Knowledge created with Protégé	117
II	Petroleum Contamination Remediation Selection Ontology : Dynamic Knowledge created with Dyna	122
III	Petroleum Contamination Remediation Selection Ontology: Static Knowledge Created with Protégé (OWL File Format)	150
IV	Petroleum Contamination Remediation Selection Ontology: Dynamic Knowledge created with Dyna (XML File Format)	152
V	Petroleum Contamination Remediation Selection Ontology: Dynamic Knowledge created with Dyna (OWL File Format)	155
VI	Dynamic Knowledge Elements (OWL File Format)	159

1. INTRODUCTION

1.1 Overview

Knowledge Base Systems (KBS) are often limited to problem solving in a narrow domain of knowledge. The high development costs involved mean that the development efforts must be focused on a specific area of knowledge. The major cost is construction of the knowledge base. Knowledge bases are often constructed from scratch because KBS development occurs in a distributed and heterogeneous environment (different locations, system types, knowledge representations, etc.) making it very difficult to share and re-use existing knowledge base components.

Sharing and re-using knowledge can help reduce costs and make it possible to create systems capable of more powerful problem solving. The Semantic Web (SW), which is widely recognized as the next evolutionary step for the World Wide Web, can be used for sharing and re-use of knowledge between KBS's in a distributed and heterogeneous environment. The SW provides semantics to data on the Web, enabling computers to more easily share and perform problem solving on the data (Berners-Lee et al., 2001).

A requirement for the SW is that data on the Web are structured semantically for machine processing, and ontologies can facilitate this process of knowledge structuring. Ontologies can also become the basis for building knowledge-based systems. An ontology is an "explicit specification of a shared conceptualization" (Gruber, 1993); it can be used for structuring the knowledge in a KBS and on the SW. The main benefit of an ontology is that it enables the sharing and re-use of application domain knowledge across distributed and heterogeneous software systems (Guarino, 1998). Ontologies

implemented in XML¹ based languages, such as RDF² and OWL³, enable different KBS development groups or different Semantic Web applications to share and re-use their knowledge and data. However, the construction of ontologies is difficult and time-consuming.

1.2 Contribution of Thesis

Software tools can help reduce the effort required to construct ontologies and knowledge bases. The general objective of this work is to provide software tool support for knowledge creation for the Semantic Web. The approach employed involved examining existing software tools for knowledge creation, and this examination provided the basis for design of two new tools, named Dyna and Class Editor, for supporting knowledge creation. The new tools aim to address deficiencies noted in some existing tools. The deficiencies in some existing tools are in the areas of knowledge management and dynamic knowledge modeling. Therefore, the focus of the work was to develop a tool to support knowledge management and another tool to support dynamic knowledge modeling. The specific objectives of the tools are listed as follows and summarized in Table 1.

1. Support for a knowledge modeling technique (we adopted for this purpose, the Inferential Modeling Technique (IMT).
2. Support for ontology management.
3. Support for testing of the developed knowledge model.

¹ T. Bray et al., "Extensible Markup Language (XML) 1.0", 4th ed., Sep 29, 2006, <http://www.w3.org/TR/2006/REC-xml-20060816/>

² Resource Description Framework (RDF), <http://www.w3.org/RDF/>

³ Web Ontology Language (OWL), <http://www.w3.org/TR/owl-features/>

4. Support for diverse types of knowledge required to model an ontology of an industrial domain, in this case, the Petroleum Contamination Remediation Selection domain.

Table 1. Summary of Objectives for Class Editor and Dyna

Tool	Technique	Static Knowledge Modeling	Dynamic Knowledge Modeling	Dynamic Knowledge Testing	Knowledge Management	Language
Class Editor	IMT	Yes			Yes	Python
Dyna	IMT		Yes	Yes		Java

Both of the Class Editor and Dyna were applied for building an application ontology in the petroleum contamination remediation selection domain to demonstrate their respective capabilities in knowledge management and dynamic knowledge modeling. Through implementation of the application ontology, it was found that the most significant contribution of this thesis is in the area of dynamic knowledge testing.

1.3 Organization of Thesis

This thesis is organized as follows.

Chapter 2 presents relevant background literature; knowledge based systems, ontologies, and the Semantic Web and their association with each other are described. The components of an ontology are presented. Next, some weaknesses in existing ontology tools are discussed which provide a basis for describing the specific research objectives of this work. The context in which this work takes place is the domain of selecting a technology for cleaning a petroleum contaminated site, which is described.

Chapter 3 describes the development of a tool for knowledge management. The knowledge representation format is presented, followed by the knowledge modeling and management support features.

Chapter 4 describes the development of a tool for dynamic knowledge modeling. The knowledge representation format is also presented, followed by the dynamic knowledge modeling features.

Chapter 5 presents the application of the tools to model the problem domain. As one of the tools was designed for managing knowledge, the application of the tool is more focused on its managing capabilities, rather than its modeling capabilities. The application of the dynamic knowledge modeling tool demonstrates its dynamic knowledge modeling capabilities.

Chapter 6 discusses the advantages, disadvantages noted, and interesting observations made during application of the tools to the modeling of the petroleum contamination remediation selection domain.

Chapter 7 provides some conclusions.

Chapter 8 discusses many directions for future work.

2. BACKGROUND LITERATURE

This chapter discusses concepts and issues relevant to the design and application of the ontology construction tools. An overview of knowledge based systems, the Semantic Web, and ontologies is provided in sections 2.1 and 2.2.

The tools, called the Class Editor and Dyna, were developed based on the Inferential Modeling Technique (IMT), which proposed a template of knowledge types for organizing domain knowledge acquired for knowledge-based system construction. The IMT provided the initial foundation for building the Knowledge Modeling System (KMS), which was the first prototype built at Energy Informatics Laboratory, for automation of the ontology construction process. The Class Editor and Dyna presented here were built to improve on areas of weakness in KMS. The IMT and KMS are discussed in sections 2.3 and 2.4 respectively.

Some background on the research area of ontology construction tools and methodologies is presented in section 2.5. This includes a discussion on some weaknesses in the existing tools and the approach adopted to address those weaknesses.

Since the primary application area for the ontology construction tools is energy and environmental system analysis, applications of the Class Editor and Dyna are illustrated with the sample domain of selection of remedial technology for a petroleum contaminated site, which is presented briefly in section 2.6.

2.1 Knowledge Based Systems, the Semantic Web, and Ontologies

A Knowledge Based System (KBS) is a system that performs problem solving by querying a database of knowledge for a particular domain using artificial intelligence techniques. A Decision Support System (DSS) is a type of KBS used for assisting people

in making complex decisions. For example, ATHENA is a DSS for assisting health care workers assess and treat hypertension (high blood pressure). The user inputs various characteristics of the patient into ATHENA, and then ATHENA outputs a recommendation for a drug that should be prescribed to the patient (Chan et al., 2005).

There are difficulties in constructing the knowledge base component of a KBS, which makes KBS development very expensive. The process of acquiring the knowledge for building the knowledge base is known to be a major bottleneck. Knowledge bases are also usually constructed from scratch at considerable expense. KBS development occurs in a distributed and heterogeneous environment making it very difficult to re-use existing knowledge base components. Thus work to create a KBS is often duplicated. Due to the high costs involved, KBS's are usually limited to problem solving in a very narrow range of knowledge. Sharing and re-using knowledge can reduce costs and make it possible to create systems with more powerful problem solving capabilities.

The World Wide Web (Web) is a vast, distributed knowledge base enabling billions of humans to share and use knowledge for solving problems. The emergence of blogs and social web sites, such as Flickr⁴ (picture sharing) and YouTube⁵ (video sharing), have made it very easy for people to share ideas, pictures, and videos. However, it is very difficult to create a computer program that can share and perform any reasoning or problem solving on this information as a human would. Web pages are implemented with syntactic display technologies, such as HTML⁶ and Flash (Trademark of Adobe), which lack the structure necessary for computers to perform problem solving.

⁴ <http://www.flickr.com>

⁵ <http://www.youtube.com>

⁶ Hyper Text Markup Language (HTML), <http://www.w3.org/TR/html401/>

The Semantic Web (SW) has been proposed as the next evolutionary step for the Web. The SW aims to provide semantics to data on the Web, enabling computers to more easily share and perform problem solving on the data (Berners-Lee et al., 2001). For example, a picture on a web page can be described with meta-data⁷ in a standardized language, making it easily processable by a computer program. The computer program could then perform some reasoning or decision making on the image. The computer program could even interact with other computer programs and share the information about the image so that some shared decision can be made. There are a number of unresolved problems with the SW; nonetheless, some of its technologies and ideas can be applied to KBS development.

Data in the SW is represented by an ontology. As described in the Introduction, an ontology is a “specification of a shared conceptualization of a domain” (Gruber, 1993). An ontology contains concepts, concept taxonomies, relationships between concepts, and properties that describe concepts. An ontology can also contain axioms and constraints that add further meaning to the terms in the ontology (Gomez-Perez, et al., 2005).

Ontologies can also be used to represent knowledge in a KBS. Ontologies help facilitate the sharing and re-use of knowledge across distributed and heterogeneous systems. The Ontology-based Holonic Diagnostic System (OHDS) is a KBS for diagnosing unknown diseases and is an example of the combination of KBS, ontology, and the Semantic Web (Ulieru et al., 2006). OHDS consists of a number of knowledge based systems around the world. Users can use a web-based user interface to input

⁷ Meta-data is information that describes data

symptoms that a patient is showing, and the system will query across the many different systems, and then output a possible illness and treatment.

2.2 Ontology Components & Characteristics

An ontology contains many of the features found in object-oriented design and programming, but also contains some additional features, such as axioms and restrictions, to support the description of logic and knowledge reuse. Ontologies for the Semantic Web are represented in eXtensible Markup Language (XML) based languages. XML is a text-based file format for describing the structure of electronic data for transfer on the Internet. The following XML languages are used to represent ontologies in the SW:

1. XML
2. XML Schema
3. RDF(S)
4. OWL

XML is the base language and can be used to represent only the most basic ontology elements. Each successive language listed builds upon its predecessor, enabling the representation of more complex ontology elements. The purpose of this section is to present the components and characteristics of an ontology within the context of the XML based ontology representation languages. It is important to be aware of this context because ontology tools must support the standard XML based ontology representation languages for their models to be shareable and reusable by other systems. The basic components and characteristics are presented first, followed by each successively more complex ontology component and characteristic.

The most basic component of an ontology is the “concept”, also known as a “class”. Using XML, the concept of a “Person” can be described as shown below:

```
<Person>
  <first_name>Darth</first_name>
  <last_name>Vader</last_name>
  <age>40</age>
</Person>
```

Another component of an ontology is the “property”. Concepts can have properties or attributes that provide more details of the concept. In the case above, the Person concept has the properties “first_name”, “last_name”, and “age”. The XML element <first_name> denotes that person's first name is “Darth”.

A very important characteristic of an ontology is semantics. A limitation of basic XML is that it does not provide a method of specifying the semantics (or meaning) of the elements (Antoniou & van Harmelen, 2004). For example, it is not possible to specify that the value of <first_name> is a string or that the value of <age> is an integer. It is also not possible to specify that a <Person> element represents a concept and a <first_name> element represents a property. The ability to specify semantics of elements is necessary for describing an ontology because without them, the ontology is open for mis-interpretation. Using XML Schema, some basic semantics can be specified. XML Schema is a language for specifying restrictions on the structure of XML elements (Antoniou & van Harmelen, 2004). Using the person example, XML Schema can be used to specify that <Person> consists of the sequence of subelements: <first_name>, <last_name>, and <age>. Additionally, XML Schema can specify that the values <first_name> and <last_name> are strings and the value of <age> is an integer. This is shown as follows:

```

<xsd:complexType name="Person">
  <xsd:sequence>
    <xsd:element name="first_name" type="xsd:string"/>
    <xsd:element name="last_name" type="xsd:string"/>
    <xsd:element name="age" type="xsd:int"/>
  </xsd:sequence>
</xsd:complexType>

```

A defining characteristic that separates an ontology model from other models is that an ontology model is designed for sharing and reusability. XML provides a mechanism, called a “namespace”⁸, for reusing existing data that is defined in external files. A namespace enables an XML document to re-use data that is defined in an external XML document, also known as a “resource”. The location of a resource is specified by a Uniform Resource Identifier (URI). A type of URI is a web address (e.g. <http://www.uregina.ca>). In the XML Schema example above, for example, “complexType” is defined in the namespace “xsd”. At the top of the file that contains the XML Schema example above, there would be: *xmlns:xsd=*["http://www.w3.org/2001/XMLSchema"](http://www.w3.org/2001/XMLSchema). This line defines that any XML element that has the prefix “xsd”, is defined in the file located at the specified URI. Similarly, Person could be reused in another file, by performing the following actions:

1. Publish the Person XML file on the Web, thus making Person a resource
2. In a second XML file, create a namespace that refers to the Person resource:

```
xmlns:person=http://example.com/Person.xml
```

3. Use the person resource in the second XML file.

Other ontology components include the relationship between two concepts (e.g. inheritance), instance of a concept, and further restrictions on properties and values. XML and XML Schema are not able to support these components on their own.

⁸ XML Namespace, <http://www.w3.org/TR/REC-xml-names/>

Semantic networks can provide a much richer description of knowledge than basic XML and XML Schema and are able to support these additional components. A semantic network is a directed graph with labelled nodes and edges. The nodes represent concepts, instances of concepts, and values of properties, while the edges represent relations between concepts or properties of concepts (Gomez-Perez et al., 2005). Figure 1 shows the Person represented as a semantic network. The node labelled "Person" represents the concept of a *Person*. The node labelled "DarthVader" represents the instance of the *Person* concept, *DarthVader*. The relationship between the *Person* concept and the *DarthVader* instance is shown by the directed edge, *type*, which points from the *DarthVader* node to the *Person* node. The *DarthVader* instance has the property *age*, denoted by the *age* edge. The *age* edge points from the *DarthVader* node to the 40 node, which means that Darth Vader's age has the value of 40.

Figure 1. Person represented as a semantic network

A language for representing a semantic network is Resource Description Framework⁹ (RDF). RDF uses both XML and XML Schema, but adds additional structures to specify semantics. The nodes in an RDF graph represent resources. A resource is any data that can be identified with a Uniform Resource Identifier (URI), such as a web page or a person. The following is an example of a “Darth Vader” person resource identified by a URI: <http://example.com/person#DarthVader> .

RDF consists of two additional types of objects: properties and statements. Properties define the attributes and relations of a resource. Statements are used to specify the value of a property for a particular resource. RDF does not have the ability to define relationships between properties and resources. For example, it is not possible to restrict the *first_name* property to the *person* concept. In RDF, one could define a *rock* concept and give it a *first_name*, which does not really make sense. To define relationships between properties and resources, RDF Vocabulary Description Language¹⁰ (also known as RDF Schema or RDFS), is used. RDF and RDFS are usually referred to collectively as RDF(S). The following is an example of the *Person* class and the *first_name* property defined in RDF(S):

```
<rdfs:Class rdf:about="Person" rdfs:label="Person">
  <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdf:Property rdf:about="first_name" rdfs:label="first_name">
  <rdfs:domain rdf:resource="Person"/>
  <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
```

The Person class can then be instantiated in RDF as follows:

```
<Person rdf:about="DarthVader" first_name="Darth"/>
```

⁹ RDF, <http://www.w3.org/RDF/>

¹⁰ RDFS, <http://www.w3.org/TR/rdf-schema/>

Ontologies can also contain various other types of information to provide more semantics to the data, including cardinality and logic. A limitation of semantic networks, such as RDF(S), is that they do not have the ability to represent cardinality and descriptive logic. Cardinality is important when you want to specify the number of something. Descriptive logic (DL) consists of concepts (or classes), roles (relations between concepts, such as properties), and individuals (or instances). Descriptive logic uses first-order logic¹¹, therefore it includes relations between concepts such as intersection, union, and negation. DL also enables restrictions of roles (properties) including inverse and transitivity. DL can support a much richer knowledge representation than RDF(S) or XML.

Web Ontology Language¹² (OWL) is an eXtensible Markup Language (XML) based language and is the most recently developed language for describing ontologies on the Semantic Web. The Web Ontology Language (OWL) extends RDF(S) and is based on another ontology language, DAML+OIL¹³. It is capable of representing cardinality and descriptive logic. There are three increasingly expressive versions of OWL: 1. OWL Lite, 2. OWL DL, and 3. OWL Full. Each version allows a richer description of an ontology. An example of the Person class in OWL DL is as follows:

```
<owl:Class rdf:ID="Person"/>
<owl:FunctionalProperty rdf:ID="first_name">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:domain rdf:resource="#Person"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:FunctionalProperty>
<Person rdf:ID="DarthVader">
  <first_name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    Darth
  </first_name>
```

¹¹ First-order logic, http://en.wikipedia.org/wiki/First-order_logic

¹² Web Ontology Language, <http://www.w3.org/TR/owl-features/>

¹³ DAML+OIL, <http://www.w3.org/TR/daml+oil-reference>

`</Person>`

To show an example of representing cardinality in OWL, a “childOf” property is added to the class “Person”. Since a person has exactly two parents, the “childOf” property is restricted to cardinality equal to two. This is shown as follows.

```
<owl:Class rdf:ID="Person">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >2</owl:cardinality>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="childOf"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
```

An example of descriptive logic is the inverse of the “childOf” property, which is “parentOf”. This can be represented in OWL as follows:

```
<owl:ObjectProperty rdf:about="#childOf">
  <owl:inverseOf>
    <owl:ObjectProperty rdf:ID="parentOf"/>
  </owl:inverseOf>
```

Formal methods are techniques that use mathematics to create a specification of software that can be proved correct (Pressman, 2004). The purpose of formal methods is to improve software quality (Pressman, 2004). It may not be possible to prove that an ontology is correct, since an ontology is a shared view or perspective of a domain that others may disagree with. However, an ontology can be proven to be consistent. An ontology is consistent if it does not contain any circularity errors (e.g. a class is a subclass of itself), partition errors (e.g. two disjoint classes have common instances), or semantic errors (e.g. a class contains an incorrect property) (Gomez-Perez et al., 2005). Verifying or proving that an ontology is consistent can improve the quality of the ontology. Ontology features such as restrictions (e.g. cardinality) and logic are used by

ontology tools, such as Protégé (Gennari et al., 2003), to verify consistency. The “parentOf” property defined above will be used as an example. The “parentOf” property has the restriction of cardinality equals two. Therefore if an instance of Person contains only one value for the “parentOf” property, Protégé will show that the ontology is inconsistent by highlighting the “parentOf” input field in red. Verifying ontology consistency is related to ontology testing, which is discussed in Section 2.5.3.

To summarize, an ontology contains the following features:

- Semantics
- Share-able, re-usable
- Concept (or class)
- Property
- Relation (e.g. inheritance)
- Instances (or individuals)
- Restrictions on properties and values
- Logic

Many of the discussed features of an ontology were used in the application ontology, which is first presented in Section 2.6 and later implemented in Chapter 5. The features found in object-oriented design and programming that were used to implement the application ontology include classes, properties, instances, inheritance relationships, aggregation relationships, and composition relationships (see Section 5.2 and Section 5.3). OWL’s functional restriction was used (see Section 5.3.1). Logic structures, such as if-statements and conditional expressions, were also used (see Section 5.3.2).

2.3 Inferential Modeling Technique

The Inferential Modeling Technique (IMT) is a technique for modeling the static and dynamic knowledge elements of a problem domain. Static knowledge consists of observable domain objects (classes), properties of classes, and relationships between classes. Dynamic knowledge consists of tasks (or processes) that manipulate the static knowledge to achieve an objective. The IMT is an iterative process of knowledge modeling, and the procedure is listed below. For further details on the technique, see (Chan, 2004).

1. Specify static knowledge:
 - 1.1 Specify the physical objects in the domain
 - 1.2 Specify the properties of objects
 - 1.3 Specify the values of the properties or define the properties as functions or equations
 - 1.4 Specify the relations associated with objects and properties as functions or equations
 - 1.5 Specify the partial order of the relations in terms of strength factors and criteria associated with the relations
 - 1.6 Specify the inference relations derived from the objects and properties
 - 1.7 Specify the partial order of the inference relations in terms of strength factors and criteria associated with the relations
2. Specify the dynamic knowledge:
 - 2.1 Specify the tasks
 - 2.2 Decompose the tasks identified into inference structures or subtasks

- 2.3 Specify the partial order of the inference and subtask structures in terms of strength factors and criteria
 - 2.4 Specify strategic knowledge in the domain
 - 2.5 Specify how strategic knowledge identified is related to task and inference structures specified
3. Return to Step 1 until the specification of knowledge types is satisfactory to both the expert and knowledge engineer.

2.4 Knowledge Modeling System

Knowledge Modeling System (KMS) is a software tool for modeling static and dynamic knowledge as defined in the IMT. KMS contains a “class” module for modeling static knowledge and a “task” module for modeling dynamic knowledge. The “class” module enables the user to model concepts or classes and properties of classes. The “class” module also supports the creation of inheritance and association relationships between classes. The “task” module enables the user to create tasks and objectives. A type of strategic knowledge can be specified by adding a prioritized list of tasks to an objective. The “task” module also supports the user in linking static knowledge created in the “class” module to tasks. Task behaviour or operations that manipulate objects can also be defined. KMS provided a basis for the work presented in this thesis.

2.5 Ontology Construction Support Tools

Ontology languages, such as those based on XML, are very difficult for people to read and write. Software tools enable ontology authors to ignore the complexities of the ontology languages used in developing application ontologies. The tools support efficient application development, so that the development process can be completed with

fewer errors and involves a lower learning curve. A few common features found among ontology tools are a graphical user interface (GUI), a visualization of the ontology, and the ability to import/export the ontology from/to an ontology representation language (e.g. OWL). (Gomez-Perez et al., 2005) found six specific categories of features found in ontology tools:

1. Evaluation – evaluate the content of an ontology
2. Merge and Alignment – used to resolve problems when combining ontologies
3. Annotation – used to markup web pages with ontology information
4. Querying and Inference – perform reasoning on an ontology
5. Learning – generate an ontology automatically or semi-automatically from a knowledge source (e.g. database or text file)
6. Editing – ontology construction

A seventh, additional category of feature found in ontology tools is ontology management. Ontology management features help support the user in sharing and re-using knowledge.

The two categories this thesis focuses on are ontology editing and ontology management. A prototype ontology editor, Ontology Modeler, shown in Figure 2, was developed. Ontology Modeler was implemented in C# using Microsoft Visual Studio .Net 2002 (Trademark of Microsoft). Ontology Modeler provides some of the basic features commonly found in other ontology editors including a GUI, a UML class-diagram like visualization of the knowledge model, and can save the knowledge model to XML. Ontology Modeler can also generate an ontology from a database, thus speeding

up development time. From a survey of research work on ontology tools (Harrison & Chan, 2005b) we found a number of areas that require improvement. These areas include ontology engineering, dynamic knowledge modeling, ontology testing, and ontology management, which are discussed in the following sub-sections.

Figure 2. Ontology Modeler

2.5.1 Ontological Engineering Methodology

The process of developing an ontology consists of activities in three different areas. First, there are ontology development activities such as specification, implementation, and maintenance. Second, there are ontology management activities such as reusing existing ontologies and quality control. And third, there are ontology support activities such as knowledge acquisition and documentation (Gomez-Perez et al.,

2005). An ontological engineering methodology specifies the relationships among the activities and when the activities should be performed.

A tool that supports an ontological engineering methodology or technique can expedite the ontological engineering process. Currently, there are few tools that directly support an ontological engineering methodology; some existing tools include: OntoEdit which supports On-To-Knowledge (Fensel et al., 2002), WebODE which supports METHONTOLOGY (Gomez-Perez et al., 2003), and the Knowledge Modeling System (KMS) which supports IMT (Chan, 2004).

On-To-Knowledge is an iterative methodology that consists of five steps: 1. Feasibility Study, 2. Kickoff, 3. Refinement, 4. Evaluation, and 5. Maintenance (Gomez-Perez et al., 2005). OntoEdit provides support for ontology implementation, which happens in the third step of refinement, and maintenance (Fensel et al., 2002).

METHONTOLOGY is based on the software development process and provides support for the entire process during the development life cycle of an ontology. WebODE's ontology editor consists of a number of services that enable it to support various activities in the ontology life cycle defined in METHONTOLOGY. WebODE's services include ontology implementation, documentation, reasoning, and evaluation (Gomez-Perez et al., 2005).

As discussed in Section 2.3, the IMT is a technique for developing a classification of the knowledge elements of a domain. KMS provides automated support for developing the static and dynamic knowledge elements of a domain, and the tool was developed based on the IMT (Chan, 2004).

2.5.2 Ontology Modeling

An ontological engineering tool enables developing a knowledge or ontological model of a problem domain. A brief survey of the different ontological engineering support tools reveal there are diverse methods employed. One method of editing an ontology is to use a text-based method. (Kalyanpur et al., 2004) proposed a shorthand notation for typing OWL. A quick and simple text based interface provides significant benefits to expert users (Kalyanpur et al., 2004). However, most people are not expert users and a text based interface would slow them down.

The method used by Protégé (Gennari et al., 2003), Ontolingua (McGuinness et al., 2000), and OntoBuilder (Gal et al., 2006) for modeling ontologies involves using input fields to capture characteristics of concepts and listing all the concepts in a hierarchical tree. Hierarchical tree listings of concepts are not sufficient for displaying concepts with multiple inheritance because they can only show single inheritance. Protégé compensates for this limitation of hierarchical tree, by supporting multiple inheritance in a Superclasses pane. However, with this feature, managing the class hierarchy becomes cumbersome because a class in the Class Hierarchy must be selected first and then the user needs to remember to look at the Superclasses window to see all the parents of the class. It can be concluded that using a tree to model an ontology with multiple inheritance hinders the ontology development process.

A more effective method of visualizing ontologies is to use a graph because it can show multiple inheritance and follows more closely the semantic web ontology languages RDF(S) and OWL. A graphical method of modeling ontologies is employed in tools such as KAON OI-Modeler (Gabel et al., 2004) and Protégé OWLViz Plug-in (Horridge,

2005b), in which the representational method uses nodes to represent concepts and edges to represent relationships between concepts. A limitation of the graph method is that a graph containing hundreds of nodes and edges can be difficult for the user to navigate and comprehend.

The Unified Modeling Language (UML) tools are commonly used for visual representation and communication of software design. Due to the similarities between ontologies and object-oriented design, UML class diagrams can be used for modeling ontology classes and their properties, and relationships between classes (Cranefield et al., 2003). However, UML's expressiveness for modeling ontologies is limited; for example, standard UML cannot express more advanced ontologies that contain descriptive logic (Gasevic et al., 2005).

Due to the limitations of UML, there is research work in progress to develop a graphical notation for ontologies, called the Ontology Definition Metamodel (ODM) (Gasevic et al., 2005). ODM is based on Model Driven Architecture (MDA). MDA has similar goals to the Semantic Web; one of the goals of MDA is to "allow definition of machine readable application and data models". IBM has developed a tool that supports ODM, called the Integrated Ontology Development Toolkit (Ma et al., 2006). One of the features of ODM is that it supports descriptive logic.

The major "bottleneck" during the ontology development process is knowledge acquisition. Knowledge acquisition involves talking to experts to acquire their knowledge in a particular domain. This is a very time-consuming and expensive process. Many tools have features to help bypass much of this knowledge acquisition. Much knowledge is contained within text documents and databases. There are tools to

automatically generate an ontology from these knowledge sources. The process of doing this is called ontology learning (Maedche & Staab, 2003). Ontology learning is not perfect and errors are common, but it can quickly assist the ontology author in generating some base classes or populating a knowledge base with instances.

Existing ontological engineering tools can model static knowledge to varying degrees. One issue is data type support. Not all tools are capable of supporting all data types (string, integer, float, etc.). The data types supported by the tool should be investigated before the modeling process begins.

While most tools can support representation of static knowledge, there are significant difficulties when these tools are used for modeling dynamic knowledge. The general procedure to model dynamic knowledge in tools such as Protégé, KAON OI-Modeler, and KMS consists of the following three steps:

1. Create classes to represent Task and Objective. These are the main structures that will be used for modeling dynamic domain knowledge. It should be noted that this step is not required in KMS, as it was designed for modeling dynamic domain knowledge and most of the necessary structures are built-in.
2. Model the static domain knowledge.
3. Model the dynamic domain knowledge by instantiating the Task and Objective classes that were created in Step 1.

To better illustrate the problems encountered when modeling dynamic knowledge in existing tools, application of the above steps using Protégé to model a small part of the petroleum contamination remediation selection ontology. Protégé uses a form-based user

interface for inputting knowledge; there are windows containing text fields for inputting knowledge. The application of the above steps using Protégé is described as follows:

1. Create classes to represent Task and Objective.
 - a) Create a new Protégé project for containing the Task and Objective classes.
 - b) Create a class for representing task knowledge as shown in Figure 3. The **Task** class properties include *behaviour*, associated *objects*, and a list of sub (or dependent) tasks, as shown in the “Properties and Restrictions” box in Figure 3. In task interactions, data can be exchanged. To capture this important information, two additional properties, *inputs* and *output* are added to the **Task** class. A weakness of KMS is its inability to capture inputs and outputs of tasks, which are important components of the interactions between tasks. We believe that without modeling the inputs and outputs of tasks, it is unlikely that an ontology author will consider the details of the interactions between tasks, and an incomplete model will be developed as a consequent.

Figure 3. Creating a class for representing Task

- c) “Task Priority” is a type of strategic knowledge that is most likely to be found in an industrial domain, such as petroleum contamination remediation selection. For any particular objective, the Task Priority specifies which Tasks are required and the order in which they are to be performed. Task Priority can be embedded in a prioritized list of tasks inside the Objective class. However, the existing ontology tools, including Protégé, lack support for specifying the order (or priority) of specific items. This problem can be solved with the addition of an intermediate class, called “Task Priority”, to relate a Task to a priority. The class for representing “Task Priority” is shown in Figure 4. It contains the properties *task* (an instance of a Task) and *priority* (an integer), thus enabling the association of a priority to a task.

Figure 4. Create a class for representing TaskPriority

d) Create a class to represent an objective as shown in Figure 5. The

Objective class has a property for the *taskPriorityList*.

Figure 5. Create a class for representing Objective

2. Model the static domain knowledge.
 - a) Create a new Protégé project for containing the static domain knowledge.
 - b) Create classes, properties, and relations.
3. Model the dynamic domain knowledge.
 - a) Create a new Protégé project for containing the dynamic domain knowledge.
 - b) Import both the Protégé project created in Step 1 (Task, Objective, and TaskPriority classes) and the Protégé project created in Step 2 (static domain knowledge) in the dynamic domain knowledge project.
 - c) Choose an objective to model. In this example, the objective, **Determine Contamination Level** would be modeled. This is done by creating an instance of the Objective class identified as **DetermineContaminationLevel**. The specification of task priority (strategic knowledge) is discussed later.
 - d) This step describes how to create a task that is required to achieve the chosen objective. In this example, a task to **Calculate Weighted Normalized Benzene Concentration** will be created. To model this task, the Task class is instantiated and identified with the name **CalculateWeightedNormalizedBenzeneConcentration**. Next the task behaviour and objects can be modeled. Task behaviour is input into the “dynamic:behaviour” box and objects are input into the “dynamic:objects” box as shown in Figure 6. A major weakness of existing ontology tools, including Protégé, for modeling tasks is that the structure for task behaviour is not formally represented, which results in inconsistent syntax and grammar. Consequently, machine processing of the task behaviour is not possible.

◆ CalculateWeightedNormalizedBenzeneConcentration (Instance)

INDIVIDUAL EDITOR

For Individual: ◆ CalculateWeightedNormalizedBenzeneConcentration

◆ □ ◆ □ ◆ □

Property	Value
□ rdfs:comment	

dynamic:behaviour 🔍 ✕

soilSamplingExp.benzeneConcentration =
soilSamplingExp.benzeneConcentration /
skStandard.skBenzene *
benzene.contaminationWeight

dynamic:objects ◆ ◆ ◆

◆ skStandard
◆ soilSamplingExp
◆ benzene

dynamic:inputs ◆ ◆ ◆

dynamic:subTasks ◆ ◆ ◆

Figure 6. Create task for Calculate Weighted Normalized Benzene Concentration

- e) This step describes how to specify task priority, which is a type of strategic knowledge. First, an instance of TaskPriority is created and identified as “tp_CalculateWeightedNormalizedBenzeneConcentration”. The “tp_” prefix is used because Protégé does not allow instances to have the same name. Task Priority consists of a task and a priority number as shown in Figure 7.

tp_CalculateWeightedNormalizedBenzeneConce

INDIVIDUAL EDITOR

For Individual: ◆ tp_CalculateWeightedNormalizedBenzeneConcentration

Property
rdfs:comment	

dynamic:priority

1

dynamic:task

◆ CalculateWeightedNormalizedBenzeneConcentration

Figure 7. Task Priority for Calculate Weighted Normalized Benzene Concentration

- f) After a task priority instance has been created, it can be added to an objective as shown in Figure 8.

Figure 8. Determine Contamination Level Objective

There are two problems with this process for modeling task priority. First, the software tool requirement that the user has to create a separate entity of TaskPriority to link tasks to objectives distracts the users from the real activity of specifying strategic knowledge. The second problem is related to the visualization of the task priority list in the objective. Figure 9 illustrates the problem.

In Figure 9, the task priorities for SetContaminationLevel and CalculateWeightedNormalizedTolueneConcentration have been added to the DetermineContaminationLevel objective. From looking at the taskPriorityList, one might think the tasks are in the priority:

1. CalculateWeightedNormalizedBenzeneConcentration
2. SetContaminationLevel
3. CalculateWeightedNormalizedTolueneConcentration.

However, the actual priority of tasks is as follows:

1. CalculateWeightedNormalizedBenzeneConcentration
2. CalculateWeightedNormalizedTolueneConcentration
3. SetContaminationLevel.

Figure 9. Determine Contamination Level Objective with multiple task priorities

In a form-based user interface, which is the kind adopted in Protégé, task priorities are displayed in the order that they were added. In graph-based user

interfaces, such as Protégé OWLViz or KAON OI-Modeler, there is no order to the display; the user must look at a separate window or screen for the priority. Again this problem distracts the user for the real process of knowledge modeling.

To summarize in terms of modeling dynamic knowledge, existing ontology tools demonstrate three main weaknesses. First, their inability to enforce a consistent syntax for task behaviour makes computer processing of the task behaviour impossible. Second, the lack of support for input and output of tasks can result in specification of an incomplete model. Third, the visualization/input fields are not user friendly for enabling the user to easily specify/visualize priority among tasks. Our research work addresses all three weaknesses.

2.5.3 Ontology Testing

The third area in need of improvement is support for ontology testing. Software testing is an important part of the software development life cycle because it identifies defects and can support production of a more stable software application. It is considerably cheaper to fix defects early in the development process. Unit testing is a method of testing that verifies that the modules of a program are functioning as expected. Unit testing provides a number of benefits. One benefit is that it facilitates changes to the code; the software engineer is notified of errors caused by changes to the code. A second benefit is if the Test Driven Development (TDD) (Janzen & Saiedian, 2005) approach is adopted, then unit testing facilitates the design of the software.

Software testing techniques can be applied to ontology testing and help the knowledge engineer develop a more complete model of the domain. In the ontological

engineering field, ontology testing is also called ontology evaluation. According to (Gomez-Perez et al., 2005), ontology evaluation should be performed on the following:

- Every individual definition and axiom
- Collections of definitions and axioms stated explicitly in the ontology
- Definitions imported from other ontologies
- Definitions that can be inferred from other definitions

Existing ontology testing systems such as the OWL Unit Test Framework (Horridge, 2005a) and Chimaera's test suite (McGuinness et al., 2000), evaluate the consistency and completeness of ontologies. An ontology is consistent if it does not contain any circularity errors (e.g. a class is a sub-class of itself), partition errors (e.g. two disjoint classes have common instances), or semantic errors (e.g. a class contains an incorrect property) (Gomez-Perez et al., 2005). Chimaera performs tests in the following four areas: (1) Missing argument names, documentation, constraints, etc., (2) syntactic analysis (occurrence of words, possible acronym expansion), (3) taxonomic analysis (unnecessary super classes and types), and (4) semantic evaluation (slot/type mismatch, class definition cycle, domain/range mismatch) (McGuinness et al., 2000). Such testing tools are sufficient for testing static knowledge, but are not suitable for testing the interactions between behaviour and objects.

This research work aims to contribute to the field of ontological evaluation by addressing the difficult issue of testing behaviour or dynamic knowledge. Our approach attempts to combine unit testing techniques with the adoption of test cases in Test Driven Development (TDD) (Janzen & Saiedian, 2005). This is a useful hybrid approach for addressing the complex interactions of task behaviour and objects. The general intuition

adopted from the TDD approach of testing is that it should be “done early and done often”. In TDD, a test case is written first and then the actual module is written. Writing test cases first can be beneficial because instead of thinking of test cases as “how do I break something”, writing test cases first make you consider “what do I want the program to do”. In other words, writing test cases places focus on defining the required functionality and objects. Ontology development could also benefit from this kind of approach.

2.5.4 Ontology Management

The fourth area that needs improvement in ontology construction support tools is the need for support for ontology management. Ontology development occurs within a collaborative, distributed and heterogeneous environment in that developers around the world work together to create, reuse, and/or extend concepts using different system platforms and different knowledge representations. To document, track, and distribute ontologies in such an environment, an ontology management system is needed, which is analogous to a database management system. In general, ontology editors and ontology management systems are integrated in a client-server architecture, where the ontology editor is the client and the ontology management system is the server. Examples include the KAON Tool Suite (Volz & Oberle, 2003), Ontology Builder (Das et al., 2001), and Protégé (with Change Management and PROMPT plugins) (Noy et al., 2006). In such client-server systems, the server performs the bulk of the ontology management work, which includes ontology storage, distribution, security, and provides interfaces for the clients to access these services. Weaknesses of the systems include security/trust

vulnerabilities and insufficient support for knowledge evolution. These two weaknesses are discussed further as follows.

a. Security/Trust

As previously discussed, knowledge on the Semantic Web is to be represented in XML based file formats. For example, an XML element `<Person>` could be used to represent the person “Robert” by the following: `<Person>Robert</Person>`. A computer program that encounters this XML code processes the text between the `<Person>` and `</Person>` elements and interprets “Robert” to be a person. A major criticism of the Semantic Web is that the text between the elements cannot be trusted (Norvig, 2006). For example, a computer program could process `<Person>table</Person>`, and interpret “table” to be a Person, which is incorrect. There are two reasons why knowledge on the SW cannot be trusted. First, people make mistakes. On the current Web, many people do not write properly formed HTML. Web browsers are generally lenient when processing improperly formed HTML, therefore minor HTML errors often do not affect the appearance of a web page. Even if the appearance of a web page is severely affected, the human brain is powerful enough to understand the meaning of the web page. However, the processing of SW languages (RDF or OWL) cannot be so lenient as even minor errors can change the semantics of an ontology or even make the ontology unprocessable. RDF and OWL are much more complicated than HTML, so it is unrealistic to expect people to write proper RDF or OWL. The second reason why knowledge on the SW cannot be trusted is because of criminal activity. The current Web is full of garbage, scams, and tricks to deceive people. The SW may make it easier for criminals to deceive people. For example, if people use RDF or OWL to annotate

personal information on a web page, it will be very easy for spammers to gather their email addresses.

These same SW trust issues also apply to distributed KBS development. Ontology authors can make modeling mistakes and bad people may create malformed models on purpose. Malformed ontology models used by a KBS can cause a KBS to incorrectly function or not function at all.

To address these “trust” issues on the SW, the layers Proof and Trust have been proposed (Koivunen & Miller, 2001). The mechanisms for Proof and Trust are still under research and as of yet, there is still not a satisfactory approach. One approach to Proof and Trust is to have an administrator that monitors and controls what ontologies and changes are allowed. Ontology authors submit their work to the administrator, who evaluates the work and decides whether or not it should be placed in the system. In other words, the administrator performs quality control.

A second approach to Proof and Trust is to develop a “web of trust” (Richardson et al., 2003). In a “web of trust” each user is responsible for deciding what knowledge is considered credible. An example of a “web of trust” is Friend of a Friend (FOAF) (Richardson et al., 2003). In FOAF A trusts B and B trusts C. A does not know about the existence of C. However, through transitivity, A trusts C. Using FOAF, eventually, a “web of trust” will form.

A third approach to trust is to use a role based security model as implemented in Ontology Builder (Das et al., 2001). Each user has an account that specifies the roles (or permissions) they have within an ontology. Such a security model enables great control

over the actions of the users; users can be given permission to edit certain ontologies, and denied access to other ontologies.

An important aspect of trust is the specification of authorship/ownership of some object, which is important for enabling the creator of an object to obtain credit for the work they have done. The existing Semantic Web ontology languages do not have any elements for specifying the author or owner of an ontology. Some ontology editors have input fields for author name, organization, etc. and save this information to the top of the ontology file in the form of comments. Not only are comments very difficult for computer programs to process, but they also cannot be trusted for the same reasons described above. A technique proposed for identifying authors and the ontologies they create is to use digital signatures. A digital signature is “a data string which associates a message (in digital form) with some originating entity” (Menezes et al., 1997). An ontology object that is digitally signed by its author can be verified by other systems to determine if the author really is who they say they are.

Ontologies can contain private information. Ontologies are vulnerable to prying eyes as they are sent from ontology editor to ontology server and back. A commonly used technique for protecting data from those who should not have access, is data encryption. An ontology can be encrypted, making it unrecognizable to those who do not have the key.

Most security and trust information processing is handled by a server running an ontology management system. Client applications, such as ontology editors, simply provide user interfaces for the user to input their information. The client may then digitally sign and encrypt the information before sending it to the server.

b. Ontology Versioning & Evolution

Many versions of an ontology can be generated whenever a change is made in an ontology. An ontology can be changed due to (1) domain change, (2) a change in the conceptualization, (3) a change in the user's perspective, or (4) a specification change (Klein & Fensel, 2001). Tracking and managing the different versions of an ontology is called "ontology versioning". Ontology versioning provides two main benefits. First, ontology versioning enables change recovery (Huang & Stuckenshmidt, 2005). That is, when changes to the latest version of an ontology have negative consequences, the ontology can be rolled back to the state of the last "good" version. The second benefit of ontology versioning is compatibility (Huang & Stuckenshmidt, 2005). That is, when a new version of an ontology does not function on an older system, ontology versioning ensures backwards compatibility in that older versions of an ontology can be accessed and used by systems that require them.

There are many components required to achieve ontology versioning and reap the benefits described above. An important component is the mechanism for detecting and determining changes in an ontology. The PROMPT plug-in for Protégé compares the structure of two ontologies and notifies the user of any detected differences (Noy et al., 2006). The disadvantage of this versioning method is that backwards compatibility issues may not become apparent.

(Klein & Fensel, 2001) proposed a technique for implementing ontology versioning using a [major].[minor] numbering scheme. For example, in the version number "1.5", 1 is the major version number and 5 is the minor version number. The major version number corresponds to an ontology conceptualization. A change in

ontology conceptualization is a significant change to the semantics of the ontology, e.g. the addition of a new class. A minor change in an ontology version corresponds to a conceptualization refinement, which is a small change that does not affect the semantics of the ontology. Using this technique, backwards compatibility support can be easily supported. An application is backwards compatible with an ontology of the same major version number and same or lower minor version number.

In addition to ontology versioning, an ontology management system should also support ontology evolution, which involves supporting the creation of new versions of an ontology, while ensuring there are no inconsistencies among the different versions (Yildiz, 2006). There are different methods used by ontology tools to support ontology evolution. The major/minor versioning scheme proposed by (Klein & Fensel, 2001) applied the property of monotonicity¹⁴, which enables an ontology conceptualization to evolve and guarantees that existing concepts are not removed. Creating a new conceptualization from an existing conceptualization may create dangling references to obsolete or deleted concepts. In this situation, concepts can be removed after passing through a document consistency engine. This scheme can be further augmented by using a meta-ontology to store the ontological changes between minor versions.

KAON OI-Modeler (Gabel et al., 2004), which is one of the tools in the KAON Tool Suite, focuses on the effects of changes. When a knowledge component, e.g. concept or property, is added, a window is displayed, alerting the user to the effect the addition of the concept will have on rest of the ontology. The same functionality also applies when knowledge elements are modified or deleted. Notifications to the user can

¹⁴ Monotonic functions are *non-decreasing functions*, in which the value of the dependent variable never decreases as the value of the independent variable increases

indicate that child classes and properties may be deleted. Users are also provided with many options to control the behaviour of the ontology evolution. For example, the evolution can be set so that orphaned properties can be deleted, reconnected to super-properties, or left as-is. This feature enables the ontology author to review and consider the changes they are about to make.

Protégé has support for ontology evolution via the PROMPT plug-in and Change-management plug-in (Noy et al., 2006). As described earlier, PROMPT compares two versions of an ontology to detect changes. The Change-management plug-in provides an interface for accepting/rejecting changes and view change history. The change history logs include information on the author of the changes, the date the changes occurred, the action that occurred (e.g. a class has been created), and a description of the action taken by the user.

2.6 Application Problem Domain

2.6.1 Overview of Petroleum Contamination Remediation Selection

Petroleum contamination of soil and groundwater is an important environment issue because it can adversely impact the health and well-being of a community and the surrounding natural environment. Petroleum contamination is often the result of leaks and spills from petroleum storage tanks and pipelines, as shown in Figure 10. From the gas tank, contaminants first leak into the top layer of soil, then eventually through the soil, into the lower, groundwater layer. The petroleum contaminants include chemicals such as Benzene, Toluene, Ethyl Benzene, Xylene, and Total Petroleum Hydrocarbon. These chemicals can potentially cause serious health problems to humans (Chan et al., 2002).

Figure 10. Overview of contaminant problem from petroleum production activities

(Chen, 2001)

The process of cleaning a petroleum-contaminated site is called Petroleum Contamination Remediation. A variety of remediation methods/technologies are available. However, different contaminated sites have different characteristics depending on the pollutants' properties, hydrological conditions, and a variety of physical (e.g. mass transfer between different phases), chemical (e.g. oxidation and reduction), and biological processes (e.g. aerobic biodegradation). Thus, the methods selected for different sites vary significantly. The decision making process for selecting a suitable method at a given site often requires expertise on both the remediation technologies and site hydrological conditions. Since the selection process is complex, an automated system for supporting decision-making on site remediation techniques is useful. Development of such a decision support system (DSS) can benefit from the use of Semantic Web technology like ontology construction (Chan et al., 2002).

2.6.2 Details of Petroleum Contamination Remediation Selection

There are two categories of remediation techniques: 1. in-situ and 2. ex-situ. In-situ remediation techniques use treatments on the soil or groundwater; some examples of in-situ remediation include soil flushing, biostimulation, chemical treatment, and phytoremedization (Chan et al., 2002). Ex-situ remediation techniques involve the removal of the contaminated soil by excavation. Some examples of ex-situ remediation include land treatment, chemical extraction and excavation, air stripping, and carbon adsorption (Chan et al., 2002).

The selection of a remediation technology involves the following five factors (Chen, 2001):

1. Contaminated Media: The contaminated media (soil or groundwater) can be either unsaturated or saturated. Not all of the media require cleaning. There are three possible requirements for a remedy: (a) only the unsaturated zone needs to be cleaned, (b) only the saturated groundwater zone needs to be cleaned, or (c) both the unsaturated and saturated zones need to be cleaned.
2. Site hydraulic conditions: The hydraulic properties of a site include the following considerations: (a) soil permeability, (b) site heterogeneity, and (c) isotropism. According to these properties, a site can be classified as simple or complex.
3. Estimated plume size: If the contaminated site size is small, an ex-situ remediation technique is preferred, otherwise, an in-situ remediation technique is recommended.
4. Current phase of the immiscible contaminants: If the immiscible contaminants are present in the free phase, then oil recovery has to be considered. If the immiscible

contaminants are present in the residual phase, then more complex techniques like the integrated remediation technology is needed.

5. Concentration of the contaminants: The concentration of the contaminants can be grouped into the three ranges of low, medium, and high. Different concentration levels of the contamination require different remediation actions.

2.7 Conclusions

This chapter presented an overview of the ontology field, software tools used to construct ontologies and the challenges encountered while using ontology tools to model an industrial domain. A brief introduction to the petroleum contamination remediation selection domain was also presented.

3. DESIGN AND IMPLEMENTATION OF A TOOL FOR KNOWLEDGE MANAGEMENT

3.1 Overview

To address the objective of knowledge management, a static knowledge modeler, simply called the Class Editor, has been developed. As its name suggests, the Class Editor captures information about classes, properties of classes, and relationships between classes. The Class Editor is the front-end application for an underlying knowledge management system, called the Distributed Framework for Knowledge Evolution (DFKE) (Obst, 2006). Both the Class Editor and DFKE were implemented in the Python¹⁵ programming language.

A high level view of the architecture of the Class Editor and its interaction with the DFKE is shown in Figure 11. The user enters all of their input into the Class Editor. The user creates classes, properties, and relationships between classes using the Diagram module. A UML class diagram-like visualization is used to support the user in their knowledge modeling. There are modules for supporting the modification of the static knowledge elements. A Project module provides storage for the knowledge model. User information, such as their name and web page, is created and appended to the knowledge elements with the User module. The Class Editor interacts with the modules of the Distributed Framework for Knowledge Evolution via the ClientServices module. ClientServices provides functions for representing knowledge in a generic knowledge representation and importing and exporting knowledge models from and to XML. DFKE uses a peer-to-peer (P2P) network architecture¹⁶. The “Feeder Node” is a node in this

¹⁵ Python, <http://www.python.org>

¹⁶ Peer-to-peer network architecture, <http://en.wikipedia.org/wiki/Peer-to-peer>

P2P network. The DFKE stores the knowledge projects in a knowledge repository or database, which is attached to a “Feeder Node”. Network communication between the Client Services and the FeederNode (and Knowledge Repository) is achieved using Common Object Request Broker Architecture (CORBA)¹⁷. DFKE also handles security, by encrypting network communication and digitally signing objects. There is also a Project Browser for assisting the user in finding projects on the P2P ontology network.

¹⁷ CORBA, <http://www.omg.org/gettingstarted/corbafaq.htm>

Figure 11. High-level architecture of Class Editor

The following describes the general process of modeling knowledge with the Class Editor and using some of the DFKE features. A detailed description of the process applied to the modeling of a real-world problem domain is presented in Chapter 5.

1. Create a new project

2. Create a class
3. Create a property for the class
4. Create a relation from one class to another
5. Repeat steps 1 – 3 until satisfied that the static knowledge model is complete.
6. Click “Commit” to upload the ontology project to the network.
7. Use the Project Browser to see that the ontology project is on the ontology network.

3.2 Knowledge Representation

Tools based on the XML representation languages are difficult to develop and maintain, especially if the representation language changes. There are representational deficiencies inherent in representing objects within XML and prior knowledge of how to interpret the language structure is required. (Kivela & Hyvonen, 2002) proposed several domain independent constructs which can be used as a basis for creating an ontology representation language:

- **Class** : a generic concept similar to a frame
- **Superclass relationship** : a conceptual hierarchy allowing subsumption
- **Class properties** : a property similar to a slot allowing for simple inferencing and inheritance
- **Individual** : an instance of one or more classes by defining an *is-a* relationship
- **Axioms and constraints** : based on formal logic constructs such as axioms, constraints, and inference rules

Using the independent constructs defined in (Kivela & Hyvonen, 2002), a generic ontology representation was created by (Obst & Chan, 2005). This representation uses an

object-oriented structure to define the knowledge elements. The knowledge elements include structures for defining classes, properties of classes, and relationships between classes. The representation is enhanced with many more structures to support ontology management. These knowledge representation elements and ontology management support structures are organized into the packages **Common**, **Knowledge**, **Monotonic Properties**, and **Ontology**, which are briefly described as follows:

a. Common Package

The Common package contains classes, shown in Figure 12, that define attributes that are common among the many other classes in the generic representation. The Description class represents the attributes that can be used to describe an entity in the generic representation. The DFKE uses the Version class to track the versions of KnowledgeComponent objects (described in part d.) using the major/minor versioning scheme proposed by (Klein & Fensel, 2001). The Person class represents the attributes to describe a creator or author of an entity in the generic representation. The DatedComment class enables generic representation entities to have comments with a time or datestamp associated with them.

Figure 12. Classes in the Common Package

b. Knowledge Package

The Knowledge package contains the classes KnowledgeComponent and Knowledge Project as show in Figure 13. The KnowledgeComponent class is used as a

base class for defining other knowledge components. KnowledgeComponent is derived from the Description and Version classes defined in the Common package, enabling knowledge components to be described and versioned. KnowledgeProject is a KnowledgeComponent and represents a project.

Figure 13. Classes in the Knowledge Package

c. Monotonic Properties Package

The Monotonic Properties Package defines the monotonic properties previously defined by (Klein & Fensel, 2001). This package contains a single class, AccessCertificate, which is shown in Figure 14. AccessCertificate contains five Boolean attributes:

1. **monotonic** – Specifies whether or not the knowledge element is monotonic. If the knowledge element is monotonic, then it cannot be removed from the ontology.
2. **readOnly** – Specifies whether or not the knowledge element is read only. If the knowledge element is read only, then it cannot be modified.

3. deprecated – Specifies whether or not the knowledge element is obsolete and no longer in use.
4. allowComment – Specifies whether or not comments are allowed to be added to the knowledge element.

Figure 14. Monotonic Properties Package

d. Ontology Package

The Ontology package contains classes, shown in Figure 15, for defining the four primitive knowledge elements Class, DataProperty, Relationship, and RelationshipEndPoint. Class is for representing concepts. DataProperty is for representing attributes or properties of classes. Relationship and RelationshipEndPoint are for representing “connections” between classes. Of the set of four primitive knowledge elements, Class and Relationship are considered to be major knowledge elements, and are thus derived from KnowledgeComponent and AccessCertificate. The Concept element is derived from KnowledgeProject. Concept defines attributes for representing a conceptualization of an ontology. The Ontology element is derived from Concept. The Ontology element defines lists of classes and relationships, as well as some attributes for specifying refinement of an ontology.

Figure 15. Classes in the Ontology package

Using a standard knowledge representation format makes the model interoperable with other systems. It is very important that the generic representation be translatable into a standard format, such as one of the semantic web ontology languages. (Volz & Oberle, 2003) proposed three requirements for developing semantic web ontology language support: 1) support current SW standards (OWL and DAML) and future languages, 2) translate between different ontology languages with different semantics, and 3) translate between different ontologies of the same language. The generic representation fulfills these requirements. The generic representation can be translated into any XML based representation language and vice versa provided the language supports the previously mentioned independent constructs (Kivela & Hyvonen, 2002). This allows the generic representation to be interoperable with existing formal ontology languages, including OWL. The generic representation can also be used as an

intermediate format when translating from one formal representation to another if a translation table does not exist. New functionality can be added to the representation by updating the structures and interfaces provided by each primitive. The object-oriented properties of data hiding and encapsulation can be used to store processing logic within each component, making the representation more portable. Access policies can be created and directly supported by each primitive, reducing usage errors when creating an ontology.

3.3 Static Knowledge Modeling

A graphical, UML class diagram-like method is used for modeling static knowledge in the Class Editor. The modeling process in the Class Editor is similar to drawing on a canvas with pre-defined shapes. Classes are created by clicking the “Class” button from the toolbar and then clicking anywhere in the Diagram Window to create the class at that location. The class is drawn as a rectangle and initially only contains the name of the class. Datatype properties are created by selecting a class in the Diagram Window, right-clicking, and then selecting “Add DataType Property” from the pop-up menu. When a data type property is added to a class, the class’s rectangle expands and shows two compartments. The top compartment of the class contains the name of the class. The lower compartment of the class contains the names of the data type properties of the class. Relationships can be created, however they have no explicit meaning, such as “is-a”, “has-a”, etc. For example, the two classes “Media” and “Soil” may have a relationship connecting them. Since the relationship created in the Class Editor has no explicit meaning, it is left to the user to interpret the meaning of the relationship. To create a relationship between two classes, first click the “Relationship”

button from the toolbar, then click once on the first class, and then click once on the second class. This will draw a line representing a relationship between the two classes.

Ontology elements can be deleted by selecting them and pressing Delete. Only the selected class and its connecting relations are deleted. Any classes that are connected will remain. When a property is deleted from a class, the class simply resizes itself to fit the remaining properties.

All the ontology elements in the Diagram Window are moveable. Classes can be moved by dragging them with the mouse. A problem with graph methods of modeling is that the visualization of the model can quickly become difficult for the user to navigate and understand. The Class Editor provides a function so that the user can make the tool automatically organize and place the classes in locations that are easier to view and understand. Although the automatic placement of the classes is very good, when there are hundreds of classes, it can be difficult to find the class you are looking for. For this reason, the Class Editor also has an alphabetical listing of all the classes in the ontology. With the class listing, the user can scroll through all the classes and select the ones they are interested in.

Not all ontology information can be displayed and modified graphically. Non-graphical data is modified through various windows for each type of knowledge. These windows use a tabbed user interface, which enables the user to quickly and easily switch between different ontology information types, including ontology management information. These windows are accessed by double left-clicking on an ontology element.

3.4 Ontology Management

As previously stated, the Class Editor is the front-end application for the Distributed Framework for Knowledge Evolution (DFKE) developed by (Obst, 2006). One of the goals of both the Class Editor and DFKE was to hide as much of the “guts” of ontology management from the user as possible; the user should only be concerned with ontology modeling, not ontology management. With this goal in mind, much of the ontology management work is done automatically in the background for the user.

Documentation is an important part of ontology management, however it can be neglected under tight deadlines. When creating ontologies for sharing and reuse it is particularly important to provide information describing the author of the ontology. Author information includes the user’s name, email address, organization, etc. Since author information is the same for all knowledge elements a user creates, the author information can be gathered once and then used by the system to automatically fill the input fields. The first time the user launches the Class Editor, a window requesting user information is displayed. This window has input fields for gathering personal identifying information such as the user’s name, email, web page, and organization. When a user creates any knowledge element, this gathered information is automatically attached to the knowledge element. All of this user information can be viewed in the properties window of any knowledge element.

The window also has an input field for a digital signature. A digital signature is “a data string which associates a message (in digital form) with some originating entity” (Menezes et al., 1997). DFKE uses digital signatures to identify the author of an ontology element and to restrict access to an ontology element to only those people who

should have access, thereby protecting the ontology from malicious ontology authors. To generate a digital signature for the user, the user enters a passphrase into an input field in the user information window, and clicks a button to generate the digital signature. In the background, the DFKE, uses the user information and passphrase to generate the digital signature. This digital signature is used to identify each user and the knowledge elements the user creates. If another user attempts to modify a knowledge element that was created by the user, the other user will be denied.

The distributed and heterogeneous nature of the Semantic Web makes it very difficult for users to find ontologies. An idea of a Unified Ontology View (UOV) was proposed (Harrison et al., 2005c), which provides a layer that hides the heterogeneous and distributed nature of the ontology network. The user will be able to access the ontology network from anywhere and obtain a unified view of the ontologies in the network. Figure 16 shows the ontology projects from the perspective of the servers. The servers exchange ontology project information to present to the Client Application a single view of all the ontology projects in the network. In this network, whether the Client is connected to server A, B, or C is irrelevant; the Client will be presented with the same view of ontology projects in the ontology network.

Figure 16. Server View of Ontology Projects

Figure 17 shows the ontology projects from the perspective of the Client. When the Client connects to the ontology network, it only sees the ontology projects. It is possible that the ontology projects are distributed among many different types of systems, however the Client is unaware of this fact.

Figure 17. Unified Ontology View

DFKE implemented the idea of a UOV by using a peer-to-peer (P2P) architecture, as shown in Figure 18. Knowledge projects created with the Class Editor are stored in a database (or knowledge repository), which is connected to a Feeder node. The Feeder node that the Class Editor connects to may be local or remote. Feeder nodes can share knowledge projects with each other. The Feeder nodes discover other feeder nodes by contacting a central Hub node. Ontology projects are automatically added to the network when they are created with the Project Wizard. Once a project is added to a node on the network, it can be accessed from any node in the network. To provide a literal “unified ontology view” and assist the user in finding ontology projects, the Class Editor has a support tool, called the Project Browser. The Project Browser displays all projects found in the ontology network. Projects can be selected from the Project Browser and opened in the Class Editor.

Figure 18. P2P Architecture

Projects can be used outside of the network by saving them in an eXtensible Markup Language (XML) file.

In a step to support knowledge evolution, the DFKE uses the Concept and Ontology elements described in section 3.2 for representing conceptualizations and refinements of an ontology. Conceptualizations are considered to be major changes to an ontology, while refinements are minor changes. The major/minor versioning scheme proposed by (Klein & Fensel, 2001) is used to keep track of the versions of the ontology as conceptualizations and refinements are added. Removing a conceptualization may break other systems that depend on the conceptualization. Therefore, conceptualizations cannot be removed. The DFKE uses the concept of monotonicity¹⁸ to guarantee that existing concepts are not removed. The Class Editor helps support monotonicity by denying users the ability to remove knowledge elements that have been marked as monotonic by DFKE.

3.4.1 Knowledge Evolution in Class Editor

The only remaining major ontology management component yet to be completed in the Class Editor is the Knowledge Evolution component. A general scenario outlining how knowledge evolution could be performed by the user of the Class Editor is as follows:

1. The user loads an ontology project from the ontology repository.
2. The user browses the components of another ontology project and selects a class to import.

¹⁸ Monotonic functions are non-decreasing functions, in which the value of the dependent variable never decreases as the value of the independent variable increases

3. The Class Editor examines the effect of adding the class to their ontology and notifies the user of what changes will occur. The user accepts the changes.
4. The selected class is imported into the user's ontology project and appears in the Diagram Window.
5. The user is finished, so they click "commit". A window appears enabling the user to enter some comments describing what actions they just performed. When the user clicks "finished", the ontology project is submitted to the ontology repository and is made available across the ontology network.
6. The user views the change history log and sees all the changes that have occurred on this project, including their own.

This scenario helps identify which components are required before the Class Editor can support knowledge evolution. The Class Editor supports monotonicity, which is a basic component for building support for knowledge evolution. First, a mechanism will have to be built to support the importing of an ontology element from one project into another. Second, a module will have to be built that will examine the consequences of modifying the ontology and notifying the user. The third component to be constructed is the Change History Log. The Change History Log would be similar to the Protégé Change-Management plug-in in that it would provide a record of all the modifications that occur to the ontology.

An optional, but very useful component would be visual cues for the ontology elements. Borrowing from the software code versioning system, Subversion¹⁹ (with TortoiseSVN²⁰), we can colour code the ontology elements, depending on their state. If

¹⁹ <http://subversion.tigris.org/>

²⁰ <http://tortoisesvn.tigris.org/>

an ontology element has not been versioned, then it is coloured in grey-scale. If an ontology element has been versioned and is up-to-date, then it can have a green mark. If an ontology element has been versioned, but is not the latest version, then it can have a red mark.

There is the issue of how to handle ontology elements that have been imported from external projects. When the ontology element is updated and has a new version in the external project, should the ontology element in your own project be automatically updated? The answer is probably no. A good solution would be, if there is an update available, notify the user of the update, show the user the changes that would occur if the update is made, and finally, if the user wishes, he or she can accept the update.

3.5 Conclusions

This chapter discussed the design and implementation of the Class Editor, a static knowledge modeling tool which supports knowledge management. The Class Editor supports knowledge sharing and re-use by automatically managing author information and access rights in a peer-to-peer environment via an underlying Distributed Framework for Knowledge Evolution (DFKE) (Obst, 2006). Also discussed was how to implement the knowledge evolution component in the Class Editor.

4. DESIGN AND IMPLEMENTATION OF A TOOL FOR DYNAMIC KNOWLEDGE MODELING

4.1 Overview

As previously discussed, the purpose of this work is to construct a suite of tools to support the following objectives: 1. support for a knowledge modeling technique, 2. support for ontology management, 3. support for testing of the developed knowledge model, and 4. support for diverse types of knowledge required to model an ontology of an industrial domain. To address the objectives of modeling and testing dynamic knowledge, a Protégé plug-in, called Dyna, has been developed. Protégé, an ontology editing tool created by researchers at Stanford University (Gennari et al., 2003), is an open-source system programmed in Java and is extensible through its plug-in architecture. Dyna is a “Tab Widget” that works with both Protégé-Frames and Protégé-OWL. At the time of writing, Dyna has been tested on Protégé 3.2.1.

A high level view of the architecture of Dyna and its interaction with Protégé is shown in Figure 19. The user creates static knowledge (classes, properties, relations) in Protégé and the dynamic knowledge of objectives and tasks in Dyna. Dyna contains two main modules for creating objectives and tasks. From the Objective module, tasks can be linked to objectives. The Task module can support the functions of (1) definition of task behavior, (2) instantiation and manipulation of objects and properties created in Protégé, and (3) creation and running of test cases for verifying the task behaviour is working as expected. Both Protégé and Dyna can import and export the models in the OWL file format. Dyna can also import and export the models in the XML file format.

Figure 19. High-level architecture of Dyna

The following describes the general process of modeling knowledge with Protégé and Dyna. A detailed description of the application process for modeling of a real-world problem domain of remedial technology selection is presented in Chapter 5.

1. Model static knowledge in Protégé:

7.1 Create a class

7.2 Create a property for the class

7.3 Specify any additional attributes for the class, such as restrictions.

7.4 Repeat steps 1.1 – 1.3 until satisfied that the static knowledge model is complete or the knowledge engineer has enough information for progressing to Step 2 to develop the dynamic knowledge model.

2. Model dynamic knowledge in Dyna:

2.1 Create an objective.

- 2.2 Create a task
 - 2.2.1 Specify task behaviour
 - 2.2.2 Specify objects (instances of classes) used in the behaviour
 - 2.2.3 Specify and run test cases. Object attributes can be modified and checked if they are the expected values.
 - 2.2.4 Link task to objective
- 2.3 Repeat steps 2.1 and 2.2 until satisfied that the model is complete.
- 3. Optionally, the static and dynamic knowledge models may be exported in the OWL file format.

4.2 Knowledge Representation

The static knowledge components are handled by Protégé, which uses both a Frames-based knowledge model and an OWL-based knowledge model. Both knowledge models provide classes, properties (or slots) of classes, parent-child relations between classes, and individuals (or instances) of classes. Protégé-OWL also supports the many additional knowledge components defined in OWL. See <http://protege.stanford.edu> for more details on the Protégé knowledge models.

The dynamic knowledge components as defined by the IMT are organized into the object oriented class hierarchy shown in Figure 20. The top-level component is **KnowledgeComponent**, which defines the properties **name** and **documentation**, which are used for identification and description of a knowledge component. The components **Project**, **Task**, and **Objective** are derived from **KnowledgeComponent** as denoted by the arrows pointing from **Project**, **Task**, and **Objective** to **KnowledgeComponent**. A **Project** is composed of a list of tasks and a list of objectives as denoted by the

composition relationships from **Project** to **Task** and **Project** to **Objective**. An **Objective** consists of a prioritized list of tasks required to achieve the objective. A **Task** is an activity that is performed to achieve an **Objective**. A **Task** consists of behaviour, input values, one output value, pre-conditions, objects, and dependencies. A task can also be decomposed into sub-tasks, so a **Task** contains a list of tasks as denoted by the aggregation relationship pointing from **Task** to itself. An **Objective** is composed of a prioritized list of tasks, represented by **TaskPriority**. **TaskPriority** consists of an aggregate Task, as denoted by the aggregation relationship from **TaskPriority** to **Task**, and also a priority, thus enabling the association of a **Task** with a priority number.

Figure 20. Dyna Knowledge Representation

4.3 Dynamic Knowledge Modeling

4.3.1 Objective Modeling.

Objectives are modeled in an Objective Window, which contains input fields for the name of the objective, documentation, and tasks associated with the objective. The tasks associated with an objective are in a prioritized order so that a task with higher priority should be executed first. Test cases for the tasks can also be run, and tests are run in the order of task priority. This function is discussed further in Section 4.4.

4.3.2 Task Modeling.

Tasks are modeled in a Tasks Window, which contains input fields for the name of a task, documentation, behaviour, objects, dependencies, and test cases of a task. The main components of the task module are the Task Behaviour Language (TBL) and its interpreter. Together, TBL and its interpreter enable the specification and manipulation of objects associated with a task and tasks dependent on other tasks. They also enable dynamic knowledge to be tested. These features are described as follows.

a. Task Behaviour Language.

A weakness of the Knowledge Modeling System (KMS) (Chan, 2004) is that the system does not support a formal and systematic representation of task behaviour. As a result, the representation of task behaviour is not formal and involves inconsistent syntax and grammar, which renders machine processing of the task behaviour impossible. Dyna solves this problem by using a strictly enforced, formalized and high-level language, called Task Behaviour Language (TBL), for representing task behaviour.

TBL supports the following basic structures that are common to most programming languages:

- e) Types: integer, float, Boolean, string
- f) Mathematical operations: +, -, *, /, %
- g) Logical operators: and, or, not, xor
- h) Conditional operators: <, >, ==, <=, >=, !=
- i) Assignment operator: =
- j) If statement, While loop, Assert statement, Return statement, Print statement
- k) Class Object Instantiation

l) Function definition and calling

m) Comments: #

Dyna enforces the structure of the task behaviour with an interpreter, which also enables the task behaviour to be run. A high-level view of the architecture of the TBL interpreter is shown in Figure 21. The interpreter consists of a lexical analyzer and a parser, which were generated using JavaCC²¹, a tool for generating compilers and interpreters. The lexical analyzer breaks the input task behaviour into sequences of tokens. The parser then analyzes the sequences of tokens according to the TBL grammar and generates an abstract syntax tree (AST). If the input task behaviour has errors, then the parser outputs an error message. The functionality of each language element was implemented in the AST nodes; and there is an AST node for each TBL language structure. Interpretation of TBL is achieved by performing a depth-first traversal of the AST. As the AST is traversed, encountered symbols or identifiers are stored in the symbol table, values are pushed/popped on/off the stack, and instances of classes in Protégé are modified.

Two questions that might be asked are “why create TBL?” and “why not use an existing language?”. Any programming language that supports the basic structures (math operators, loops, conditions, etc.) could have been used for modeling the behaviour component of dynamic knowledge. The grammar describing the language, for example C++, would have been processed with a tool similar to JavaCC resulting in a lexer and parser, from which a custom interpreter could be generated. Existing programming languages have many features and language attributes, many of which are not applicable

²¹ JavaCC, <https://javacc.dev.java.net/>

for modeling dynamic knowledge. We believe processing these languages would involve significantly more work as the extra features would have to be navigated and dealt with in some manner. Hence, it would be easier and more efficient to create a simple language and interpreter built specifically for processing task behaviour. On the other-hand, there are non-programming languages that could have been used too, for example, the Semantic Web Rule Language (SWRL)²². SWRL is a combination of OWL and Horn-like rules. SWRL can be used to model processes (Happel & Stojanovic, 2006), so it may be possible to adapt SWRL to model tasks and objectives. SWRL would be a radically different approach to modeling task behaviour than the approach adopted by KMS, the system on which Dyna is based. Since the objective of Dyna was to improve upon KMS, and not a different approach, Dyna adopted TBL, which is like a programming language, for modeling task behaviour.

²² SWRL, <http://www.w3.org/Submission/SWRL/>

Figure 21. Architecture of TBL Interpreter

b. Task Objects

Task behaviour consists of calculations and operations on static knowledge represented in a class of objects. The Task Window enables the user to select a Class in Protégé to instantiate and specify a name for the resulting object. Operations and

calculations can be performed on the object and its properties using TBL. For example: *car.color = "red"*, where *car* is an instance of a Car class, and its property *color* is set to "red". When a test case is run through the interpreter, the defined operations and calculations on the object are actually performed.

c. Task Dependencies

Tasks can have interactions with other tasks. When a task requires a second task to perform its behaviour, this second task is considered to be a "dependency" of the first task. Task dependencies are represented in TBL by: *DependentTaskName()*. Calls to dependent tasks are similar to function calls found in common programming languages. Values can be passed to the dependent task via input parameters. An output value can also be returned from the dependent task. When a dependency is encountered by the interpreter, the dependency is first added to the task's list of dependencies, then the interpreter evaluates the behaviour of the dependency.

4.4 Testing Dynamic Knowledge

Two existing ontology testing systems are the OWL Unit Test Framework (Horridge, 2005a) and the test suite in Chimaera (McGuinness et al., 2000). The OWL Unit Test Framework is a Protégé plug-in that checks that classes are classified as expected. Such testing tools are sufficient for testing static knowledge, but are not suitable for testing dynamic knowledge; specifically, the interactions between task behaviour and objects.

Test Driven Development (TDD) (Janzen & Saiedian, 2005) can be a very useful technique for dynamic knowledge modeling. Writing test cases for tasks first can help the ontology author better understand which classes and properties are required in the

task behaviour, resulting in a more complete model. However, since writing test cases first may initially prove difficult for some users, Dyna supports the creation of test cases at any time during the ontology development process.

Dyna's testing framework is based on JUnit²³, a unit testing framework for Java. To test a class in JUnit, a test class that contains methods representing test cases is created. A JUnit test case consists of an initialization section, a run test section, and a verification section. The initialization section defines the data required for running the test case. The run test section is typically a call to a method that is to be tested. The verification section contains assert functions to verify that the test is a success. Since the initialization code can be the same for multiple test cases, JUnit provides a special method called `setup()` for doing initialization that is common among the test cases.

Similarly in Dyna, each task contains a test suite, which provides a "setup" module and facilities for creating test cases and defining them in TBL. The "setup" module is for defining initialization that is common among multiple test cases. The run test section is simply a call to the task to be tested, and the call can take input parameters and return a value. The verification section uses the "assert" function, which takes as a parameter a condition that is necessary for the test case to succeed. When the test case is run through a test interpreter, the interpreter first performs any necessary initialization, then it executes the behaviour of the task defined by the call to the task, and finally it evaluates the condition of the "assert" function. Depending on whether the "assert" function returns a "true" or "false" value, the interpreter would display a message notifying the user if the test case has "passed" or "failed" respectively.

²³ JUnit, <http://www.junit.org>

To illustrate Dyna’s testing capabilities that were described above, an example from the Financial domain is used. This Financial ontology will contain a task for paying the bill for an account, which will be tested. The static knowledge for the Financial ontology consists of a class to represent an Account. The Account class has the property balance, which is a float data type. The dynamic knowledge for the Financial ontology consists of the task “BillPayment”. The properties of the task are as follows:

Objects: account (instance of Account class)

Inputs: float amount

Behaviour: `account.balance = account.balance – amount`

Figure 22 shows the test suite for the BillPayment task. Highlighted on the left-side of Figure 22 is name of the current test case, “testBillPayment”. On the right-side of Figure 22 are the following three lines of TBL code that define the test case of testBillPayment:

```
account.balance = 1000.0
BillPayment(1000.0)
assert(account.balance == 0.0)
```

The first line initializes the test case by setting the value of account.balance. The second line creates a call to the task behaviour for BillPayment and passes the input value 1000.0, which is the amount of the bill to pay. The third and final line is an assert function that will evaluate the condition `account.balance == 0.0`. In this case, the assertion will be “true”, and therefore the test passes. Figure 23 shows the result of running the test case.

Figure 22. Test Suite for BillPayment Task

Figure 23. Result of Running Test Case testBillPayment

4.5 Knowledge Storage and Interoperability

4.5.1 XML

Dyna projects are stored in the XML file format, enabling them to be shared and re-used by other systems. Most of the dynamic knowledge components and their properties are represented as a hierarchy of XML elements. The following is a sample of the XML for a task:

```

<Task>
  <Name/>
  <Documentation/>
  <SubTaskList/>
  <DependencyList/>
  <Behaviour/>
  <ObjectList>
    <Object/>
  </ObjectList/>
  <PreCondition/>
  <TestSuite>
    <TestSetup/>
    <TestCaseList/>
  </TestSuite>
</Task>

```

Since Task is represented as a class in the object-oriented representation, it is a top-level element in the XML. All the attributes of a Task are represented in XML as sub-elements of the Task element. Some of the attributes are lists of items. To represent lists, an ItemList element is defined whose sub-elements are the items. Also, some of the attributes are classes themselves and contain their own set of attributes. Representation of attributes that are classes is performed by creating an element for the class and the sub-elements for each attribute.

Task Objects have a link to their instances defined in Protégé. Linking to the instance defined in Protégé from Dyna XML was achieved by using RDF's "resource" element, shown as follows:

```
<Object rdf:resource="http://example.com/petrol_rem_sel.owl#benzene"/>
```

rdf:resource defines where the definition of this object can be found. The static knowledge is stored in the OWL file, *petrol_rem_sel.owl* on a hypothetical web server at *http://example.com*. *benzene*, is the identifier of the object.

Dyna XML provides limited interoperability with other systems. It is difficult to share and re-use knowledge in this format because it is primarily XML, which lacks representation of semantics. Representing semantics is important because without it, the information that, for example, a Task is a class or a Name is a string, cannot be conveyed to other systems. Moreover, without semantics, relationships between XML elements are not explicit. For example, it is impossible for other systems to know that a DependencyList is a list of Task instances. While XML has these limitations, its expressiveness is sufficient for representing dynamic knowledge within the confines of

Dyna as it knows how to interpret the XML elements. A better storage format is needed in the future for true interoperability to be realized.

4.5.2 OWL

A weakness of Dyna XML is that it lacks semantics, making it difficult to use the dynamic knowledge in other systems. To achieve true interoperability with other systems, Dyna projects can be exported to OWL, which provides semantics to XML. In this way, it is possible for another system to know for example, that a Task is an instance of a task in the dynamic knowledge model.

The modeling of the dynamic knowledge elements was done in Protégé-OWL. As described in Section 2.2, there are three types of OWL considered: (1) OWL Lite, (2) OWL DL, and (3) OWL Full. OWL Lite is the simplest and least expressive. OWL DL is more expressive than OWL Lite. OWL Full is the most expressive, but performing reasoning on it is very difficult. The type of OWL chosen for the project was OWL DL because (1) it is sufficiently expressive, and (2) the purpose of modeling the petroleum contamination remediation selection domain is to eventually create a DSS, which means automated reasoning is required. OWL DL proved to be indeed expressive enough, and every dynamic knowledge element in the remediation domain was represented. The final OWL dynamic knowledge model was verified as valid OWL DL by using the WonderWeb OWL Ontology Validator²⁴. Since the model is in valid OWL DL, any other system that can read OWL DL, will be able to use this model.

The following is a sample of the OWL code used to model the dynamic knowledge components. The OWL in this example defines a class **Task** and states that it

²⁴ WonderWeb OWL Ontology Validator, <http://phoebus.cs.man.ac.uk:9999/OWL/Validator>

is a sub-class of **KnowledgeComponent**. There is also a definition of the property **name**.

```
<owl:Class rdf:about="#Task">
  <rdfs:subClassOf rdf:resource="#KnowledgeComponent"/>
</owl:Class>
<owl:FunctionalProperty rdf:ID="name">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:domain rdf:resource="#KnowledgeComponent"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:FunctionalProperty>
```

Dyna's projects are exported to OWL simply by traversing the internal knowledge model and outputting each element's OWL representation to a file. The following is a sample of the OWL code for representing a task for deciding the number of sampling points. **dyn:Task** defines an instance of the **Task** class, with the **name** property set to "DecideNumberOfSamplingPoints".

```
<dyn:Task rdf:ID="DecideNumberOfSamplingPoints">
  <dyn:name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">DecideNumberOfSampling
Points</dyn:name>
</dyn:Task>
```

Dyna's exported OWL projects have also been verified as valid OWL DL by the WonderWeb OWL Ontology Validator.

4.6 Conclusions

This chapter presented the design and implementation of Dyna, a software tool for modeling dynamic knowledge. Dyna has been developed by extending Protégé so as to render it capable of supporting IMT. A new language called TBL has been proposed for representing task behaviour. TBL supports formal expression of task behaviour so that task knowledge defined in an ontology can be more shareable and re-usable. The

dynamic knowledge model can be tested by running test cases on the task behaviour. The dynamic knowledge is stored in an XML format, but can also be exported to OWL so that it can be shared and re-used by other systems.

5. APPLICATION OF THE TOOLS

5.1 Overview

Both the Class Editor and Protégé/Dyna were applied to create ontologies of the petroleum contamination remediation selection domain. The applications of the tools had two separate motivations. The motivation to apply the Class Editor was to demonstrate its ontology management features, while the motivation to apply Protégé/Dyna was to demonstrate Dyna's dynamic knowledge modeling features.

The workflow for creating an ontology consists of a knowledge acquisition phase, followed by an ontology implementation phase. Knowledge acquisition was performed by L. L. Chen (2001), who consulted with an expert in the petroleum contamination remediation selection domain. The process of selecting a remediation technology for a petroleum contaminated site involves many steps which interact with a number of different objects. According to the IMT, knowledge is considered to be either static or dynamic, therefore the knowledge elements in the petroleum contamination remediation selection domain are categorized into the two types/groups of knowledge. Static knowledge includes concrete objects such as soil, water, and contaminants. Dynamic knowledge includes objectives, such as "Select Remediation Method" and actions or tasks that are required to achieve the objective. Chen organized the results of the knowledge acquisition into five knowledge tables (Chen, 2001); three knowledge tables described the domain objects with their attributes, possible values, and relationships; two knowledge tables described the dynamic knowledge. These knowledge tables by Chen were used as the source material for the ontologies constructed in the implementation phase, which is the phase that involved the use of the Class Editor, Protégé, and Dyna.

The static knowledge was modeled using first the Class Editor, (described in Section 5.2) and then again in Protégé-OWL 3.2.1 (described in Section 5.3.1). The dynamic knowledge is modeled in Dyna and is described in greater detail in Section 5.3.2.

5.2 Application of Class Editor

The Petroleum Contamination Remediation Selection ontology was implemented in the Class Editor through an iterative process of creating a class and then its properties, followed by relationships between classes. The process of class creation and using the Class Editor's ontology management features will be described with an example. Since the **media** (soil, water, groundwater) that have been contaminated constitutes an important knowledge element in the Petroleum Contamination Remediation Selection domain, this element is used in the following example for illustrating the process of creating a class and other characteristics:

1. Enter user information and create digital signature (if running Class Editor first time), as shown in Figure 24.

The screenshot shows a window titled "User" with the following fields and controls:

- First:** Robert
- Last Name:** Harrison
- Email:** harrison@uregina.ca
- Home:** http://www.uregina.ca
- Public Key:** A text area containing a PGP public key block, with a "Generate Key" button to its right.
- Buttons:** OK, Apply, and Cancel at the bottom right.

Figure 24. Class Editor: acquiring user information

2. Create a new project: **PetroleumContaminationRemediationSelection**
3. Create the knowledge elements (class, data type property, relationship) as shown in Figure 25. A class, for example Media, is created by clicking the square “Class” button on the toolbar and then clicking anywhere in the workspace area. A property, for example, “site_volume”, is created by right-clicking on a class and then selecting “Add DataType Property” from the pop-up menu. A relationship between two classes is created by clicking the button that looks like a line from the toolbar and then clicking on two classes. The relationships have no semantics, so it is left to the user to interpret the relationships. The relationships in Figure 25 are to be interpreted as inheritance relationships: **Water** and **Soil** are children of **Media**; **GroundWater** is a child of **Water**; **Soil_GroundWater** is a child of **GroundWater** and **Soil**.

Figure 25. Class Editor

4. If required, detailed information on the knowledge elements can be modified using a set of screens that are accessed by right-clicking on a knowledge element and selecting “Edit” from the pop-up menu. Figures 26 and 27 show some information about the **Media** class. Figure 26 shows the screen for modifying the name, description, and home page of the **Media** class.

The screenshot shows a window titled "Media" with a tabbed interface. The "General" tab is selected, showing three input fields: "Name" (containing "Media"), "Description" (containing "Media is an object consisting of Note 1."), and "Home Page" (containing "http://www.uregina.ca"). At the bottom are "OK", "Apply", and "Cancel" buttons.

General	Creator	Data Properties	Object Properties	Comments	Version	Access
Name <input type="text" value="Media"/>						
Description <input type="text" value="Media is an object consisting of Note 1."/>						
Home Page <input type="text" value="http://www.uregina.ca"/>						
<input type="button" value="OK"/> <input type="button" value="Apply"/> <input type="button" value="Cancel"/>						

Figure 26. Editing documentation for the Media class in Class Editor.

Figure 27 shows the screen for modifying the access certificate information (described in Section 3.2) for the **Media** class. Each of the items in the Access screen is briefly described as follows. Monotonic is checked, so the **Media** class cannot be removed from the ontology project. Read Only is unchecked, so the **Media** class can be modified (renamed, properties added/deleted, etc.). Complete is unchecked, so the **Media** class is not yet finished. Deprecated is unchecked, so

the **Media** class is not obsolete. Allow Comment is checked, so the user can add comments to the **Media** class.

Figure 27. Access information for the Media class in Class Editor

5. Continue steps 2 – 4 until all the classes, data type properties, and relationships are entered.
6. Commit the project to the ontology network
7. View the project in the Project Browser, shown in Figure 28

Figure 28. Project Browser

5.3 Application of Dyna to Modeling of Petroleum Contamination Remediation Selection

5.3.1 Static Knowledge

The static knowledge of the Petroleum Contamination Remediation Selection domain was implemented in Protégé-OWL 3.2.1 using an existing ontology developed by (Chen, 2001) with the assistance of an environmental engineer (Z. Hu). Again, OWL DL was adopted because of its expressiveness and its support of automated reasoning. The static knowledge was modeled using an iterative process of creating a class and then its properties. The process of class creation will be described with an example. As mentioned earlier, since the **media** (soil, water, groundwater) that has been contaminated is an important knowledge element in the Petroleum Contamination Remediation Selection domain, this element is used in the following example for illustrating the process of creating a class and other characteristics:

1. Create a class: **Media**
2. Create a property for the class. OWL supports two types of properties: datatype and object. Datatype properties are for properties of simple types such as “integer” or “string”. Object properties are for properties that are individuals of other classes. The size of Media can be classified as “small”, “medium”, or “large”. Since these are strings, a datatype property identified as **siteSize** was created.
 - 2.1 Specify the domain(s) of the property. For **siteSize**, the domain was set to **Media**.
 - 2.2 Specify the range of the property. The possible values for **siteSize** (“small”, “medium”, “large”) are all strings, so the range of **siteSize** was set to **string**.
 - 2.3 Specify restrictions for the property.

- 2.3.1 Restrict the possible allowed values. The only possible values for the size of a site are “small”, “medium”, and “large”. To enforce this restriction, these values were input into the Allowed Values field.
- 2.3.2 Specify whether or not the property is Functional. OWL individuals or instances can have multiple values of a property. For example a “nameList” property may have the values “John”, “Jane”, and “Joe”. In such a situation with multiple values of a property, the property is considered to be “non-functional”. If a property is limited to one value for an individual, for example a “name” property has the value “Joe”, then that property is considered to be “functional”. An individual of **Media** can only have one value of **siteSize** at a time (a medium cannot be both “small” and “large”), so **siteSize** is set to “functional”.
3. Repeat steps 1 and 2 until either all the classes are represented or the knowledge engineer has enough information to develop the dynamic knowledge model.
4. Specify disjointness. OWL individuals can be of more than one type. For example, it is possible to specify that an individual is both a **Media** and a **Remediation**. This type of scenario may result in unforeseen negative consequences, therefore such modeling errors should be prevented. A class can be made disjoint with one or more other classes, thus preventing the possibility of an individual becoming specified as more than one type. In the case of the **Media** class, it is disjoint with all of its sibling classes (**Contaminant**, **Experiment**, **Gas**, **Mathematics**, and **Remediation**).

The result of the static knowledge modeling process is the class hierarchy shown in Figure 29.

Figure 29. Class Hierarchy in Protege

5.3.2 Dynamic Knowledge

The dynamic knowledge was constructed in Dyna using an existing ontology by (Chen, 2001). The dynamic knowledge was modeled in Dyna and involved an iterative

process of creating an objective, and then its tasks. An example of dynamic knowledge for the Petroleum Contamination Remediation Selection domain is determining the classification of the size of a site (or media). This example is used in the following to show the process of creating objectives and tasks:

1. Create an objective: **DetermineSiteSize**

2. Create a task: **SetSiteSize**

- 2.1 Specify task behaviour and any inputs and output. The behaviour for **SetSiteSize** is as follows:

Inputs: int area, int volume

```
if (area < 1600 and volume < 25000)
{
    site.siteSize = "small"
}
else if (area >= 1600 and area <= 2000 and
        volume >= 25000 and volume <= 30000)
{
    site.siteSize = "medium"
}
else if (area > 2000 and volume > 30000)
{
    site.siteSize = "large"
}
```

This task behaviour is input into the Task Behaviour screen shown in Figure 30.

Figure 30. Dyna – Task Behaviour

- 2.2 Specify any objects, i.e. instances or individuals of classes, which are used in the behaviour. **SetSiteSize** uses the object **site**, which is an individual of **Media**.
- 2.3 Specify unit tests for the task behaviour. The approach to unit testing the task behaviour in Dyna involved creating a test case for each scenario in the task behaviour. A scenario is a situation that will cause the TBL interpreter to visit a unique path in the task behaviour. A path is a route from the start of the task behaviour to the end of the task behaviour. Below, the lines of the task behaviour for **SetSiteSize** have been labelled. In the case of **SetSiteSize**, there are four

paths – one path for setting each possible site size (small, medium, and large) and an additional path when no site size is set. The path for setting the site size to “small” are the lines labelled in order: (a), (b), (g). The four paths through this task behaviour are listed in Table 2.

```

(a)  if (area < 1600 and volume < 25000)
      {
(b)    site.siteSize = "small"
      }
(c)  else if (area >= 1600 and area <= 2000 and
           volume >= 25000 and volume <= 30000)
      {
(d)    site.siteSize = "medium"
      }
(e)  else if (area > 2000 and volume > 30000)
      {
(f)    site.siteSize = "large"
      }
(g)

```

Table 2. Paths for task behaviour “SetSiteSize”

Path #	Lines	Test Case
1	(a), (b), (g)	Test Small Site Size
2	(a), (c), (d), (g)	Test Medium Site Size
3	(a), (c), (e), (f), (g)	Test Large Site Size
4	(a), (c), (e), (g)	Test Site Size Not Set

Testing each path in a task behaviour gives the ontology author confidence that the task behaviour is working as expected. The test case for verifying that the size of a site is “small” is as follows:

```

site.siteArea = 1000
site.siteVolume = 1000

SetSiteSize(site.siteArea, site.siteVolume)

assert(site.siteSize == "small")

```

The test case begins with setting the area and volume of the site. Then the task, **SetSiteSize**, is called and its behaviour is run through the TBL interpreter. The TBL interpreter will evaluate the condition of the first “if” statement and determine that the condition is true and then the interpreter will set the value of `site.siteSize` to “small”. When the TBL interpreter is finished interpreting **SetSiteSize**, the size of the site is verified with the assert function. In this case, the assertion will be “true”, therefore the following message will be output to a screen:

```
SetSiteSize
testSmall : Passed
```

If the assertion was “false”, the message output to a screen would be:

```
SetSiteSize
testSmall : Failed
```

The testing suite, shown in Figure 31, provides the facilities for adding, deleting, and running test cases. Test cases may be run individually or together in sequence. The test cases shown in Figure 31 are for the task of **SetSiteSize**. The test case that is highlighted is for testing of a “small” site size, which was described above.

Figure 31. Dyna – Testing Task Behaviour

3. Link the task to the objective. The link between the task **SetSiteSize** and the objective **DetermineSiteSize** is shown in Figure 32.

Objective : DetermineSiteSize									
Name:	DetermineSiteSize								
Documentation:									
Tasks:	<table border="1"><thead><tr><th>Task</th><th>Priority</th></tr></thead><tbody><tr><td>MeasureSiteArea</td><td>1</td></tr><tr><td>MeasureSiteVolume</td><td>2</td></tr><tr><td>SetSiteSize</td><td>3</td></tr></tbody></table>	Task	Priority	MeasureSiteArea	1	MeasureSiteVolume	2	SetSiteSize	3
Task	Priority								
MeasureSiteArea	1								
MeasureSiteVolume	2								
SetSiteSize	3								

Figure 32. Dyna – DetermineSiteSize Objective

4. Adjust the priority of the task in the objective, if necessary. **SetSiteSize** has a priority of 3, which means that it is to be performed after the tasks **MeasureSiteArea** and **MeasureSiteVolume**.
5. Repeat Steps 1-4 until the knowledge engineer is satisfied that the model is complete. While it is difficult to know when the model is complete, a useful heuristic is to assess whether all the tasks have test cases and if they run successfully. If both criteria are satisfied, then it is likely the model is complete.

The final resulting ontology is listed in the Appendix. The root objective is “SelectRemediationMethod”; all the other objectives and tasks fall under this objective. “SelectRemediationMethod” is described in Table 3 to briefly discuss the final ontology:

Table 3. Select Remediation Method Objective

Objective	Tasks
SelectRemediationMethod	1. DetermineSiteSize
	2. SoilSampling
	3. DetermineSiteHydraulicConductivity
	4. DetermineContaminantionLevel
	5. DetermineContaminationPhase
	6. SetRemediationMethod

The first task, DetermineSiteSize, measures the area and volume of the site and then classifies the size of the site. The SoilSampling task involves a number of different activities such as collecting soil samples, measuring thicknesses and percentages of the soil types in the soil samples, and measuring contaminant concentrations. DetermineSiteHydraulicConductivity finds the degree to which water can move between the spaces between the particles of soil. DetermineContaminationLevel finds the contamination level by calculating the concentrations of benzene, ethyl benzene, toluene, and xylenes. DetermineContaminationPhase simply sets the phase as “free” or “residual”. The final task, SetRemediationMethod, considers the following five factors, which are found in the preceding tasks, to determine the appropriate remedial technology:

1. Media Type (soil or soilgroundwater)
2. Hydraulic Conductivity (simple or complex)
3. Site Size (small or large)
4. Contaminant Phase (free or residual)

5. Contamination Level (low or high)

The entire ontology that was implemented in Protégé and Dyna can be found in the Appendix. Most of the dynamic knowledge was successfully implemented in Dyna. There are some limitations of Dyna, which prevented some knowledge from being modeled. These limitations are discussed further in Chapter 6. There were a few tasks, whose behaviour could not be modeled due to lack of information. The values for soil texture contribution factor were stored in a database; there were no calculations on how the values were derived. Therefore, for testing purposes a value for soil texture contribution factor has been hard coded into the behaviour of the task `CalculateSoilTextureContributionFactor`.

5.4 Conclusions

This chapter presented the implementation of a petroleum contamination remediation selection ontology in both the Class Editor and Protégé/Dyna. The application of the Class Editor demonstrated its ontology management features, while the application of Protégé/Dyna demonstrated Dyna's dynamic knowledge modeling features.

6. DISCUSSION

During the implementation of the petroleum contamination remediation selection ontology in the Class Editor and Dyna, a number of interesting observations were made. While discussing these observations it is important to keep in mind that at the time of writing the tools have only been tested on a single domain of knowledge. Most of these points may be applicable when the tools are applied to other domains since the tools are based on the IMT, which has been used to model knowledge in a legal domain (Chan, 1995), a reforestation domain (Chan & Johnston, 1996), and a pipeline monitoring domain (Chan, 2000). This chapter discusses those observations.

6.1 Class Editor

The Class Editor was developed specifically to address weaknesses in the area of ontology management. Since the focus was ontology management, its modeling capabilities are limited. The Class Editor can create classes, data type properties, and “generic” relationships between classes. One modeling limitation is that the data type properties do not have any actual “type” information attached to them, so it is not possible to specify that sandThickness is a float, for example. A second modeling limitation is that object type properties cannot be modeled. The Class Editor does not support the instantiation of classes, thus making it impossible to create object type properties. These limitations resulted in a very incomplete static knowledge model of the petroleum remediation selection domain.

The Class Editor supports knowledge sharing and re-use by automatically managing author information and access rights in a peer-to-peer environment. The automatic entry of user data for knowledge components that are created by the user is a great benefit. Great discipline would be required by the user if they were left to manually

type in all their information for each knowledge element. The automatic enforcement of monotonicity ensures that knowledge elements aren't deleted and is a step in the right direction for supporting knowledge evolution. Further work is required to implement support for knowledge evolution.

6.2 Dyna

A number of observations can be drawn about Dyna from the process of modeling the petroleum contamination remediation selection ontology; some comments can be made about the modeling process itself. The following presents some observations about the modeling process, and some advantages and weaknesses of Dyna.

In modeling with the IMT, a distinction is made between a “task” and an “objective”. High-level operations are objectives (e.g. “SelectRemediationMethod”) and operations with specific actions are tasks (e.g. “MeasureSandThickness”). Tasks can be identified using keywords that describe simple actions, such as “set”, “measure”, or “add”; and keywords that indicate more general or higher-level action words, “determine” or “find” can be used for identifying objectives. However, when this distinction is applied for modeling the remediation domain, it became apparent that the general guideline that objectives are high-level operations and tasks are specific actions, cannot always be followed. For example, the original knowledge table in (Chen, 2001) listed the objective “Select Remediation Method” as containing the task “Determine Site Size”. Following the guideline, “Determine Site Size” is an objective as it is a high-level operation that contains a few different actions. The ideal solution would be to add the “Determine Site Size” objective to the list of tasks of “Select Remediation Method”, but this is not possible as the knowledge representation in Dyna does not support this. The

solution used was to create a task called “DetermineSiteSize”. The behaviour for such a task duplicates the objective’s prioritized tasks. This solution is not satisfactory because the task behaviour duplicates the objective’s prioritized tasks.

Using the correct names for tasks and objectives is very important as it greatly affects the intended meaning of the model. Many of the tasks are “Measure” tasks, such as “MeasureBenzeneConcentration”. These “Measure” tasks set the value of some property for an object. In the case of “MeasureBenzeneConcentration”, the SoilSamplingExperiment object's benzeneConcentration value is set. This is typical behaviour of a “setter” function, which is used to set the value of a member variable for a class. Therefore, instead of “MeasureBenzeneConcentration”, we could have “SetBenzeneConcentration”. However, “Set” is not a suitable prefix in this situation. “Set” and “Measure” have different meanings. “Measure” implies that we are performing some action to find a value. “Set”, on the other hand, implies that we already know the value, and are simply recording it.

The IMT has suggested that a knowledge engineer should model static knowledge first, and when the static knowledge model has been developed, then the dynamic knowledge is modeled. However, we have found this suggested sequence of modeling is not always practical. Often, the ontology author is uncertain what classes and properties are required. As a consequence, creating static knowledge first leads to errors, such as incorrect names for objects are specified and properties are attached to the wrong classes. From our experience, we found a more fruitful approach is to use an iterative process of creating some dynamic knowledge first, and using it as a guide to create the static knowledge required to ensure the dynamic knowledge can be implemented. This is

similar to the process of writing test cases first in Test Driven Development (TDD), described in Section 2.5.3. By creating tasks and objectives first, the user has to think about what functionality and objects are required, which greatly reduces the amount of errors when objects are specified.

The two main components that were most helpful in revealing missing information and modeling errors were (1) the fields for task behaviour input and output and (2) dynamic knowledge testing. Both components were helpful in revealing new classes and properties that need to be added to the static knowledge. The fields for task behaviour input and output can be used for prompting the user to think about the relationships between tasks and the objects manipulated by the tasks. The testing of dynamic knowledge can be used in identifying new classes and properties that need to be added to the static knowledge. The creation of test cases can also reveal logical errors in the task behaviour. For example, an erroneous “if statement” for classifying site size was as follows:

```
if (area < 1600 and volume < 25000)
{
    site.siteSize = “small”
}
else if (area >= 1600 and area < 2000 and volume >= 25000 and volume <= 30000)
{
    site.siteSize = “medium”
}
else if (area > 2000 and volume > 30000)
{
    site.siteSize = “large”
}
```

The logical error is in bold and underlined in the above behaviour. If **area** equals 2000, then the size of the site will not be set. Running the test cases for this task would generate a failure, thereby alerting us of the error in the model. When the error is

corrected, the test cases should all run successfully. A minor weakness of Dyna's testing suite is that it cannot perform interactions with the descriptive logic structures found in Protégé due to the fact that TBL does not support descriptive logic. The petroleum contamination remediation selection ontology did not require descriptive logic, so this was never an issue. Overall, dynamic ontology testing facilitated verification and enabled creation of a more complete knowledge model.

Not all of the dynamic knowledge in the original ontology could be modeled in Dyna. The main weakness of Dyna is its lack of support for the array data type. Many of the tasks perform operations on groups (or arrays) of objects. For example, one of the tasks measures the percentage of sand for each soil sample in the soil sampling experiment. This task requires a loop that iterates through one or more soil samples. However, since Dyna does not support arrays, only one soil sample can be measured. Therefore, in implementing the ontology model, it is assumed that the number of sampling points equals one (`numSamplingPoints=1`), and there is one soil sample (`soilSample1`). The lack of support for arrays also made it impossible to select multiple remediation methods. Under certain conditions more than one remediation method can be selected. However, without arrays, it is not possible to model those conditions.

Another weakness of Dyna is its visualization of Objectives and Tasks. Objectives and Tasks occur in a sequence or flow. Figure 33, shows the objective **DetermineSiteHydraulicConductivity** and its prioritized list of tasks. The prioritized list of tasks is a sequence. A visualization of the sequence of Tasks and Objectives can give the user a better understanding of the interactions between the different objectives and tasks. It is difficult for the user to picture the entire sequence of Tasks and

Objectives due to two problems with the visualization. First, the view of the sequence is limited to a single objective. In Figure 33 for example, **DetermineSoilType** is followed by **CalculateSoilTextureContributionFactor**. This sequence is missing tasks of the objective **DetermineSoilType**. The second problem is that there is no context for which the Objective takes place in. In Figure 33, the screen does not show that **DetermineSiteHydraulicConductivity** takes place within the context of the objective **SelectRemediationMethod** or that the objective **SoilSampling** and its tasks precede **DetermineSiteHydraulicConductivity**. Due to these two problems the user is required to picture the sequence of Tasks and Objectives in their mind, thus making the modeling process more difficult. A graphical view that can show the full sequence of objectives and tasks would be very beneficial.

Objective : DetermineSiteHydraulicConductivity

Name: DetermineSiteHydraulicConductivity

Documentation:

Tasks:

Task	Priority
DetermineSoilType	1
CalculateSoilTextureContributionFactor	2
CalculateHorizontalHydraulicConductivity	3
CalculateVerticalHydraulicConductivity	4
DetermineSoilHeterogeneity	5
DetermineSoilIsotropy	6
DetermineSoilHydraulicPermeability	7
DetermineSoilHydraulicConductivity	8
SetSiteHydraulicConductivity	9

Run Tests

Figure 33. Objective : DetermineSiteHydraulicConductivity

Maintaining the consistency between the static knowledge modeled in Protégé and the dynamic knowledge modeled in Dyna is difficult and error prone. The task

behaviour and task behaviour test cases are dependent on the classes and properties defined in the static knowledge model. The static knowledge cannot be modified or removed without having an effect on the dynamic knowledge. If a static knowledge element, such as a class, is deleted from Protégé, there is no notification to the user in Dyna that the knowledge element has been deleted. The user may not realize that the knowledge element has been removed until the test cases are run and error messages alert the user regarding an “undefined” knowledge element. An ontology management system capable of automatically maintaining consistency would be very beneficial. Such an ontology management system could alert the user to the effect their modifications on the static knowledge will have on the dynamic knowledge and possibly prevent any possible problems that will result from their changes.

6.3 Conclusions

This chapter discussed observations made during the implementation of the petroleum contamination remediation selection ontology in the Class Editor and Dyna. These observations include advantages and disadvantages of the implemented tools and also some comments on the modeling process itself.

7. CONCLUSIONS

Sharing and re-using knowledge can help reduce the high costs of constructing knowledge based systems. Technologies and ideas from the emerging Semantic Web can be useful for developing knowledge based systems. One of these technologies is the ontology, which can be defined as a shareable, computer processable model of a domain of knowledge. However, there are difficulties in constructing ontologies.

The objective of this work is to construct a suite of software tools for modeling and managing knowledge that would support ontology construction. Existing systems, including Protégé, KAON, and KMS were examined and the following four weaknesses were found: 1. Lack of support for an ontological engineering methodology, 2. Lack of support for dynamic knowledge modeling, 3. Lack of support for knowledge testing, and 4. Lack of support for ontology management. A tool that supports an ontological engineering methodology or technique can expedite the ontological engineering process, however, there are few tools that directly support an ontological engineering methodology. While most tools can support representation of static knowledge, there are significant difficulties when using these tools (including KMS) for modeling dynamic knowledge. Ontology testing can also help the knowledge engineer develop a more complete model of the domain. Existing ontology testing systems such as the OWL Unit Test Framework (Horridge, 2005a) and Chimaera's (McGuinness et al., 2000) test suite, evaluate the correctness and completeness of ontologies. Such testing tools are sufficient for testing static knowledge, but are not suitable for testing the interactions between behaviour and objects. To document, track, and distribute ontologies in such a distributed and heterogeneous environment, an ontology management system is needed.

Further weaknesses of the systems include security/trust vulnerabilities and insufficient support for knowledge evolution.

Two new tools were developed to address these four areas of weakness. Both tools were based on the Inferential Modeling Technique and the Knowledge Modeling System.

The Class Editor was created to address the area of ontology management. The Class Editor enables users to model static knowledge using a UML class diagram like visualization. The Class Editor is the front-end application for the ontology management system called Distributed Framework for Knowledge Evolution (DFKE) (Obst, 2006). Through the DFKE, the Class Editor can access ontology projects stored in a peer-to-peer network, automatically document authorship of ontology elements, and provide the ability to digitally sign ontology elements. Using the Class Editor, a basic static knowledge model of the petroleum contamination remediation selection domain was created. The incompleteness of the static knowledge model highlights the incompleteness of the Class Editor's modeling functions.

To address the areas of ontological engineering methodology, dynamic ontology modeling, and ontology testing, Dyna was created. Dyna extended Protégé with the ability to support IMT. The tool also enables task behaviour to be modeled in TBL, which enforces a formalized and structured representation of the task behaviour, so that tasks are more shareable and re-usable. The dynamic knowledge model can be tested by running test cases on the task behaviour. The dynamic knowledge is stored in XML and OWL formats, so that it may be shared and re-used by other systems.

Dyna has been applied for creating an ontology in the petroleum remediation selection domain. Modeling the petroleum contamination remediation selection ontology revealed some advantages and weaknesses of Dyna. During the modeling process, close attention must be paid to the naming of knowledge elements. Certain keywords are helpful in identifying tasks and objectives. The name of a task or objective could also change the intended semantics of the knowledge element. The most significant contribution of this thesis is in the area of dynamic knowledge testing. No other ontology tools support dynamic knowledge testing. Dynamic knowledge testing was very helpful in eliminating errors and creating a more complete knowledge model. The main weakness of Dyna is the lack of support for arrays in TBL, which made it difficult to model operations on groups of objects.

Dyna can be used to model the knowledge in other domains besides petroleum contamination remediation selection. The Inferential Modeling Technique (IMT) has been used to model a number of diverse domains including a legal domain (Chan, 1995), a reforestation domain (Chan & Johnston, 1996), and a pipeline monitoring domain (Chan, 2000). Since the IMT has been use to model such diverse domains and the IMT is the theoretical foundation on which Dyna is based, Dyna can be used to model the knowledge in other domains.

8. FUTURE WORK

Future directions that can be pursued are discussed as follows.

8.1 Knowledge Evolution in Dyna

Since Dyna does not currently support ontology management, a possible direction for future work is to integrate DFKE with Dyna. However, the integration is challenging in two aspects. The first challenge is the need to integrate the diverse knowledge representation schemes because DFKE and Dyna use different knowledge representation mechanisms for both static and dynamic knowledge. Dyna has been built on Protégé, which is implemented in Java, while DFKE is implemented in Python. The second challenge involves compatibility and communication between Dyna and DFKE due to the diverse platforms on which the two tools are built. Tackling these challenges remain on the future research agenda.

8.2 Semi-Automatic Ontology Generation

As previously discussed, a major problem in ontology development is the knowledge acquisition bottleneck. Reusing existing knowledge sources such as databases to create an ontology can greatly decrease the amount of work required. Ontology Learning techniques can be used to generate an ontology semi-automatically from a knowledge source, such as a database or text file. The process of generating an ontology from a database is called Data Correlation (Maedche & Staab, 2003) or Reverse Engineering (Meersman, 2001). In Data Correlation or Reverse Engineering, the data in the tables and relations between tables and in the database are reverse engineered into an ontology.

Reverse engineering a database into an ontology consists of three main steps. First, a class is created for each table in the database. Second, relations between tables are mapped to relations between classes. Third the rows of data in the tables are mapped to instances of the classes.

The effectiveness of generating an ontology from a database is highly dependent on the database schema. If the database schema has lots of relations, then most likely, the ontology generated will be of some use. However, if the database schema is flat, then the ontology generated will be flat also.

Data Correlation could be added to the Class Editor. The functions required for data correlation would be very similar to those already implemented in the Ontology Modeler. Protégé currently has plug-ins for performing data correlation in the same method described. However, such a method of data correlation is only useful for static knowledge. A direction for possible future research is to investigate how data correlation can be performed on dynamic knowledge.

8.3 Support Arrays in TBL

As described in the Discussion (Chapter 6), a weakness of Dyna is its lack of support for arrays. Without arrays, it was not possible to perform operations on groups of objects, such as soil samples. For example, calculations on multiple soil samples could not be performed without arrays. Efforts to implement arrays in Dyna that will work with the Protégé knowledge model have so far proved to be very challenging and unsuccessful. Support for arrays in TBL is fairly straight-forward to implement. The syntactic structures for arrays are added to the TBL grammar, which is then used to

generate the Lexer, Parser, and AST. The challenging part is interfacing the behaviour for an array with the objects in Protégé.

8.4 Visualizing Flow or Sequence of Tasks and Objectives

The objectives and tasks in the petroleum contamination remediation selection ontology occur in a flow or sequence that is specified by task priority. For example, in the objective **SelectRemediationMethod**, the task **DetermineSiteSize** is followed by the task **SoilSampling**. Visualizing the sequence in the Objective window is very difficult because (1) the view of the sequence is limited to a single objective and (2) there is no context for which the Objective takes place in. It could be very beneficial to have a visualization of the sequence because it may make it easier for the user to do the modeling as humans often prefer to do things visually. A process flow diagram could be adopted for this purpose. Figure 34 shows a possible flow diagram for modeling tasks and objectives. In this flow diagram, the white boxes along the top represent objectives. In this example, there are the objectives, “SelectRemediationMethod”, “DetermineSiteSize”, and “SoilSampling”. Beneath each objective are its tasks, represented by grey boxes, in order of priority. The first task in “SelectRemediationMethod” is “DetermineSiteSize”. A dashed line points from the “DetermineSiteSize” task in the “SelectRemediationMethod” column to the “MeasureSiteArea” task in the “DetermineSiteSize” column. The dashed line shows the flow from one task to another within an objective and also across to another objective.

Figure 34. Task/Objective Flow Diagram

8.5 Code Coverage Metrics

Code Coverage is a metric that can be used to measure the percentage of the total source code that has been tested successfully in a program. Combined with unit testing, code coverage can give the software engineer confidence that a certain percentage of the code has been tested and is functioning as expected; the end result of which is improved software quality.

As discussed in Section 5.3.2, the approach to testing the dynamic knowledge of the petroleum contamination remediation selection ontology involved creating a test case for each unique path through the task behaviour. The percentage of code coverage could be calculated as the TBL interpreter traverses the code for each path. After the TBL interpreter has finished running all the tests, the calculated code coverage percentage

could be output to a screen. This is an informal metric that would give the ontology author an idea of how much of the task behaviour has been tested, which may help in improving the quality of the final ontology.

8.6 Top-Level Ontology

The petroleum contamination remediation selection ontology created in Dyna and Class Editor contains some concepts that are very general (e.g. water) and can be found in other domains. A type of ontology that defines concepts that are common to many domains is called a top-level ontology. Examples of top-level ontologies are Cyc²⁵ and Standard Upper Ontology²⁶. Top-level ontologies are meant to be re-used by other ontologies, such as domain specific ontologies, thus reducing effort required to implement them. Although the application ontology is already complete, it may be a worthwhile exercise from a learning perspective, to link some of the more general concepts in the petroleum contamination remediation selection ontology to the general concepts in a top-level ontology.

8.7 Evaluation of Dyna

A preliminary evaluation should be performed by comparing the completeness of the petroleum contamination remediation selection ontology created in Dyna to the same ontology created in KMS. Further research is required to find an appropriate formal evaluation method for assessing the ontology tool.

8.8 Miscellaneous Ideas

There are many future directions in which this work could proceed. This section describes some ideas that may be very useful, but have not been fully thought through.

²⁵ <http://www.cyc.com/>

²⁶ <http://suo.ieee.org/>

8.8.1 Uncertainty Modeling

In the real world, we are not always totally certain that the information from which we base our decisions is accurate and complete. Noise (extra information) has an effect on our certainty that the information is accurate and complete. The degree to which we are certain or uncertain of the accuracy or completeness of information, has an effect on our decision making. Modeling uncertainty could be very beneficial as it would make a Decision Support System (DSS) behave more realistically. However, the standard ontology representation formats are not capable of representing and reasoning with uncertain information.

Bayesian Networks (BN) can be used to represent knowledge and uncertainty. BN is a directed acyclic graph, whose nodes represent various events and the directed edges represent the dependencies between events. Each event has a conditional probability table. The probability of an event is in the interval $[0, 1]$, where 0 means the event is not certain at all and 1 means that the event is certain. The probability of an event can be calculated given the probabilities of the event's dependencies (Jensen, 1996).

The knowledge representation capability of BN is quite limited. Both (Koller & Pfeffer, 1998) and (Yang & Calmet, 2005) have combined ontologies with Bayesian Networks. The result is an ontology that can represent complex domains and uncertainty. (Yang, 2006) applied ontology-Bayesian Network (OntoBayes) to create a DSS for insurance and natural disaster management.

Some of the ideas of combining ontologies and BN can be applied to modeling uncertainty in Dynamic Knowledge. The concepts between BN echo some of the ideas in

an Abstract Syntax Tree (AST), which is used by Dyna for storing Task Behaviour for processing. Like BN, the AST consists of nodes and edges. As previously discussed, the AST is processed using a depth-first traversal of the nodes. The AST for the simple statement **a = b + c** is:

Using a depth-first traversal, the nodes in the above AST would be processed in the order: =, a, +, b, c. A more likely structure that would contain uncertainty is an “if statement”. For example:

```

if (x < y)
{
    z = 1
}

```

The AST for the above “if-statement” would be:

The nodes for the AST of the “if-statement” would be processed in the order (assuming the statement is true): if, < x, y, =, z, 1. With the existing technique for modeling dynamic knowledge, the easiest method of modeling uncertainty would be to introduce a TBL statement for specifying uncertainty. Using the “if statement” as an example:

```

if (x < y) : uncertainty = u
{
    z = 1
}

```

}

In the above situation, **uncertainty** is the keyword, and states that the condition has an uncertainty factor of *u*. If the uncertainty is above a certain amount, then the body of the “if statement” ($z = 1$), will never be executed, regardless of the result of the “if statement” condition ($x < y$). The AST for the if statement with uncertainty would be as follows:

```

      if
    /  |  \
unc. <  =
 |  / \  / \
u   x  y z  1

```

The value for **u** would be calculated using the BN. If *u* is below a threshold level, then the rest of the if statement is ignored. In other words, the nodes would be processed in the following order: *u*, uncertainty, if. Further research is required to find the feasibility and practicality of uncertainty modeling with TBL.

8.8.2 Semi-Automatic Application Generation

Ontologies can be used as a source to automatically generate application program code. Protégé can export OWL to Java using the JSave plug-in²⁷. JSave converts OWL classes to Java classes and OWL properties to Java attributes of classes. The generated Java code simply contains classes and class attributes; there are no functions or methods. There are similarities between dynamic knowledge and functions or methods found in programs. The dynamic knowledge in Dyna could be used to generate Java methods. The TBL for defining task behaviour is represented as an AST for processing. The current set of AST nodes enable the dynamic knowledge to interact with the static knowledge. A second set of AST nodes could be created that instead of manipulating

²⁷ <http://protegewiki.stanford.edu/index.php/JSave>

dynamic and static knowledge, they output Java representations of the TBL statements. The process of generating Java methods from dynamic knowledge would be semi-automatic. Methods in Java must belong to a class, however, dynamic knowledge is not attached to any specific class. User input is required to choose a class for the dynamic knowledge, thus making the process semi-automatic. Using such a system would greatly reduce the time required to develop knowledge based software.

8.8.3 Time and Parallel Processes

Some domains require tasks to be performed at a specific time or in parallel. The ability to specify the exact time that a task should be executed or that multiple tasks are to be executed in parallel is a very powerful feature that is missing in Dyna. The petroleum contamination remediation selection ontology did not have any processes that had to be executed at a particular time or in parallel, so this feature was never considered for implementation. However, some domains may require such processes. Tasks can be specified that they are to be executed in parallel in the Objective's task priority list. Two or more tasks that have the same priority, can mean that the tasks are to be executed in parallel. The TBL interpreter is limited to running one task at a time, so the tests would not be performed in parallel. The time that a task is to be executed cannot be specified in Dyna. E-LOTOS is a language for the specification of time and parallel processes (Huecas, et al., 1999). A study of E-LOTOS may provide insight into how TBL could be extended to support time and parallel processes.

8.9 Conclusions

This chapter discussed a number of future directions for this work. One possible direction for future is to integrate the Distributed Framework for Knowledge Evolution

(DFKE) into Dyna, thus enabling it to support ontology management. A second possible direction for future work is to use ontology learning techniques to semi-automatically generate an ontology from a knowledge source, such as a database or text file. As described in the Discussion (Chapter 6), a weakness of Dyna is its lack of support for arrays. Implementing support for arrays in Dyna is another direction for future work. Objectives and tasks occur in a sequence, which is difficult to visualize in the current Dyna user interface. A fourth direction for future work is to develop a user interface that uses a process flow diagram for visualizing the sequences of objectives and tasks. A preliminary evaluation should also be performed by comparing the completeness of the petroleum contamination remediation selection ontology created in Dyna to the same ontology created in KMS.

There are also some ideas that may be very useful, but have not been fully thought through. These ideas include uncertainty modeling, semi-automatic application generation, and modeling time and parallel processes.

REFERENCES

- Antoniou, G., & van Harmelen, F. (2004). "A Semantic Web Primer", The MIT Press, 2004, <http://www.books24x7.com>
- Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The Semantic Web, *Scientific American*, May 2001.
- Chan, A. S., Martins, S. B., Coleman, R. W., Bosworth, H. B., Oddone, E. Z., Shlipak, M. G., Tu, S. W., Musen, M. A., Hoffman, B. B., & Goldstein, M. K. (2005). Post-fielding Surveillance of a Guideline-Based Decision Support System. K. Henriksen, J. B. Battles, E. S. Marks and D. I. Lewin (Eds). *Advances in Patient Safety: From Research to Implementation*. Vol. 1. Research Findings AHRQ Pub Numb 05-0021-1. Rockville, MD 20850, AHRQ. 1: 331-339. 2005.
- Chan, C. W. (2004). "From Knowledge Modeling to Ontology Construction", *Int. Journal of Software Engineering and Knowledge Engineering*, 14(6), Dec 2004.
- Chan, C. W., Huang, G., & Hu, Z. (2002). Development of an Expert Decision Support System for Selection of Remediation Technologies for Petroleum-Contaminated Sites, *Final Report for Petroleum Technology Research Center, Project No. 00-03-018*, University of Regina, Regina, SK, Canada, May 2002.
- Chan, C. W. (2000). A Knowledge Modeling Technique and Industrial Applications, invited book chapter, Chapter 34 in Volume 4 of C. Leondes, Ed., *Knowledge-Based Systems Techniques and Applications*, 4 volumes (Academic Press, USA, 2000).
- Chan, C. W. & Johnston, M. (1996). Knowledge Modeling for Constructing an Expert System to Support Reforestation Decisions, In *Knowledge-Based Systems Journal*, 9, 1996, pp. 41-59.
- Chan, C. W. (1995). Development and Application of A Knowledge Modeling Technique, In *Journal of Experimental and Theoretical Artificial Intelligence*, 7(2), 1995, pp. 217-236.
- Chen, L. L. (2001). "Construction of an ontology for the domain of selecting remediation techniques for petroleum contaminated sites", M.A.Sc. Thesis, University of Regina, Canada, 2001.
- Cranefield, S., Pan, J., & Purvis, M. (2003). A UML Ontology and Derived Content Language for a Travel Booking Scenario, *OAS 2003*, pp. 55-62.
- Das, A., Wu, W., & McGuinness, D. (2001). "Industrial Strength Ontology Management", in *Proceedings of SWWS'01, The First Semantic Web Working Symposium*, Aug, 2001, pp. 17-38.

Fensel, D., Harmelen, F. van, Ding, Y., Klein, M., Akkermans, H., Broekstra, J., Kampman, A., Meer, J. van der, Studer, R., Sure, Y., Davies, J., Duke, A., Engels, R., Iosif, V., Kiryakov, A., Lau, T., Reimer, U., & Horrocks, I. (2002). On-To-Knowledge in a Nutshell, *IEEE Computer*, 2002.

Gabel, T., Sure, Y., & Voelker, J. (2004). KAON- An Overview, April 7, 2004.

Gal, A., Eyal, A., Roitman, H., Jamil, H., Anaby-Tavor, A., Modica, G., & Enan, M. (2006). OntoBuilder, <http://iew3.technion.ac.il/OntoBuilder/>

Gasevic, D., Djuric, D., & Devedzic, V. (2005). Ontology Modeling and MDA, In *Journal of Object Technology*, 4(1), 2005, pp. 109-128.

Gennari, J. H., Musen, M. A., Fergerson, R. W., Grosso, W. E., Crubezy, M., Eriksson, H., Noy, N. F., Tu, S. W. (2003). The evolution of Protégé: an environment for knowledge-based systems development, In *International Journal of Human-Computer Studies*, 58(1), January 2003, pp. 89-123.

Gomez-Perez, A., Fernandez-Lopez, M., & Corcho, O. (2005). Ontological Engineering: with examples from the areas of Knowledge Management, e-Commerce, and the Semantic Web. Springer, 2005, pp. 112, 179.

Gomez-Perez, A., Fernandez-Lopez, M., & Corcho, O. (2003). WebODE Ontology Engineering Platform, <http://webode.dia.fi.upm.es/WebODEWeb/index.html>

Gruber, T. (1993). Towards Principles for the Design of Ontologies Used for Knowledge Sharing, In *Guarino & Poli (eds): Formal Ontology in Conceptual Analysis & Knowledge Representation*, Padova, Italy: Kluwer, 1993.

Guarino, N. (1998). Formal Ontology in Information Systems, In *Guarino (ed) 1st International Conference on Formal Ontology in Information Systems (FOIS'98)*, Trento, Italy, IOS Press, Amsterdam, 1998, pp. 3-15.

Happel, H.-J. & Stojanovic, L. (2006). "Ontoprocess – a prototype for semantic business process verification using SWRL", *Proceedings of the 3rd European Semantic Web Conference (ESWC2006)*, Budva, Montenegro, June 11 – 14, 2006.

Harrison, R. & Chan, C. W. (2008). "A Dynamic Knowledge Modeler", Accepted for *Artificial Intelligence for Engineering Design, Analysis and Manufacturing: Special Issue on Developing and Using Engineering Ontologies*, Winter 2008/9, Volume 23, Number 1, Cambridge University Press, 2008.

Harrison, R. & Chan, C. W. (2007a). "A Tool for Dynamic Knowledge Modeling", In *Proceedings of the 6th International Conference on Cognitive Informatics 2007 (ICCI'07)*, Lake Tahoe, California, USA, Aug. 6-8, 2007.

Harrison, R. & Chan, C. W. (2007b). "Tools for Industrial Knowledge Modeling", In *Proceedings of the 20th Annual Canadian Conference on Electrical and Computer Engineering (CCECE'07)*, Vancouver, BC, Canada, April 22-26, 2007.

Harrison, R. & Chan, C.W. (2005a). "Distributed Ontology Management System", *Proceedings of the 18th Annual Canadian Conference on Electrical and Computer Engineering (CCECE 2005)*, Saskatoon, Saskatchewan, Canada, May 1-4, 2005, p. 661-664.

Harrison, R. & Chan, C. W. (2005b). "Implementation of an Application Ontology: A Comparison of Two Tools", *Artificial Intelligence Applications and Innovations II: Second IFIP TC12 and WG12 Conference on Artificial Intelligence Applications and Innovations (AIAI-2005)*, Beijing, China, Sept. 7-9, 2005, pp. 131-143.

Harrison, R., Obst, D., & Chan, C.W. (2005c). "Design of an Ontology Management Framework", In *Proceedings of the 4th International Conference on Cognitive Informatics 2005 (ICCI'05)*, University of California, Irvine, USA, Aug. 8-10, 2005, pp. 260-268.

Horridge, M. (2005a). OWL Unit Test Framework, <http://www.co-ode.org/downloads/owlunittest/>

Horridge, M. (2005b). Protégé OWLViz, <http://www.co-ode.org/downloads/owlviz/co-ode-index.php>

Huang, Z. & Stuckenschmidt, H. (2005). "Reasoning with Multi-Version Ontologies: a Temporal Logic Approach", *Proceedings of the 2005 International Semantic Web Conference*.

Huecas, G., Llana-Diaz, L. F., Robles, T., & Verdejo, A. (1999). E-LOTOS: An Overview, In M. A. Marsan, J. Quemada, T. Robles, and M. Silva, editors, *Formal Methods and Telecommunications (FM&T'99)*, pp. 94-102. Prensas Universitarias de Zaragoza, Sept. 1999.

Janzen, D. & Saiedian, H. (2005). Test Driven Development: Concepts, Taxonomy, and Future Direction, *IEEE Computer*, September 2005.

Jensen, F. V. (1996). "An Introduction to Bayesian Networks", New York: Springer, 1996, pp. 14-20.

Kalyanpur, A., Hashmi, N., Golbeck, J., & Parsia, B. (2004). "Lifecycle of a Casual Web Ontology Development Process", WWW2004 Workshop.

Kivela, A. & Hyvonen, E. (2002). "Ontological Theories for the Semantic Web", *Semantic Web Kick-Off in Finland*, pp. 111-136, May 2002.

Klein, M. & Fensel, D. (2001). "Ontology Versioning on the Semantic Web", Proceedings of SWWS '01, The First Semantic Web Working Symposium, pp. 75-92, August 2001.

Koivunen, M. J. & Miller, E. (2001). W3C Semantic Web Activity, <http://www.w3.org/2001/12/semweb-fin/w3csw>

Koller, D. & Pfeffer, A. (1998). "Probabilistic Frame-Based Systems", In Proceedings of the 5th National Conference on Artificial Intelligence (AAAI-98), pp.580-587, Madison, WI, July, 1998.

Ma, L., Xie, G., Yang, Y., & Zhang, L. (2006). IBM Integrated Ontology Development Toolkit, <http://www.alphaworks.ibm.com/tech/semanticstk>

Maedche, A. & Staab, S. (2003). "Ontology Learning", In: Staab, S., Studer, R., eds.: Handbook of Ontologies in Information Systems, Springer Verlag, pp 173-190, 2003.

McGuinness, D., Fikes, R., Rice, J., & Wilder, S. (2000). An Environment for Merging and Testing Large Ontologies, In *Proceedings of KR 2000*, pp. 485-493.

Meersman, R. (2001). "Ontologies and Databases: More than a fleeting resemblance", V.U.B. Star Lab, 2001.

Menezes, A. J., van Oorschot, P. C., & Vanstone, S. A. (1997). "Handbook of Applied Cryptography", pp. 426.

Norvig, P., (2006). AAAI: Google and the Semantic, Satanic, Romantic Web, http://www.nodalpoint.org/2006/07/19/aaai_google_and_the_semantic_satanic_romantic_web

Noy, N. F., Chugh, A., Liu, W., & Musen, M. A. (2006). "A Framework for Ontology Evolution in Collaborative Environments", In Proceedings of 5th International Semantic Web Conference, Athens, GA, USA, 2006.

Obst, D. (2006). "Distributed Framework for Knowledge Evolution", Presentation at *University of Regina Grad. Student Conf.*, Regina, SK, Canada, 2006.

Obst, D. & Chan, C.W. (2005). "Towards a Framework for Ontology Evolution", accepted for 18th Annual Canadian Conference on Electrical and Computer Engineering (CCECE05), Saskatoon, Saskatchewan, Canada, May 1-4, 2005.

Pressman, R. S. (2004). Software Engineering: A Practitioner's Approach. McGraw-Hill, 2004.

Richardson, M., Agrawal, R., & Domingos, P. (2003). "Trust Management for the Semantic Web". In *Proceedings of International Semantic Web Conference (ISWC'2003)*, 2003.

Ulieru, M., Hadzic, M., & Chang, E. (2006). Soft Computing Agents for E-Health in Application to the Research and Control of Unknown Diseases. In *Information Sciences*, 176 (2006), Elsevier, pp. 1190-1214.

Volz, R. & Oberle, D. (2003). "KAON SERVER - A Semantic Web Management System", May 2003.

Yang, Y. & Calmet, J. (2006). "From the OntoBayes Model to a Service Oriented Decision Support System". In *Proceedings of International Conference on Computational Intelligence for Modelling, Control and Automation (CIMCA'06)*, IEEE, November, 2006, Sydney, Australia.

Yang, Y. & Calmet, J. (2005). "OntoBayes: An Ontology-Driven Uncertainty Model". In *Proceeding of the International Conference on Intelligent Agents, Web Technologies and Internet Commerce (IAWTIC'05)*, IEEE Computer Society, vol. I, pp. 457-464.

Yildiz, B. (2006). "Ontology Evolution and Versioning: The State of the Art", Asgaard-TR-2006-3, October 2006.

Appendices

Appendix I

Petroleum Contamination Remediation Selection Ontology : Static Knowledge created with Protégé

This appendix contains knowledge tables for the static knowledge of the petroleum contamination remediation selection ontology created with Protégé.

Class	Parent Class	Properties	Individuals
Contaminant		contaminantPhase owl:oneOf{ "free" "residual" "free and residual" } [functional] contaminationLevel owl:oneOf{ "low" "medium" "high" "extremely high" } [functional] chemicalConcentration float [functional] measuredContaminantValue float [functional] contaminationWeight float [functional] resideIn Media	
ChemicalContaminant	Contaminant		toluene xylenes toluene benzene
NonOrganicChemicalContaminant	ChemicalContaminant		
OrganicChemicalContaminant	ChemicalContaminant		
Experiment			
SamplingExperiment	Experiment	mediaType owl:oneOf{ "soil" "groundwater" "soilgroundwater" } [functional] numSamplingPoints int [functional] samplingMethodName owl:oneOf{ "device soil sampling" "direct push probes" "hollow-stem augers device" "continuous split-spoon sampling device" } [functional]	
AirSamplingExperiment	SamplingExperiment		
SoilSamplingExperiment	SamplingExperiment	contaminantPhase owl:oneOf{ "free" "residual" "free and residual" } [functional] ethylBenzeneConcentration float [functional] tolueneConcentration float [functional]	soilSamplingExp

		contaminationLevel owl:oneOf{"low" "medium" "high" "extremely high"} [functional] hydraulicConductivity owl:oneOf{"simple" "complex"} [functional] moisturePerctange float [functional] tphConcentration float [functional] porosity float [functional] contaminationConcentration float [functional] xylenesConcentration float [functional] benzeneConcentration float [functional] sample Media soilSample SoilSample contaminantType ChemicalContaminant [functional] remediationMethod PetroleumContaminationRemediation [functional] substanceContainedInSoil ChemicalContaminant [functional] samplingSite Media [functional]	
WaterSamplingExperiment	SamplingExperiment		
Gas			
Media		siteArea int [functional] depthOfSite float [functional] siteVolume int [functional] hydraulicConductivity owl:oneOf{"simple" "complex"} [functional] siteSize owl:oneOf{"small" "medium" "large"} [functional]	site
Soil	Media	horizontalHydraulicConductivity float [functional] verticalHydraulicConductivity float [functional] soilHeterogeneity owl:oneOf{"homogeneous" "heterogeneous" "extremely heterogeneous"} [functional] soilThickness float [functional] soilDepth float [functional] isotropy owl:oneOf{"isotropic" "anisotropic"} [functional] soilType owl:oneOf{"sand" "clay" "sandy loam" "clay loam" "sandy clay" "silt" "loamy sand" "loam" "sandy clay loam" "silty clay loam" "silty clay" "silt loam"} [functional] soilHydraulicPermeability owl:oneOf{"extremely low" "low" "medium" "high" "extremely high"} [functional]	soil
SoilGroundWater	Soil		

	GroundWater		
Water	Media	pH float [functional]	
GroundWater	Water	waterType owl:oneOf{ "potable water" "non-potable water" } [functional]	
Remediation		efficiency float [functional]	
		reference string [functional]	
		cost float [functional]	
		remediationName string [functional]	
		dateOfUpdate date [functional]	
		siteMediaCondition Soil or GroundWater or SoilGroundWater [functional]	
AirPollutionRemediation	Remediation		
PetroleumContaminationRemediation	Remediation		solventExtraction
			excavation
			chemicalOxidationSystem
			compostingTechnology
			bioremediation
			airSparging
			thermalDesorption
			soilVaporExtraction
ExSituRemediation	PetroleumContaminationRemediation	exsituRemediationName string [functional]	desorptionSystemProcess
InSituRemediation	PetroleumContaminationRemediation	insituRemediationName string [functional]	exSituChemicalExtraction
			inSituSlurryWalls
			inSituSteamExtractionTreatment
			inSituChemicalExtraction
			inSituLeachingAndChemical
WaterPollutionRemediation	Remediation		inSituPassiveRemediation
Sample			
SoilSample	Sample	clayLoamThickness float [functional]	soilSample1
		measuredSiltPercentage float [functional]	
		calculatedSiltPercentage float [functional]	
		siltyClayThickness float [functional]	
		horizontalHydraulicConductivity float [functional]	
		sandThickness float [functional]	

		verticalHydraulicConductivity float [functional]	
		siltyLoamThickness float [functional]	
		hydraulicConductivity owl:oneOf{"simple" "complex"} [functional]	
		measuredSandPercentage float [functional]	
		soilHeterogeneity owl:oneOf{"homogeneous" "heterogeneous" "extremely heterogeneous"} [functional]	
		calculatedClayPercentage float [functional]	
		siltThickness float [functional]	
		loamySandThickness float [functional]	
		gravelPercentage float [functional]	
		siltyClayLoamThickness float [functional]	
		soilTextureContributionFactor float [functional]	
		sandyClayLoamThickness float [functional]	
		sampleDepth float [functional]	
		sandyLoamThickness float [functional]	
		isotropy owl:oneOf{"isotropic" "anisotropic"} [functional]	
		soilType owl:oneOf{"sand" "clay" "sandy loam" "clay loam" "sandy clay" "silt" "loamy sand" "loam" "sandy clay loam" "silty clay loam" "silty clay" "silt loam"} [functional]	
		measuredClayPercentage float [functional]	
		measuredGravelPercentage float [functional]	
		loamThickness float [functional]	
		clayThickness float [functional]	
		soilHydraulicPermeability owl:oneOf{"extremely low" "low" "medium" "high" "extremely high"} [functional]	
		sandyClayThickness float [functional]	
		calculatedSandPercentage float [functional]	
		clayPercentage float [functional]	
		sandPercentage float [functional]	
		siltPercentage float [functional]	
Standard			
AirPollutionStandard	Standard		
PetroleumContaminationStandard	Standard	standardContaminantConcentrationInSoilAndGroundWater owl:oneOf{"residential" "industrial" "agricultural" "potable water" "non-potable water"} [functional]	

		standardContaminantConcentrationInGroundWater owl:oneOf{ "potable water" "non-potable water" } [functional]	
		standardContaminantConcentrationInSoil owl:oneOf{ "residual" "industrial" "agricultural" } [functional]	
AlbertaStandard	PetroleumContaminationStandard		
SaskatchewanStandard	PetroleumContaminationStandard	skEthylBenzene float [functional]	skStandard
		skToluene float [functional]	
		skBenzene float [functional]	
		skXylenes float [functional]	
		skTotalHydrocarbon float [functional]	
USEPAStandard	PetroleumContaminationStandard		
WaterPollutionStandard	Standard		

Appendix II

Petroleum Contamination Remediation Selection Ontology : Dynamic Knowledge created with Dyna

This appendix contains knowledge tables for the dynamic knowledge for the petroleum contamination remediation selection ontology created with Dyna.

1. Objectives

#	Objective	Tasks
0	SoilSampling	1. SetSamplingMethod 2. SetNumSamplingPoints 3. CollectSoilSamples 4. MeasureMoisturePercentage 5. DetermineMedia 6. MeasureSoilThickness 7. CalculateSamplingDepth 8. MeasureSoilTypePercentages 9. CalculateSoilTypePercentages 10. MeasureContaminantConcentration
1	DetermineSiteHydraulicConductivity	1. DetermineSoilType 2. CalculateSoilTextureContributionFactor 3. CalculateHorizontalHydraulicConductivity 4. CalculateVerticalHydraulicConductivity 5. DetermineSoilHeterogeneity 6. DetermineSoilIsotropy 7. DetermineSoilHydraulicPermeability 8. DetermineSoilHydraulicConductivity 9. SetSiteHydraulicConductivity
2	DetermineSoilHeterogeneity	1. SetSoilHeterogeneity
3	DetermineMedia	1. SetMediaType
4	DetermineSoilType	1. SetSoilType
5	DetermineSiteSize	1. MeasureSiteArea 2. MeasureSiteVolume 3. SetSiteSize
6	DetermineContaminationLevel	1. CalcWeightedNormalizedBenzeneConcentration 2. CalcWeightedNormalizedTolueneConcentration 3. CalcWeightedNormalizedEthylBenzeneConcentration 4. CalcWeightedNormalizedXylenesConcentration 5. SumPollutantConcentrationValues 6. SetContaminationLevel
7	SelectRemediationMethod	1. DetermineSiteSize 2. SoilSampling 3. DetermineSiteHydraulicConductivity 4. DetermineContaminantionLevel 5. DetermineContaminationPhase 6. SetRemediationMethod
8	DetermineSoilHydraulicPermeability	1. SetSoilHydraulicPermeability
9	DetermineSoilIsotropy	1. SetSoilIsotropy

10	DetermineSoilHydraulicConductivity	1. SetSoilHydraulicConductivity
11	DetermineContaminationPhase	1. SetContaminationPhase

2. Tasks

#	0				
Name	SetNumSamplingPoints				
Behaviour	soilSamplingExp.numSamplingPoints = numSamplingPoints				
Input(s)	int numSamplingPoints				
Output					
Objects	SoilSamplingExperiment soilSamplingExp				
Tests	<table> <tr> <td>0. testSetNumSamplingPoints</td><td>SetNumSamplingPoints(1)</td></tr> <tr> <td></td><td>assert(soilSamplingExp.numSamplingPoints == 1)</td></tr> </table>	0. testSetNumSamplingPoints	SetNumSamplingPoints(1)		assert(soilSamplingExp.numSamplingPoints == 1)
0. testSetNumSamplingPoints	SetNumSamplingPoints(1)				
	assert(soilSamplingExp.numSamplingPoints == 1)				
#	1				
Name	DetermineSiteHydraulicConductivity				
Behaviour	DetermineSoilType(soilSample1.calculatedSandPercentage, soilSample1.calculatedSiltPercentage, soilSample1.calculatedClayPercentage) CalculateSoilTextureContributionFactor() CalculateHorizontalHydraulicConductivity() CalculateVerticalHydraulicConductivity() DetermineSoilHeterogeneity(soilSample1.soilTextureContributionFactor) DetermineSoilIsotropy(soilSample1.horizontalHydraulicConductivity, soilSample1.verticalHydraulicConductivity) DetermineSoilHydraulicPermeability() DetermineSoilHydraulicConductivity(soilSample1.soilIsotropy, soilSample1.soilHeterogeneity) SetSiteHydraulicConductivity(soilSample1.soilHeterogeneity, soilSample1.soilIsotropy, soilSample1.soilPermeability)				
Input(s)					
Output					
Objects	SoilSample soilSample1				
Tests	□				
#	2				
Name	SoilSampling				
Behaviour	SetSamplingMethod(samplingMethod) SetNumSamplingPoints(numSamplingPoints) CollectSoilSamples() MeasureMoisturePercentage(moisturePercentage) string mediaType mediaType = DetermineMedia() MeasureSoilThickness(sandThickness, sandyClayThickness, sandyLoamThickness, sandyClayLoamThickness, siltThickness, siltyClayThickness, siltyLoamThickness, siltyClayLoamThickness, clayThickness, clayLoamThickness, loamThickness, loamySandThickness) CalculateSamplingDepth(sandThickness, sandyClayThickness, sandyLoamThickness, sandyClayLoamThickness, siltThickness, siltyClayThickness, siltyLoamThickness, siltyClayLoamThickness, clayThickness, clayLoamThickness, loamThickness, loamySandThickness)				

	MeasureSoilTypePercentages(sandPercentage, siltPercentage, clayPercentage, gravelPercentage) CalculateSoilTypePercentages(soilSample1.measuredSandPercentage, soilSample1.measuredSiltPercentage, soilSample1.measuredClayPercentage) MeasureContaminantConcentration()		
Input(s)	string samplingMethod, int numSamplingPoints, float moisturePercentage, float sandThickness, float sandyClayThickness, float sandyLoamThickness, float sandyClayLoamThickness, float siltThickness, float siltyLoamThickness, float siltyClayLoamThickness, float clayThickness, float clayLoamThickness, float loamThickness, float loamySandThickness, float sandPercentage, float siltPercentage, float clayPercentage, float gravelPercentage		
Output			
Objects	SoilSample soilSample1		
Tests	□		
#	3		
Name	SetSamplingMethod		
Behaviour	soilSamplingExp.samplingMethodName = samplingMethodName		
Input(s)	string samplingMethodName		
Output			
Objects	SoilSamplingExperiment soilSamplingExp		
Tests	0. testSelectSamplingMethod	SetSamplingMethod("device soil sampling") assert(soilSamplingExp.samplingMethodName == "device soil sampling")	
#	4		
Name	CalculateSamplingDepth		
Behaviour	soilSample1.sampleDepth = clayThickness + siltyClayThickness + sandyClayThickness + siltyClayLoamThickness + clayLoamThickness + sandyClayLoamThickness + siltyLoamThickness + siltThickness + loamThickness + sandyLoamThickness + loamySandThickness + sandThickness		
Input(s)	float sandThickness, float sandyClayThickness, float sandyLoamThickness, float sandyClayLoamThickness, float siltThickness, float siltyClayThickness, float siltyLoamThickness, float siltyClayLoamThickness, float clayThickness, float clayLoamThickness, float loamThickness, float loamySandThickness		
Output			
Objects	SoilSample soilSample1		
Tests	0. testCalculateSamplingDepth	CalculateSamplingDepth(15.0, 14.0, 13.0, 12.0, 11.0, 10.0, 9.0, 8.0, 7.0, 6.0, 5.0, 4.0) assert(soilSample1.sampleDepth == 114.0)	
#	5		
Name	DetermineSoilType		
Behaviour	SetSoilType(calcSandPercentage, calcSiltPercentage, calcClayPercentage)		
Input(s)	float calcSandPercentage, float calcSiltPercentage, float calcClayPercentage		
Output			
Objects	SoilSample soilSample1		
Tests	0. testClay	CalculateSoilTypePercentages(0.05, 0.05, 0.8) DetermineSoilType(soilSample1.calculatedSandPercentage, soilSample1.calculatedSiltPercentage, soilSample1.calculatedClayPercentage) assert(soilSample1.soilType == "clay")	
#	6		
Name	MeasureSoilThickness		
Behaviour	MeasureSandThickness(sandThickness) MeasureSandyClayThickness(sandyClayThickness) MeasureSandyLoamThickness(sandyLoamThickness) MeasureSandyClayLoamThickness(sandyClayLoamThickness)		

	MeasureSiltThickness(siltThickness) MeasureSiltyClayThickness(siltyClayThickness) MeasureSiltyLoamThickness(siltyLoamThickness) MeasureSiltyClayLoamThickness(siltyClayLoamThickness) MeasureClayThickness(clayThickness) MeasureClayLoamThickness(clayLoamThickness) MeasureLoamThickness(loamThickness) MeasureLoamySandThickness(loamySandThickness)		
Input(s)	float sandThickness, float sandyClayThickness, float sandyLoamThickness, float sandyClayLoamThickness, float siltThickness, float siltyClayThickness, float siltyLoamThickness, float siltyClayLoamThickness, float clayThickness, float clayLoamThickness, float loamThickness, float loamySandThickness		
Output			
Objects			
Tests	0. testMeasureSoilThickness	float sandThickness sandThickness = 15.0 float sandyClayThickness sandyClayThickness = 14.0 float sandyLoamThickness sandyLoamThickness = 13.0 float sandyClayLoamThickness sandyClayLoamThickness = 12.0 float siltThickness siltThickness = 11.0 float siltyClayThickness siltyClayThickness = 10.0 float siltyLoamThickness siltyLoamThickness = 9.0 float siltyClayLoamThickness siltyClayLoamThickness = 8.0 float clayThickness clayThickness = 7.0 float clayLoamThickness clayLoamThickness = 6.0 float loamThickness loamThickness = 5.0 float loamySandThickness loamySandThickness = 4.0 # Run test MeasureSoilThickness(sandThickness, sandyClayThickness, sandyLoamThickness, sandyClayLoamThickness, siltThickness, siltyClayThickness, siltyLoamThickness, siltyClayLoamThickness, clayThickness, clayLoamThickness, loamThickness, loamySandThickness) # Verify assert(soilSample1.sandThickness == sandThickness and soilSample1.sandyClayThickness == sandyClayThickness and soilSample1.sandyLoamThickness == sandyLoamThickness and soilSample1.sandyClayLoamThickness == sandyClayLoamThickness and soilSample1.siltThickness == siltThickness and soilSample1.siltyClayThickness == siltyClayThickness and soilSample1.siltyLoamThickness == siltyLoamThickness and soilSample1.siltyClayLoamThickness == siltyClayLoamThickness and soilSample1.clayThickness == clayThickness and soilSample1.clayLoamThickness == clayLoamThickness and	

		soilSample1.loamThickness == loamThickness and soilSample1.loamySandThickness == loamySandThickness)	
#	7		
Name	MeasureSiteArea		
Behaviour	site.siteArea = siteArea		
Input(s)	int siteArea		
Output			
Objects	Media site		
Tests	0. testMeasureSiteArea	MeasureSiteArea(100) assert(site.siteArea == 100)	
#	8		
Name	MeasureSiteVolume		
Behaviour	site.siteVolume = siteVolume		
Input(s)	int siteVolume		
Output			
Objects	Media site		
Tests	0. testMeasureSiteVolume	MeasureSiteVolume(1000) assert(site.siteVolume == 1000)	
#	9		
Name	MeasureSandPercentage		
Behaviour	soilSample1.sandPercentage = sandPercentage		
Input(s)	float sandPercentage		
Output			
Objects	SoilSample soilSample1		
Tests	0. testMeasureSandPercentage	MeasureSandPercentage(50.0) assert(soilSample1.sandPercentage == 50.0)	
#	10		
Name	MeasureSiltPercentage		
Behaviour	soilSample1.siltPercentage = siltPercentage		
Input(s)	float siltPercentage		
Output			
Objects	SoilSample soilSample1		
Tests	0. testMeasureSiltPercentage	MeasureSiltPercentage(25.0) assert(soilSample1.siltPercentage == 25.0)	
#	11		
Name	MeasureClayPercentage		
Behaviour	soilSample1.clayPercentage = clayPercentage		
Input(s)	float clayPercentage		
Output			
Objects	SoilSample soilSample1		
Tests	□		
#	12		

Name	MeasureContaminantConcentration	
Behaviour	MeasureBenzeneConcentration() MeasureEthylBenzeneConcentration() MeasureTolueneConcentration() MeasureXylenesConcentration()	
Input(s)		
Output		
Objects	SoilSamplingExperiment soilSamplingExp	
Tests	0. testMeasureContaminantConcentration	benzene.chemicalConcentration = 10.0 ethylBenzene.chemicalConcentration = 8.0 toluene.chemicalConcentration = 6.0 xylenes.chemicalConcentration = 4.0 MeasureContaminantConcentration() assert(soilSamplingExp.benzeneConcentration == 10.0 and soilSamplingExp.ethylBenzeneConcentration == 8.0 and soilSamplingExp.tolueneConcentration == 6.0 and soilSamplingExp.xylenesConcentration == 4.0)
#	13	
Name	MeasureBenzeneConcentration	
Behaviour	soilSamplingExp.benzeneConcentration = benzene.chemicalConcentration	
Input(s)		
Output		
Objects	ChemicalContaminant benzene SoilSamplingExperiment soilSamplingExp	
Tests	0. testMeasureBenzeneConcentration	benzene.chemicalConcentration = 10.0 MeasureBenzeneConcentration() assert(soilSamplingExp.benzeneConcentration == 10.0)
#	14	
Name	MeasureEthylBenzeneConcentration	
Behaviour	soilSamplingExp.ethylBenzeneConcentration = ethylBenzene.chemicalConcentration	
Input(s)		
Output		
Objects	ChemicalContaminant ethylBenzene SoilSamplingExperiment soilSamplingExp	
Tests	0. testMeasureEthylBenzeneConcentration	ethylBenzene.chemicalConcentration = 10.0 MeasureEthylBenzeneConcentration() assert(soilSamplingExp.ethylBenzeneConcentration == 10.0)
#	15	
Name	MeasureTolueneConcentration	
Behaviour	soilSamplingExp.tolueneConcentration = toluene.chemicalConcentration	
Input(s)		
Output		
Objects	ChemicalContaminant toluene SoilSamplingExperiment soilSamplingExp	
Tests	0. testMeasureTolueneConcentration	toluene.chemicalConcentration = 10.0 MeasureTolueneConcentration()

		assert(soilSamplingExp.tolueneConcentration == 10.0)	
#	16		
Name	MeasureSandThickness		
Behaviour	soilSample1.sandThickness = sandThickness		
Input(s)	float sandThickness		
Output			
Objects	SoilSample soilSample1		
Tests	0. testMeasureSandThickness	MeasureSandThickness(10.0) assert(soilSample1.sandThickness == 10.0)	
#	17		
Name	MeasureSiltThickness		
Behaviour	soilSample1.siltThickness = siltThickness		
Input(s)	float siltThickness		
Output			
Objects	SoilSample soilSample1		
Tests	0. testMeasureSiltThickness	MeasureSiltThickness(10.0) assert(soilSample1.siltThickness == 10.0)	
#	18		
Name	MeasureClayThickness		
Behaviour	soilSample1.clayThickness = clayThickness		
Input(s)	float clayThickness		
Output			
Objects	SoilSample soilSample1		
Tests	0. testMeasureClayThickness	MeasureClayThickness(10.0) assert(soilSample1.clayThickness == 10.0)	
#	19		
Name	MeasureLoamThickness		
Behaviour	soilSample1.loamThickness = loamThickness		
Input(s)	float loamThickness		
Output			
Objects	SoilSample soilSample1		
Tests	0. testMeasureLoamThickness	MeasureLoamThickness(10.0) assert(soilSample1.loamThickness == 10.0)	
#	20		
Name	MeasureSandyClayThickness		
Behaviour	soilSample1.sandyClayThickness = sandyClayThickness		
Input(s)	float sandyClayThickness		
Output			
Objects	SoilSample soilSample1		
Tests	0. testMeasureSandyClayThickness	MeasureSandyClayThickness(10.0)	

		assert(soilSample1.sandyClayThickness == 10.0)	
#	21		
Name	MeasureSandyLoamThickness		
Behaviour	soilSample1.sandyLoamThickness = sandyLoamThickness		
Input(s)	float sandyLoamThickness		
Output			
Objects	SoilSample soilSample1		
Tests	0. testMeasureSandyLoamThickness	MeasureSandyLoamThickness(10.0) assert(soilSample1.sandyLoamThickness == 10.0)	
#	22		
Name	MeasureSandyClayLoamThickness		
Behaviour	soilSample1.sandyClayLoamThickness = sandyClayLoamThickness		
Input(s)	float sandyClayLoamThickness		
Output			
Objects	SoilSample soilSample1		
Tests	0. testMeasureSandyClayLoamThickness	MeasureSandyClayLoamThickness(10.0) assert(soilSample1.sandyClayLoamThickness == 10.0)	
#	23		
Name	MeasureSiltyClayThickness		
Behaviour	soilSample1.siltyClayThickness = siltyClayThickness		
Input(s)	float siltyClayThickness		
Output			
Objects	SoilSample soilSample1		
Tests	0. testMeasureSiltyClayThickness	MeasureSiltyClayThickness(10.0) assert(soilSample1.siltyClayThickness == 10.0)	
#	24		
Name	MeasureSiltyLoamThickness		
Behaviour	soilSample1.siltyLoamThickness = siltyLoamThickness		
Input(s)	float siltyLoamThickness		
Output			
Objects	SoilSample soilSample1		
Tests	0. testMeasureSiltyLoamThickness	MeasureSiltyLoamThickness(10.0) assert(soilSample1.siltyLoamThickness == 10.0)	
#	25		
Name	MeasureSiltyClayLoamThickness		
Behaviour	soilSample1.siltyClayLoamThickness = siltyClayLoamThickness		
Input(s)	float siltyClayLoamThickness		
Output			
Objects	SoilSample soilSample1		
Tests	0. testMeasureSiltyClayLoamThickness	MeasureSiltyClayLoamThickness(10.0)	

		assert(soilSample1.siltyClayLoamThickness == 10.0)	
#	26		
Name	MeasureClayLoamThickness		
Behaviour	soilSample1.clayLoamThickness = clayLoamThickness		
Input(s)	float clayLoamThickness		
Output			
Objects	SoilSample soilSample1		
Tests	0. testMeasureClayLoamThickness	MeasureClayLoamThickness(10.0) assert(soilSample1.clayLoamThickness == 10.0)	
#	27		
Name	MeasureLoamySandThickness		
Behaviour	soilSample1.loamySandThickness = loamySandThickness		
Input(s)	float loamySandThickness		
Output			
Objects	SoilSample soilSample1		
Tests	0. testMeasureLoamySandThickness	MeasureLoamySandThickness(10.0) assert(soilSample1.loamySandThickness == 10.0)	
#	28		
Name	MeasureSoilTypePercentages		
Behaviour	MeasureSandPercentage(sandPercentage) MeasureSiltPercentage(siltPercentage) MeasureClayPercentage(clayPercentage) MeasureGravelPercentage(gravelPercentage)		
Input(s)	float sandPercentage, float siltPercentage, float clayPercentage, float gravelPercentage		
Output			
Objects	SoilSample soilSample1		
Tests	0. testMeasureSoilTypePercentages	MeasureSoilTypePercentages(50.0, 25.0, 15.0, 10.0) assert(soilSample1.sandPercentage == 50.0 and soilSample1.siltPercentage == 25.0 and soilSample1.clayPercentage == 15.0 and soilSample1.gravelPercentage == 10.0)	
#	29		
Name	MeasureMoisturePercentage		
Behaviour	soilSamplingExp.moisturePercentage = moisturePercentage		
Input(s)	float moisturePercentage		
Output			
Objects	SoilSamplingExperiment soilSamplingExp		
Tests	0. testMeasureMoisturePercentage	MeasureMoisturePercentage(10.0) assert(soilSamplingExp.moisturePercentage == 10.0)	
#	30		
Name	DetermineSoilHydraulicConductivity		
Behaviour	SetSoilHydraulicConductivity(isotropy, heterogeneity)		
Input(s)	string isotropy, string heterogeneity		

Output		
Objects	SoilSample soilSample1	
Tests	0. testDetermineSoilHydraulicConductivity	DetermineSoilHydraulicConductivity("isotropic", "heterogeneous") assert(soilSample1.hydraulicConductivity == "simple")
#	31	
Name	SetSoilHeterogeneity	
Behaviour	<pre> if (contributionFactor <= 0.2) { soilSample1.soilHeterogeneity = "homogeneous" } else if (contributionFactor > 0.2 and contributionFactor < 0.8) { soilSample1.soilHeterogeneity = "heterogeneous" } else { soilSample1.soilHeterogeneity = "extremely heterogeneous" } </pre>	
Input(s)	float contributionFactor	
Output		
Objects	SoilSample soilSample1	
Tests	0. testHomogeneous	SetSoilHeterogeneity(0.1) assert(soilSample1.soilHeterogeneity == "homogeneous")
	1. testHeterogeneous	SetSoilHeterogeneity(0.7) assert(soilSample1.soilHeterogeneity == "heterogeneous")
	2. testExtremelyHeterogeneous	SetSoilHeterogeneity(0.8) assert(soilSample1.soilHeterogeneity == "extremely heterogeneous")
#	32	
Name	CalculateSoilTextureContributionFactor	
Behaviour	<pre> # the values are loaded from a database soilSample1.soilTextureContributionFactor = 0.1 </pre>	
Input(s)		
Output		
Objects	SoilSample soilSample1	
Tests	□	
#	33	
Name	CalculateVerticalHydraulicConductivity	
Behaviour	<pre> float soilThickness soilThickness = soilSample1.sandThickness + soilSample1.sandyClayThickness + soilSample1.sandyLoamThickness + soilSample1.sandyClayLoamThickness + soilSample1.siltThickness + soilSample1.siltyClayThickness + soilSample1.siltyLoamThickness + soilSample1.siltyClayLoamThickness + soilSample1.clayThickness + soilSample1.clayLoamThickness + soilSample1.loamThickness + soilSample1.loamySandThickness soilSample1.verticalHydraulicConductivity = soilSample1.sampleDepth / (soilThickness / soilSample1.horizontalHydraulicConductivity) </pre>	
Input(s)		
Output		
Objects	SoilSample soilSample1	
Tests	0. testCalculateVerticalHydraulicConductivity	soilSample1.sandThickness = 15.0 soilSample1.sandyClayThickness = 14.0 soilSample1.sandyLoamThickness = 13.0 soilSample1.sandyClayLoamThickness = 12.0 soilSample1.siltThickness = 11.0

		soilSample1.siltyClayThickness = 10.0 soilSample1.siltyLoamThickness = 9.0 soilSample1.siltyClayLoamThickness = 8.0 soilSample1.clayThickness = 7.0 soilSample1.clayLoamThickness = 6.0 soilSample1.loamThickness = 5.0 soilSample1.loamySandThickness = 4.0 soilSample1.sampleDepth = 200.0 soilSample1.horizontalHydraulicConductivity = 10.0 CalculateVerticalHydraulicConductivity() assert(soilSample1.verticalHydraulicConductivity == 17.54386)
#	34	
Name	DetermineSoilHydraulicPermeability	
Behaviour	SetSoilHydraulicPermeability()	
Input(s)		
Output		
Objects		
Tests	☐	
#	35	
Name	DetermineSoilHeterogeneity	
Behaviour	SetSoilHeterogeneity(contributionFactor)	
Input(s)	float contributionFactor	
Output		
Objects	SoilSample soilSample1	
Tests	0. testDetermineSoilHeterogeneity	DetermineSoilHeterogeneity(0.1) assert(soilSample1.soilHeterogeneity == "homogeneous")
#	36	
Name	DetermineSoilIsotropy	
Behaviour	SetSoilIsotropy(horizontalHydraulicConductivity, verticalHydraulicConductivity)	
Input(s)	float horizontalHydraulicConductivity, float verticalHydraulicConductivity	
Output		
Objects	SoilSample soilSample1	
Tests	0. testDetermineSoilIsotropy	SetSoilIsotropy(1.0, 2.0) assert(soilSample1.isotropy == "isotropic")
#	37	
Name	SetSiteHydraulicConductivity	
Behaviour	if ((soilHeterogeneity == "homogeneous" and soilIsotropy == "isotropic" and soilPermeability == "high") or (soilHeterogeneity == "homogeneous" and soilIsotropy == "isotropic" and soilPermeability == "medium") or (soilHeterogeneity == "homogeneous" and soilIsotropy == "isotropic" and soilPermeability == "low") or (soilHeterogeneity == "heterogeneous" and soilIsotropy == "isotropic" and soilPermeability == "high") or (soilHeterogeneity == "heterogeneous" and soilIsotropy == "isotropic" and soilPermeability == "medium") or (soilHeterogeneity == "heterogeneous" and soilIsotropy == "isotropic" and soilPermeability == "low") or (soilHeterogeneity == "extremely heterogeneous" and soilIsotropy == "anisotropic" and soilPermeability == "high")) { site.hydraulicConductivity = "simple" }	

	<pre> else if ((soilHeterogeneity == "homogeneous" and soilIsotropy == "isotropic" and soilPermeability == "extremely low") or (soilHeterogeneity == "extremely heterogeneous" and soilIsotropy == "isotropic" and soilPermeability == "high") or (soilHeterogeneity == "extremely heterogeneous" and soilIsotropy == "isotropic" and soilPermeability == "medium") or (soilHeterogeneity == "heterogeneous" and soilIsotropy == "anisotropic" and soilPermeability == "high") or (soilHeterogeneity == "heterogeneous" and soilIsotropy == "anisotropic" and soilPermeability == "medium")) { site.hydraulicConductivity = "complex" } </pre>	
Input(s)	string soilHeterogeneity, string soilIsotropy, string soilPermeability	
Output		
Objects	Media site	
Tests	0. testSimpleHydraulicConductivity	SetSiteHydraulicConductivity("homogeneous", "isotropic", "high") assert(site.hydraulicConductivity == "simple")
	1. testComplexHydraulicConductivity	SetSiteHydraulicConductivity("homogeneous", "isotropic", "extremely low") assert(site.hydraulicConductivity == "complex")
#	38	
Name	CollectSoilSamples	
Behaviour	# soilSample1 should really be an input, but objects cannot yet be used as input parameters soilSamplingExp.soilSample = soilSample1	
Input(s)		
Output		
Objects	SoilSamplingExperiment soilSamplingExp SoilSample soilSample1	
Tests	0. testCollectSoilSamples	CollectSoilSamples() assert(true)
#	39	
Name	SetMediaType	
Behaviour	<pre> if (moisturePercentage < 25.0) { soilSamplingExp.mediaType = "soil" } else if (moisturePercentage >= 25.0 and moisturePercentage < 100.0) { soilSamplingExp.mediaType = "soilgroundwater" } else if (moisturePercentage >= 100.0) { soilSamplingExp.mediaType = "groundwater" } </pre>	
Input(s)	float moisturePercentage	
Output		
Objects	SoilSamplingExperiment soilSamplingExp	
Tests	0. testSoilType	SetMediaType(10.0) assert(soilSamplingExp.soilType == "soil")
	1. testSoilGroundwaterType	SetMediaType(50.0) assert(soilSamplingExp.soilType == "soilgroundwater")
	2. testGroundwaterType	SetMediaType(100.0) assert(soilSamplingExp.soilType == "groundwater")

#	40
Name	SetSoilType
Behaviour	<pre> string soilType float distance float minDis int i int j minDis = 1.0 i = 1 while (i <= 12) { float iclay iclay = GetIdealClayValue(i) float isand isand = GetIdealSandValue(i) float isilt isilt = GetIdealSiltValue(i) distance = sqrt((calcClayPercentage - iclay) ** 2.0 + (calcSandPercentage - isand) ** 2.0 + (calcSiltPercentage - isilt) ** 2.0) if (minDis >= distance) { minDis = distance j = i } if (j == 1) { soilType = "clay" } else if (j == 2) { soilType = "silty clay" } else if (j == 3) { soilType = "sandy clay" } else if (j == 4) { soilType = "silty clay loam" } else if (j == 5) { soilType = "clay loam" } else if (j == 6) { soilType = "sandy clay loam" } else if (j == 7) { soilType = "silt loam" } else if (j == 8) { soilType = "silt" } else if (j == 9) { soilType = "loam" } else if (j == 10) { soilType = "sandy loam" } </pre>

	<pre> } else if (j == 11) { soilType = "loamy sand" } else if (j == 12) { soilType = "sand" } } i = i + 1 } soilSample1.soilType = soilType </pre>	
Input(s)	float calcSandPercentage, float calcSiltPercentage, float calcClayPercentage	
Output		
Objects	SoilSample soilSample1	
Tests	0. testClay	<pre> CalculateSoilTypePercentages(0.05, 0.05, 0.8) SetSoilType(soilSample1.calculatedSandPercentage, soilSample1.calculatedSiltPercentage, soilSample1.calculatedClayPercentage) assert(soilSample1.soilType == "clay") </pre>
	1. testClayLoam	<pre> CalculateSoilTypePercentages(0.25, 0.25, 0.4) SetSoilType(soilSample1.calculatedSandPercentage, soilSample1.calculatedSiltPercentage, soilSample1.calculatedClayPercentage) assert(soilSample1.soilType == "clay loam") </pre>
#	41	
Name	SetSoilIsotropy	
Behavior	<pre> float result result = horizontalHydraulicConductivity / verticalHydraulicConductivity if (result < 2.0) { soilSample1.isotropy = "isotropic" } else if (result >= 2.0) { soilSample1.isotropy = "anisotropic" } </pre>	
Input(s)	float horizontalHydraulicConductivity, float verticalHydraulicConductivity	
Output		
Objects	SoilSample soilSample1	
Tests	0. testIsotropic	<pre> float horizontalHydraulicConductivity horizontalHydraulicConductivity = 1.0 float verticalHydraulicConductivity verticalHydraulicConductivity = 2.0 SetSoilIsotropy(horizontalHydraulicConductivity, verticalHydraulicConductivity) assert(soilSample1.isotropy == "isotropic") </pre>
	1. testAnisotropic	<pre> float horizontalHydraulicConductivity horizontalHydraulicConductivity = 5.0 float verticalHydraulicConductivity verticalHydraulicConductivity = 2.0 SetSoilIsotropy(horizontalHydraulicConductivity, verticalHydraulicConductivity) </pre>

	assert(soilSample1.isotropy == "anisotropic")	
#	42	
Name	MeasureSiteArea	
Behaviour	site.siteArea = area	
Input(s)	int area	
Output		
Objects	Media site	
Tests	□	
#	43	
Name	MeasureSiteVolume	
Behaviour	site.siteVolume = volume	
Input(s)	int volume	
Output		
Objects		
Tests	□	
#	44	
Name	CalcWeightedNormalizedBenzeneConcentration	
Behaviour	soilSamplingExp.benzeneConcentration = soilSamplingExp.benzeneConcentration / skStandard.skBenzene * benzene.contaminationWeight	
Input(s)		
Output		
Objects	SaskatchewanStandard skStandard SoilSamplingExperiment soilSamplingExp ChemicalContaminant benzene	
Tests	0. testCalcWeightedNormalizedBenzeneConcentration soilSamplingExp.benzeneConcentration = 0.5 skStandard.skBenzene = 0.2 benzene.contaminationWeight = 0.1 CalcWeightedNormalizedBenzeneConcentration() assert(soilSamplingExp.benzeneConcentration == 0.25)	
#	45	
Name	CalcWeightedNormalizedTolueneConcentration	
Behaviour	soilSamplingExp.tolueneConcentration = soilSamplingExp.tolueneConcentration / skStandard.skToluene * toluene.contaminationWeight	
Input(s)		
Output		
Objects	SoilSamplingExperiment soilSamplingExp SaskatchewanStandard skStandard ChemicalContaminant toluene	
Tests	0. testCalcWeightedNormalizedTolueneConcentration soilSamplingExp.tolueneConcentration = 0.5 skStandard.skToluene = 0.2 toluene.contaminationWeight = 0.1 CalcWeightedNormalizedTolueneConcentration() assert(soilSamplingExp.tolueneConcentration == 0.25)	
#	46	
Name	CalcWeightedNormalizedEthylBenzeneConcentration	
Behaviour	soilSamplingExp.ethylBenzeneConcentration = soilSamplingExp.ethylBenzeneConcentration / skStandard.skEthylBenzene * ethylBenzene.contaminationWeight	
Input(s)		
Output		

Objects	ChemicalContaminant ethylBenzene SaskatchewanStandard skStandard SoilSamplingExperiment soilSamplingExp	
Tests	0. testCalcWeightedNormalizedEthylBenzeneConcentration	soilSamplingExp.ethylBenzeneConcentration = 0.5 skStandard.skEthylBenzene = 0.2 ethylBenzene.contaminationWeight = 0.1 CalcWeightedNormalizedEthylBenzeneConcentration() assert(soilSamplingExp.ethylBenzeneConcentration == 0.25)
#	47	
Name	MeasureXylenesConcentration	
Behaviour	soilSamplingExp.xylenesConcentration = xylenes.chemicalConcentration	
Input(s)		
Output		
Objects	ChemicalContaminant xylenes SoilSamplingExperiment soilSamplingExp	
Tests	0. testXylenesConcentration	xylenes.chemicalConcentration = 10.0 MeasureXylenesConcentration() assert(soilSamplingExp.xylenesConcentration == 10.0)
#	48	
Name	CalcWeightedNormalizedXylenesConcentration	
Behaviour	soilSamplingExp.xylenesConcentration = soilSamplingExp.xylenesConcentration / skStandard.skXylenes * xylenes.contaminationWeight	
Input(s)		
Output		
Objects	ChemicalContaminant xylenes SaskatchewanStandard skStandard SoilSamplingExperiment soilSamplingExp	
Tests	0. testCalcWeightedNormalizedXylenesConcentration	soilSamplingExp.xylenesConcentration = 0.5 skStandard.skXylenes = 0.2 xylenes.contaminationWeight = 0.1 CalcWeightedNormalizedXylenesConcentration() assert(soilSamplingExp.xylenesConcentration == 0.25)
#	49	
Name	SumPollutantConcentrationValues	
Behaviour	soilSamplingExp.contaminantConcentration = soilSamplingExp.benzeneConcentration + soilSamplingExp.tolueneConcentration + soilSamplingExp.xylenesConcentration	
Input(s)		
Output		
Objects	SoilSamplingExperiment soilSamplingExp	
Tests	0. testSumPollutantConcentrationValues	soilSamplingExp.benzeneConcentration = 0.5 skStandard.skBenzene = 0.2 benzene.contaminationWeight = 0.1 CalcWeightedNormalizedBenzeneConcentration() soilSamplingExp.tolueneConcentration = 0.5 skStandard.skToluene = 0.2 toluene.contaminationWeight = 0.1 CalcWeightedNormalizedTolueneConcentration() soilSamplingExp.ethylBenzeneConcentration = 0.5

		skStandard.skEthylBenzene = 0.2 ethylBenzene.contaminationWeight = 0.1 CalcWeightedNormalizedEthylBenzeneConcentration() soilSamplingExp.xylenesConcentration = 0.5 skStandard.skXylenes = 0.2 xylenes.contaminationWeight = 0.1 CalcWeightedNormalizedXylenesConcentration() SumPollutantConcentrationValues() assert(soilSamplingExp.contaminationConcentration == 1.0)
#	50	
Name	DetermineSiteSize	
Behaviour	MeasureSiteArea(siteArea) MeasureSiteVolume(siteVolume) SetSiteSize(site.siteArea, site.siteVolume)	
Input(s)	int siteArea, int siteVolume	
Output	site.siteSize	
Objects	Media site	
Tests	0. testDetermineSiteSize string result result = DetermineSiteSize(100, 1000) assert(result == "small")	
#	51	
Name	DetermineMedia	
Behaviour	SetMediaType(moisturePercentage)	
Input(s)	float moisturePercentage	
Output	soilSamplingExp.mediaType	
Objects	SoilSamplingExperiment soilSamplingExp	
Tests	0. testDetermineMedia string result result = DetermineMedia(5.0) assert(result == "soil")	
#	52	
Name	DetermineContaminantType	
Behaviour	SoilSampling() DetermineSubstancesInSample() DeterminePollutantTypesInSoil() MeasurePollutantConcentration()	
Input(s)		
Output		
Objects		
Tests	□	
#	53	
Name	DetermineContaminationLevel	
Behaviour	CalcWeightedNormalizedBenzeneConcentration() CalcWeightedNormalizedTolueneConcentration() CalcWeightedNormalizedEthylBenzeneConcentration() CalcWeightedNormalizedXylenesConcentration() SumPollutantConcentrationValues() SetContaminationLevel()	
Input(s)		
Output		
Objects		

Tests	☐	
#	54	
Name	DetermineContaminationPhase	
Behaviour	SetContaminationPhase(phase)	
Input(s)	string phase	
Output		
Objects	SoilSamplingExperiment soilSamplingExp	
Tests	0. testDetermineContaminationPhase	DetermineContaminationPhase("free") assert(soilSamplingExp.contaminantPhase == "free")

#	55	
Name	SetRemediationMethod	
Behaviour	<pre> if (soilSamplingExp.mediaType == "soil" and soilSamplingExp.hydraulicConductivity == "complex" and site.siteSize == "large" and soilSamplingExp.contaminantPhase == "residual" and soilSamplingExp.contaminationLevel == "low") { soilSamplingExp.remediationMethod = compostingTechnology } else if (soilSamplingExp.mediaType == "soilgroundwater" and soilSamplingExp.hydraulicConductivity == "complex" and site.siteSize == "large" and soilSamplingExp.contaminantPhase == "free" and soilSamplingExp.contaminationLevel == "low") { soilSamplingExp.remediationMethod = inSituSteamExtractionTreatment } else if (soilSamplingExp.mediaType == "soilgroundwater" and soilSamplingExp.hydraulicConductivity == "complex" and site.siteSize == "small" and soilSamplingExp.contaminantPhase == "residual" and soilSamplingExp.contaminationLevel == "low") { soilSamplingExp.remediationMethod = chemicalOxidationSystem } else if (soilSamplingExp.mediaType == "soil" and soilSamplingExp.hydraulicConductivity == "complex" and site.siteSize == "small" and soilSamplingExp.contaminantPhase == "residual" and soilSamplingExp.contaminationLevel == "low") { soilSamplingExp.remediationMethod = desorptionSystemProcess } else if (soilSamplingExp.mediaType == "soil" and soilSamplingExp.hydraulicConductivity == "simple" and site.siteSize == "small" and soilSamplingExp.contaminantPhase == "residual" and soilSamplingExp.contaminationLevel == "low") { soilSamplingExp.remediationMethod = inSituVitrification } else if (soilSamplingExp.mediaType == "soilgroundwater" and soilSamplingExp.hydraulicConductivity == "simple" and site.siteSize == "small" and soilSamplingExp.contaminantPhase == "residual" and soilSamplingExp.contaminationLevel == "low") { soilSamplingExp.remediationMethod = solventExtraction } else if (soilSamplingExp.mediaType == "soilgroundwater" and soilSamplingExp.hydraulicConductivity == "simple" and site.siteSize == "large" and soilSamplingExp.contaminantPhase == "free" and soilSamplingExp.contaminationLevel == "low") { soilSamplingExp.remediationMethod = airSparging } else if (soilSamplingExp.mediaType == "soilgroundwater" and soilSamplingExp.hydraulicConductivity == "simple" and site.siteSize == "large" and soilSamplingExp.contaminantPhase == "residual" and soilSamplingExp.contaminationLevel == "low") { soilSamplingExp.remediationMethod = bioremediation } else if (soilSamplingExp.mediaType == "soil" and soilSamplingExp.hydraulicConductivity == "simple" and site.siteSize == "large" and soilSamplingExp.contaminantPhase == "free" and soilSamplingExp.contaminationLevel == "low") </pre>	

	<pre> { soilSamplingExp.remediationMethod = soilVaporExtraction } else if (soilSamplingExp.mediaType == "soil" and soilSamplingExp.hydraulicConductivity == "complex" and site.siteSize == "small" and soilSamplingExp.contaminantPhase == "free" and soilSamplingExp.contaminationLevel == "low") { soilSamplingExp.remediationMethod = thermalDesorption } else if (soilSamplingExp.mediaType == "soilgroundwater" and soilSamplingExp.hydraulicConductivity == "simple" and site.siteSize == "large" and soilSamplingExp.contaminantPhase == "free" and soilSamplingExp.contaminationLevel == "high") { soilSamplingExp.remediationMethod = inSituChemicalExtraction } else if (soilSamplingExp.mediaType == "soil" and soilSamplingExp.hydraulicConductivity == "simple" and site.siteSize == "large" and soilSamplingExp.contaminantPhase == "residual" and soilSamplingExp.contaminationLevel == "high") { soilSamplingExp.remediationMethod = inSituLeachingAndChemical } else if (soilSamplingExp.mediaType == "soilgroundwater" and soilSamplingExp.hydraulicConductivity == "simple" and site.siteSize == "small" and soilSamplingExp.contaminantPhase == "free" and soilSamplingExp.contaminationLevel == "high") { soilSamplingExp.remediationMethod = exSituChemicalExtraction } else if (soilSamplingExp.mediaType == "soil" and soilSamplingExp.hydraulicConductivity == "complex" and site.siteSize == "small" and soilSamplingExp.contaminantPhase == "residual" and soilSamplingExp.contaminationLevel == "high") { soilSamplingExp.remediationMethod = excavation } else if (soilSamplingExp.mediaType == "soil" and soilSamplingExp.hydraulicConductivity == "complex" and site.siteSize == "small" and soilSamplingExp.contaminantPhase == "residual" and soilSamplingExp.contaminationLevel == "high") { soilSamplingExp.remediationMethod = excavation } else if (soilSamplingExp.mediaType == "soilgroundwater" and soilSamplingExp.hydraulicConductivity == "simple" and site.siteSize == "large" and soilSamplingExp.contaminantPhase == "residual" and soilSamplingExp.contaminationLevel == "high") { soilSamplingExp.remediationMethod = inSituPassiveRemediation } else if (soilSamplingExp.mediaType == "soilgroundwater" and soilSamplingExp.hydraulicConductivity == "complex" and site.siteSize == "small" and soilSamplingExp.contaminantPhase == "residual" and soilSamplingExp.contaminationLevel == "high") { soilSamplingExp.remediationMethod = inSituSlurryWalls } </pre>
Input(s)	
Output	
Objects	SoilSamplingExperiment soilSamplingExp Media site PetroleumContaminationRemediation compostingTechnology InSituRemediation inSituSteamExtractionTreatment PetroleumContaminationRemediation chemicalOxidationSystem PetroleumContaminationRemediation desorptionSystemProcess InSituRemediation inSituVitrification PetroleumContaminationRemediation solventExtraction PetroleumContaminationRemediation airSparging PetroleumContaminationRemediation bioremediation PetroleumContaminationRemediation soilVaporExtraction PetroleumContaminationRemediation thermalDesorption InSituRemediation inSituChemicalExtraction InSituRemediation inSituLeachingAndChemical

	ExSituRemediation exSituChemicalExtraction PetroleumContaminationRemediation excavation InSituRemediation inSituPassiveRemediation InSituRemediation inSituSlurryWalls	
Tests	0. testCompostingTechnology	soilSamplingExp.mediaType = "soil" soilSamplingExp.hydraulicConductivity = "complex" site.siteSize = "large" soilSamplingExp.contaminantPhase = "residual" soilSamplingExp.contaminationLevel = "low" SetRemediationMethod() assert(soilSamplingExp.remediationMethod == compostingTechnology)
	1. testInSituSteamExtractionTreatment	soilSamplingExp.mediaType = "soilgroundwater" soilSamplingExp.hydraulicConductivity = "complex" site.siteSize = "large" soilSamplingExp.contaminantPhase = "free" soilSamplingExp.contaminationLevel = "low" SetRemediationMethod() assert(soilSamplingExp.remediationMethod == inSituSteamExtractionTreatment)
	2. testChemicalOxidationSystem	soilSamplingExp.mediaType = "soilgroundwater" soilSamplingExp.hydraulicConductivity = "complex" site.siteSize = "small" soilSamplingExp.contaminantPhase = "residual" soilSamplingExp.contaminationLevel = "low" SetRemediationMethod() assert(soilSamplingExp.remediationMethod == chemicalOxidationSystem)
	3. testDesorptionSystemProcess	soilSamplingExp.mediaType = "soil" soilSamplingExp.hydraulicConductivity = "complex" site.siteSize = "small" soilSamplingExp.contaminantPhase = "residual" soilSamplingExp.contaminationLevel = "low" SetRemediationMethod() assert(soilSamplingExp.remediationMethod == desorptionSystemProcess)
	4. testInSituVitrification	soilSamplingExp.mediaType = "soil" soilSamplingExp.hydraulicConductivity = "simple" site.siteSize = "small" soilSamplingExp.contaminantPhase = "residual" soilSamplingExp.contaminationLevel = "low" SetRemediationMethod() assert(soilSamplingExp.remediationMethod == inSituVitrification)
	5. testSolventExtraction	soilSamplingExp.mediaType = "soilgroundwater" soilSamplingExp.hydraulicConductivity = "simple" site.siteSize = "small" soilSamplingExp.contaminantPhase = "residual" soilSamplingExp.contaminationLevel = "low" SetRemediationMethod() assert(soilSamplingExp.remediationMethod == solventExtraction)
	6. testAirSparging	soilSamplingExp.mediaType = "soilgroundwater" soilSamplingExp.hydraulicConductivity = "simple" site.siteSize = "large" soilSamplingExp.contaminantPhase = "free" soilSamplingExp.contaminationLevel = "low" SetRemediationMethod()

	assert(soilSamplingExp.remediationMethod == airSparging)
7. testBioremediation	soilSamplingExp.mediaType = "soilgroundwater" soilSamplingExp.hydraulicConductivity = "simple" site.siteSize = "large" soilSamplingExp.contaminantPhase = "residual" soilSamplingExp.contaminationLevel = "low" SetRemediationMethod() assert(soilSamplingExp.remediationMethod == bioremediation)
8. testSoilVaporExtraction	soilSamplingExp.mediaType = "soil" soilSamplingExp.hydraulicConductivity = "simple" site.siteSize = "large" soilSamplingExp.contaminantPhase = "free" soilSamplingExp.contaminationLevel = "low" SetRemediationMethod() assert(soilSamplingExp.remediationMethod == soilVaporExtraction)
9. testThermalDesorption	soilSamplingExp.mediaType = "soil" soilSamplingExp.hydraulicConductivity = "complex" site.siteSize = "small" soilSamplingExp.contaminantPhase = "free" soilSamplingExp.contaminationLevel = "low" SetRemediationMethod() assert(soilSamplingExp.remediationMethod == thermalDesorption)
10. testInSituChemicalExtraction	soilSamplingExp.mediaType = "soilgroundwater" soilSamplingExp.hydraulicConductivity = "simple" site.siteSize = "large" soilSamplingExp.contaminantPhase = "free" soilSamplingExp.contaminationLevel = "high" SetRemediationMethod() assert(soilSamplingExp.remediationMethod == inSituChemicalExtraction)
11. testInSituLeachingAndChemical	soilSamplingExp.mediaType = "soil" soilSamplingExp.hydraulicConductivity = "simple" site.siteSize = "large" soilSamplingExp.contaminantPhase = "residual" soilSamplingExp.contaminationLevel = "high" SetRemediationMethod() assert(soilSamplingExp.remediationMethod == inSituLeachingAndChemical)
12. testExSituChemicalExtraction	soilSamplingExp.mediaType = "soilgroundwater" soilSamplingExp.hydraulicConductivity = "simple" site.siteSize = "small" soilSamplingExp.contaminantPhase = "free" soilSamplingExp.contaminationLevel = "high" SetRemediationMethod() assert(soilSamplingExp.remediationMethod == exSituChemicalExtraction)
13. testExcavation	soilSamplingExp.mediaType = "soil" soilSamplingExp.hydraulicConductivity = "complex" site.siteSize = "small" soilSamplingExp.contaminantPhase = "residual" soilSamplingExp.contaminationLevel = "high" SetRemediationMethod() assert(soilSamplingExp.remediationMethod == excavation)
14. testInSituPassiveRemediation	soilSamplingExp.mediaType = "soilgroundwater" soilSamplingExp.hydraulicConductivity = "simple"

		site.siteSize = "large" soilSamplingExp.contaminantPhase = "residual" soilSamplingExp.contaminationLevel = "high" SetRemediationMethod() assert(soilSamplingExp.remediationMethod == inSituPassiveRemediation)
	15. testInSituSlurryWalls	soilSamplingExp.mediaType = "soilgroundwater" soilSamplingExp.hydraulicConductivity = "complex" site.siteSize = "small" soilSamplingExp.contaminantPhase = "residual" soilSamplingExp.contaminationLevel = "high" SetRemediationMethod() assert(soilSamplingExp.remediationMethod == inSituSlurryWalls)

#	56	
Name	SetContaminationLevel	
Behaviour	if (contaminationConcentration < 0.1) { soilSamplingExp.contaminationLevel = "low" } else if (contaminationConcentration >= 0.1 and contaminationConcentration <= 0.8) { soilSamplingExp.contaminationLevel = "high" } else if (contaminationConcentration > 0.8) { soilSamplingExp.contaminationLevel = "extremely high" }	
Input(s)	float contaminationConcentration	
Output		
Objects	SoilSamplingExperiment soilSamplingExp	
Tests	0. testLow	SetContaminationLevel(0.01) assert(soilSamplingExp.contaminationLevel == "low")
	1. testHigh	SetContaminationLevel(0.5) assert(soilSamplingExp.contaminationLevel == "high")
	2. testExtremelyHigh	SetContaminationLevel(0.9) assert(soilSamplingExp.contaminationLevel == "extremely high")

#	57	
Name	CalculateHorizontalHydraulicConductivity	
Behaviour	# Need the calculation for this. soilSample1.horizontalHydraulicConductivity = 1.0	
Input(s)		
Output		
Objects	SoilSample soilSample1	
Tests		

#	58	
Name	SetSoilHydraulicConductivity	
Behaviour	if (heterogeneity == "homogeneous") { soilSample1.hydraulicConductivity = "simple" } else if (heterogeneity == "heterogeneous") { if (isotropy == "isotropic")	

	<pre> { soilSample1.hydraulicConductivity = "simple" } else { soilSample1.hydraulicConductivity = "complex" } } else if (heterogeneity == "extremely heterogeneous") { if (isotropy == "isotropic") { soilSample1.hydraulicConductivity = "complex" } else { soilSample1.hydraulicConductivity = "simple" } } } </pre>		
Input(s)	string isotropy, string heterogeneity		
Output			
Objects	SoilSample soilSample1		
Tests	0. testSimple	SetSoilHydraulicConductivity("isotropic", "heterogeneous") assert(soilSample1.hydraulicConductivity == "simple")	
	1. testComplex	SetSoilHydraulicConductivity("anisotropic", "heterogeneous") assert(soilSample1.hydraulicConductivity == "complex")	
#	59		
Name	MeasureGravelPercentage		
Behaviour	soilSample1.gravelPercentage = gravelPercentage		
Input(s)	float gravelPercentage		
Output			
Objects	SoilSample soilSample1		
Tests	□		
#	60		
Name	CalculateSoilTypePercentages		
Behaviour	CalculateClayPercentage(measuredSandPercentage, measuredSiltPercentage, measuredClayPercentage) CalculateSandPercentage(measuredSandPercentage, measuredSiltPercentage, measuredClayPercentage) CalculateSiltPercentage(measuredSandPercentage, measuredSiltPercentage, measuredClayPercentage)		
Input(s)	float measuredSandPercentage, float measuredSiltPercentage, float measuredClayPercentage		
Output			
Objects	SoilSample soilSample1		
Tests	0. testCalculateSoilTypePercentages	CalculateSoilTypePercentages(50.0, 25.0, 15.0) assert(soilSample1.calculatedSandPercentage == 0.55555556 and soilSample1.calculatedSiltPercentage == 0.27777778 and soilSample1.calculatedClayPercentage == 0.16666667)	
#	61		
Name	CalculateClayPercentage		
Behaviour	soilSample1.calculatedClayPercentage = measuredClayPercentage / (measuredClayPercentage + measuredSandPercentage + measuredSiltPercentage)		
Input(s)	float measuredSandPercentage, float measuredSiltPercentage, float measuredClayPercentage		
Output			
Objects	SoilSample soilSample1		
Tests	0. testCalculateClayPercentage	CalculateClayPercentage(50.0, 25.0, 15.0)	

		assert(soilSample1.calculatedClayPercentage == 0.16666667)
#	62	
Name	CalculateSandPercentage	
Behaviour	soilSample1.calculatedSandPercentage = measuredSandPercentage / (measuredClayPercentage + measuredSandPercentage + measuredSiltPercentage)	
Input(s)	float measuredSandPercentage, float measuredSiltPercentage, float measuredClayPercentage	
Output		
Objects	SoilSample soilSample1	
Tests	0. testCalculateSandPercentage	CalculateSandPercentage(50.0, 25.0, 15.0) assert(soilSample1.calculatedSandPercentage == 0.55555556)
#	63	
Name	CalculateSiltPercentage	
Behaviour	soilSample1.calculatedSiltPercentage = measuredSiltPercentage / (measuredClayPercentage + measuredSandPercentage + measuredSiltPercentage)	
Input(s)	float measuredSandPercentage, float measuredSiltPercentage, float measuredClayPercentage	
Output		
Objects	SoilSample soilSample1	
Tests	0. testCalculateSiltPercentage	CalculateSiltPercentage(50.0, 25.0, 15.0) assert(soilSample1.calculatedSiltPercentage == 0.27777778)
#	64	
Name	GetIdealClayValue	
Behaviour	float value value = 0.0 if (soilType == 1) { value = 0.77 } else if (soilType == 2) { value = 0.5 } else if (soilType == 3) { value = 0.46 } else if (soilType == 4) { value = 0.35 } else if (soilType == 5) { value = 0.35 } else if (soilType == 6) { value = 0.29 } else if (soilType == 7) { value = 0.14 } else if (soilType == 8) { value = 0.06 } else if (soilType == 9)	

	<pre> { value = 0.18 } else if (soilType == 10) { value = 0.1 } else if (soilType == 11) { value = 0.08 } else if (soilType == 12) { value = 0.05 } </pre>
Input(s)	int soilType
Output	value
Objects	
Tests	☐
#	65
Name	GetIdealSandValue
Behaviour	<pre> float value value = 0.0 if (soilType == 1) { value = 0.22 } else if (soilType == 2) { value = 0.1 } else if (soilType == 3) { value = 0.54 } else if (soilType == 4) { value = 0.1 } else if (soilType == 5) { value = 0.33 } else if (soilType == 6) { value = 0.62 } else if (soilType == 7) { value = 0.25 } else if (soilType == 8) { value = 0.1 } else if (soilType == 9) { value = 0.38 } else if (soilType == 10) { value = 0.6 } else if (soilType == 11) { </pre>

	<pre> value = 0.8 } else if (soilType == 12) { value = 0.93 } </pre>
Input(s)	int soilType
Output	value
Objects	
Tests	☐
#	66
Name	GetIdealSiltValue
Behaviour	<pre> float value value = 0.0 if (soilType == 1) { value = 0.2 } else if (soilType == 2) { value = 0.5 } else if (soilType == 3) { value = 0.1 } else if (soilType == 4) { value = 0.56 } else if (soilType == 5) { value = 0.34 } else if (soilType == 6) { value = 0.19 } else if (soilType == 7) { value = 0.25 } else if (soilType == 8) { value = 0.9 } else if (soilType == 9) { value = 0.39 } else if (soilType == 10) { value = 0.25 } else if (soilType == 11) { value = 0.15 } else if (soilType == 12) { value = 0.08 } </pre>
Input(s)	int soilType
Output	value

Objects		
Tests		
#	67	
Name	SetSiteSize	
Behaviour	<pre> if (area < 1600 and volume < 25000) { site.siteSize = "small" } else if (area >= 1600 and area <= 2000 and volume >= 25000 and volume <= 30000) { site.siteSize = "medium" } else if (area > 2000 and volume > 30000) { site.siteSize = "large" } </pre>	
Input(s)	int area, int volume	
Output		
Objects	Media site	
Tests	0. testSmall	<pre> site.siteArea = 1000 site.siteVolume = 1000 SetSiteSize(site.siteArea, site.siteVolume) assert(site.siteSize == "small") </pre>
	1. testMedium	<pre> site.siteArea = 2000 site.siteVolume = 30000 SetSiteSize(site.siteArea, site.siteVolume) assert(site.siteSize == "medium") </pre>
	2. testLarge	<pre> site.siteArea = 10000 site.siteVolume = 300000 SetSiteSize(site.siteArea, site.siteVolume) assert(site.siteSize == "large") </pre>
#	68	
Name	DetermineSoilHydraulicPermeability	
Behaviour	<pre> CalculateHorizontalHydraulicConductivity() CalculateVerticalHydraulicConductivity() SetSoilHydraulicConductivity() </pre>	
Input(s)		
Output		
Objects		
Tests		
#	69	
Name	SetContaminationPhase	
Behaviour	soilSamplingExp.contaminantPhase = phase	
Input(s)	string phase	
Output		
Objects	SoilSamplingExperiment soilSamplingExp	
Tests	0. testFree	<pre> SetContaminationPhase("free") assert(soilSamplingExp.contaminantPhase == "free") </pre>

	1. testResidual	SetContaminationPhase("residual") assert(soilSamplingExp.contaminantPhase == "residual")	
	2. testFreeAndResidual	SetContaminationPhase("free and residual") assert(soilSamplingExp.contaminantPhase == "free and residual")	
#	70		
Name	SetSoilHydraulicPermeability		
Behaviour	soilSample1.soilHydraulicPermeability = "medium"		
Input(s)			
Output			
Objects	SoilSample soilSample1		
Tests	☐		

Appendix III

Petroleum Contamination Remediation Selection Ontology : Static Knowledge

Created with Protégé (OWL File Format)

This appendix contains the static knowledge for the petroleum contamination remediation selection ontology created with Protégé in the OWL file format.

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns="http://142.3.27.23/~harrisonr/petrol_rem_sel.owl#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://142.3.27.23/~harrisonr/petrol_rem_sel.owl">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:about="#Media">
    <owl:disjointWith rdf:resource="#Standard"/>
    <owl:disjointWith>
      <owl:Class rdf:about="#Remediation"/>
    </owl:disjointWith>
    <owl:disjointWith rdf:resource="#Gas"/>
    <owl:disjointWith>
      <owl:Class rdf:about="#Experiment"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:about="#Contaminant"/>
    </owl:disjointWith>
  </owl:Class>
  <owl:FunctionalProperty rdf:ID="depthOfSite">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <rdfs:domain rdf:resource="#Media"/>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
      in feet</rdfs:comment>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="hydraulicConductivity">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <rdfs:range>
      <owl:DataRange>
        <owl:oneOf rdf:parseType="Resource">
          <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
            simple</rdf:first>
          <rdf:rest rdf:parseType="Resource">
            <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
          </rdf:rest>
        </owl:oneOf>
      </owl:DataRange>
    </rdfs:range>
  </owl:FunctionalProperty>
```

```

        <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
        >complex</rdf:first>
    </rdf:rest>
</owl:oneOf>
</owl:DataRange>
</rdfs:range>
<rdfs:domain>
    <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
            <owl:Class rdf:about="#Media"/>
            <owl:Class rdf:about="#SoilSample"/>
            <owl:Class rdf:about="#SoilSamplingExperiment"/>
        </owl:unionOf>
    </owl:Class>
</rdfs:domain>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="siteArea">
    <rdfs:domain rdf:resource="#Media"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="siteVolume">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
    <rdfs:domain rdf:resource="#Media"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</owl:FunctionalProperty>
<Media rdf:ID="site">
    <siteVolume rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
    >1000</siteVolume>
    <siteArea rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
    >100</siteArea>
    <depthOfSite rdf:datatype="http://www.w3.org/2001/XMLSchema#float"
    >0.0</depthOfSite>
    <hydraulicConductivity rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >complex</hydraulicConductivity>
    <siteSize rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >small</siteSize>
</Media>
</rdf:RDF>

```

Appendix IV

Petroleum Contamination Remediation Selection Ontology : Dynamic Knowledge created with Dyna (XML File Format)

This appendix contains a sample of the XML output of Dyna for the petroleum contamination remediation selection ontology.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<DynaProject xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <ProjectName>Project</ProjectName>
  <Documentation/>
  <Url>http://142.3.27.23/~harrisonr/test.owl</Url>
  <NamespaceList>
    <Namespace>
      <Prefix>rdf</Prefix>
      <Uri>http://www.w3.org/1999/02/22-rdf-syntax-ns#</Uri>
    </Namespace>
    <Namespace>
      <Prefix>xsd</Prefix>
      <Uri>http://www.w3.org/2001/XMLSchema#</Uri>
    </Namespace>
    <Namespace>
      <Prefix>rdfs</Prefix>
      <Uri>http://www.w3.org/2000/01/rdf-schema#</Uri>
    </Namespace>
    <Namespace>
      <Prefix>owl</Prefix>
      <Uri>http://www.w3.org/2002/07/owl#</Uri>
    </Namespace>
    <Namespace>
      <Prefix>dyn</Prefix>
      <Uri>http://142.3.27.23/~harrisonr/dynamic.owl#</Uri>
    </Namespace>
    <Namespace>
      <Prefix>petrol</Prefix>
      <Uri>http://142.3.27.23/~harrisonr/petrol_rem_sel.owl#</Uri>
    </Namespace>
  </NamespaceList>
  <TaskList>
    <Task>
      <Name>SetSiteSize</Name>
      <Documentation/>
      <SubTaskList/>
      <DependencyList/>
      <TaskArgList>
```

```

<TaskArg>
<VarType>int</VarType>
<VarName>area</VarName>
</TaskArg>
<TaskArg>
<VarType>int</VarType>
<VarName>volume</VarName>
</TaskArg>
</TaskArgList>
<TaskReturn/>
<Behaviour>if (area &lt; 1600 and volume &lt; 25000)
{
    site.siteSize = "small"
}
else if (area &gt;= 1600 and area &lt;= 2000 and
        volume &gt;= 25000 and volume &lt;= 30000)
{
    site.siteSize = "medium"
}
else if (area &gt; 2000 and volume &gt; 30000)
{
    site.siteSize = "large"
}</Behaviour>
<ObjectList>
<Object rdf:resource="http://142.3.27.23/~harrisonr/petrol_rem_sel.owl#site"/>
</ObjectList>
<PreCondition/>
<TestSuite>
<TestSetup/>
<TestCaseList>
<TestCase>
<TestCaseName>testSmall</TestCaseName>
<TestCaseCode>site.siteArea = 1000
site.siteVolume = 1000

SetSiteSize(site.siteArea, site.siteVolume)

assert(site.siteSize == "small")</TestCaseCode>
</TestCase>
<TestCase>
<TestCaseName>testMedium</TestCaseName>
<TestCaseCode>site.siteArea = 2000
site.siteVolume = 30000

SetSiteSize(site.siteArea, site.siteVolume)

```

```

assert(site.siteSize == "medium")</TestCaseCode>
</TestCase>
<TestCase>
<TestCaseName>testLarge</TestCaseName>
<TestCaseCode>site.siteArea = 10000
site.siteVolume = 300000

SetSiteSize(site.siteArea, site.siteVolume)

```

```

assert(site.siteSize == "large")</TestCaseCode>
</TestCase>
</TestCaseList>
</TestSuite>
</Task>
</TaskList>
<ObjectiveList>
<Objective>
<Name>DetermineSiteSize</Name>
<Documentation/>
<TaskPriorityList>
<TaskPriority>
<Name>tp_SetSiteArea</Name>
<Documentation/>
<Task>MeasureSiteArea</Task>
<Priority>1.0</Priority>
</TaskPriority>
<TaskPriority>
<Name>tp_SetSiteVolume</Name>
<Documentation/>
<Task>MeasureSiteVolume</Task>
<Priority>2.0</Priority>
</TaskPriority>
<TaskPriority>
<Name>tp_SetSiteSize</Name>
<Documentation/>
<Task>SetSiteSize</Task>
<Priority>3.0</Priority>
</TaskPriority>
</TaskPriorityList>
</Objective>
</ObjectiveList>
</DynaProject>

```

Appendix V

Petroleum Contamination Remediation Selection Ontology : Dynamic Knowledge created with Dyna (OWL File Format)

This appendix contains a sample of the OWL output of Dyna for the petroleum contamination remediation selection ontology.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns="http://142.3.27.23/~harrisonr/test.owl"
xmlns:dyn="http://142.3.27.23/~harrisonr/dynamic.owl#"
xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:petrol="http://142.3.27.23/~harrisonr/petrol_rem_sel.owl#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
xml:base="http://142.3.27.23/~harrisonr/test.owl#">
<owl:Ontology rdf:about="">
<owl:imports rdf:resource="http://142.3.27.23/~harrisonr/dynamic.owl"/>
<owl:imports rdf:resource="http://142.3.27.23/~harrisonr/petrol_rem_sel.owl"/>
</owl:Ontology>
<dyn:Project rdf:ID="Project">
<dyn:name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Project</dyn:name>
<dyn:documentation rdf:datatype="http://www.w3.org/2001/XMLSchema#string"/>
<dyn:taskList>
<dyn:Task rdf:ID="SetSiteSize">
<dyn:name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">SetSiteSize</dyn:name>
<dyn:documentation rdf:datatype="http://www.w3.org/2001/XMLSchema#string"/>
<dyn:behaviour rdf:datatype="http://www.w3.org/2001/XMLSchema#string">if (area
&lt; 1600 and volume &lt; 25000)
{
    site.siteSize = "small"
}
else if (area &gt;= 1600 and area &lt;= 2000 and
        volume &gt;= 25000 and volume &lt;= 30000)
{
    site.siteSize = "medium"
}
else if (area &gt; 2000 and volume &gt; 30000)
{
    site.siteSize = "large"
}</dyn:behaviour>
<dyn:objectList rdf:resource="http://142.3.27.23/~harrisonr/petrol_rem_sel.owl#site"/>
<dyn:preCondition rdf:datatype="http://www.w3.org/2001/XMLSchema#string"/>
```

```

<dyn:testSuite>
<dyn:KnowledgeTestSuite rdf:ID="ts_SetSiteSize">
<dyn:name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">ts_SetSiteSize</dyn:name
>
<dyn:documentation rdf:datatype="http://www.w3.org/2001/XMLSchema#string"/>
<dyn:setup rdf:datatype="http://www.w3.org/2001/XMLSchema#string"/>
<dyn:testCaseList>
<dyn:KnowledgeTestCase rdf:ID="testSmall">
<dyn:name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">testSmall</dyn:name>
<dyn:documentation rdf:datatype="http://www.w3.org/2001/XMLSchema#string"/>
<dyn:testCode
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">site.siteArea = 1000
site.siteVolume = 1000

```

SetSiteSize(site.siteArea, site.siteVolume)

```

assert(site.siteSize == "small")</dyn:testCode>
</dyn:KnowledgeTestCase>
</dyn:testCaseList>
<dyn:testCaseList>
<dyn:KnowledgeTestCase rdf:ID="testMedium">
<dyn:name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">testMedium</dyn:name>
<dyn:documentation rdf:datatype="http://www.w3.org/2001/XMLSchema#string"/>
<dyn:testCode
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">site.siteArea = 2000
site.siteVolume = 30000

```

SetSiteSize(site.siteArea, site.siteVolume)

```

assert(site.siteSize == "medium")</dyn:testCode>
</dyn:KnowledgeTestCase>
</dyn:testCaseList>
<dyn:testCaseList>
<dyn:KnowledgeTestCase rdf:ID="testLarge">
<dyn:name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">testLarge</dyn:name>
<dyn:documentation rdf:datatype="http://www.w3.org/2001/XMLSchema#string"/>
<dyn:testCode
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">site.siteArea = 10000
site.siteVolume = 300000

```

SetSiteSize(site.siteArea, site.siteVolume)

```

assert(site.siteSize == "large")</dyn:testCode>
</dyn:KnowledgeTestCase>
</dyn:testCaseList>
</dyn:KnowledgeTestSuite>
</dyn:testSuite>
</dyn:Task>
</dyn:taskList>
<dyn:objectiveList>
<dyn:Objective rdf:ID="DetermineSiteSize">
<dyn:name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">DetermineSiteSize</dyn:n
ame>
<dyn:documentation rdf:datatype="http://www.w3.org/2001/XMLSchema#string"/>
<dyn:taskPriorityList>
<dyn:TaskPriority rdf:ID="tp_SetSiteArea">
<dyn:name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">tp_SetSiteArea</dyn:name
>
<dyn:documentation rdf:datatype="http://www.w3.org/2001/XMLSchema#string"/>
<dyn:task rdf:resource="#MeasureSiteArea"/>
<dyn:priority
rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1.0</dyn:priority>
</dyn:TaskPriority>
</dyn:taskPriorityList>
<dyn:taskPriorityList>
<dyn:TaskPriority rdf:ID="tp_SetSiteVolume">
<dyn:name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">tp_SetSiteVolume</dyn:n
ame>
<dyn:documentation rdf:datatype="http://www.w3.org/2001/XMLSchema#string"/>
<dyn:task rdf:resource="#MeasureSiteVolume"/>
<dyn:priority
rdf:datatype="http://www.w3.org/2001/XMLSchema#int">2.0</dyn:priority>
</dyn:TaskPriority>
</dyn:taskPriorityList>
<dyn:taskPriorityList>
<dyn:TaskPriority rdf:ID="tp_SetSiteSize">
<dyn:name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">tp_SetSiteSize</dyn:name
>
<dyn:documentation rdf:datatype="http://www.w3.org/2001/XMLSchema#string"/>
<dyn:task rdf:resource="#SetSiteSize"/>
<dyn:priority
rdf:datatype="http://www.w3.org/2001/XMLSchema#int">3.0</dyn:priority>
</dyn:TaskPriority>
</dyn:taskPriorityList>

```



```
</dyn:Objective>  
</dyn:objectiveList>  
</dyn:Project>  
</rdf:RDF>
```

Appendix VI

Dynamic Knowledge Elements (OWL File Format)

When the dynamic knowledge in Dyna is exported to OWL, the below OWL statements are instantiated to represent Dyna's dynamic knowledge elements in OWL.

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns="http://www.owl-ontologies.com/unnamed.owl#"
  xml:base="http://www.owl-ontologies.com/unnamed.owl">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="TaskPriority"/>
  <owl:Class rdf:ID="Task"/>
  <owl:Class rdf:ID="Objective"/>
  <owl:ObjectProperty rdf:ID="objects">
    <rdfs:domain rdf:resource="#Task"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="inputs">
    <rdfs:domain rdf:resource="#Task"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="taskPriorityList">
    <rdfs:domain rdf:resource="#Objective"/>
    <rdfs:range rdf:resource="#TaskPriority"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="subTasks">
    <rdfs:domain rdf:resource="#Task"/>
    <rdfs:range rdf:resource="#Task"/>
  </owl:ObjectProperty>
  <owl:DatatypeProperty rdf:ID="DatatypeProperty_7"/>
  <owl:FunctionalProperty rdf:ID="documentation">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="output">
    <rdfs:domain rdf:resource="#Task"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="preCondition">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:FunctionalProperty>
```

```

</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="task">
  <rdfs:range rdf:resource="#Task"/>
  <rdfs:domain rdf:resource="#TaskPriority"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="priority">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  <rdfs:domain rdf:resource="#TaskPriority"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="behaviour">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Task"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="name">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</owl:FunctionalProperty>
</rdf:RDF>

```