**ONTOLOGIES OF ELECTRONIC DEVICES IN
DAML+OIL FOR AUTOMATED PRODUCT DESIGN
SERVICES IN THE SEMANTIC WEB**

by

Syed I Ahmed

A thesis submitted to the Faculty of Graduate Studies and Research

in partial fulfillment of the requirements for the degree of

Master of Engineering in Telecommunication Technology Management

Department of Systems and Computer Engineering

Carleton University

Ottawa, Canada, K1S 5B6

January 08, 2003

© 2003, Syed I Ahmed

National Library
of Canada

Acquisitions and
Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale
du Canada

Acquisitions et
services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

ISBN: 0-612-83492-1

Our file Notre référence

ISBN: 0-612-83492-1

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

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

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

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

Canada

The undersigned recommend to
the Faculty of Graduate Studies and Research
acceptance of the thesis

**“ONTOLOGIES OF ELECTRONIC DEVICES IN DAML+OIL FOR
AUTOMATED PRODUCT DESIGN SERVICES IN THE SEMANTIC WEB”**

submitted

by Syed I Ahmed

in partial fulfillment of the requirements for the degree of
Master of Engineering in Telecommunication Technology Management

R.A. Goubran, Department Chair

A.J. Bailetti, Co-Supervisor

M. Weiss, Co-Supervisor

Carleton University

January 8, 2003

ABSTRACT

The DARPA Agent Mark-up Language + Ontology Inference Layer (DAML+OIL) is a new language that is currently in research stage for the development of the next generation Internet envisioned to become the Semantic Web. The Semantic Web is anticipated to understand meaning and offer advanced Artificial Intelligence (AI) based services to the users. Among the many types of services that can potentially be constructed in the Semantic Web is automated electronic product design service. A primary asset that is required for such a service is ontologies for electronic devices in machine interpretable form. In this thesis, prototype partial ontologies were developed for a number of electronic devices in DAML+OIL. It was assessed that DAML+OIL can be readily used to develop ontologies for describing the DC electrical characteristics of the devices but not the AC characteristics. It was also determined that Unified Modeling Language (UML) will require extensions to fully model ontologies developed in DAML+OIL.

ACKNOWLEDGEMENTS

I would like to express my gratitude to my supervisors, Prof. Tony Bailetti of Department of Systems and Computer Engineering and my co-supervisor Prof. Michael Weiss, of Computer Science Department for their guidance, patience, advice and inspiration, throughout the research work. Their constant striving for creative and rigorous research influenced many aspects of my thinking on the subject matter.

I also thank my family for their support, understanding and love during my study and research.

TABLE OF CONTENTS

<u>1. INTRODUCTION</u>	1
<u>1.1 The DARPA DAML+OIL project</u>	4
<u>1.2 The need for research in electronic device ontologies</u>	5
<u>1.3 Research question</u>	6
<u>1.4 Research Approach</u>	7
<u>1.5 Objective</u>	7
<u>1.6 Scope</u>	8
<u>1.7 Electronic product design services in the Semantic Web</u>	8
<u>1.7.1. What is involved in electronic product development?</u>	9
<u>1.7.2. Automated product development scenario</u>	9
<u>1.8 Architecture for product design services in the Semantic Web</u>	12
<u>1.8.1. Service Access Plane (SAP)</u>	13
<u>1.8.2. The DAML+OIL Agent Plane (DAAP)</u>	14
<u>1.8.3. The Virtual Reference Plane (VRP)</u>	14
<u>1.9 Visualizing ontologies and knowledge domains in the web</u>	15
<u>1.9.1. Different domains and ontologies in the Semantic Web</u>	16
<u>1.10 Machine readability, machine comprehension and ontology</u>	17
<u>1.10.1. Readability of electronic device specifications</u>	17
<u>1.10.2. Machine comprehension of meaning</u>	18
<u>1.10.3. Relationship between readability, ontology & comprehension</u>	19

<u>2.</u>	<u>LITERATURE REVIEW</u>	20
<u>2.1</u>	<u>Semantic Web</u>	20
<u>2.2</u>	<u>The Layer Cake Model of the Semantic Web</u>	21
<u>2.2.1.</u>	<u>Unicode</u>	22
<u>2.2.2.</u>	<u>Uniform Resource Indicator (URI)</u>	23
<u>2.2.3.</u>	<u>XML</u>	24
<u>2.2.4.</u>	<u>Namespace</u>	25
<u>2.2.5.</u>	<u>XML Schema</u>	26
<u>2.2.6.</u>	<u>RDF</u>	27
<u>2.2.7.</u>	<u>Logic and Trust</u>	28
<u>2.3</u>	<u>DAML+OIL</u>	29
<u>2.4</u>	<u>Ontologies</u>	31
<u>2.4.1.</u>	<u>Relation between Ontologies and Schema-languages</u>	32
<u>2.4.2.</u>	<u>Agents and the Semantic Web</u>	33
<u>2.5</u>	<u>Agents</u>	33
<u>2.6</u>	<u>Unified Modeling Language (UML) and Ontologies</u>	34
<u>2.7</u>	<u>DAML-Services</u>	35
<u>2.8</u>	<u>Product development on the web</u>	35
<u>2.9</u>	<u>Electronic Device specification documents</u>	36
<u>2.10</u>	<u>Other initiatives in machine readability of device specification</u>	36
<u>2.10.1.</u>	<u>Goal of machine readability</u>	37
<u>2.10.2.</u>	<u>Platform</u>	38
<u>2.10.3.</u>	<u>DAML+ OIL ontologies vs. Si2 Open Access</u>	38
<u>2.10.4.</u>	<u>DAML+OIL device ontologies vs. RosettaNet™</u>	39
<u>2.10.5.</u>	<u>DAML+ OIL device ontologies vs. EDIF</u>	40

2.10.6.	<u>DAML+ OIL device ontologies vs. EDA</u>	40
2.11	<u>Modeling XML Applications with UML</u>	41
2.11.1.	<u>XML Metadata Interchange (XMI)</u>	41
2.11.2.	<u>Mapping UML to XML</u>	42
2.12		43
2.12.1.	<u>Mapping UML Model of a electronic device catalog to XML</u>	44
3.	<u>PROTOTYPE ONTOLOGIES FOR ELECTRONIC DEVICE IN DAML+OIL</u>	45
3.1	<u>DAML+OIL Class Constructors and Axioms</u>	45
3.1.1.	<u>DAML+OIL Axioms</u>	46
3.2	<u>Approach to electronic device ontology development</u>	46
3.2.1.	<u>Levels in electronic device ontology development</u>	47
3.2.2.	<u>Ad hoc approach to device ontology development</u>	48
3.2.3.	<u>Global ontologies</u>	48
3.2.4.	<u>Standard ontology based on classes of devices</u>	49
3.3	<u>Electronic devices as a tree of DAML+OIL classes</u>	51
3.4	<u>Device ontology development experiment</u>	52
3.4.1.	<u>Setting up Namespaces for devices</u>	52
3.4.2.	<u>Defining device classes and properties in DAML+OIL</u>	54
3.4.3.	<u>Declaring universal class 'Devices'</u>	54
3.4.4.	<u>Active Devices class</u>	55
3.4.5.	<u>Passive Devices class</u>	56
3.4.6.	<u>Distinguishing between Resistors, Capacitors, Inductors</u>	58
3.4.7.	<u>Class of devices with RLC</u>	58
3.4.8.	<u>PassiveDiode</u>	59
3.5	<u>Properties of electronic devices in DAML +OIL</u>	60

3.5.1.	<u>Device Meta Data and RDF</u>	60
3.5.2.	<u>Device Property and property datatype</u>	60
3.5.3.	<u>Device property</u>	62
3.5.4.	<u>Survey of device mechanical properties</u>	63
3.5.5.	<u>Device Thermal Properties</u>	65
3.5.6.	<u>Electrical property</u>	66
3.5.7.	<u>Ohms Law electrical property</u>	67
3.5.8.	<u>Voltage property</u>	67
3.5.9.	<u>Current property</u>	67
3.5.10.	<u>Impedance property</u>	68
3.5.11.	<u>Resistance Property</u>	68
3.5.12.	<u>Inductance Property</u>	68
3.5.13.	<u>Capacitance Property</u>	69
3.5.14.	<u>Distinguishing Electrical, Mechanical and Thermal properties</u>	69
3.5.15.	<u>Has Ground Pins Property</u>	75
3.5.16.	<u>Judicious use of rdfs: domain and rdfs: range for devices</u>	75
3.6	<u>DAML + OIL DatatypeProperty for Electronic Devices</u>	76
3.7	<u>Devices classes with the device property included</u>	77
3.7.1.	<u>Passive Diode Class</u>	78
3.8	<u>Ontology for active devices in DAML+OIL</u>	78
3.8.1.	<u>Active Devices</u>	78
3.8.2.	<u>DiscreteActiveComponents</u>	79
3.8.3.	<u>Digital Devices</u>	79
3.8.4.	<u>Integrated Circuits (IC) Class in DAML+OIL</u>	80
3.8.5.	<u>Operation Amplifiers Class</u>	81
3.8.6.	<u>Multiple super-classes & Mixed Signal Devices</u>	83
3.8.7.	<u>Transmit and Receive as Inverse relations</u>	84

<u>3.9</u>	<u><i>Ontology of Programmable Devices in DAML+OIL</i></u>	84
<u>3.9.1.</u>	<u><i>Classes of serial data comm. devices in DAML+OIL</i></u>	85
<u>3.10</u>	<u><i>Properties of Programmable Devices</i></u>	87
<u>3.10.1.</u>	<u><i>Tree of properties for the PC16550 chip</i></u>	87
<u>3.10.2.</u>	<u><i>Device Application Level Ontology</i></u>	88
<u>3.10.3.</u>	<u><i>Functional Level Ontology</i></u>	89
<u>3.10.4.</u>	<u><i>Hardware Properties of devices</i></u>	90
<u>3.10.5.</u>	<u><i>Firmware Level Ontology</i></u>	93
<u>3.10.6.</u>	<u><i>Device Bit Rate in XML</i></u>	94
<u>3.11</u>	<u><i>Instance of Application Level description</i></u>	94
<u>3.12</u>	<u><i>Semantic difference between device parameters</i></u>	95
<u>3.13</u>	<u><i>Electronic Devices with similar properties</i></u>	97
<u>3.14</u>	<u><i>EXAMPLE Ontology of LASER DIODE in DAML+OIL</i></u>	100
<u>3.14.1.</u>	<u><i>Ontology header</i></u>	101
<u>3.14.2.</u>	<u><i>Defining Properties of Laser Diode in DAML+OIL</i></u>	102
<u>3.14.3.</u>	<u><i>Data Types for Laser Diode properties</i></u>	104
<u>3.15</u>	<u><i>Instantiations of Laser Diode</i></u>	107
<u>3.16</u>	<u><i>Summary of device ontology development experiment</i></u>	107
<u>4.</u>	<u><i>MODELING DAML+OIL ELECTRONIC DEVICE ONTOLOGY IN UML</i></u>	109
<u>4.1</u>	<u><i>Background and distinctive features of UML and DAML+OIL</i></u>	110
<u>4.2</u>	<u><i>Concepts in UML and DAML+OIL</i></u>	112
<u>4.3</u>	<u><i>Classes in DAML+OIL and UML</i></u>	113
<u>4.3.1.</u>	<u><i>Potential use of UML Class Diagrams in DAML+OIL</i></u>	113
<u>4.3.2.</u>	<u><i>Constraints</i></u>	120

<u>4.4</u>	<u><i>Difference between DAML+OIL and UML</i></u>	120
<u>4.4.1.</u>	<u><i>No Universal Classes in UML</i></u>	120
<u>4.4.2.</u>	<u><i>DAML+OIL Property and UML Association</i></u>	121
<u>4.4.3.</u>	<u><i>Domain, property, association and range</i></u>	122
<u>4.4.4.</u>	<u><i>Non-binary association in UML</i></u>	123
<u>4.4.5.</u>	<u><i>Multiplicity Constraints</i></u>	124
<u>4.4.6.</u>	<u><i>Comparing Cardinality Constraints</i></u>	125
<u>4.4.7.</u>	<u><i>Imposition of Constraints</i></u>	126
<u>4.5</u>	<u><i>DAML+OIL effect on modularity of software design</i></u>	126
<u>4.6</u>	<u><i>Properties, Association and Attributes</i></u>	127
<u>4.6.1.</u>	<u><i>Properties without domain</i></u>	127
<u>4.6.2.</u>	<u><i>Property as a sub property</i></u>	128
<u>4.7</u>	<u><i>Namespaces in DAML+OIL and UML</i></u>	128
<u>4.8</u>	<u><i>Linking & protection in UML and DAML+OIL</i></u>	129
<u>4.8.1.</u>	<u><i>Comparing DAML+OIL with property patterns</i></u>	129
<u>4.8.2.</u>	<u><i>Example of Real, virtual, view and agent pattern</i></u>	132
<u>4.8.3.</u>	<u><i>Design pattern properties in terms of DAML+OIL</i></u>	132
<u>4.8.4.</u>	<u><i>Pattern in terms of DAML+OIL</i></u>	133
<u>4.9</u>	<u><i>UML, Semantic Web and the layer cake model</i></u>	134
<u>4.10</u>	<u><i>DAML+OIL and ASN.1</i></u>	135
<u>5.</u>	<u><i>CONCLUSIONS AND CLAIMS</i></u>	137
<u>5.1</u>	<u><i>Summary of the conclusion</i></u>	137
<u>5.2</u>	<u><i>Summary of contributions</i></u>	137
<u>5.3</u>	<u><i>Claim of original contribution</i></u>	141

5.4	<i>Future work</i>	141
6.	<u>APPENDIX 1: WHAT IS ONTOLOGY?</u>	143
6.1	<i>Various definitions of Ontology</i>	143
6.1.1.	<i>Ontology as Controlled Vocabulary</i>	146
6.1.2.	<i>Ontology as a glossary</i>	147
6.1.3.	<i>Ontology as a Thesauri</i>	147
6.1.4.	<i>Test of a well developed ontology</i>	148
6.1.5.	<i>Ontology as data structure</i>	148
6.1.6.	<i>Ontology as classes</i>	149
6.1.7.	<i>Ontology vs. classes in Object Oriented Design</i>	149
6.1.8.	<i>Ontology as a subclass hierarchies</i>	149
6.1.9.	<i>Expressiveness of ontology</i>	150
6.2	<i>Ontology and the Semantic Web</i>	150
6.3	<i>Reasons for developing ontology for engineering services</i>	151
6.3.1.	<i>Separating the domain knowledge from the operational knowledge</i>	152
6.4	<i>Conclusion</i>	152
7.	<u>APPENDIX 2: XML IN TELECOMMUNICATION</u>	153
8.	<u>APPENDIX 3 : DATA SHEETS OF ELECTRONIC DEVICES</u>	155
8.1	<i>Phillips Semiconductor</i>	155
8.2	<i>National Semiconductor Universal Asynchronous Receiver Transmitter</i>	155
8.3	<i>Texas Instrument Chip 16C550 data sheet</i>	155
8.4	<i>Zarlink Semiconductor</i>	155
8.5	<i>Digikey Catalog</i>	155

8.6	<u>Texas Instrument Mixed Signal Processors</u>	155
9.	<u>APPENDIX 4 KNOWLEDGE REPRESENTATION LABS</u>	156
10.	<u>APPENDIX 5: DAML+OIL ONTOLOGY LISTING FOR LASER</u>	157
11.	<u>APPENDIX 6 : SUMMARY OF RDF</u>	159
12.	<u>APPENDIX 7: SUMMARY OF DAML+OIL LANGUAGE ELEMENTS</u>	164
13.	<u>APPENDIX 8: VIRTUAL OBJECTS</u>	173
14.	<u>BIBLIOGRAPHY</u>	179

TABLE OF FIGURES

<u>Figure 1.7.2-1: Architecture for product design service in the Semantic Web</u>	12
<u>Figure 1.8.3-1 Graphical representation of different domains</u>	15
<u>Figure 1.9.1-1 Representations of many different domains in Internet</u>	16
<u>Figure 1.10.3-1: Readability, Ontology and Machine Comprehension</u>	19
<u>Figure 1.10.3-1 Layer cake model of the Semantic Web.</u>	21
<u>Figure 2.2.7-1: Three roots of Ontology inference Layer (OIL)</u>	30
<u>Figure 2.11.2-1: Simple electronic device catalog model in UML</u>	43
<u>Figure 2.12.1-1 : Mapping UML model of device catalog to XML</u>	44
<u>Figure 3.2.4-1: Tree Diagram for subclasses of electronic devices</u>	51
<u>Figure 3.5.2-1: Tree diagram for electronic device properties</u>	62
<u>Figure 3.5.4-1 : Example of Mechanical properties of PC 16550 chip</u>	63
<u>Figure 3.9.1-1: Tree diagram of subclasses of serial data comm. devices</u>	85
<u>Figure 3.9.1-1: Generic properties of programmable electronic devices</u>	87
<u>Figure 3.10.1-1: Tree diagram developed for PC16550 chip ontology levels</u>	88
<u>Figure 3.10.4-1 : PC16550 Chip Hardware</u>	90
<u>Figure 3.10.4-2; Pin description for PC 16550 chip</u>	91
<u>Figure 3.10.6-1: Family of TI MSP 430 devices with similar properties</u>	98
<u>Figure 4.3.1-1: DAML+OIL Property Restriction</u>	114
<u>Figure 4.3.1-2: Example of sub-property</u>	115
<u>Figure 4.3.1-3: Example of a Sub-Property of a Device</u>	116
<u>Figure 4.3.1-4: DAML+OIL Type Property of an example Device</u>	116
<u>Figure 4.3.1-5 : DAML+OIL Cardinality and UML multiplicity</u>	117

<u>Figure 4.3.1-6: Example of DAML+OIL Unambiguous Property</u>	118
<u>Figure 4.3.1-7: DAML+OIL Transitive Property</u>	119
<u>Figure 4.8.1-1 : Examples of geometrical design patterns</u>	129
<u>Figure 4.8.1-2: DAML+OIL abstraction of properties of patterns</u>	131

LISTING OF ONTOLOGY SEGMENT

<u>Listing: 2.2-1: Example of an XML tags and attribute</u>	25
<u>Listing: 3.4-1 Namespaces for device ontology.....</u>	53
<u>Listing: 3.4-2 Ontology declaration in DAML+OIL.....</u>	53
<u>Listing: 3.4-3 Universal Device class</u>	54
<u>Listing: 3.4-4 ActiveDevices class</u>	55
<u>Listing: 3.4-5: PassiveDevices class.....</u>	56
<u>Listing: 3.4-6: Resistors class.....</u>	57
<u>Listing: 3.4-7 Inductors class.....</u>	57
<u>Listing: 3.4-8: Capacitors class.....</u>	57
<u>Listing: 3.4-9 : Distinguishing between RLC in DAML+OIL</u>	58
<u>Listing: 3.4-10 RLC Class</u>	58
<u>Listing: 3.4-11 PassiveDiode class.....</u>	59
<u>Listing: 3.5-1: Device property</u>	62
<u>Listing: 3.5-2: Device Mechanical Properties.....</u>	64
<u>Listing: 3.5-3: Thermal Property of device</u>	65
<u>Listing: 3.5-4: Electrical property of the device.....</u>	66
<u>Listing: 3.5-5: Ohm's Law Electrical Property of the device</u>	67
<u>Listing: 3.5-6 : Voltage property.....</u>	67
<u>Listing: 3.5-7 : Electrical current property.....</u>	67
<u>Listing: 3.5-8: Inductance Property.....</u>	68
<u>Listing: 3.5-9 Capacitance Property</u>	69
<u>Listing: 3.5-10 : DC property of devices</u>	71

<u>Listing: 3.5-11: AC property of the of device</u>	73
<u>Listing: 3.5-12: Example of device AC properties as XML tags</u>	73
<u>Listing: 3.5-13 : Address Strobe width</u>	74
<u>Listing: 3.6-1: Device Total pin count property</u>	76
<u>Listing: 3.6-2: Device temperature property data type</u>	77
<u>Listing: 3.7-1: Passive Diodes class with property included</u>	78
<u>Listing: 3.9-1: Class Serial-data-devices-devices</u>	85
<u>Listing: 3.9-2: Class Synchronous devices</u>	86
<u>Listing: 3.9-3: Class TDM devices</u>	86
<u>Listing: 3.9-4: Class Sync-Async devices</u>	86
<u>Listing: 3.9-5: Class USART devices</u>	86
<u>Listing: 3.9-6: Class CUSCC devices</u>	86
<u>Listing: 3.9-7: Class Asynchronous devices</u>	86
<u>Listing: 3.10-1: Grouping modem control pins & XML representation</u>	92
<u>Listing: 3.10-2: Modem signals in XML</u>	92
<u>Listing: 3.10-3: Maximum data speed property in DAML+ OIL</u>	93
<u>Listing: 3.10-4 Bit rate in XML Schema form</u>	94
<u>Listing: 3.11-1 Application Level representation of data link speed</u>	94

TABLES

<u>Table 3.1-1 : DAML+OIL Class Constructors</u>	45
<u>Table 3.1-2 : DAML+OIL AXIOMS.....</u>	46
<u>Table 3.5-1: Example of Thermal Properties of device</u>	65
<u>Table 3.5-2: Example of Device DC Electrical Characteristics</u>	71
<u>Table 3.5-3: Example of AC Electrical Characteristics.....</u>	72
<u>Table 3.12-1 : Programming Register differences in same class of device.....</u>	96
<u>Table 3.14-1: Zarlink Laser Diode device data sheet.....</u>	100
<u>Table 4.2-1: Comparison between UML with DAML+OIL.....</u>	112
<u>Table 6.2-1 : Comparision Ontology and Semantic Web.....</u>	150

ACRONYMS

ARPANET	Advanced Research Projects Agency Network
AC	Alternating Current
CAD	Computer Aided Design
DAAP	DAML+OIL Agent Plane
DAML	DARPA Agent Markup Language
DC	Direct Current
DARPA	Defense Advanced Research Project Agency
DL	Descriptive Logic
IEEE	Institute of Electrical and Electronic Engineers
ITU	International Telecommunication Union
MSP	Mixed Signal Processor
OIL	Ontology Inference Layer
RDF	Resource Description Framework
SAP	Service Access Plane
UML	Unified Modeling Language
URI	Uniform Resource Indicator
VRP	Virtual Reference Plane
XML	Extensible Markup Language
XPATH	XML Path
XPOINTER	XML Pointer

1. INTRODUCTION

The World Wide Web (www) is being increasingly used for e-mail, document transfer, business transactions, entertainment, education, government services and many other applications. Vast amounts of engineering knowledge exist in various web based digital libraries, such as in the IEEE digital libraries [50][51] and in the web sites of universities, research institutions, government, businesses and corporations that are connected to the Internet. The web is also being used for publishing research papers, engineering knowledge, technical standards [49], advertising catalogues [66], and specifications of devices, components and products.

Heflin [2] points out that the vast intellectual resources and the global connectivity of the Internet offer “virtually unlimited potential”.

To utilize the potential of the web, attempts could be made to semantically interconnect quality information and knowledge in the Internet and configure it to offer advanced intelligent services. The successful construction of such services in the Internet could provide new powerful capabilities and features that are not possible with software tools on standalone workstations.

Unfortunately, the current software in the Internet is unaware of the meaning of the information that resides in the different web sites and servers. Among the many reasons for this drawback is the fact that information is not in machine interpretable form and is primarily targeted at people and not at information processing machines. The final destination of the information on the Internet is the user's computer monitor screen or the

printer from where the information is 'consumed' by humans. The machines of the future are expected to process not only information but also the meaning. The World Wide Web (W3C) Consortium [52] is developing new language standards and recommendations for the development of next generation Internet that is being envisioned to become the *Semantic Web*. The Semantic Web [1], as described by T. Berners-Lee, J. Hendler et al, is expected to understand the meaning and have the capabilities to offer advanced intelligent services and exhibit *anthropomorphic* characteristics. When using the Semantic Web the users would feel, as if they are interacting with intelligent and knowledgeable human beings rather than reciprocating with machines. The Internet of the future is expected to not only become a global library of almost all human knowledge but also ubiquitous computing resource that will offer Artificial Intelligence (AI) [25] based services to both humans and other machines. Advanced AI based services in the future Internet are expected to become available to industries, homes, government, business, travel, health, education, science, technology institutions and to entertainment sectors.

The current web does not have the infrastructure to provide such advanced services. One of the many reasons for the weakness is that the documents in the current web are mostly written in Hypertext Markup Language (HTML) and such documents are not machine-interpretable. The capability of HTML is limited to document format, text transfer, remote access and display to humans. HTML does not have the capability to indicate semantics nor to support machine comprehension of the meaning by the machines. It is therefore unsuitable for supporting the implementation of the envisioned Semantic Web. The Semantic Web is being conceived to have distributed computing architecture and extensively software agent based. Vast amount of knowledge has already been developed

about software agents and published in several academic literatures and most modern computer systems developers implement the concept of software agents. A fundamental requirement for the success of the envisioned Semantic Web is the development of a common language for communicating of meaning among software agents in the Internet and between humans and machines or software agents. Towards that objective, the World Wide Web Consortium (W3C) [52] has developed the Extensible Markup Language (XML) [4] and the XML-Schema [5] [6] specifications. The Consortium is also developing a family of associated supporting recommendations and specifications, such as the Resource Description Framework (RDF) [7] and RDF-Schema [8]. A Schema is an abstract definition and an overall 'blueprint' that represent an object's particular characteristics and interrelationships between its parts. The W3C specifications are expected to form the foundation on which the next generation Internet could be built.

The language defined by W3C for the next generation Internet is XML [4], which is actually a language to construct other languages. An example of a mature XML based language, is XOL [30]. XOL is an XML-based ontology-exchange language that was designed for the exchange of bio-informatics ontologies, and it is claimed that XOL can be used for ontologies in any domain. (The notion of ontology is discussed in this thesis in APPENDIX 1: WHAT IS ONTOLOGY?).

In the electronics area, some of the XML based languages developed for the telecommunication sector are listed in APPENDIX 2.

An important XML based language for the Semantic Web that is currently in research stage is “*Defense Advanced Research Projects Agency (DARPA) Agent Markup Language*”. It is referred in research literature in abbreviated form as ‘DAML’ [55].

1.1 The DARPA DAML+OIL project

In the year 2000, the U.S. Defense Advanced Research Projects Agency (DARPA) [53] initiated a project called, “DARPA Agent Mark-Up Language”, or DAML [47] to develop the language and tools [54] for the next generation Internet. DARPA funds large U.S. military research projects and is the organization that originally funded the ARPANET [46], which subsequently led to the development of the now popular Internet. DAML is currently an on-going well-funded research project.

The DARPA project [21] is expected to contribute to the development of the next generation Internet that is envisioned to be extensively ontology and software agent based. A part of the DAML project is the development of DAML+OIL. OIL, which stands for “Ontology Inference Layer”, which is an extension to DAML contributed by a few European countries and DAML+OIL is now a joint United States and European effort. DAML+OIL merges the two languages (DAML and OIL) into a single proposed standard for the Semantic Web, which is currently a W3C draft document [23]. The Ontology Inference Layer (OIL) [60] is being added to DAML because ontologies constitute the foundation or the skeleton for virtually all the innovative Knowledge-based systems (KBS) [APPENDIX 4].

With the progress of DAML+OIL, Knowledge Representation (KR) and Knowledge Based Systems (KBS), *machine comprehension of meaning* could become a reality. With that technological achievement, the Internet of the future will exhibit anthropomorphism and intelligence and have the capability to offer advanced knowledge based services to both humans and machines. It may be assumed that among the many Artificial Intelligence (AI) based services that are being envisaged in the Semantic Web will be engineering services for product and system design. The future Internet may become the Virtual Engineering Workstation with built-in distributed intelligence that may globally span over the entire Internet. The services in the web of the future may vary in capability from a very simple service, like the design a wooden box, to a highly complex system like the design of a useable space shuttle and inter planetary vehicles. The concepts on the architectures and design of on-line services in the Semantic Web are currently being developed in draft specification called DAML-S [56].

An essential asset that will be required for the development of automated electronic product design services, are *ontologies for electronic devices*. In this thesis, the potential of using DAML+OIL to develop electronic device ontologies is determined through experimentation.

1.2 The need for research in electronic device ontologies

The development of device ontology is important because electronics is extensively used in the modern world. The exercise of developing device ontologies may contribute towards the knowledge of developing ontologies of other [37] non-electronic products

and systems as well. This is because electronic device ontology requires representation of the both static and dynamic characteristics, and the lessons learnt may be applicable to other products and systems, such as mechanical systems. The dynamic characteristics of electronic devices can vary in speed from zero to hundred GigaHertz (GHz) frequencies. Mechanical systems are slower than electronic systems in performance and electronics have been used to model mechanical systems. DAML+OIL is expected to have the capability to interface with ontologies in any domain and not limited to the technology domain.

As shown in the DAML web site [47], a large number of university professors and other researchers are currently engaged in DAML+OIL and Semantic Web research. None of the published or unpublished literature has so far focused on the potential of using DAML+OIL to develop ontologies for electronic devices.

1.3 Research question

At least two fundamental research questions arise when considering the development of ontologies for electronic devices. These are:

- *Can DAML+OIL be used to develop ontologies for electronic devices? If so, what are the limitations of DAML+OIL, if any?*
- *Can Unified Modeling Language (UML) be applied to model device ontologies written in DAML+OIL? If so, what are its limitations, if any?*

This thesis attempts to find answers to the above two questions. These are fundamentally important questions because DAML+OIL is a candidate language for the Semantic Web, which is envisaged to offer advanced artificial intelligence based services. Such services will require support from developed ontologies. In addition, the modeling of ontologies, perhaps in UML will be important for the software engineering that will be required to construct the Semantic Web.

1.4 Research Approach

An *experimental approach* is taken to conduct this research using the following four steps:

1. Literature review
2. Take random samples of a number of different commercial electronic device specifications, attempt to develop prototype ontologies using DAML+OIL, and observe the complexities and limitations to be successful.
3. Compare DAML+OIL with UML for modeling ontologies
4. Draw conclusions from the results.

1.5 Objective

The objective of this research is to contribute to the knowledge of building services in the Semantic Web.

1.6 Scope

The scope of the research is limited to the following:

- Formulation of an ontology for electronic devices in DAML+OIL
- Comparison of DAML+OIL with UML

1.7 Electronic product design services in the Semantic Web

The business use of Internet is increasing and it is being realized that the Internet can also be utilized [81] for product development. Brown and Eisenhardt [70] has pointed out that new product development is critical because it becoming the nexus of competition and that efficient product design is important for two reasons:

- It is a source of competitive advantage, firms that can quickly develop exciting designs for products that people are anxious to buy will likely win
- It is the means by which members of an organizations diversify, adapt, and reinvent their firms to match evolving market and technical conditions

Product design services in the Semantic Web will not simply be a matter of moving the current design tools, such as CAD systems, from stand alone workstations and making them available on the Internet for global access. The vision of the Semantic Web is to build intelligence in the Internet that is globally interoperable and which can be

configured to offer advanced AI automated services that can exceed the capabilities of stand-alone workstations.

Presently, at least two important constituents required for the development of the envisioned automated electronic product design services are missing. These are:

- Ontology of electronic devices
- Language to markup information so that it is machine readable and understandable

1.7.1. What is involved in electronic product development?

Electronic product and system design includes conceptualization of the market demand for the functionality that the product can offer, the use of electronic theory, tools, application of engineering design rules, integration, testing and compliance to industry standards or conventions. Automated product design implies the generation of valid designs using machines with little or no human intervention. Automated product development in the Semantic Web implies the use of artificial intelligence in the Internet to produce the acceptable designs.

1.7.2. Automated product development scenario

The product development processes that are typically followed in the electronics and telecommunication industries are more or less similar. These are listed below:

1. Analysis of the functional requirements of the product

2. Determination and application of electronic systems design rules
3. Compliance to the industry standards
4. Search for most appropriate components and devices
5. Application of design rules
6. Generation of schematics and other drawings
7. Generation of Bill of Materials (BOM)
8. Printed Circuit board layout or silicon chip layout
9. Integration of devices and components and prototyping
10. Programming (if required)
11. Testing
12. Generation and execution of test vectors
13. Change and revision control
14. Manufacturing
15. Generation of various documentation

The activities listed above correspond to the many stages required in the product development process. The processes may be executed sequentially or in parallel. It requires mental and physical work by specialist. In the industry, the responsibilities are assigned to teams who have expertise in the particular areas. In the Semantic Web, analogous to the activities and expertise executed by humans, groups of software agents could be developed to emulate the corresponding activities now performed by humans. With reference to the Service Architecture proposed in section 1.8.2, dedicated DAML+OIL software Agents could be developed that specialize in the different domains

and aspects of product development. The proposed Virtual Reference Plane (VRP) in section 1.8.3 could contain the knowledge base required by the service agents.

However, it is unlikely that the product development processes can be totally automated in the near future. It should be realized at the outset that, human *input* will always be required not only for creativity and also because it may be not be possible to automate all aspects of product development. In reality, it would be prudent to architecture the envisioned advanced services in the Semantic Web to facilitate only *semi-automatic product development*, rather attempt to build a fully unmanned development system in the Semantic Web.

It should be recognized that at present, the capability of machines to interpret *data*, *knowledge* and *process* is extremely weak to support the envisioned automation of processes listed above. For example, the device data shown in Figure 3.10.4-2 cannot be translated into machine interpretable form and therefore the automation cannot be readily implemented. Extensive amount of research and achievement of success and confidence in machine comprehension of meaning [Section 1.10.2] needs to be demonstrated before automated services will become possible in the web.

1.8 Architecture for product design services in the Semantic Web.

The architecture for services in the Semantic Web has not yet been determined. In this thesis, the architecture proposed is shown below:

Figure 1.7.2-1: Architecture for product design service in the Semantic Web

It consists of three virtual planes:

1. Service Access Plane (SAP)
2. DAML+OIL Agent Plane (DAAP)
3. Virtual Reference Plane (VRP)

1.8.1. *Service Access Plane (SAP)*

In the proposed service architecture, product development services are accessed by humans and/or by machine agents from the Service Access Plane (SAP). This virtual plane provides the 'user interface' (human or machine) for communicating with the DAML+OIL Agents. This plane may be conceptualized as a platform for inputting data or meaning that needs to be processed by the Semantic Web Agents and for receiving the processed data or meaning generated by the Agents. The SAP may be modeled with the Unified Modeling Language (UML) [42] Use Case Diagrams.

The output generated by the service can be in any suitable form, such as textual, engineering drawings, algorithms, equations, software, processes diagrams, parts list, bill of materials, methodology or machine control programs etc. At the human interface level, the solution generated by the service is expected to be in human readable form [Section 1.10]. All outputs into the SAP may not necessarily always require to be in human readable form in every instance, as for example, in the case of compiled object code generated for C++ program. However, when the output is to be directed to another machine, it must always be in machine-readable form.

1.8.2. *The DAML+OIL Agent Plane (DAAP)*

This proposed plane in the envisioned Semantic Web, contain the DAML+OIL agents that together forms a community of software agents that cooperate to process the product design problems received from the Service Access Plane (SAP). These agents have access to the Virtual Reference Plane (VRP) and recognize the ontologies of their respective domains that contain the corresponding Knowledge Base System (KBS) that the agents can utilize to do the analysis and generate valid solutions. Each agent may also advertise its service capabilities [Section 2.2.6] using Resource Description Framework RDF [7].

1.8.2.1. *Distributed processing & specialized agents*

The proposed architecture Figure 1.7.2-1 shows a community of agents. It is reasonable to assume that the complex set of design problems will need to be processed with distributed [39] software agents. In the development of AI based services, each agent can be designed to provide a specialized knowledge or process expertise as services to humans or to other machine agents.

1.8.3. *The Virtual Reference Plane (VRP)*

The intelligence in the Internet may be conceptualized to reside in the Virtual Reference Plane (VRP) containing the ontologies [Section 1.9.1], knowledge bases systems, constants, equations, algorithm, design rules etc.

1.9 Visualizing ontologies and knowledge domains in the web

The future ontologies and knowledge domains in the next generation Internet may be graphically visualized as shown in the figure below.

Figure 1.8.3-1 Graphical representation of different domains

Ovals: The ovals represent ontologies or knowledge domains

Overlapping ovals: The *overlapping* (intersection) ovals imply areas of knowledge or ontology that is common between two domains e.g. chemistry domain and electricity chemistry domain, where electrochemistry would be a common between the two.

Non-overlapping ovals: The non-overlapping (disjoint) oval represents disjoint domains e.g., Metallurgy and microbiology domains are unrelated domains.

Arrows: The *arrow* represents an element within a knowledge base system or ontology that can reference (or point to) another domain as a whole, or to a specific element within

that domain. The arrows represent URIs [9] (Uniform Resource Indicators) [Section 2.2.2].

1.9.1. *Different domains and ontologies in the Semantic Web*

The many different ontologies and knowledge domains that will be required in the Semantic Web may be graphically visualized as shown in the figure below:

Figure 1.9.1-1 Representations of many different domains in Internet

The Figure 1.9.1-1 shows many more domains compared to Figure 1.8.3-1 to imply the very large number of ontologies and knowledge domains that will be required to construct the envisioned Semantic Web. Dedicated software agents would access these ontologies in the Internet as a resource that aid in determining meaning and knowledge source.

1.10 Machine readability, machine comprehension and ontology

It is important to clarify the difference between machine readability, machine comprehension and ontology.

1.10.1. *Readability of electronic device specifications*

Presently, the electronic device datasheets files at the manufacturers web sites files are in one or more of the following formats: HTML, AcrobatTM PDF, MS WordTM, and PostScriptTM. These files are machine-readable only in the sense that the machine can recognize the alphanumeric characters, but the meaning in the document is not machine-interpretable. Electronic device specifications are written in natural language, with tables and graphs. Natural language is unsuitable for supporting machine comprehension of meaning. The algorithms that machines require to comprehend the meaning of information written natural language is at presently insufficiently developed to be reliable.

The reading activity does not necessarily result in the assimilation of meaning by the reader, be it human or machine. Machine readability is merely the ability of the machine to recognize the sequence of character strings and the format of the document but is often mistakenly assumed to imply machine understandability. Comprehension of meaning goes beyond machine readability.

1.10.2. *Machine comprehension of meaning*

Since the Semantic Web is expected to understand meaning, it is essential to clarify what is meant by *machine comprehension*. Comprehension of meaning involves making automatic references, forming association with other entities appropriately and includes the ability of machines to make logically correct assessment of a given condition about a previous, present or future outcome of a reality or abstraction. Meaning may also involve generating correct predictions of future results from a given set of conditions or the output of a valid question by the machine. Machine comprehension of electronic device specification may be described as the ability of the machine to read and understand the device characteristics, signals or parameters and forming automatic association with appropriate electronic theory or design rules. It may involve automatic parts selection, calculations, and software reconfiguration, linking, or binding with additional software resources, interfaces and knowledge bases that enhance the capability of the machine to generate valid design or to ask valid questions.

1.10.3. Relationship between readability, ontology & comprehension

The relationship between machine readability, ontology and machine comprehension may be diagrammatically represented as shown below:

Figure 1.10.3-1: Readability, Ontology and Machine Comprehension

In the context of Figure 1.10.3-1 ontology may be considered as a medium for facilitating machine comprehension of meaning. The diagram shows that at first, the electronic device data must arrive at the machine in machine-readable form. That data then has to be assembled and recognized to be in an ontological form before the determination of meaning can occur. Meaning determination may require additional support from the domain reference logic and Knowledge Representation (KR).

2. LITERATURE REVIEW

The basic concepts required for this research can be listed from the *key words* in the title of this research. These are:

- Semantic Web
- Ontology
- DAML+OIL
- Agent
- Services
- Product Development
- Electronic devices

2.1 Semantic Web

In the May 2001 issue of Scientific American, [1] in the article, “Semantic Web” T. Berners-Lee, J. Hendler, and O. Lasila described a new form of Web content that will convey meaning to computers. It is being envisaged that computers in the Internet of the future will process *meaning* rather than text. The Semantic Web is expected to unleash a revolution of new services based on Artificial Intelligence (AI). Software agents in the web would carry out sophisticated tasks for both humans and machines. It would be an

extension of the features available in the current Internet but with the addition, that information is given well-defined meaning. It will enable computers and people to work in cooperation.

Berners-Lee et al. identifies that the Semantic Web must provide a language to express both the data and the rules for *reasoning* about the data. It must also allow rules from any existing knowledge-representation (KR) base to be exported into the Web and must have access to collections of information and inference rules that can be used to conduct *automated reasoning*.

2.2 The Layer Cake Model of the Semantic Web

The article in the IEEE INTELLIGENT SYSTEMS March/April 2001 page 30 illustrates the Layer Cake Model of the Semantic Web shown below:

Source: IEEE INTELLIGENT SYSTEMS March/April 2001 page 30

Figure 1.10.3-1 Layer cake model of the Semantic Web.

This model was presented by Tim Berners-Lee at the XML 2000 conference for the Semantic Web. It is being supported by some of the leading researchers in Ontology and Semantic Web, such as McGuinness [28], Hendler, [20], McIllith [14] and others. It is assumed that the layer cake model will be designed with software agents. However, the Layer Cake Model [Figure 1.10.3-1] does not mandate the use of DAML+OIL agents.

The above diagram shows that the fundamental technologies for developing the Semantic Web are the Unicode [3], URI [9], eXtensible Markup Language (XML) [4], Namespaces [24], XML-Schema [5] [6], Resource Description Framework (RDF) [7], RDF-Schema [8], Ontologies, Logic, Proof and Trust.

2.2.1. Unicode

In the Semantic Web Model, at the lowest layer, at first all data (including engineering design data) will need to be translated into Unicode [3]. Unicode is a character coding system designed to support the worldwide interchange, processing, and display of the written texts of the diverse languages of the modern world. It supports classical and historical texts of many written languages. With respect to the envisaged product design services, all technical data to be processed by the Semantic Web service will first need to be converted to Unicode.

2.2.2. *Uniform Resource Indicator (URI)*

The Layer Cake Model [Figure 1.10.3-1] uses the URI standard at the lowest layer. The well-known Internet term, URL (Uniform Resource Locator) refers to a subset of the Uniform Resource Indicator (URI). RFC 2396 describes URI [9] as a compact string of characters for identifying an abstract or a physical resource. It can be used to point to the whole information or to any segment of the information segment (of any granularity) that are on the Internet. URI can also be used to point to object that are not physically connected to the Internet. URI defines the generic syntax, including both absolute and relative forms e.g. a paragraph chapter of a book. An absolute identifier refers to a resource independent of the context in which the identifier is used. In contrast, a relative identifier refers to a resource by describing the difference within a hierarchical namespace between the current context and an absolute identifier of the resource.

The URI mechanism of referencing will be supported by powerful mechanisms consisting of XPOINTER [11], XPATH [12] and XLINK [10]. These specifications can be used to link information resources in the Internet in unidirectional, bi-directional and in multi-directional manner. The powerful features of the URI can be used for macroscopic addressing e.g. to address a whole document or a web site or microscopic addressing, e.g. one alphabet on a page.

2.2.3. XML

XML (eXtensible Markup Language) is derived from simplified version of the well-established document-encoding standard in the publishing industry called SGML [84] (ISO 8879) and has been adapted by W3C [52] for use in the Internet. XML is both human and machine-readable language. The current Internet based on extensive use of HTML document transfer has fixed number of tags to indicate document format. In XML, tags are user definable that can be used to imply meaning rather than the format of the document. Any number of XML tags in a document can be mutually defined and agreed upon between the sender and the receiver of a document or by a standards body. Each institution may define its own XML tags for internal use within the organization or affiliations.

HTML tags indicate format, such as for bold and <i> for *italics etc.* but there are no tags to indicate the semantics of the words or sentence in a document. For example, there is no tag named <voltage> to imply that the text is about electricity. Unknown tags, such as, <water> are simply ignored by the Internet Browsers, like NetscapeTM and Internet ExplorerTM. In the Semantic Web <water> would automatically be associated with hydrogen and oxygen, river transportation, blood, clouds etc. It is intended that in the Semantic Web, XML tags like <voltage> would be automatically associated with the subject matter of electricity without the need for the users to specify such an association

each time. It could also be automatically linked with electrical Knowledge Base System (KBS) containing electrical theory, design rules and device ontologies. These resources would be in human and in machine-readable and comprehensible form.

XML documents are tree structured and consist of elements, starting from a root element. An element may contains tags, e.g. <name>, <address>, and a tag may have attributes (e.g. "university"). An example of XML document is shown below:

```
<?xml version="1.0"? encoding="UTF-8" ?>
<name institution = "university"> Carleton </name>
<address> 1125 Colonel By </address>
<city> Ottawa </city>
<postal-code> K1S 5B6 </postal-code>
```

Listing: 2.2-1: Example of an XML tags and attribute

Both SGML and XML are "meta" languages because they are used for defining markup languages.

2.2.4. *Namespace*

Namespaces [24] are used to indicate the context or the category under which certain terms are used. Namespaces in XML differ from the "namespaces" conventionally used in computing disciplines; the XML version has internal structure and is not a mathematical set. Namespace helps to resolve potential conflicts with duplicate names. Last name or family name can be considered as a namespace that may help to differentiate two people with identical first names. Namespaces can also be utilized implicitly for making the

meaning of certain terms dependant on the namespace under which it is used. As for example, the meaning of the word “frequency”, used in the context of circuit design usually pertains to sine or clock waves which involves time base, whereas in statistics, “frequency” usually means the number of events. Namespaces are essential elements in ontology. In device ontology, Namespaces [Section 3.4.2] can be used to group device characteristic under a common category, such as ‘analog’, ‘digital’, ‘thermal’, ‘mechanical’ etc

2.2.5. XML Schema

The structure of XML document was initially validated to check if the document was well-formed with another reference document called Document Type Definition (DTD), but DTD has many weaknesses and the problems were identified by many. For example, it was recognized by Boris Chidlovskii [13] that DTD has tightness problem in information retrieval in tree-structured document. The weakness of DTD is overcome by another standard called XML Schema [5] [6] developed by W3C. The XML Schema can be used for validation of the structure, format and type of the input data. The Schema is described in two documents listed below:

XML Schema Part 1: Structures [5] describes the facilities for defining the structure and constraints in the contents of XML documents, including the use of the XML Namespace facility. The schema language is itself represented in XML, it uses namespaces and extends the capabilities described in XML 1.0 Document Type definitions (DTDs).

XML Schema Part 2: Datatypes [6] defines the various data types that can be used in XML document. Schemas specify data types (e.g. integers, string etc) of textual elements and attributes.

2.2.6. RDF

The Resource Description Framework (RDF) Model and Syntax Specification [7], is considered to be an essential ingredient of the Semantic Web. It provides a foundation for metadata (metadata is data summary of data). Metadata can facilitate interoperability between applications that exchange machine-understandable information on the Web. The Service Applications can read the meta-data to determine if it is accessing the right resource that it requires. RDF can be used to advertise the meta-data of a resource and as an aid to search engines, software agents and humans to recognize the resource of interest in the Internet. RDF can be conceptualized as dealing with *resources*, *properties* of the resources and with *values* associated with those resources, as attribute /value pair entities.

In the context of Semantic Web, a resource is anything that can be represented by a URI [9]. RDF can provide the basic ontological primitives i.e. the basics for indicating meaning and relationships. RDF represents an abstract model, but it does not define any particular vocabularies [19]. The vocabularies are provided by XML for the concrete textual representation of the resource model.

The RDF Vocabulary Description Language 1.0 RDF Schema [8] describes how to use RDF to describe *RDF vocabularies*. It also defines basic vocabulary for this purpose and

conventions that can be used by the applications in the Semantic Web to support sophisticated RDF vocabulary description. RDF can be used in resource discovery, cataloging and in building the trustful web. RDF specification has the notion of data container, such as *bag*, *seq*, and *alt*. (These have been replaced by *List* in DAML+OIL).

2.2.7. Logic and Trust

The Logic layer in the Layer Cake Model is located on top of the Ontology layer. Tim Berners-Lee, states that the model allows: “*the deduction of one type of document from a document of another type; the checking of a document against a set of rules of self-consistency; and the resolution of a query by conversion from terms unknown into terms known*”.

It may be noted that one of the biggest difficulties for the Semantic Web community is to find interoperable ways of incorporating the inference rules into ontologies. There is a need for much research in this area.

The Proof and Trust relationship are located at higher layers in the layer cake model. Presently, these layers of the model are undefined and research work is currently mainly focused in the Ontology and Logic layer of the model.

2.3 DAML+OIL

The DAML+OIL (DARPA Agent Mark-up Language + Ontology Inference Layer) [23] is a semantic markup language for Web resources. This language is currently in research and development stage. The development included incorporation of RDF [7] and RDF Schema [8] specification and the adding of extensions to these specifications. DAML+OIL is being designed to support much richer variety of constraints and logical inference. It has rich modeling primitives and features that are commonly found in Frame Based System [62]. The DAML+OIL has a clean and well-defined semantics.

The Ontology Inference Layer is [60] is a common language for specifying and exchanging ontologies over the web. It provides decidable logical layer. The ontology portion of DAML+OIL is based on the subset of the original OIL (Ontology Inference Layer) specification. OIL is described in literature as having germinated from three roots, the Description Logic (DL) Frame Based System [62] and from Web Languages [4,5,6,7,8].

Figure 2.2.7-1 may be used to conceptualize the Ontology Inference Layer (OIL) as shown below:

Figure 2.2.7-1: Three roots of Ontology inference Layer (OIL)

Description Logic (DL) is a sub field of Knowledge Representation (KR). DL has formal semantics and reasoning support. It illustrates knowledge in terms of concepts and role restrictions that are used to derive classification taxonomies. In DL, classes are called *concepts*. Frame Based System [62] describes Epistemological Modeling Primitives.

Frame Based Systems: The idea of Frame originated in 1970's and it was described by Marvin Minsky [62]. It subsequently gained popularity as basic tools for representing knowledge and was further advanced by Chaudhri [65], Fikes & Kehler [63] and by Karp [64]. The basic idea of using a Frame System is very simple: A Frame can represent an object or a concept. A collection of attributes (or slots) can be filled initially with values and put in the frame. During use, the values in frame slots can be altered to make the frame correspond to a particular situation.

This background of OIL, indicates the potential built into DAML+OIL. The constructors of the Semantic Web and designers of product development services requires to know this background in order to utilize the potential of DAML+OIL and to efficiently develop ontologies and web services.

2.4 Ontologies

Ontology is hierarchically structured set of terms for describing a domain that can be used as a skeletal or a foundation for the knowledge base. The subject of ontology is the study of categories of things that exist or may exist in some domain. Ontologies provide a shared and common understanding of a domain between people and software. Ontology does not include a judgmental description of what is good or bad about a domain.

Several authors [20], [22], have identified that the definition of ontology in literature is inconsistent and even contradictory. Due to the uncertainty expressed in the academic literature on the meaning of ontology, an analysis was done in this thesis and results placed in the APPENDIX 1. In summary, it was discovered that a widely cited definition of ontology is Gruber's [29] definition: "*A specification of a conceptualization*". It should be realized that on the web, the meaning of ontology has a different slant from the classical philosophical notions of ontology.

Ontologies [31] are used in knowledge engineering, knowledge representation, qualitative modeling, language engineering, database design, information retrieval and extraction,

knowledge management and organization. It includes areas of library science and ontology-enhanced search. McGuinness [28] indicates that ontology should be tailored and developed as per the needs of its particular application.

Data languages and Knowledge Representation (KR) languages can express schemas that define the structure and constraints of data (such as instances or objects) conforming to the schema. A schema in a KR language is called ontology [67] [68] [69]. Ontology is an explicit, formal semantic model and does not include instances/annotations.

2.4.1. *Relation between Ontologies and Schema-languages*

Klein et al [18] described the relation between Ontologies and Schema-languages with the intent of translating the OIL-specifications in XML-schema [5]. The authors identifies that ontology provides a *domain theory* and is not a data container unlike XML-Schema, which can be thought of as ‘container’ for data. They illustrated that Ontology Inference Layer (OIL) is more expressive than the XML Schema is.

This concept stated by Klein et al [18] can be applied to extend the capabilities of the present XML-Schema based B2B (business to business) and enhance it towards AI based transactions.

2.4.2. Agents and the Semantic Web

James Hendler [20] indicates that ontologies on the web could significantly affect the use of Web services. It enable tasks to be performed more efficiently and with less human effort. Ontologies allow reuse and large ontologies can be created from combining small ones [45]. Hendler states that ontologies will be highly significant in the Semantic Web and agents, which will provide the services, will use ontologies extensively.

2.5 Agents

Computationally, an agent is a software robot or 'softbot' that can apply general guidelines in deciding how to respond to a specific situation. A large variety of definitions and description exist in the literature for software agents. Shoham [32] points out that there is no comprehensive definition of the notion of agent and that each person presents his or her version of the definition of agent. Weiss and Deugo [40] pointed out that agent system development has occurred on an *ad hoc* basis that has led to many problems. Weiss has argued strongly against use of an ad hoc model for agents.

Agents have properties such as *autonomy*, *adaptiveness*, *intelligence*, *awareness*, *mobility* and *anthropomorphism*. On going execution, is a property of the agent that indicates that unlike software routines (that are invoked when required), agents run continuously [32]. The *adaptiveness* feature enables agents to change their behavior to suite the preferences and behavior of individual users. Although this feature is attractive, it introduces

complexity and uncertainty because the same agent can behave differently after a period due to the new adaptation. Software based systems are usually highly sensitive to even minor changes that can have large negative effects. The risks vs. benefits of such self-adaptive features in the Semantic Web needs to be explored.

2.6 Unified Modeling Language (UML) and Ontologies

The Unified Modeling Language (UML) [42] is a graphical system modeling language. It can be used to describe the static features and dynamic behavior of systems. It was initially developed for conveying concepts from one human to another for describing systems that were to be developed in *object oriented programming languages*. UML is now widely adopted by the software engineering community. The scope of UML is now broadening to include diverse applications. Cranefield [41] states that UML although developed to support analysis and design in software engineering, is beginning to be used for other modeling problems including Meta Data Coalition [79]. Cranefield proposes application of UML to the Semantic Web and claims that UML class diagrams provide a static modeling capability that is well suited for representing ontologies and that UML object diagrams can be interpreted as declarative representations of knowledge.

Stephen Cranefield and Martin Purvis [78] illustrates how UML can be used as an Ontology Modeling Language and presents an ontology representation language based on a subset of the Unified Modeling Language together with its associated Object Constraint Language [42].

2.7 DAML-Services

Under the DARPA Agent Markup Language Program, Web Service Ontology (currently named DAML-S) is being developed to support tools and agent technology to enable automation of services on the Semantic Web. The draft DAML-S [56], builds on top of the current DAML+OIL specifications with a core set of markup language constructs for describing the properties and capabilities of Web services in computer-interpretable, unambiguous form. It will facilitate the automation of Web *service discovery, service execution, interoperation, composition and monitoring*. DAML-S document also describes profile, process, and service grounding. It is assumed that in the future, product design services in the Semantic Web could be constructed based on the DAML-S specifications.

2.8 Product development on the web

The concept of using the Internet as a product development environment is becoming increasingly popular among academicians and in the industry. For example, at MIT, Wallace [81] et al believes that Internet based Computer-aided design (CAD) tools will evolve enterprise-wide to facilitate the publishing of live design services. A prototype system called DOME is a pilot study with Ford Motor Company that is being used to illustrate the concept of Web based distributed design service that could provide a

marketplace for virtual environment for product development. The development of the Semantic Web and DAML+OIL can augment and extent the efforts such as the DOME project to a more global virtual environment.

2.9 Electronic Device specification documents

A survey of electronic device manufacturers web sites and a review of their device specifications [APPENDIX 3], catalogs [66] and application notes show that the information is published largely in human readable form and not in machine readable or machine interpretable form. Therefore, these on-line documents cannot be readily deployed in the construction of the envisioned electronic product design services for the Semantic Web. It is also realized from the review that conversion of the device specification documents into machine-interpretable ontological form is highly complex.

2.10 Other initiatives in machine readability of device specification

A number of technical standards currently exist for machine-readability of the electronic component specification. These standards contribute to a certain extent towards electronic product design automation. Some of these standards, like RosettaNetTM also support automatic B2B (business to business) and technical data transactions between manufacturers, suppliers, vendors, designers and users. The common machine-readable electronic standards are from organizations like, SI2 [72], EDIF [90], RosettaNet [71],

and Electronic Design Automation EDA [91]. It is pointed out [Section 1.10.2] that machine readability should not be confused with machine interpretability of meaning.

The motivation of these organizations may be compared with the main theme of this thesis, in terms of the goal, approach and languages.

2.10.1. *Goal of machine readability*

The technical goal of this thesis and those of the above-mentioned organizations are common, i.e. to achieve machine readability of device specification to support design automation. The additional goal of this thesis is to enhance machine interpretability of the device specifications that can be facilitated by ontological representation of the device properties. Ontological representation is required for the Artificial Intelligence (AI) based envisioned services in the Semantic Web.

In this thesis, a high-level approach to device ontology is taken. This approach can augment Knowledge Representation (KR) as it facilitates better abstract interfaces to Knowledge Base Systems (KBS) and AI. The approach in this thesis is to contribute to design automation due to machine *reasoning* rather than through *rigid solutions*. The approach of the above mentioned organizations can be considered 'rigid solutions' that are almost at low machine-level (e.g. transistor and digital logic gates level). This is obvious from the fact that some of the offered solutions to design automation are downloadable intellectual property (IP) from proprietary design libraries. The approach in this thesis is not based on such "pre-canned" solutions but is based on the philosophy

that open knowledge on the web and intelligent reasoning by the machine should generate the proper product design solutions.

2.10.2. *Platform*

In this thesis, the platform for machine readability and machine comprehension of device data is hardware and software platform independent. In contrast, the solution platform from the above mentioned standards organizations are tightly bound to particular operating systems (OS) like UNIX, and programming languages like C++. Their solution may also require source code download, APIs (Application Programming Interface) and parameter passing between Application software. In this thesis, the support for automated services is in the context of developing device ontologies independent of source code, programming language, and compiler and O.S. platforms.

2.10.3. *DAML+ OIL ontologies vs. Si2 Open Access*

The focus of Si2 [72], OpenAccess is on Integrated Circuit (IC) design. The OpenAccess API is a C++ level program interface to IC design data. It provides a low-level interface to databases i.e. in terms of source code where API's may be strongly data typed. The OpenAccess [72], enables unambiguous exchange of electrical component information, comprehensive dictionary of component properties and computer interpretable timing diagrams. The DAML+OIL based ontologies for devices in this thesis are not necessarily focused on IC design. Unlike, the Si2 [72] approach, there are no concepts of APIs, databases etc. in DAML+OIL.

2.10.4. DAML+OIL device ontologies vs. RosettaNetTM

The goal of RosettaNetTM [71] which merged with the Uniform Code Council, Inc. (UCC) [92] is to “*create open e-business process standards that form a common e-business language, aligning processes between supply chain partners on a global basis*”. RosettaNetTM standards are aimed at supply chain, including manufacturers, distributors, resellers, shippers and end users. RosettaNetTM is B2B (business to business) orientated, whereas the main objectives of this thesis [Section 1.4] are to develop device ontologies in DAML+OIL, which may only indirectly contribute to B2B transactions.

Examination of the excerpts of RosettaNet ITTD shows that RosettaNetTM function at XML and XML-Schema level. With respect to the layer cake model [Figure 1.10.3-1] of the Semantic Web, RosettaNetTM is limited to XML-Schema level and below and does not reach the ontology level of the layer cake model. This means that RosettaNet does not describe electronic devices in ontological terms as DAML+OIL can (as shown in this thesis). Therefore, it can be assumed that RosettaNetTM does not provide interface to Knowledge Base System (KBS) in the same manner as DAML+OIL can which inherently supports such abstract interfaces. The class the constructors and axioms [Ref: Table 3.1-2] that DAML+OIL provides support to KBS which is lacking in RosettaNetTM.

RosettaNetTM defines electronic specification in terms of XML tags, whereas in DAML+OIL, the device specification parameters can be declared in XML and as well as with DAML+OIL property ID, using `daml: ObjectProperty` [Section 3.5.6]. DAML+OIL can relate device properties to other properties using `daml: subPropertyOf` etc. As will be obvious from the subsequent sections of this thesis, DAML+OIL offers additional advantages to relate signals at high-level relationships.

However, it may be possible to use the RosettaNetTM infrastructure advantageously in the DAML+OIL based ontologies because the low-level specification of the devices requires support of XML and XML Schema, which, RosettaNetTM uses extensively.

2.10.5. *DAML+ OIL device ontologies vs. EDIF*

EDIF [90] is a widely used electronic design interchange format. EDIF files are machine-readable and may be interchanged electronically. Computer Aid Design (CAD) systems that can read EDIF are often referred to as Electronic CAD (ECAD) systems or Electronic Design Automation (EDA) Systems. These are low, or near machine level code, whereas the DAML+OIL approach is high level, where electronic devices are an abstraction with certain properties.

2.10.6. *DAML+ OIL device ontologies vs. EDA*

The philosophy of Electronic Design Automation EDA [91] consortium is different from this research. Whereas, EDA provides tools for electronic design that can be downloaded,

the approach in this thesis is *the inverse of the EDA concept* of using downloads for on-line services. In the Semantic Web, the intelligence to do design would reside in the web *without requiring the need to download the design tools*. In EDA, most tools are low levels ad hoc tools developed by different tool vendors to increase the individual engineer's productivity, such as circuit design, design for test tools, fault coverage and performing debugging work etc. One of the ideas behind the Semantic Web is to have distributed intelligence and to make the web a virtual Engineering Workstation rather than only a source of downloadable tools.

2.11 Modeling XML Applications with UML

David Carlson [75] showed how to integrate XML with object-oriented technology UML (Unified Modeling Language) [42] to create dynamic, interactive Web applications. The application example that Carlson described can be referenced with the layer cake [Figure 1.10.3-1] model of the Semantic Web. It can be noted that such web applications extends only up to the XML-Schema level of the layer cake model. It does not include the RDF layer and above. Such applications may be sufficient for the bulk of B2B transactions, but it will not have the advantages of RDF based metadata and ontology based interfaces to Knowledge Base Systems (KBS).

2.11.1. XML Metadata Interchange (XMI)

XMI specification [74] is a specification for generating DTD from meta-model. XMI is also a specification for creating XML documents from the models that are instances of

meta-models. The document creation rules are based on Meta Object Facility (MOF). XMI can provide support in the development of electronic design services in the Semantic Web.

2.11.2. *Mapping UML to XML*

UML [42] as a modeling tool may be used for the *software engineering* required to design the envisaged services in the Semantic Web. For the services, UML models of electronic device catalogs may need to be created on the web sites with a set of vocabularies that describe the parameters of the device. It was observed that different device manufacturers often use different vocabularies to name, describe and specify their products. It can therefore be assumed that the services in the Semantic Web will require handling of many different sets of vocabularies that may be different even for the same class of devices.

Carlson [75] has shown with examples how the different vocabularies can be transformed from one style to another using XSLT [16] and how UML class diagrams can be mapped to XML.

An attempt is made below to apply Carlson's example of a catalog to a hypothetical electronic device catalog for use in the Semantic Web.

Figure 2.11.2-1: Simple electronic device catalog model in UML

2.12.1. Mapping UML Model of a electronic device catalog to XML

Figure 2.12.1-1 : Mapping UML model of device catalog to XML

It can be observed from Carlson's [75] example (modified here for the hypothetical electronic device catalog) that the UML model for electronic catalog maps quite well into XML.

3. PROTOTYPE ONTOLOGIES FOR ELECTRONIC DEVICE IN DAML+OIL

Electronic devices have *static*, *dynamic*, *thermal* and *mechanical* characteristics. These characteristics will need to be represented in ontological form for electronic design service in web. The ontology of electronic devices must not only include the vocabulary for the device parameters, but also indicate the relationships between them. The relationship can be shown using DAML+OIL Class Constructors and Axioms.

3.1 DAML+OIL Class Constructors and Axioms

Constructor	Abbreviation	Potential use in device ontology
intersectionOf	$C1 \wedge \dots \wedge Cn$	Magnetic $\wedge$ Electric e.g. electro-magnetic
UnionOf	$C1 \vee \dots \vee Cn$	Resistors $\vee$ Capacitors
complementOf	$\neg C$	Software $\neg$ Hardware
One of	$\{x1 \dots xn\}$	$\{input1, input2, input3 \dots inputn\}$
ToClass	$\forall P.C$	$\forall gain.Amplifier$
hasClass	$\exists P.C$	$\exists attenuation.channel$
hasValue	$\exists P.\{x\}$	$\exists valueOf.\{Register\}$
minCardinalityQ	$\geq nP.C$	$\geq 3 terminals.transistors$
maxCardinalityQ	$\leq nP.C$	$\leq 8bits.ByteRegister$
cardinalityQ	$= n P.C$	$= 1 Value.Register$

Table 3.1-1 : DAML+OIL Class Constructors

3.1.1. DAML+OIL Axioms

Axiom	Abbreviation	Potential use device ontology
subClassOf	$C_1 \subseteq C_2$	Capacitors $\subseteq$ PassiveDevices
sameClassAs	$C_1 \equiv C_2$	Insulator $\equiv$ dielectric
subPropertyOf	$P_1 \subseteq P_2$	Bandwidth $\subseteq$ Channel
samePropertyAs	$P_1 \equiv P_2$	Light $\equiv$ Electro-magnetic-waves
sameIndividualAs	$x_1 \equiv x_2$	Voltage $\equiv$ ElectroMotiveForce
disjointWith	$C_1 \subseteq \neg C_2$	Wireless $\subseteq \neg$ Fiber-Optics
differentIndividualFrom	$x_1 \subseteq \neg x_2$	Capacitor $\subseteq \neg$ Resistor
InverseOf	$P_1 \subseteq P_2^{-}$	transmit $\subseteq$ receive-
transitiveProperty	$P_1^{+} \subseteq P_2$	subchannel $^{+} \subseteq$ channel
uniqueProperty	$\text{Thing} \subseteq \leq 1P$	amplitude $\subseteq \leq$ sine-wave
UnambiguousProperty	$\text{Thing} \subseteq \leq 1P^{-}$	amplitude $\subseteq \leq$ sine-wave ⁻

Table 3.1-2 : DAML+OIL AXIOMS

[REF: APPENDIX 7: SUMMARY OF DAML+OIL LANGUAGE ELEMENTS]

3.2 Approach to electronic device ontology development

Currently, no literature exists that illustrates an approach to develop ontology for electronic devices. Fonseca et al [88] has discussed that ontology can be constructed at various levels of granularity. This principle may be applied to develop ontologies for complex devices, where the device functionality may be viewed at different levels of

granularity. The idea of creating large ontologies [45] by combining smaller components is not unique to the Semantic Web.

3.2.1. *Levels in electronic device ontology development*

It is claimed that ontologies are classified [88] in four general groups, according to their dependence on a specific task or point of view. *Top-level ontologies* describe very general concepts. *Domain ontologies* describe the vocabulary related to a generic domain, for example optical, wireless, biology etc. *Task ontologies* describe a task or activity, such as specification interpretation or requirement assessment. *Application ontologies* describe concepts that depend on both a particular domain and a task, and are usually a specialization of them. Application ontologies are created from combination of high-level ontologies and the user needs.

In the context of software agents, Wooldridge and Nwana [86] classify ontologies in three categories:

- Ad hoc ontologies
- Global ontologies
- Standard ontologies

The potential use of these categories may be considered for the development of electronic device ontologies.

3.2.2. *Ad hoc approach to device ontology development*

An ad hoc approach would be to develop the ontology for devices in whichever way one likes. With the ad hoc approach, a huge disparate vocabulary is likely to result with many duplicates or even with contradictory names of device signals. The ad hoc approach may lead to unmanageable large varieties of incompatible vocabularies and ontologies. Clearly, this approach would be inefficient and confusing for interoperability between software agents in the Semantic Web. Hendler [20] warns about the possibility of the Semantic Web becoming “...a complex Web of semantics ruled by the same sort of anarchy that rules the rest of web”. Ad hoc device ontology development would also make device search and comparisons by software agents an extremely difficult task.

3.2.3. *Global ontologies*

Global ontologies are not domain specific and can be used in many domains. An example of a global ontology is CYC [89] that was developed for building common sense knowledge. It contains concepts and common sense axioms. The applicability of CYC axioms in the development of device ontology is not considered, because the focus of this thesis on the potential use of DAML+OIL in device ontologies.

3.2.4. *Standard ontology based on classes of devices*

This approach appears to be more suited for the development of device ontology since it involves categorization [33] according to some standard. Compared to the ad hoc approach, ontologies based on classes of devices can vastly reduce the vocabulary required to construct the ontologies, which have consistency in the semantics. In the *standards* based approach, devices with similar characteristics or technology may be grouped into a common class. Device can be classified according to application, function, hardware or firmware, physical dimensions, source or on a particular parameter e.g. micro voltage.

Classes can be grouped under a common namespace [24], for example, the “amplifier” class, or the “digital” class etc. In the electronic industry, devices are usually classified as:

- active and passive
- analog, digital or mixed signal
- Discrete or Integrated Circuits
- High and low voltages
- ASIC or Commercial
- Optical

Since the language elements of DAML+OIL [23] do not inherently have the notion of subject matter, well-developed ontologies can facilitate the construction of services in the web by providing support to the software agents to interpret the device characteristics and parameters.

A tree diagram may be constructed to represent the classification and properties of electronic devices. In the context of DAML+OIL, where classes can be subclasses of two or more super classes, the tree diagram is not the best graphical method for showing the classes, subclasses, properties and sub properties of devices. Branches from multiple super classes may need to be merged to a subclass lower in the hierarchy, resulting in a diagram that will not remain a tree diagram anymore. For simplicity in this research, the tree diagram is used.

3.3 Electronic devices as a tree of DAML+OIL classes

Figure 3.2.4-1: Tree Diagram for subclasses of electronic devices

3.4 Device ontology development experiment

It can be realized from the examination of the device data sheets [Appendix 8] that it is extremely difficult to translate the functionality of devices totally into ontological form, Therefore, in this thesis the approach taken is to experiment at developing partial ontologies, rather than at attempting to construct a complete useable ontology.

3.4.1. *Setting up Namespaces for devices*

In device ontology development, each namespace can signify a classification or a domain, IDs, or the context in which the terms, classes or properties of devices are used. Namespace also helps to isolate the possible duplicate names and avoid potential confusion. The namespaces declared below are: rdf (Resource Description Framework), rdfs (RDF Schema), xsd (XML Schema definition) davps (device application), dvfnc (device function), dvtyp (device type), dvfw (device firmware).

Some of the namespaces used below, are located at Carleton University server where laser device ontology developed under this thesis was placed.

[Ref: www.sce.carleton.ca/~siahmed/laser]

```

<?xml version="1.0" encoding="ISO-8859-1" ?>

rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:xsd="http://www.w3.org/2000/10/XMLSchema#"
xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
xmlns:dvaps="http://www.sce.carleton.ca/~siahmed/deviceapps.daml#"
xmlns:dvfnc="http://www.sce.carleton.ca/~siahmed/devicevaps.daml#"
xmlns:dvtyp="http://www.sce.carleton.ca/~siahmed/devicetyp.daml#"
xmlns:dvthw="http://www.sce.carleton.ca/~siahmed/devicethw.daml#"
xmlns:dvfw="http://www.sce.carleton.ca/~siahmed/devicefw.daml#"
xmlns="http://www.sce.carleton.ca/~siahmed/laser.daml#"
>

```

Listing: 3.4-1 Namespaces for device ontology

The fact that an excerpt in a DAML+OIL document is Ontology can be declared by using the language element **daml: ontology**

```

<daml : Ontology rdf : about = "" ">
<daml:versionInfo> Oct 2002 </daml:versionInfo>
<rdf:comment> Ontology for electronic devices </rdf:comment>

```

Listing: 3.4-2 Ontology declaration in DAML+OIL

The rdf: about attribute shown in the above XML element daml: Ontology is empty (it does not contain a URI) to imply that the subject of this assertion is this document and not some other document in the web. The next line, <rdfs: comment> element is for documentation purposes only, for the human readers, for example, developers of the code for the Semantic Web services. Here, the comment is in plain English but it can be in any natural language.

3.4.2. Defining device classes and properties in DAML+OIL

In DAML+OIL, instances of a class can also be a class or subclass. In other Object Orientated languages, instance of a class is an object. In DAML+OIL, classes and properties can be defined independently of each other. Properties can later be associated with appropriate classes. For example, an electronic property, such as “*Gain*” can be selectively assigned only to certain subclasses of devices such as Amplifiers, as required [Diagram 3.8.5-1]. The *Gain* property should not be assigned to passive devices, such as resistor, capacitors since electronic theory proves that such devices cannot have voltage gain.

3.4.3. Declaring universal class ‘Devices’

With reference to the tree, diagram constructed in Figure 3.2.4-1, at first, a universal class called “Devices” is declared at the root of the tree. This includes all possible types i.e. sub classes of electronic devices.

```
<daml:Class rdf:ID="Devices">
<rdfs:label> Devices </rdfs:label>
<rdfs:comment> This universal class include all types of devices </rdfs:comment>
</daml:Class>
```

Listing: 3.4-3 Universal Device class

For simplicity, only two lines are introduced in the above definition of the class, a *label* and a *comment*.

shows that the Active and Passive devices are subclasses of the root class. These can be declared in DAML+OIL as shown below.

3.4.4. *Active Devices class*

```
<daml:Class rdf:ID="ActiveDevices">  
  <rdfs:subClassOf rdf:resource="#Devices"/>  
</daml:Class>
```

Listing: 3.4-4 ActiveDevices class

The # indicates that Devices class is already defined elsewhere within the current document and not in at another location in the Internet. In DAML+OIL, it is legal to declare a class as a subclass of class that is defined on another web site, in which case the # would be replaced with the full URI [9]. This means that the ontology infrastructure could span globally and different segments of the ontology can reside anywhere in the Internet. The URI mechanism can link the various segments of the ontology.

(In this thesis it is suggested that in the Semantic Web, the DAML+OIL classes for electronics and telecommunication ontology reside in standards bodies, like the ITU [49], IEEE etc. that are connected to the Internet in order to maintain universal consistency).

Although, intuitively one may associate electrical properties in the mind with the term “ActiveDevices”, machines following the DAML+OIL rules will not assign any property to the class until properties are explicitly associated with that class.

Therefore, in the above class definition of an electronic device, no electrical properties will be automatically assumed based on the class definition [Listing: 3.4-4] above. In DAML+OIL, the relationship expressed above, is logically the same as “*man is a subclass of the humans class*”, since DAML+OIL has no notion of content. Therefore, the class and subclass relationship is valid for other subclass relationships shown in subsequent sections of this chapter.

3.4.5. *Passive Devices class*

Passive devices class needs to be distinguished from active devices. This can be done as shown next.

```
<daml:Class rdf:ID="PassiveDevices">
  <rdfs:subClassOf rdf:resource="#Devices"/>
  <daml:disjointWith rdf:resource="#ActiveDevices"/>
</daml:Class>
```

Listing: 3.4-5: PassiveDevices class

Compared to the “ActiveDevices” declared earlier, **daml: disjointWith** is associated with in the “PassiveDevices” declaration in order to distinguish it from “ActiveDevices”. The relationship expressed with the use of daml: disjointWith language element in DAML+OIL is essentially of the same logic as for example, “women are subclass of the humans class but distinct from the man class”. This implies that the same software code segment in the Semantic Web that process the ActiveDevices as subclass of Devices can also be used for processing humans vs. women relationship.

3.4.5.1. Resistors class

Electrical resistors are categorized as a passive device element and may be declared as follows:

```
<daml:Class rdf:ID="Resistors">  
  <rdfs:subClassOf rdf:resource="#PassiveDevices"/>  
</daml:Class>
```

Listing: 3.4-6: Resistors class

3.4.5.2. Inductor class

Electrical inductors are categorized as a passive device element.

```
<daml:Class rdf:ID="Inductors">  
  <rdfs:subClassOf rdf:resource="#PassiveDevices"/>  
</daml:Class>
```

Listing: 3.4-7 Inductors class

3.4.5.3. Capacitor class

Electrical capacitors are also categorized as a passive device element and therefore can be represented as:

```
<daml:Class rdf:ID="Capacitors">  
  <rdfs:subClassOf rdf:resource="#PassiveDevices"/>  
</daml:Class>
```

Listing: 3.4-8: Capacitors class

3.4.6. *Distinguishing between Resistors, Capacitors, Inductors*

It will be necessary to indicate to the software agents in the Semantic Web that will process the device ontology that although Resistors (R), Capacitor (C) and Inductors (L) all belong to the same passive device class, they are distinct. This distinction can be stated in compact DAML+OIL idiom form by using **daml: collection** (to group items) and **daml: disjoint** language elements [48] as shown next:

```
<daml:disjoint rdf:parseType="daml:collection">
  <daml:Class rdf:about="#Resistor"/>
  <daml:Class rdf:about="#Inductors"/>
  <daml:Class rdf:about="#Capacitors"/>
</daml:disjoint>
```

Listing: 3.4-9 : Distinguishing between RLC in DAML+OIL

3.4.7. *Class of devices with RLC*

Certain electronic devices, such as *passive filters* are constructed from integration of resistors, capacitor and inductors. The class of such devices may be given any suitable ID, say "RLC". In DAML+OIL, it may be declared using **daml :intersectionOf** as follows:

```
<daml :class rdf:ID="RLC ">
  <rdfs:subClassOf rdf:resource="#PassiveDevices"/>
  <daml :intersectionOf rdf:parseType="daml:collection">
    <daml :Class rdfs : about ="#Resistors"/>
    <daml :Class rdfs : about ="#Inductors"/>
    <daml :Class rdfs : about ="#Capacitors"/>
  </daml :intersectionOf>
</daml:class>
```

Listing: 3.4-10 RLC Class

The above does not necessarily imply that the RLC class has the properties of an electrical filter. It simply indicates to the human reader or machine agents that a category of device is created with an ID “RLC” that includes all three classes of devices Resistors, Inductors and Capacitors defined earlier. It does not indicate how the three devices are electrically interconnected nor give any indication of many there are of each type. To associate the properties of electrical “filter”, additional constructs would be required, including the ontology for time [73] and frequency.

3.4.8. *PassiveDiode*

Passive diodes are categorized as a passive electronic device element. It may be represented as shown below:

```
<daml:Class rdf:ID="PassiveDiode">
<rdfs:subClassOf rdf:resource="#PassiveDevices"/>
<daml:disjoint rdf:parseType="daml:collection">
  <daml:Class rdf:about="#Resistors"/>
  <daml:Class rdf:about="#Inductors"/>
  <daml:Class rdf:about="#Capacitors"/>
</daml:disjoint>
</daml:Class>
```

Listing: 3.4-11 PassiveDiode class

The above states that PassiveDiode are subclass of PassiveDevices, but they are different from Resistors, Capacitors, and Inductors. This makes the declaration of “PassiveDiode” class more distinct and adds value towards machine comprehension of meaning.

3.5 Properties of electronic devices in DAML +OIL

A survey of the catalogs and data sheets of electronic devices show that in the device specification document, at first, usually a summary is provided followed by details of signal names, mechanical, electrical and thermal characteristics. [REF: APPENDIX 3 : DATA SHEETS OF ELECTRONIC DEVICES]

3.5.1. *Device Meta Data and RDF*

The summary section (or the introductory) section of the device specification document, as for example in the PhillipsTM Semiconductor [Section 8.1] may be regarded as meta-data of the device. Meta data is some information about data. The introductory summary portion could be a candidate for representation as metadata in terms of RDF [7]. RDF can facilitate the search by meaning (rather than text) for future search engines in the Internet. However, it must be recognized that much research is needed to determine how electronic device metadata can be expressed in terms of RDF. RDF is currently in research stage.

3.5.2. *Device Property and property datatype*

In DAML+OIL, property is a binary relation between two entities. Recently, the daml property was split into two: daml: ObjectProperty and the daml: DatatypeProperty. The

data type property not only indicates the type (e.g. integer, decimal, string etc) but also the constraints (e.g. if the value should be unique).

The device signal names for mechanical, thermal and electrical characteristics (AC and DC) can be regarded as *device properties* and can be represented using the daml: ObjectProperty language element of DAML+OIL. Representation of devices properties in DAML+OIL is attempted here, starting with a tree diagram drawn to show the classification of properties. Following McGuiness [28] recommendation that ontology development should be consistent with real world categorization, the properties as shown in the Figure 3.5.2-1 is consistent with the way most electronic device manufactures describe the specifications of their products. Like classes, electronic device properties may be *sub property* of several other properties and therefore the tree diagram is not the best way to represent properties of electronic devices.

However, in this thesis, for simplicity, a tree diagram is used and is illustrated below:

Figure 3.5.2-1: Tree diagram for electronic device properties

3.5.3. Device property

The properties of electronic devices may be described in a hierarchical manner [Figure 3.5.2-1] with a root property that includes all properties of the devices. It can be written as shown below:

```

<daml:ObjectProperty rdf:ID="DeviceProperty">
  <rdfs:domain rdf:resource="#Devices"/>
  <rdfs:range rdf:resource="#Devices"/>
</daml:ObjectProperty>

```

Listing: 3.5-1: Device property

The above simply assigns an ID name, “DeviceProperty” in the domain Devices class [Listing: 3.4-3] that was created earlier. The range of “DeviceProperty” as declared above covers the entire Devices class.

3.5.4. Survey of device mechanical properties

The figure next is taken from National Semiconductor web site. It shows a segment of the PC16550 [Section 8.2] device data sheet.

Figure 3.5.4-1 : Example of Mechanical properties of PC 16550 chip

The above figure is a typical example of how the mechanical properties of electronic devices are illustrated in the industry. The mechanical dimensions shown above are required, for example, in Printed Circuit Board (PCB) layout. For the envisaged product

design services in the Semantic Web (such as automated PCB layout), software agents must be able to determine the mechanical dimensions shown in the above figure. This means that the electronic device ontology must include the mechanical dimension as a part of the device ontology.

However, it can be observed that most mechanical drawings as currently drawn by the device manufacturers are for human viewing and are not usually created for machine interpretability. Unlike the list of electrical signal names, there are no or very few labels for the mechanical dimensions of the devices. For example, in the drawing there are no labels such as, <length>, <thickness>, <pin-spacing> etc. For machine readability and for ontological representation of the mechanical properties, labels or ID would be required for each of the mechanical parameter of the device. The mechanical properties can be represented, as ontology of it's own and later integrated into ontology of the device.

In Figure 3.5.2-1 a node is created under which all the mechanical dimensions of electronic devices can be listed. In DAML+OIL, this node can be written as shown below:

```
<daml:ObjectProperty rdf:ID="MechanicalProperty">  
<rdfs:subPropertyOf rdf:resource="#DeviceProperty"/>  
<rdfs:range rdf:resource="#Devices"/>  
</daml:ObjectProperty>
```

Listing: 3.5-2: Device Mechanical Properties

In the above, the domain and range is also set to 'Devices' class to mean that mechanical properties are applicable to all electronic devices. A set of vocabulary needs to be created

for the mechanical parameters. These may be expressed either as XML tag or as daml:ObjectProperty.

3.5.5. Device Thermal Properties

Electronic Devices have thermal properties, such maximum, minimum and ambient temperatures. An example of thermal properties for the PC 16550 chip [Section 8.2] is shown below:

1.0 Absolute Maximum Ratings		
Temperature Under Bias	0°C to +70°C	<i>Note: Maximum ratings indicate limits beyond which permanent damage may occur. Continuous operation at these limits is not intended and should be limited to those conditions specified under DC electrical characteristics.</i>
Storage Temperature	−65°C to +150°C	
All Input or Output Voltages with Respect to V _{SS}	−0.5V to +7.0V	
Power Dissipation	1W	

Table 3.5-1: Example of Thermal Properties of device

Device thermal property in DAML+OIL can be declared as shown below

```
<daml:ObjectProperty rdf:ID="ThermalProperty">
<rdfs:subPropertyOf rdf:resource="#DeviceProperty"/>
<rdfs:range rdf:resource="#devices"/>
</daml:ObjectProperty>
```

Listing: 3.5-3: Thermal Property of device

The sub properties of “ThermalProperty” can be noted from Table 3.5-1. The parameters listed can be used as thermal properties and each considered a sub property of the ThermalProperty. The daml:DatatypeProperty can be used to define the type of each sub-property.

The daml: ObjectProperty ID can be associated with “temperature-under-bias”, “storage-temperature”, “input-voltage-wrt-vss”, “output-voltage-wrt-vss”, “power-dissipation”.

Alternatively, the XML tags may be used for each thermal property, as shown below:

```
<temperature-under-bias>, <storage-temperature>, <power-dissipation>
```

Using XML tags makes it more private but it becomes difficult to add more sub properties to the property.

3.5.6. *Electrical property*

With respect to the tree diagram in Figure 3.5.2-1, a node in the tree is created for all the electrical properties, as a part of the ontology development. In DAML+OIL, it can be written as:

```
<daml:ObjectProperty rdf:ID="electricalProperty">  
<rdfs:subPropertyOf rdf:resource="#DeviceProperty"/>  
<rdf:range rdf:resource="Devices"/>  
</daml:ObjectProperty>
```

Listing: 3.5-4: Electrical property of the device

In the above expression, what exactly are the electrical properties are not defined, but only an ID is created that can later be used to append the properties by using rdf:about= “#electricalProperty”.

3.5.7. *Ohms Law electrical property*

```
<daml:ObjectProperty rdf:ID="OhmsLaw">  
<rdfs:subPropertyOf rdf:resource="#ElectricalProperty"/>  
</daml:ObjectProperty>
```

Listing: 3.5-5: Ohm's Law Electrical Property of the device

The above property does not enunciate what Ohm Law is, but only creates an ID to which Ohms law may be attached.

All electronic devices must have voltage, current and impedance properties. These properties may be declared in DAML+OIL as follows:

3.5.8. *Voltage property*

```
<daml:ObjectProperty rdf:ID="Voltage">  
<rdfs:subPropertyOf rdf:resource="#electricalProperty">  
<rdfs:domain rdf:resource="#devices"/>  
<rdfs:range rdf:resource="#devices"/>  
</daml:ObjectProperty>
```

Listing: 3.5-6 : Voltage property

3.5.9. *Current property*

```
<daml:ObjectProperty rdf:ID="electricalCurrent">  
<rdfs:subPropertyOf rdf:resource="#electricalProperty">  
<rdfs:domain rdf:resource="#devices"/>  
<rdfs:range rdf:resource="#devices"/>  
</daml:ObjectProperty>
```

Listing: 3.5-7 : Electrical current property

3.5.10. *Impedance property*

Impedance can be defined as a property of all electronic devices, therefore `rdf:range` can be set to `#devices` to imply that is applicable to all devices defined under the `#devices` class declared earlier.

```
<daml:ObjectProperty rdf:ID="hasImpedance">
<rdf:subPropertyOf   rdf:resource="#electricalProperty">
<rdfs:domain rdf:resource="#devices"/>
<rdfs:range rdf:resource="#devices"/>
</daml:ObjectProperty>
```

Listing 3-1: Impedance Property

The above assigns “hasImpedance” property to the entire device class.

3.5.11. *Resistance Property*

In terms of DAML+OIL it can be written as shown below:

```
<daml:ObjectProperty rdf:ID="hasResistance">
<rdf:subPropertyOf rdf:resource="hasImpedance"/>
<rdfs:domain rdf:resource="#Devices"/>
<rdfs:range rdf:resource="#Devices"/>
</daml:ObjectProperty>
```

Listing: 3.5-1 Resistance Property

3.5.12. *Inductance Property*

Similarly, inductance can be defined as sub property of the impedance property and applicable to all devices.

```
<daml:ObjectProperty rdf:ID="hasInductance">
<rdf:subPropertyOf rdf:resource="hasImpedance"/>
<rdfs:domain rdf:resource="#Devices"/>
<rdfs:range rdf:resource="#Devices"/>
</daml:ObjectProperty>
```

Listing: 3.5-8: Inductance Property

3.5.13. *Capacitance Property*

Similarly, capacitance can be defined as sub property of impedance, which is not necessarily limited to the Capacitors class.

```
<daml : ObjectProperty rdf:ID="hasCapacitance">  
<rdfs : subPropertyOf: rdf:resource = "hasImpedance"/>  
<rdfs:domain rdf:resource = "#Devices"/>  
<rdfs:range rdf:resource="#Devices"/>  
</daml:ObjectProperty>
```

Listing: 3.5-9 Capacitance Property

Here the domain and range is set to Devices because many devices exhibit property of capacitance and as defined here, the capacitance property is not limited to capacitor class only. It may also be defined to be limited to capacitor class using `rdf:range`.

3.5.14. *Distinguishing Electrical, Mechanical and Thermal properties*

DAML+OIL has no notion of content. Therefore, it will be necessary to indicate to the software agents that will browse the device ontologies written in DAML+OIL that although electrical, mechanical and thermal are all properties of devices, they are not equivalent to each other.

This can be stated in compact DAML+OIL idiom form by using daml: collection element and daml: disjoint element as shown below:

```
<daml:disjoint rdf:parseType="daml:collection">  
  < daml:ObjectProperty rdf:about="#ElectricalProperty"/>  
  < daml:ObjectProperty rdf:about="#MechanicalProperty"/>  
  < daml:ObjectProperty rdf:about="#ThermalProperty"/>  
</daml:disjoint>
```

Listing 3-2: Distinguishing between device properties

3.5.14.1. Has Terminals Property

A property of all electronic devices is that they have electrical terminals, pins or connectors for the flow of electricity. It can be defined as shown below:

```
<daml:ObjectProperty rdf:ID="hasTerminals">  
  <rdfs : subPropertyOf: rdf:resource ="MechanicalProperty"/>  
  <rdfs:domain rdf:resource ="#Devices"/>  
  <rdfs:range rdf:resource ="#Devices"/>  
</daml:ObjectProperty>
```

Listing 3-3: daml:Object Property “hasTerminals”

In Listing 3-3, the use of rdf: range and rdf: domain in the declaration of “hasTerminals”, property forces all devices to inherit the “hasTerminals” as one of the mechanical properties (Figure 3.5.2-1). Therefore, Resistors, Capacitors, AnalogDevices, DigitalDevices etc. will all inherit the “hasTerminals” property. This is appropriate because all electronic devices must have terminals. This inheritance of property facilitated by rdfs:range and rdfs:domain is a very useful feature for assigning property to a large group of electronic devices.

3.5.14.2. Electrical DC property of devices

The figure below shows a sample of the way DC Electrical Characteristics of electronic device are expressed. The illustration is from a section of device specification for the National Semiconductor or PC16550 chip [Section 8.2]

2.0 DC Electrical Characteristics $T_A = 0^\circ\text{C to } +70^\circ\text{C}$, $V_{DD} = +5\text{V} \pm 10\%$, $V_{SS} = 0\text{V}$, unless otherwise specified.					
Symbol	Parameter	Conditions	Min	Max	Units
V_{ILX}	Clock Input Low Voltage		~ 0.5	0.8	V
V_{IHx}	Clock Input High Voltage		2.0	V_{DD}	V
V_{IL}	Input Low Voltage		~ 0.5	0.8	V
V_{IH}	Input High Voltage		2.0	V_{DD}	V
V_{OL}	Output Low Voltage	$I_{OL} = 1.6\text{ mA on all (Note 1)}$		0.4	V
V_{OH}	Output High Voltage	$I_{OH} = -1.0\text{ mA (Note 1)}$	2.4		V
$I_{CC(AV)}$	Average Power Supply Current	$V_{DD} = 5.5\text{V}$, $T_A = 25^\circ\text{C}$ No Loads on output SIN, DSR, DCD, CTS, RI = 2.0V All other inputs = 0.8V		15	mA

Table 3.5-2: Example of Device DC Electrical Characteristics

The DC characteristics can be defined as a sub property of the electricalProperty that was declared earlier.

```

<daml:ObjectProperty rdf:ID = "DC-characteristics">
<rdfs:subPropertyOf rdf:resource="# electricalProperty"/>
<rdfs:range rdf:resource="#devices"/>
</daml:ObjectProperty>

```

Listing: 3.5-10 : DC property of devices

The above does not fully describe all the DC characteristics of PC16550 chip.

Representation of the signals parameters in the device data sheet [Table 3.5-2] may be represented using XML tags for each of the device parameters, and a set of XML tags corresponding to the signals may be written such as:

<clock-input-low-voltages>, <clock-input-high-voltage>, <input-low-voltage>, <input-high-voltage>, <output-low-voltage>, <output-high-voltage>, <average-power-supply-current> etc. etc

Alternatively, the device signals may also be written terms of daml: ObjectProperty, and declared with IDs, such as:

“clock-input-low-voltages”, “clock-input-high-voltage”, “input-low-voltage”, “input-high-voltage”, “output-low-voltage”, “output-high-voltage”, “average-power-supply-current” etc.

The table next shows a partial list of the AC Electrical characteristics of PC 16550 chip [Ref: section 8.2].

3.5.14.3. Electrical AC Characteristics of devices

3.0 AC Electrical Characteristics $T_A = 0^\circ\text{C to } +70^\circ\text{C}, V_{DD} = +5\text{V} \pm 10\%$					
Symbol	Parameter	Conditions	Min	Max	Units
t_{ADS}	Address Strobe Width		60		ns
t_{AH}	Address Hold Time		0		ns
t_{AR}	RD, RD Delay from Address	(Note 1)	30		ns
t_{AS}	Address Setup Time		60		ns
t_{AW}	WR, WR Delay from Address	(Note 1)	30		ns
t_{CH}	Chip Select Hold Time		0		ns
t_{CS}	Chip Select Setup Time		60		ns
t_{CSH}	RD, RD Delay from Chip Select	(Note 1)	30		ns
t_{CSW}	WR, WR Delay from Select	(Note 1)	30		ns

Table 3.5-3: Example of AC Electrical Characteristics

Examination of Table 3.5-3 shows that *AC Electrical Characteristics* include the notion of *time*, e.g. 'Address Hold Time' under the Parameters column above. It should be realized that none of the DAML+OIL language elements [48] deal with time. (Separate time ontology is required). The AC-characteristics can be declared as subPropertyOf of the electricalProperty and declared in the same manner as the parameter of DC-characteristics, as shown below:

```
<daml:ObjectProperty rdf:ID="AC-characteristics">
<rdfs:subPropertyOf rdf:resource="# electricalProperty"/>
<rdfs:range rdf:resource="#Devices"/>
</daml:ObjectProperty>
```

Listing: 3.5-11: AC property of the of device

Some of the AC parameters of devices are shown in the Table 3.5-3. The type of AC parameters can vary from device to device. Each of these AC device signals may be represented with corresponding XML tags as shown below:

```
<address-strobe-width>, <address-hold-time>, <rd-delay-from-address>
<address-set-up-time>, <write-delay-from-address>, <chip-select-hold-time>
<chip-select-set-up-time>, <rd-delay-from-chip-select>,
<write-delay-from-select>
```

Listing: 3.5-12: Example of device AC properties as XML tags

It may be noted from the Listing: 3.5-12, and Listing: 3.5-10 that the semantics of the signal names are common with other similar electronic devices, even though the names and spelling may be vary among the different manufacturers. McGuinness [22] states that a properly developed ontology should be applicable to all similar objects in that class, therefore device ontology developed for one manufacturer should applicable to a similar

device from another manufacturer. Instead of using XML tags, the above AC characteristics may be declared in DAML+OIL and the device AC parameters may be assigned with unique daml: ObjectProperty IDs as shown below:

```
<daml:ObjectProperty rdf:ID= "address-strobe-width ">  
<rdfs:subPropertyOf rdf:resource="#AC-characteristics "/>  
<rdfs:range rdf:resource="#Devices"/>  
</daml:ObjectProperty>
```

Listing: 3.5-13 : Address Strobe width

Similar to the above, IDs may be assigned daml: ObjectProperty to each of the other AC characteristics parameters listed below:

```
"address-hold-time", "rd-delay-from-address" "address-set-up-time", "write-delay-from-  
address", "chip-select-hold-time" "chip-select-set-up-time", "rd-delay-from-chip-select", "write-  
delay-from-select"
```

3.5.14.4. Time in DAML+OIL

It should be noted from the AC parameters listed in Table 3.5-3 that AC Electrical Characteristics of device are associated with the notion of time. Time is not one of the defined language elements

[48] of DAML+OIL. In DAML+OIL, Time Ontology needs to be defined separately. The development of time ontology is currently work in progress. [73]

3.5.15. *Has Ground Pins Property*

Ground pins can be declared as a sub property of “hasTerminals”.

```
<daml:ObjectProperty rdf:ID="GroundPins">  
<rdf:subPropertyOf rdf:resource = "hasTerminals"/>  
</daml:ObjectProperty>
```

Listing 3-4: daml:Object Property “GroundPins”

3.5.16. *Judicious use of rdfs: domain and rdfs: range for devices*

DAML+OIL allows the use of rdfs: domain and rdfs: range. These can be used to indicate global information about properties, regardless of the particular class.

This feature can be advantageously used, as for example; the property “hasTerminals” is applicable to all types devices, regardless of the fact that the device is active or passive.

It must be realized that rdfs: domain and rdfs: range needs to be used judiciously and with great care in device ontology development. It may be noted that the “hasGroundPins” property was not declared with rdf: domain and rdf: range because some types of devices do not have ground pins, such as diodes. If the “hasGroundPins” were assigned all electronic devices, it would include diodes. On encountering “hasGroundPins”, the software agents in the Semantic Web could attempt to always ground that pin, including one on the diode. However, the electrical theory shows that diodes need not be grounded in many applications. Such an oversight in assigning properties in the creation of device

ontologies can cause unrealistic logical deduction by the agent in the Semantic Web and this could inadvertently cause generation of faulty designs. The solution to such risks is to use the `daml: toClass` element in the creation of ontology for some types of devices. The `daml: toClass` limits the application of that property to the named class.

3.6 DAML + OIL DatatypeProperty for Electronic Devices

The concept of `daml: DatatypeProperty` in DAML+OIL is somewhat similar to data type (e.g. characters, Boolean, integer etc) in programming languages, but is much richer because it can include additional information, for example. if the type take only unique values. The `daml: DatatypeProperty` feature can be used advantageously to match the data type of the device parameters published in the electronic device manufactures specification sheet. For example, the properties of some devices, such as voltage need to be both negative and positive numbers, but the “TotalPins” or “PinCount” parameter requires only positive integer. (PinCount cannot be negative number). In DAML+OIL, the type associated with such parameter can be written as shown in the example below:

```
<daml:DatatypeProperty      rdf:ID="TotalPins">
  <rdfs:comment>
    TotalPins is a DatatypeProperty whose range are non-negative integers
    TotalPins is also a UniqueProperty (can only have one total of pin count)
  </rdfs:comment>
  <rdf:type    rdf:resource="http://www.daml.org/2001/03/daml+oil#UniqueProperty"/>
  <rdfs:range  rdf:resource="http://www.w3.org/2000/10/XMLSchema#positiveInteger"/>
</daml:DatatypeProperty>
```

Listing: 3.6-1: Device Total pin count property

Another example is the property of silicon chip 16550 [APPENDIX 3 National Semiconductor Universal Asynchronous Receiver Transmitter] data sheet page 3, under section 1.0 Absolute Maximum Rating; a property called “MaxTemperatureUnderBias”, in the range 0 to 70⁰C. This parameter can be considered a unique property of that device and this parameter may be represented as shown below.

```
<daml:DatatypeProperty rdf:ID="MaxTemperatureUnderBias">
  <rdfs:comment>
    MaxTemperatureUnderBias is a DatatypeProperty whose range is xsd:decimal.
    MaxTemperatureUnderBias is also a UniqueProperty (can only have one maximum length)
  </rdfs:comment>
  <rdf:type    rdf:resource="http://www.daml.org/2001/03/daml+oil#UniqueProperty"/>
  <rdfs:range  rdf:resource="http://www.w3.org/2000/10/XMLSchema#decimal"/>
</daml:DatatypeProperty>
```

Listing: 3.6-2: Device temperature property data type

3.7 Devices classes with the device property included

In the previous sections, the device classes and properties were declared separately. In DAML+OIL, it is possible to include property in the declaration of a class. This can be illustrated using a class with internal structure. Taking passive diodes as an example, and by utilizing the daml:onProperty, daml:cardinality, and daml:Restriction language elements, device classes can be declared with the property included.

3.7.1. *Passive Diode Class*

```
<daml:Class rdf:ID="PassiveDiodes">
  <rdfs:comment>
    Passive diodes have exactly two terminals
  </rdfs:comment>
  <rdfs:subClassOf>
    <daml:Restriction daml:cardinality="2">
      <daml:onProperty rdf:resource="#hasTerminals"/>
    </daml:Restriction>
  </rdfs:subClassOf>
</daml:Class>
```

Listing: 3.7-1: Passive Diodes class with property included

In the above, daml: Restriction defines anonymous class of all things that satisfy the restriction that it has two terminals. It can be noted that #hasTerminals property was defined earlier. For simplicity, other electrical properties of diodes are not included.

3.8 Ontology for active devices in DAML+OIL

Devices that need power (i.e. external voltage) to function are called active devices. Examples of analog devices are operational amplifiers, filters, attenuators, etc.

3.8.1. *Active Devices*

Analog devices can be classified as subclass of devices.

```
<daml:Class rdf:ID="ActiveDevices">
  <rdfs:subClassOf rdf:resource="#Devices"/>
</daml:Class>
```

Listing: 3.8-1: ActiveDevices class

The "ActiveDevices" as declared above does not differentiate between discrete and integrated devices.

3.8.2. *DiscreteActiveComponents*

Discrete active components such as transistors can be classified, as subclass of Devices but it needs to be distinguished from Integrated Circuits (IC). This distinction can be partially achieved with the use of daml: disjointWith

```
<daml:Class rdf:ID="DiscreteActiveComponents">  
  <rdfs:subClassOf rdf:resource="#ActiveDevices"/>  
  <daml:disjointWith rdf:resource="#ICs"/>  
</daml:Class>
```

Listing: 3.8-2: Discrete ActiveComponents

3.8.3. *Digital Devices*

Digital devices can be classified as subclass of Devices but its properties need to be distinguished from Analog devices that are also subclass of Devices.

```
<daml:Class rdf:ID="DigitalDevices">  
  <rdfs:subClassOf rdf:resource="#Devices"/>  
</daml:Class>
```

Listing: 3.8-3: DigitalDevices class

3.8.4. Integrated Circuits (IC) Class in DAML+OIL

Integrated Circuits (IC) class may be better declared with a structure using DAML+OIL *anonymous class* and *property restrictions*. The ICs have more than one active device inside and has power supply pins, the ground pins and usually exceeds number greater than three pins. This is not a strict criterion for ICs, but is stated here to illustrate the use of *anonymous class* in the development of device ontology.

Restriction defines the anonymous class, i.e. class of all things that satisfy the restriction.

```
<daml:Class rdf:ID="ICs">
  <rdfs:subClassOf rdf:resource="#Devices"/>
  <rdfs:subClassOf>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#ACproperty"/>
      <daml:toClass rdf:resource="#ICs"/>
    </daml:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <daml:Restriction daml:minCardinality="4">
      <daml:onProperty rdf:resource="#DevicePins"/>
    </daml:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#hasPowerSupplyPins"/>
      <daml:minCardinality>1</daml:minCardinality>
    </daml:Restriction>
  </rdfs:subClassOf>
</daml:Class>
```

Listing: 3.8-4: ICs class in DAML+OIL

With the use of cardinality, the above shows that ICs (Integrated Circuits) must have at least one pin power supply pin.

3.8.5. Operation Amplifiers Class

The next sample electronic device chosen is the Operational Amplifier. An example of a commercial operational amplifier is shown next.

Diagram 3.8.5-1: Operational Amplifier

```
<daml:Class rdf:ID="OperationalAmplifiers">
  <rdfs:subClassOf rdf:resource="#AnalogDevices"/>
</daml:Class>
```

Listing: 3.8-5: Operational Amplifier class

In the above diagram, the Burr-Brown™ 3360J [Diagram 3.8.5-1] would be an instance of the OperationalAmplifier class. The properties of the operational amplifier class can be determined from the Diagram 3.8.5-1 above and expressed either in terms of XML tags or as daml:ObjectProperty and is shown as examples below:


```
<VCC>, <Gain>, <in-minus>, <in-plus>, <offset-trim>, <FB-Com>, <FB-Out>, <output>
```

In terms of DAML+OIL, daml:ObjectProperty ID can be assigned to each of the parameters of the operational amplifier. The IDs could be named, as shown in the example below:

```
"VCC", "Gain", "in-minus", "in-plus", "offset-trim", "FB-Com", "FB-Out", "output"
```

A tree node, for example OpAmpProperty, can be created under which all operational amplifier properties can be attached.

```
<daml:ObjectProperty rdf:ID="OpAmpProperty">  
  <rdfs:domain rdf:resource="# Amplifiers "/>  
  <rdfs:range rdf:resource="#OperationalAmplifiers "/>  
</daml:Class>
```

Listing: 3.8-6: Operational Amplifier property

```
<daml:ObjectProperty rdf:ID="Gain">  
  <daml:subPropertyOf rdf:resource="#OpAmpProperty "/>  
</daml:ObjectProperty>
```

Listing: 3.8-7 Gain as sub property Operational Amplifier property

```
<daml:ObjectProperty rdf:ID="output">  
  <daml:subPropertyOf rdf:resource="#OpAmpProperty "/>  
</daml:ObjectProperty>
```

Listing: 3.8-8 Output as sub property Operational Amplifier property

Similarly, other properties of the amplifier can be listed.

3.8.6. *Multiple super-classes & Mixed Signal Devices*

In DAML+OIL, it is admissible for a class to be subclass of multiple super-classes. This feature can be advantageously used to partially describe complex electronic devices such as Mixed Signal Devices (MSP).

The term mixed signal is used in the industry to classify devices that can process both digital and analog signals, such as the Texas InstrumentTM MSP 430 series [Appendix, Section 8.3].

```
<daml:Class rdf:ID="MixedSignalDevices">
  <daml:intersectionOf rdf:parseType="daml:collection">
    <daml:Class rdf:about="#DigitalDevices"/>
    <daml:Class rdf:about="#AnalogDevices"/>
  </daml:intersectionOf>
</daml:Class>
```

Listing: 3.8-9: MixedSignalDevices class in DAML+OIL

The class as defined above using the daml:intersectionOf, excludes linear only devices, such as operational amplifiers class shown in Diagram 3.8.5-1 and digital only devices such as RAM.

It should be noted that the above is not the best way to define mixed signal devices nor a sufficient condition to define the reality of mixed signal device. No properties of devices are assigned in the above declaration.

3.8.7. *Transmit and Receive as Inverse relations*

The **daml:inverseOf** property can be used indicate inverse relations. It shows the relationships such as if X is parent of Y then Y is child of X.

In the electronic device area, this property can be used as for example if device A transmits to B, then B receives from A. In DAML+OIL it can be represented as:

```
<daml:ObjectProperty rdf:ID="Transmit">  
<daml:inverseOf rdf:resource="#Receive"/>  
</daml:ObjectProperty>
```

Listing: 3.8-10 Transmit Receive as *InverseOf* DAML+OIL relation

3.9 Ontology of Programmable Devices in DAML+OIL

In this section, programmable electronics devices are chosen as samples for device ontology development. The UART (Universal Asynchronous Receiver Transmitter) electronic device is commonly used in PCs for dial up data connections. A similar device, but for synchronous data communication is the class of devices known in the industry as USART (Universal Synchronous Asynchronous Receiver Transmitter). It includes both asynchronous and synchronous communication. The PhillipsTM USART is called CDUSCC, "CMOS Dual Universal Serial Communication Controller [Ref:8.1], which is yet another name of a similar device.

3.9.1. Classes of serial data comm. devices in DAML+OIL

A tree diagram for showing class and subclass of such devices may be drawn as shown below:

Figure 3.9.1-1: Tree diagram of subclasses of serial data comm. devices

The UART, USART and CDUSCC all belong to the same class of devices i.e. for serial data communication. The TDM (Time Division Multiplexer) is also a type of serial data communication. A part of the ontology for serial data communication devices may be written as shown in the following segments of ontology:

```
<daml:Class rdf:ID="Serial-data Devices-devices ">
  <rdfs:subClassOf rdf:resource="#IC"/>
</daml:Class>
```

Listing: 3.9-1: Class Serial-data-devices-devices

```
<daml:Class rdf:ID="Synchronous">
  <rdfs:subClassOf rdf:resource="#Serial-data Devices-devices" />
</daml:Class>
```

Listing: 3.9-2: Class Synchronous devices

```
<daml:Class rdf:ID="TDM">
  <rdfs:subClassOf rdf:resource="#Synchronous" />
</daml:Class>
```

Listing: 3.9-3: Class TDM devices

```
<daml:Class rdf:ID="Sync-Aynch">
  <rdfs:subClassOf rdf:resource="#Serial-data Devices-devices" />
</daml:Class>
```

Listing: 3.9-4: Class Sync-Async devices

```
<daml:Class rdf:ID="USART">
  <rdfs:subClassOf rdf:resource="#Sync-Aynch" />
</daml:Class>
```

Listing: 3.9-5: Class USART devices

```
<daml:Class rdf:ID="CUSCC">
  <rdfs:subClassOf rdf:resource="#Sync-Aynch" />
</daml:Class>
```

Listing: 3.9-6: Class CUSCC devices

```
<daml:Class rdf:ID="Asynchronous">
  <rdfs:subClassOf rdf:resource="#Serial-data Devices-devices" />
</daml:Class>
```

Listing: 3.9-7: Class Asynchronous devices

3.10 Properties of Programmable Devices

On examination of several programmable devices, it can be observed that the properties of such devices are generally the same and therefore a common ontology can be well suited for all programmable devices. The properties of programmable devices may be drawn as a generic tree diagram shown below:

Figure 3.9.1-1: Generic properties of programmable electronic devices

3.10.1. Tree of properties for the PC16550 chip

Examination of the PC16550 datasheet [Section 8.2] shows that the device is quite complex. It requires hardware interfacing with an external CPU and development of firmware (i.e. device register programming) and dynamic servicing to make it operational. The complexity may be managed with different levels of [88] abstractions.

The properties of PC16550 may be considered an instance of generic properties of programmable devices shown in Figure 3.9.1-1 and its instance is shown below:

Figure 3.10.1-1: Tree diagram developed for PC16550 chip ontology levels

The levels of abstraction of a device may be categorized as, Application, Functional, Hardware, and Firmware level. Each of these levels could have sub levels.

3.10.2. Device Application Level Ontology

Application level is the most abstract level that conceptualizes the device only in terms of its application and not in terms of its technical specifications.

At this level, the PC 16550 device may be described only in general terms with a set of XML tags.

```
<serial-data-data-communication/>, <max-data-rate/>, <min-data-rate/>
```

3.10.3. *Functional Level Ontology*

The ontology of a device functional level (high level) may be considered as metadata of the device.

<UART/>	<hardware>	<programmable/>	<serial/>	<data-communication/>	
<Asynchronous/>	<data-rate/>	<Serial/>	<transmit/>	<receive/>	<maximum/>
<minimum/>	<asynchronous/>				

Alternatively, a set of property IDs may be defined using `daml: ObjectProperty` as:

"UART"	"hardware"	"programmable"	"serial"	"data-communication"	
"Asynchronous"	"data-rate"	"Serial"	"transmit"	"receive"	"maximum"
"minimum"	"asynchronous"				

3.10.4. Hardware Properties of devices

Figure 3.10.4-1 : PC16550 Chip Hardware

It may be noted that different packaging of the UART device PC 16550 chip has different pins and numbers. The specific packages as shown above may be considered as instance of the UART class in DAML+OIL. It must be recognized that the ontology should remains the same for both types of packages and pin assignment.

6.0 Pin Descriptions

The following describes the function of all UART pins. Some of these descriptions reference internal circuits.

In the following descriptions, a low represents a logic 0 (0V nominal) and a high represents a logic 1 (+2.4V nominal).

A0, A1, A2, Register Select, Pins 26–28: Address signals connected to these 3 inputs select a UART register for the CPU to read from or write to during data transfer. A table of registers and their addresses is shown below. Note that the state of the Divisor Latch Access Bit (DLAB), which is the most significant bit of the Line Control Register, affects the selection of certain UART registers. The DLAB must be set high by the system software to access the Baud Generator Divisor Latches.

Register Addresses				Register
DLAB	A ₂	A ₁	A ₀	
0	0	0	0	Receiver Buffer (read), Transmitter Holding Register (write)
0	0	0	1	Interrupt Enable
X	0	1	0	Interrupt Identification (read)
X	0	1	0	FIFO Control (write)
X	0	1	1	Line Control
X	1	0	0	MODEM Control
X	1	0	1	Line Status
X	1	1	0	MODEM Status
X	1	1	1	Scratch
1	0	0	0	Divisor Latch (least significant byte)
1	0	0	1	Divisor Latch (most significant byte)

ADS, Address Strobe, Pin 25: The positive edge of an active Address Strobe (ADS) signal latches the Register Select (A₀, A₁, A₂) and Chip Select (CS₀, CS₁, CS₂) signals.

Note: An active ADS input is required when the Register Select (A₀, A₁, A₂) and Chip Select (CS₀, CS₁, CS₂) signals are not stable for the duration of a read or write operation. If not required, tie the ADS input permanently low.

BAUDOUT, Baud Out, Pin 15: This is the 16 × clock signal from the transmitter section of the UART. The clock rate is equal to the main reference oscillator frequency divided by the specified divisor in the Baud Generator Divisor Latches. The BAUDOUT may also be used for the receiver section by tying this output to the RCLK input of the chip.

CS₀, CS₁, CS₂, Chip Select, Pins 12–14: When CS₀ and CS₁ are high and CS₂ is low, the chip is selected. This enables communication between the UART and the CPU. The positive edge of an active Address Strobe signal latches the decoded chip select signals, completing chip selection. If ADS is always low, valid chip selects should stabilize according to the t_{CSW} parameter.

CTS, Clear to Send, Pin 36: When low, this indicates that the MODEM or data set is ready to exchange data. The CTS signal is a MODEM status input whose conditions can be tested by the CPU reading bit 4 (CTS) of the MODEM Status Register. Bit 4 is the complement of the CTS signal. Bit 0 (DCTS) of the MODEM Status Register indicates whether the CTS input has changed state since the previous reading of the MODEM Status Register. CTS has no effect on the Transmitter.

Note: Whenever the CTS bit of the MODEM Status Register changes state, an interrupt is generated if the MODEM Status Interrupt is enabled.

D₇–D₀, Data Bus, Pins 1–8: This bus comprises eight TRI-STATE input/output lines. The bus provides bidirectional communications between the UART and the CPU. Data, control words, and status information are transferred via the D₇–D₀ Data Bus.

DCD, Data Carrier Detect, Pin 38: When low, indicates that the data carrier has been detected by the MODEM or data set. The DCD signal is a MODEM status input whose condition can be tested by the CPU reading bit 7 (DCD) of the MODEM Status Register. Bit 7 is the complement of the DCD signal. Bit 3 (DDCD) of the MODEM Status Register indicates whether the DCD input has changed state since the previous reading of the MODEM Status Register. DCD has no effect on the receiver.

Note: Whenever the DCD bit of the MODEM Status Register changes state, an interrupt is generated if the MODEM Status Interrupt is enabled.

DDIS, Driver Disable, Pin 23: This goes low whenever the CPU is reading data from the UART. It can disable or control the direction of a data bus transaction between the CPU and the UART.

DSR, Data Set Ready, Pin 37: When low, this indicates that the MODEM or data set is ready to establish the communications link with the UART. The DSR signal is a MODEM status input whose condition can be tested by the CPU reading bit 5 (DSR) of the MODEM Status Register. Bit 5 is the complement of the DSR signal. Bit 1 (DDSR) of the MODEM Status Register indicates whether the DSR input has changed state since the previous reading of the MODEM Status Register.

Note: Whenever the DSR bit of the MODEM Status Register changes state, an interrupt is generated if the MODEM Status Interrupt is enabled.

DTR, Data Terminal Ready, Pin 33: When low, this informs the MODEM or data set that the UART is ready to establish a communications link. The DTR output signal can be set to an active low by programming bit 0 (DTR) of the MODEM Control Register to a high level. A Master Reset operation sets this signal to its inactive (high) state. Loop mode operation holds this signal in its inactive state.

INTR, Interrupt, Pin 30: This pin goes high whenever any one of the following interrupt types has an active high condition and is enabled via the IER: Receiver Error Flag; Received Data Available; Timeout (FIFO Mode only); Transmitter Holding Register Empty; and MODEM Status. The INTR signal is reset low upon the appropriate interrupt service or a Master Reset operation.

MR, Master Reset, Pin 35: When this input is high, it clears all the registers (except the Receiver Buffer, Transmitter Holding, and Divisor Latches), and the control logic of the UART. The states of various output signals (SOUT, INTR, OUT 1, OUT 2, RTS, DTR) are affected by an active MR input (Refer to Table 1.) This input is buffered with a TTL-compatible Schmitt Trigger with 0.5V typical hysteresis.

OUT 1, Output 1, Pin 34: This user-designated output can be set to an active low by programming bit 2 (OUT 1) of the MODEM Control Register to a high level. A Master Reset operation sets this signal to its inactive (high) state. Loop mode operation holds this signal in its inactive state. In the XMODEM parts this will achieve TTL levels.

Figure 3.10.4-2; Pin description for PC 16550 chip

The pin description copied from the excerpt of PC16550 chip specification sheet and produced in Figure 3.10.4-2 shows that the pin descriptions are quite complex and difficult to translate into machine interpretable form. However, the hardware ontology may be developed from the names of the pin signals by grouping them into the following categories and represented either in XML tag form or as daml:ObjectProperty or daml:subPropertyOf

Address, Chip-Select, Serial-Data, Clock, modem-control-signals, Device-Control, Power. The example for the modem group is shown below:

```
<modem-control-signals>
<request-to-send> 19 </request-to-send>
<data-terminal-ready> 20 </data-terminal-ready>
<clear-to-send> 23 </clear-to-send>,
<data-set-ready> 24 </data-set-ready>
<data-carrier-detect> 26 </data-carrier-detect>
</modem-control-signals>
```

Listing: 3.10-1: Grouping modem control pins & XML representation

The elements in XML documents can have parent child relationship. The children of the modem-control-signals are shown below:

```
<address-signals/> <address-strobe/>, <chip-select/>, <not-connected/>, <receive-clock/>,
<data-bus/>, <request-to-send/>, <data-terminal-ready/>, <clear-to-send/>, <receive-clock/>,
<receive-ready/>, <interrupt/>
```

Listing: 3.10-2: Modem signals in XML

3.10.5. *Firmware Level Ontology*

The firmware (i.e. computer program on a silicon chip) level ontology can be derived from the programmable registers of the chip. For example, in the case of PC16550 chip [Ref: section 8.2] the set of XML tags corresponding to the chip registers may be written as shown below:

```
<line-control-register/> <line-status-register/>, <interrupt-enable-register/>, <modem-control-  
register/>, <modem-status-register/>, <scratch-pad-register/l>, <programmable-baud-rate-  
generator/>
```

Various properties can be derived from the chip specification. For example, the maximum data speed of the chip may be written in terms of daml: ObjectProperty as shown below:

```
<daml:ObjectProperty rdf:ID="hasMaxSpeed">  
<rdf:domain rdf:resource = "#SerialDataCom">  
<rdf:range rdf:resource = "#SerialDataCom "/>  
</daml:ObjectProperty>
```

Listing: 3.10-3: Maximum data speed property in DAML+ OIL

3.10.6. *Device Bit Rate in XML*

Since the device can be programmed to have only one rate at a time, a table can be set up using XML Schema Definition [5] using enumeration (xsd:enumeration) as shown below:

```
<xsd:simpleType name="bitRate">
  <!-- This is an XMLS datatype valid values for which are the strings -->
  <!--The data rate varies from 50 to 12800 b/s -->
  <xsd:restriction base="string">
    <xsd:enumeration value="50"/>
    <xsd:enumeration value="75"/>
    <xsd:enumeration value="110"/>
    <xsd:enumeration value="300"/>
    <xsd:enumeration value="1200"/>
    <xsd:enumeration value="2400"/>
    <xsd:enumeration value="4800"/>
    <xsd:enumeration value="4800"/>
    <xsd:enumeration value="9600"/>
    <xsd:enumeration value="19200"/>
    <xsd:enumeration value="38400"/>
    <xsd:enumeration value="5600"/>
    <xsd:enumeration value="12800"/>
  </xsd:restriction>
</xsd:simpleType>
</xsd:schema>
```

Listing: 3.10-4 Bit rate in XML Schema form

3.11 Instance of Application Level description

As an example of application level description in DAML+OIL, the listing below shows a segment stating that the link to Toronto is at 19.2 Kb/s.

```
<SerialDataCom rdf:ID="TorontoLink">
  <bitRate> <xsd:string rdf:value="19200"/> </bitRate>
</SerialDataCom>
```

Listing: 3.11-1 Application Level representation of data link speed

3.12 Semantic difference between device parameters

McGuinness [22] indicates that vocabularies are a part of ontology. The vocabularies used by device manufacturers can be considered candidates in the development of device ontology. Some electronic device manufacturers use vocabularies that are inconsistent with the commonly used vocabulary in the electronics industry and can result in generation of different ontology.

However, many semiconductor manufacturers use their own format and vocabulary to describe their component, although the functionality of these components and signals may be similar to many other manufacturers. This is evident from the survey of device data sheets from various manufacturers. For example, on examination of data sheets it is seen that National Semiconductor [Section 8.2] uses the common industry name “UART” (Universal Asynchronous Receiver Transmitter). PhillipsTM [section 8.1] uses the name “CMOS Dual Universal Serial Communication Controller (CDUSCC)”, which is actually “Dual USART”, or (Dual Universal Synchronous Asynchronous Receiver Transmitter) in behavior i.e. different words and phases for the same functionality.

These devices can be considered to belong to the same class of devices.

The difference in vocabulary and phrases used is shown in the table below:

National Semiconductor <u>16550 REGISTERS</u> <u>PROPERTIES</u> Chip configuration Programming Registers LineControlRegister LineStatus Register Interrupt Identification Register InterruptEnableRegister ModemControl Register ModemStatusRegister ScratchpadRegister FIFOPolledModeOperation FIFOInterruptModeOperation FIFOControl Register	Phillips Semiconductor <u>SC2656C2 REGISTER</u> <u>PROPERTIES</u> Chip configuration Programming Registers Status Registers General Status Register Interrupt Control Receiver and Transmitter Status Registers (RSR, TRSR) Input and Counter/Timer Status Register (ICTSRA, ICTSRB) Transmitter/Receiver Miscellaneous Status Register (TRMSR) CDUSCC Interrupt Modes Default Interrupt Mode . Interrupt Enable Register (IERA, IERB) Interrupt Vector Register (IVR) and Modified Vector Register (IVRM) 604
--	---

Table 3.12-1 : Programming Register differences in same class of device

However, the objective should be to minimize the vocabulary required and take advantage of the commonality. A few examples of common vocabulary are: ‘address’, ‘data’, ‘clock’, ‘I/O’, ‘read’, ‘write’, ‘strobe’, ‘enable’, ‘disable’, ‘chip-select’, ‘ground’ etc. These signals names could be standardized industry wide and marked-up with known XML tags under a fixed name space [24]. Alternatively, they can be given standardized daml: ObjectProperty ID by a standards organization and accessed via URI [9].

Carlson [75] Chapter 10 “Vocabulary transformation”, has shown the technique of vocabulary transformation and that technique may be used for translating the devices different vocabularies, as for example, those in the table above.

XML Style Sheet Transformation Language [16] XSLT may be used for transformation of electronic device vendor specific vocabulary. XSLT specification defines the syntax and semantics for transforming XML documents into other XML documents. XSLT is a nonprocedural language for transforming a source document of type to another. The transformation is possible if one schema definition is similar to the semantics in the other document.

3.13 Electronic Devices with similar properties

The next sample chosen are commercial electronic devices with similar properties. It can be shown that DAML+OIL language elements

[48] can be applied to state the similarity between two devices. A survey of semiconductor manufacturers web sites, shows that family of devices have similar properties. In ontology, these can be grouped in a common class. The Texas InstrumentTM Mixed Signal Processors MSP 430 series of devices listed in web site [Ref: section 8.6] can be taken as an example.

The list below is taken from the Texas Instrument™ web site.

Device Name	Attributes	OTP / EPROM (KB)	RAM (Kbytes)	ADC	LCD Segments	Compare Cruz Timer	UART	Hardware Multiplier	Number of Timers	1KHz Pulse (1000)
☐ MSP430C1101	Show	1	128	Slope		Yes	software	No	2	Call
☐ MSP430C1111	Show	2	128	slope		Yes	software	No	2	Call
☐ MSP430C1121	Show	4	256	slope		Yes	software	No	2	Call
☐ MSP430C1321	Show	8	256	Slope		Yes	hardware	No	3	Call
☐ MSP430C1351	Show	16	512	Slope		Yes	hardware	No	3	Call
☐ MSP430C311	Show	2	128	slope	64	No	software	No	6	Call
☐ MSP430C312	Show	4	256	slope	92	No	software	No	6	Call
☐ MSP430C313	Show	8	256	slope	92	No	software	No	6	Call
☐ MSP430C314	Show	12	512	slope	92	No	software	No	6	Call
☐ MSP430C315	Show	16	512	slope	92	No	software	No	6	Call
☐ MSP430C323	Show	8	256	14-bit	84	No	software	No	6	Call
☐ MSP430C325	Show	16	512	14-bit	84	No	software	No	6	Call
☐ MSP430C336	Show	24	1024	slope	120	Yes	hardware	Yes	7	Call
☐ MSP430C337	Show	32	1024	slope	120	Yes	hardware	Yes	7	Call
☐ MSP430F110	Show	1	128	slope		Yes	software	No	1	1.02
☐ MSP430F1101	Show	1	128	slope		Yes	software	No	2	1.02
☐ MSP430F1101A	Show	1	128	slope		Yes	software	No	2	1.02
☐ MSP430F1111A	Show	2	128	slope		Yes	software	No	2	1.34
☐ MSP430F112	Show	4	256	slope		Yes	software	No	2	1.76
☐ MSP430F1121	Show	4	256	slope		Yes	software	No	2	1.76
☐ MSP430F1121A	Show	4	256	slope		Yes	software	No	2	1.75
☐ MSP430F1122	Show	4	256	ADC		Yes	software	No	2	Call
☐ MSP430F1132	Show	8	256	ADC		Yes	software	No	2	Call
☐ MSP430F122	Show	4	256	slope		Yes	hardware	No	2	2.43

Figure 3.10.6-1: Family of TI MSP 430 devices with similar properties

In DAML+OIL, the property of one entity can be represented in terms of another. For example, the Texas instrument (TI) device MSP430C1111 can be declared in terms of MSP430C1101 as shown below:

```
<daml:ObjectProperty rdf:ID="MSP430C1111">
  <daml:samePropertyAs rdf:resource="#MSP430C1101"/>
</daml:ObjectProperty>
```

Listing: 3.13-1: daml:samePropertyAs for devices

It may be noted from the Texas InstrumentTM (TI) web site that the device MSP430C1101 has only one kilobytes of Read Only Memory (ROM) whereas MSP430C1111 has two kilobytes. With respect to Description Logic, DL [76] and Frame based systems [62] on which DAML+OIL is based, this value can be considered only a difference in slot value in the ontology.

In the Figure 3.10.6-1, the UART property (column 9 of TI chart) of these devices indicates that some devices have hardware UARTs and others are software UARTs. This difference has design implications, in both the external circuitry required and the way the internal firmware will need to function to produce a valid design.

The device ontology in this case will need to distinguish between the two forms of UARTs existing in the same class of devices, perhaps with additional sub property definition or with additional XML tags. Such fine distinction (noting the difference between software and hardware UART function) should be encoded in the device ontology as it can assist software agents in the Semantic Web that will offer automated design services to produce valid designs.

3.14 EXAMPLE Ontology of LASER DIODE in DAML+OIL

Below is one of specification tables copied from the Zarlink Semiconductor data sheet

Optical and Electrical Characteristics - Case Temperature 25°C

Parameter	Symbol	Min	Typ	Max	Unit	Test Condition
Optical Power	P_o	5	7		mW	$I_F=40\text{mA}$
Slope Efficiency (dP_o/dI_F)	η		300		mW/A	$I_F=40\text{mA}$
Beam Divergence	Θ		11		deg	Full Width at $1/e^2$
Bandwidth 3dB _{opt})	f_c	1			GHz	$I_F=40\text{mA}$
Peak Wavelength	λ_p	830	845	860	nm	$I_F=40\text{mA}$
Spectral Width	$\Delta\lambda$		0.5	1.5	nm	$I_F=40\text{mA}$
Forward voltage	V_F		2.0	2.3	V	$I_F=40\text{mA}$
Threshold Current	I_{th}		14	19	mA	

Table 3.14-1: Zarlink Laser Diode device data sheet

The optical and electrical characteristics shown in the chart above can be considered properties of all similar class of VCSEL (Vertical Cavity Surface Emitting Laser) diodes. An attempt is made in this section of the thesis to develop ontology for the above device.

3.14.1. *Ontology header*

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
xmlns:xsd="http://www.w3.org/2000/10/XMLSchema#"
xmlns="http://www.sce.carleton.ca/siahamed/laser.daml#">
>
<daml:Ontology about="">
<daml:versionInfo> August 2002 </daml:versionInfo>
<rdfs:comment>Simple ontology for Laser diodes </rdfs:comment>
<daml:imports rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns" />
<daml:imports rdf:resource="http://www.w3.org/2000/01/rdf-schema" />
<daml:imports rdf:resource="http://www.daml.org/2001/03/daml+oil" />
</daml:Ontology>
```

Listing: 3.14-1: Ontology Header Laser Diodes

Devices were declared as a class earlier. Therefore, LaserDevices can be declared as a subClassOf Devices as follows:

```
<daml :Class rdf : ID = "LaserDevices">
<rdfs: subClassOf rdf: resource = #Devices/>
</daml : Class>
```

Listing: 3.14-2: Laser Devices as subClass of Devices

The ZarlinkTM, datasheet shows a type of laser called Vertical Cavity Surface Emitting Laser (VCSEL) and can be considered as a subclass of Laser Devices.

```
<daml:Class rdf:ID="VCSEL_Devices">
<rdf:subClassOf rdf:resource="#LaserDevices"/>
</daml:Class>
```

Listing: 3.14-3 VCSEL as subClass of Laser Devices

3.14.2. *Defining Properties of Laser Diode in DAML+OIL*

The characteristics of the diode, shown in the datasheet can be considered the properties of the device. These are Optical Power, Slope Efficiency, Beam Divergence, Bandwidth, Peak Wavelength, Spectral Width, Forward Voltage, and Threshold Current.

Properties can be defined by using daml : ObjectProperty element.

```
<daml:ObjectProperty rdf:ID="hasOpticalPower">
<rdf:domain rdf:resource="#LaserDevices"/>
<rdf:range rdf:resource="#VCSEL-Devices"/>
</daml:ObjectProperty>
```

Listing: 3.14-4: hasOpticalPower property

The above declaration assigns an ID and states that “hasOpticalPower” is applicable to all classes that are LaserDevices within the range of classes, which are VCSEL-Devices. Similarly, other properties of the device can each be given an ID, domain and range (corresponding to the characteristics in the datasheet).

```
<daml:ObjectProperty rdf:ID = "hasSlope-Efficiency">
<rdf:domain    rdf:resource = "#LaserDevices"/>
<rdf:range     rdf:resource = "#VCSEL-Lasers"/>
</daml:ObjectProperty>
```

Listing: 3.14-5 daml : ObjectProperty hasSlope-Efficiency

```
<daml:ObjectProperty rdf:ID = "hasBeam-Divergence">
<rdf:domain    rdf:resource = "#LaserDevices"/>
<rdf:range     rdf:resource = "#VCSEL-Lasers"/>
</daml:ObjectProperty>
```

Listing: 3.14-6: ObjectProperty “hasBeam-Divergence”

```
<daml:ObjectProperty rdf:ID = "hasBandwidth">
<rdf:domain    rdf:resource = "#LaserDevices"/>
<rdf:range     rdf:resource = "#VCSEL-Lasers"/>
</daml:ObjectProperty>
```

Listing: 3.14-7: ObjectProperty “hasBandwidth”

```
<daml:ObjectProperty rdf:ID = "hasPeakWavelength">
<rdf:domain    rdf:resource = "#LaserDevices"/>
<rdf:range     rdf:resource = "#VCSEL-Lasers"/>
</daml:ObjectProperty>
```

Listing: 3.14-8: Property “PeakWavelength”

```
<daml:ObjectProperty rdf:ID = "hasSpectralWidth">
<rdf:domain    rdf:resource = "#LaserDevices"/>
<rdf:range     rdf:resource = "#VCSEL-Lasers"/>
</daml:ObjectProperty>
```

Listing: 3.14-9: ObjectProperty “hasSpectralWidth”

```
<daml:ObjectProperty rdf:ID = "hasForwardVoltage">
<rdf:domain    rdf:resource = "#LaserDevices"/>
<rdf:range     rdf:resource = "#VCSEL-Lasers"/>
</daml:ObjectProperty>
```

Listing: 3.14-10: ObjectProperty “hasForwardVoltage”

3.14.3. *Data Types for Laser Diode properties*

Examination of the first row of the Zarlink™ datasheet, that Min (minimum), Typ (typical), Max (maximum) are data associated with the properties of the device and are integers or decimals.

```
<xsd:simpleType name="range-forward-voltage">
  <!--range-forward-voltage is an XMLS union datatype -->
  <!-- values of may be integers for some manufacturers or decimal for others -->
  <xsd:union>
    <xsd:simpleType>
      <xsd:restriction base='integer'/>
    </xsd:simpleType>
    <xsd:simpleType>
      <xsd:restriction base='decimal'/>
    </xsd:simpleType>
  </xsd:union>
</xsd:simpleType>
```

```
<xsd:simpleType name="parameter-value-range">
  <!--range of parameter is an XMLS union datatype -->
  <!-- values of may be integers for some manufacturers or decimal for others -->
  <xsd:union>
    <xsd:simpleType>
      <xsd:restriction base='integer'/>
    </xsd:simpleType>
    <xsd:simpleType>
      <xsd:restriction base='decimal'/>
    </xsd:simpleType>
  </xsd:union>
</xsd:simpleType>
```

```

<daml:DatatypeProperty rdf:ID="Forward-Voltage">
  <rdfs:comment>
    Forward-Voltage is a DatatypeProperty whose range is range-forward-voltage.
  </rdfs:comment>
  <rdf:type rdf:resource="http://www.daml.org/2001/03/daml+oil#UniqueProperty"/>
  <rdfs:range rdf:resource="http://www.daml.org/2001/03/daml+oil-ex-dt# range-forward-voltage" />
</daml:DatatypeProperty>

```

Listing: 3.14-11: daml:DatatypeProperty Property “Forward-Voltage”

The above shows that value of the parameter “range-forward-voltage” can be integer or decimal. The daml : DatatypeProperty, can be used for this purpose as shown below:

```

<daml:DatatypeProperty rdf:ID="BeamDivergence">
  <rdfs:comment>
    BeamDivergence is a DatatypeProperty whose range is xsd:non-negative integer
    BeamDivergence is also a UniqueProperty (can only have one BeamDivergence)
  </rdfs:comment>
  <rdf:type rdf:resource="http://www.daml.org/2001/03/daml+oil#UniqueProperty"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#nonNegativeInteger"/>
</daml:DatatypeProperty>

```

Listing: 3.14-12: daml:DatatypeProperty “BeamDivergence”

```

<daml:DatatypeProperty rdf:ID="BandWidth">
  <rdfs:comment>
    BandWidth is a DatatypeProperty whose range is xsd:decimal.
    BandWidth is also a UniqueProperty (can only have one BandWidth)
  </rdfs:comment>
  <rdf:type rdf:resource="http://www.daml.org/2001/03/daml+oil#UniqueProperty"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#decimal"/>
</daml:DatatypeProperty>

```

Listing: 3.14-13: daml:DatatypeProperty “BandWidth”


```

<daml:DatatypeProperty rdf:ID="Typical-Value">
  <rdfs:comment>
    Typical value a DatatypeProperty whose range is xsd:decimal.
    Typical value is also a UniqueProperty (can only have one maximum value)
  </rdfs:comment>
  <rdf:type rdf:resource="http://www.daml.org/2001/03/daml+oil#UniqueProperty"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#decimal"/>
</daml:DatatypeProperty>

```

Listing: 3.14-14: daml:DatatypeProperty "Typical-Value"

```

<daml:DatatypeProperty rdf:ID="Max-Value">
<daml:samePropertyAs rdf:resource="#Typical-Value"/>
</daml:ObjectProperty>

```

Listing: 3.14-15: daml:DatatypeProperty "Max-Value"

```

<daml:DatatypeProperty rdf:ID="Min-Value">
<daml:samePropertyAs rdf:resource="#Typical-Value"/>
</daml:ObjectProperty>

```

Listing: 3.14-16: daml:DatatypeProperty "Min-Value"

The ontology developed above is incomplete for simplicity. It does not show the interrelationship between the parameters.

3.15 Instantiations of Laser Diode

Instantiations are not a part of ontology. The properties of Zarlink™ laser diode may be partially instantiated as:

```
<VCSEL-Diode rdf:ID = " ZL6005 " >  
<Forward-Voltage>  
<Typical-Value > 2.0 </ Typical-Value >  
<Max-value> 2.3 </Max-value>  
</Forward-Voltage>  
<BandWidth> 1 </BandWidth>  
<BeamDivergence > 11 </BeamDivergence>  
</VCSEL-Diode>
```

Listing: 3.15-1: Instantiation of Laser Diode

Another instance of laser device in class of laser devices may have different numerical values and those may be substituted above while the ontology remains the same.

3.16 Summary of device ontology development experiment

A number of different types of devices were chosen for ontology development. No difficulty was encountered in defining classes and properties for the many different types of electronic devices, including active, passive, integrated and programmable circuits. Only a segment of the ontology was developed for each device due to the complexity and length of details required for expressing the static characteristics fully. It was seen that

1. Devices can be categorized rather easily and daml:class can be used to declare the device classes

2. The device parameters listed in the manufacturers device data sheet can be treated as DAML+OIL properties and can be declared with `daml:ObjectProperty`
3. The values and ranges of the device parameters can also be treated as DAML+OIL property and declared with `daml:DatatypeProperty`
4. Device properties can be declared using either `daml:ObjectProperty` or by creating XML tag for each property, such as signal name of the device.
5. At the ontology level, DAML+OIL can merely relate different classes, properties and values of the devices with each other.

The properties and parameters of devices could only be defined in high-level terms. The ontology developed as shown above is rudimentary, incomplete and not sufficiently developed to support electronic design services in the Semantic Web.

Considerable amount of research effort is further required. As for example, the representation of the meaning of “Beam Divergence”, “Optical Power” etc. in machine-interpretable form was not dealt with due to the complexity and remains unresolved problem.

4. MODELING DAML+OIL ELECTRONIC DEVICE ONTOLOGY IN UML

The common formalisms used to represent ontologies are Knowledge Interchange Format (KIF) and KL-ONE [77] style of knowledge representation languages. Knowledge Interchange Format (KIF) is a computer-oriented language for the interchange of knowledge among disparate programs. KIF is highly expressive but it is a low-level language for expressing ontologies. The previous chapter of this thesis illustrated how ontology for electronic devices could potentially be developed in DARPA Agent Markup Language +Ontology Inference Layer (DAML+OIL) [23] which is significantly different from KIF.

An increasingly popular modeling language is the Unified Modeling Language commonly known as UML [42]. Cranefield [41] has shown the use of UML for representing ontologies and domain knowledge for the Semantic Web. Cranefield identifies UML as a standardized and an expressive modeling language that has a large user community and support tools. It has associated constraint language with the expressive power of first-order logic. UML is also beginning to be used by the Meta Data Coalition [79] for representing metadata schemas and it may be a useful tool for modeling device ontologies written in DAML+OIL. In this chapter, DAML+OIL and UML are compared and an attempt is made to apply the UML to model device ontology written in DAML+OIL.

However, at the outset, it should be recognized that UML and DAML+OIL are not programming languages like ADA, C++, or Perl or other Object Orientated languages are. Functional code cannot be generated directly from DAML+OIL or from UML. The only similarity with such programming languages is in the use of Object Orientated [34] approach to concept representation, which uses *hierarchy*, *encapsulation* and *inheritance* properties. DAML+OIL does not include modularization typically used in Object Orientated Design and in UML.

4.1 Background and distinctive features of UML and DAML+OIL

UML [42] was originally developed to convey system concepts from one human to another and developed prior to the publication of ideas of the Semantic Web and DAML+OIL. It may be assumed that UML may lack features for supporting the Semantic Web and DAML+OIL. However, recently, UML is being used for designing artifacts that are more declarative [[75] (i.e., similar to ontologies) such as XML, DTDs, Schemas, Resource Description Framework (RDF) schemas [8], database schemas [80] and knowledge models [82].

The aim of the DAML+OIL project is to achieve “*semantic interoperability between Web pages, databases, programs, and sensors*”. UML is described [42] as a language for “*specifying, visualizing, constructing, and documenting the artifacts of a system-intensive process*”. Initially, the key words “process” and “visualizing” in the description of UML can be used to differentiate it from DAML+OIL.

DAML+OIL has no concept of *process*, which is normally indicated by methods, function calls or behavior. Unlike UML, DAML+OIL has no graphical constructs such as, 'State Machine', 'Activity Graphs' and 'Interaction Diagrams'. These UML diagrams give multiple perspective of the system under analysis or design.

Presently, DAML+OIL can express only static relationships in terms of classes, properties and restrictions, and therefore the UML diagrams for representing dynamic processes of a system are not directly applicable in DAML+OIL. UML diagrams are broadly classified into structure diagrams and behavior diagram. XML diagrams are most relevant for modeling XML vocabularies [75].

Unlike UML, the focus in the DAML+OIL is not on system analysis and design but on conveyance of meaning to the software agents and humans by stating the static relationship between pairs of entities. As for example, the use of `daml: sameClassAs`, `daml: sameIndividualAs`, `daml: disjointWith` imply general relationship between two entities, but not what those entities are. From a set of such relationships expressed in DAML+OIL, and with additional reasoning resources, software agents and humans can derive meaning. UML exhibits weakness with respect to such reasoning support compared to DAML+OIL, for example, there is no "sameClassAs" feature in UML. However, UML has some advanced features, such as profiles, modularity and extension mechanism that is currently not included in DAML+OIL.

The table in the next section was constructed by comparing the features of DAML+OIL.

4.2 Concepts in UML and DAML+OIL

CONCEPTS IN UML	CONCEPTS IN DAML+OIL
Modeling language for system design	Modeling language for meaning
Class	Class
Package	Thing class
Abstraction	Ontology
Data Types	Abstraction
Constrains Association Ends, including multiplicity and roles. Implicitly as class containing the attribute	Data Types
Limited Generalization for Associations, Class	Restrictions
Generalization Relations. None for Attributes	Hierarchy
Aspects of Attributes, Associations and Classes (difficult to represent sets)	Properties
Object Instances and Attribute Values	As Sets (union, disjoint)
Sharing data	Instances and Values
Not a programming language	Sharing meaning
Class Diagram	Not a programming language
Interaction Diagrams	
State Diagrams	
Activity Diagrams	
Physical Diagrams	
Method	
instance	type
type of ModelElement	type
attribute	ObjectProperty or DatatypeProperty
binary association	ObjectProperty
generalization	subClassOf
	subPropertyOf
stereotyped dependency between 2 associations	
note	comment
name	label
import	imports
multiplicity	cardinality
Multiple range A..B	A = minCardinality, B = maxCardinality
stereotype on an association	TransitiveProperty
association target with end multiplicity = 0..1 or 1	UniqueProperty
association source with end multiplicity = 0..1 or 1	UnambiguousProperty
stereotyped dependency between two associations	inverseOf

Table 4.2-1: Comparison between UML with DAML+OIL

The comparison table shows the equivalence, features that are analogous or closest in similarity or the absence of concepts between the two languages.

4.3 Classes in DAML+OIL and UML

The purpose of class in UML [42] is to declare a collection of methods, operations, and attributes that fully describe the structure and dynamic behavior of objects. However, it is not necessary for all UML classes to include methods or operations. DAML+OIL classes are without methods, functions or behavior and therefore cannot indicate dynamic processes.

4.3.1. *Potential use of UML Class Diagrams in DAML+OIL*

Cranefield [41] claims that UML class diagrams are well suited for representing ontologies. Ontologies include class/subclass hierarchies, relationships between classes, class attribute definitions and axioms that specify constraints. In this section, class diagrams are attempted for the ontology of electronic devices.

DAML+OIL class declaration can include property restrictions as shown in the example copied from DAML+OIL web site.

```
<rdfs:subClassOf>
  <daml:Restriction>
    <daml:onProperty rdf:resource="#hasParent"/>
    <daml:toClass rdf:resource="#Person"/>
  </daml:Restriction>
</rdfs:subClassOf>
```

Source: <http://www.daml.org/2001/03/daml+oil-walkthru#restrictions>

A class of electronic devices can be categorized, as “ProgrammableDevice” with the property restriction that it has internal memory, which are write-able. This can be represented in the class diagram below:

Figure 4.3.1-1: DAML+OIL Property Restriction

In the diagram above, the box on the left represents a general class called Devices. The arrow with ‘memory’ implies the memory property of the Devices class. (Devices have other properties besides memory). The text above the arrow shows the restriction on the device property that it has memory and such devices can be a subclass, ProgrammableDevice (right hand box).

The Figure 4.3.1-1, can be a graphical representation of a part of the device ontology in DAML+OIL with the corresponding ontology segment as shown in the listing next:

```

<daml:Class rdf:ID="Devices">
<daml:label>Device</daml:label>
<daml:subClassOf>
<daml:Restriction>
<daml:onProperty rdf:resource="#memory"/>
<daml:toClass rdf:resource="#ProgrammableDevice"/>
</daml:Restriction>
</daml:subClassOf>
</daml:Class>
<daml:Class rdf:ID="ProgrammableDevice">
<daml:label>Programmable</daml:label>
</daml:Class>
<daml:Property rdf:ID="memory"/>

```

Listing 4-1: DAML+OIL Translation of Figure 4.3.1-1

A property may be a refinement of another property, as for example; “mother of” is a refinement of “parent of” property. These two properties belong to the same class of persons called women, and it can be represented in a UML class diagram as shown below:

Figure 4.3.1-2: Example of sub-property

Similarly, the properties of some electronic devices can also be considered as refinement of other properties. For example, a class of devices can have a general property of having memory and those with Flash memory can be a refinement of that property. This may be represented, as shown in the figure below.

Figure 4.3.1-3: Example of a Sub-Property of a Device

The '*' indicates no upper limit.

The next figure, Figure 4.3.1-4 below, shows an instantiated UML class that can be used in DAML+OIL.

Figure 4.3.1-4: DAML+OIL Type Property of an example Device

The diagram shows that PC16550 chip [Ref: 8.2], with pin count 40 and type UART, is an instance of the class Devices.

The Figure 4.3.1-5 below shows the mapping between DAML+OIL cardinality and UML multiplicity of association. A class X has an association with another class Y if an object of class X needs to maintain a reference to an object of class Y. An association may be bi-directional or unidirectional. (if a single arrowhead is present in the UML diagram). A 'multiplicity' expression at the end of an association specifies how many objects of that class may take part in that relationship with a single object of the class at the other end of the association.

Figure 4.3.1-5 : DAML+OIL Cardinality and UML multiplicity

The association end that contains a single value would map to a specific cardinality value for the DAML+OIL property restriction. An association end that has range of values

would map to the minimum and maximum cardinality allowed for the corresponding property restriction.

The predefined cardinality restriction is defined in DAML+OIL as a relation that, given a specified target element, will always originate from the same source element.

The Unambiguous Property with cardinality restriction is shown below:

Figure 4.3.1-6: Example of DAML+OIL Unambiguous Property

Predefined cardinality restriction is also shown in Figure 4.3.1-7 that represents the UML notation concept of a DAML+OIL Transitive Property.

Transitive Property

A Transitive Property definition requires three or more elements. For example, given that 'Albert' is the ancestor of 'Bob', and 'Bob' is the ancestor of 'Casy', then 'Albert' is also the ancestor of 'Casy'. In a transitive property, if a property that holds true for the first

and second elements and second holds true for the third, and it must hold true for the first and third elements. In electronics, an example of transitive property can be observed in a hierarchy of Time Division Multiplexer (TDM) Telecommunication channels. For example, if channel B is a sub-channel of channel A and C is a sub-channel of B then C is as sub-channel of A. The diagram for sub-channels is show below:

Diagram 4.3.1-1 : Time Division Multiplexer channels in Telecommunication

Diagram 4.3.1-1 above may be represented graphically as shown below:

Figure 4.3.1-7: DAML+OIL Transitive Property

The above UML diagram [Figure 4.3.1-7] shows the followings:

If the first Device has a property that it is a sub-channel of the second Device, that is a sub-channel of the third Device, then the first Device is also a sub-channel of the third Device.

4.3.2. *Constraints*

In UML, a variety of graphical methods can be used to specify constraints such as UML Object Constraint Language (OCL) [42, chapter 6]. In DAML+OIL, the constraints are imposed by specifying that a class is a subclass of a restriction class. DAML+OIL axioms are used to specify the constraints [Ref: Table 3.1-2]

4.4 **Difference between DAML+OIL and UML**

4.4.1. *No Universal Classes in UML*

No universal classes exist in UML. DAML+OIL and RDFS have “universal” classes to include all other sub classes. In the previous chapter, electronic device ontology development was started by defining a universal class [Ref: 3.4.3] named “Devices”. In DAML+OIL, the Resource class includes every object as an instance, and the Literal includes every literal as an instance. DAML+OIL also has the Thing class. It is the most general class in DAML+OIL that has every object, including both the objects and literal. Thing class is equal to the union of any class and its complement. Therefore, universal

class like electronic “Devices” and the Thing class cannot be adequately represented in UML.

4.4.2. DAML+OIL Property and UML Association

The notion of property in DAML+OIL may be viewed as analogous to association in UML, but the two are not equivalent. This difference is a major cause of the difficulty of using UML for DAML+OIL.

In DAML+OIL, if person owns a cat and a company owns a truck, the property ‘owns’ is regarded to be the same semantic. In UML, that ownership will be regarded as having different meaning. Applying such semantics to electronics in ontology, a Processor chip may be defined to have “connectivity” property and that it can *connects* to the RAM, or connects to UARTs [Ref: 8.2] or any other device. The ‘connects’ in all cases is regarded to be the same semantic. In UML, each connection is regarded as different associations, one for each device. Therefore, the DAML+OIL *property* and UML *association* are not exactly the same semantics.

In DAML+OIL, the notion of property is a first class a modeling element. This also implies that one can specify a cardinality constraint (i.e. number of valid values) that will be applicable to all domains in which that property is applied. For example, a device property can be “*Operating-voltage*” and cardinality constraints can be that it operates at three voltages, say 110V, 220V and 240V.

The DAML+OIL notion of a property is defined independently of any context in which it might be used. In UML, associations (which are somewhat analogous to property) are not first-class modeling elements and UML associations are always with a class.

For example, in DAML+OIL, one can define “Father” as a property restricted to males.

```
<daml:ObjectProperty rdf:ID="hasFather">
  <rdfs:subPropertyOf rdf:resource="#hasParent"/>
  <rdfs:range rdf:resource="#Male"/>
</daml:ObjectProperty>
```

Source: <http://www.daml.org/2001/03/daml+oil-walkthru.html>

In device ontology, "hasResistance" property is restricted to Resistors devices

```
<daml:ObjectProperty rdf:ID="hasResistance">
  <rdfs:subPropertyOf rdf:resource="#hasImpedance"/>
  <rdfs:range rdf:resource="#Resistors"/>
</daml:ObjectProperty>
```

Another example in electronics is that, ‘addressing’ may be declared as a property of the devices and the cardinality constraints can be set to say 8-bit. Then this property can be applied to classes of devices such as 8-bit CPUs, RAM, Decoders, and UARTs etc.

4.4.3. Domain, property, association and range

In UML, binary association (between two entities) always has just one domain class and one range class, but a DAML+OIL property can have many domain classes, and one

range class. This causes the difficulty of applying UML in development of electronic device ontology. For example, in electronics an electronic property, “attenuation” can be declared. This property can be applied to optical domain as well as to copper domain, say in the range of Amplifier class.

4.4.4. *Non-binary association in UML*

DAML+OIL properties can only be binary (relationship between two entities), for example:

```
<daml: ObjectProperty rdf: ID = “hasMom”>  
<daml :samePropertyAs rdf:resource = “#hasMother”/>  
</daml: ObjectProperty>
```

UML allows associations that are non-binary (i.e. association with two or more entities), therefore UML non-binary association cannot be directly represented as a single DAML+OIL property. UML non-binary association must be reified as a DAML+OIL class with as many properties as UML associations. The meaning of reification varies according to the context in which it used. Reify according to Webster dictionary is regarding something that is abstract as a material thing. The concept of reification has a specific meaning in the context of object-oriented programming, namely representing something as an object that is not (typically) an object. Examples of this are things such as algorithms or states. When reifying an algorithm, one represents it by a class of its own, rather than the code of a method. Reification used in this thesis is in the context of DAML as discussed [83] in the W3C discussion paper.

Reification [83] is a useful design technique, but it has the drawback that the design can become highly complex. With reification, the design units may evolve to rapidly becoming no longer explicitly related to one another. A UML non-binary association must be reified. (although binary associations can also be reified). In ontology development, such design choices could be called ontological commitment. This means that ontology should require the minimal commitment such that it is sufficient to support the intended knowledge sharing activities. Ontology should make few claims about the world being modeled to allow interested parties committed to that ontology the freedom to specialize and instantiate the ontology as needed.

4.4.5. *Multiplicity Constraints*

In UML, multiplicity constraints constrain the number of objects that can be instantiated from the classes, for example, “man” and “women” can be only two instances in the “human” class. Constraints on associations can affect membership of objects in the classes related through the association, and in the human class example, excludes the membership of a person who is neither male nor female. With reference, to the Devices class as declared, could be viewed as only two objects; Active and Passive and electrochemical objects would not be a member.

Classes in UML do not directly affect associations. In DAML+OIL, constraints on properties are imposed somewhat indirectly by specifying that a class is a subtype of a class called a Restriction. Doing this may limit the scope of the properties being constrained by the restriction class.

4.4.6. Comparing Cardinality Constraints

A constraint from UML point of view is a requirement that restricts the instantiations that is possible for a given model. The cardinality constraints in DAML+OIL correspond to the UML multiplicity constraints on an association.

Unlike DAML+OIL, UML does not have anonymous class to define constraints.

```
<rdfs:subClassOf>
  <daml:Restriction>
    <daml:onProperty rdf:resource="#hasParent"/>
    <daml:toClass rdf:resource="#Person"/>
  </daml:Restriction>
</rdfs:subClassOf>
```

Source: <http://www.daml.org/2001/03/daml+oil-walkthru.html>

Listing: 4.4-1: Anonymous class in DAML+OIL

The incompatibility is due to the fact that in DAML+OIL properties are first-class model elements and one-directional.

The first-class feature of properties implies that it is possible to specify a cardinality (i.e. number of values) constraint all at once for every domain in which the property could potentially be used. In UML, it must be specified separately for each association (corresponding to the property), but in DAML+OIL, it can be only once. However, UML allows specification of the cardinality constraints at all of the ends of an association.

4.4.7. *Imposition of Constraints*

The semantics of 'constraints' is distinct between UML and DAML+OIL. DAML+OIL allows imposing of constraints on properties. From UML's perspective, this means that one can impose a constraint on a number of associations all at once. This modularization technique is similar to current work on aspect-oriented programming. In DAML+OIL, as in other logic based Knowledge Representation (KR) languages, constraints are axioms and from which one can perform logical inference.

4.5 **DAML+OIL effect on modularity of software design**

In DAML+OIL, descriptions can be distributed over various locations with the use of URIs [9]. A part of a class or property may reside in one site and another on a different site. Such fragmentation is uncommon in UML. Therefore, the DAML+OIL approach appears to go against the principles of modularization generally recommended in classical Software Engineering. However, the distribution of classes and properties over the Internet does help to support the reuse of data structures globally. This feature is particularly useful in the development of electronic design services with distributed intelligence. (The Figure 1.9.1-1 shows conceptualization of such as distribution in the Internet). In addition, the idea of a property being associated with multiple classes appears to be non-modular approach, but again, DAML+OIL is more flexible and facilitates reuse.

4.6 Properties, Association and Attributes

As in many other Knowledge Representation (KR) languages, in RDF and DAML+OIL properties are first-class. In DAML+OIL, properties can have values that are literal as well as objects, so properties subsume both the association and the attribute concepts used in UML.

4.6.1. *Properties without domain*

In DAML+OIL, a property need not have any domain or range specifications at all. This is shown in the example below:

```
<daml:DatatypeProperty rdf:ID="associatedData">
  <rdfs:comment>
    associatedData is a DatatypeProperty without a range restriction.
  </rdfs:comment>
</daml:DatatypeProperty>
```

Source: <http://www.daml.org/2001/03/daml+oil-walkthru.html> page 14

Listing: 4.6-1: No domain property

However, when it does, it may have multiple domains and only one range. When attempting to map each association to a different RDFS property, then RDFS properties having multiple domains will not be expressible in UML. Two UML associations may have the same name, but they are still different associations because the names are in different namespaces. When trying to map two UML associations having the same name

to the same RDFS property (DAML+OIL is based on RDFS), then it could violate the requirement that an RDFS property have only one range.

4.6.2. *Property as a sub property*

DAML+OIL allows one property to be a sub property of another. The close similarity of this in UML is that one association is a specialization of another i.e. mother is a specialization of parent. It may not be possible to specify accurately the semantics of property as an association specialization.

4.7 **Namespaces in DAML+OIL and UML**

XML namespace is quite simple. It is a prefix belonging to a group of elements and attributes. XML namespace unlike UML does not support hierarchy of packages. DAML+OIL objects need not be explicitly named. DAML+OIL objects can be anonymous. It can be related to other objects using XML containment (i.e. XML document is tree structured). When mapping from UML to DAML+OIL or the vice versa there must be means for ensuring that names can be distinguished. This is the most important issue would be the mapping of names.

4.8 Linking & protection in UML and DAML+OIL

Unlike, DAML+OIL, in UML, the mechanism of “linking” is different. Instead of using URIs [9], UML uses names in a large number of namespaces. Each class can have its own namespace for its attributes and associations. UML enriches this by the ability to specify private, protected and public scopes. DAML+OIL has no such scope to protect classes.

4.8.1. Comparing DAML+OIL with property patterns

UML diagrams usually model design patterns and their compositions. It is therefore appropriate to discuss DAML+OIL with respect to Property Pattern that usually occurs in Design Patterns. Design patterns are a specialized area of computer science that uses concepts analogous to the use of patterns in other branches of science, such as biology and engineering. In architecture and civil engineering, certain geometrical shapes may be reused to form patterns that solve certain problems or provide desirable features.

Figure 4.8.1-1 : Examples of *geometrical* design patterns

Analogous to geometrical patterns, [Figure 4.8.1-1] which may be used in the construction of buildings, the use of data structures (classes, objects and tags) in a certain configuration can be considered as design patterns for computer based solutions. In Design Pattern, the lessons learnt from a particular architecture and design that is a solution to one problem can be applied as is, or in a modified form to solve another similar problems. Design patterns can support large-scale reuse of software architecture and design that extends the Object Oriented approach to system design.

The DAML+OIL Class Constructors and Axioms shown in Table 3.1-1 and Table 3.1-2 can support the development of ontologies for system engineering and modeling that is to be based on the use of Design Patterns techniques.

The geometrical patterns have properties that can be described and utilized advantageously. Similarly, patterns consisting of certain configuration of data structures, objects and classes can have the desirable properties, which can be utilized favorably and reused. For example, certain data structure constructs can be considered as “*property pattern*”, “*virtual object pattern*” etc. However, it should be noted that these patterns are not identical to language elements (listed below) related to property of DAML+OIL. (Examples of DAML+OIL property elements are: daml: ObjectProperty, daml: onProperty, daml: subPropertyOf, daml: samePropertyAs, daml: transitiveProperty, daml: uniqueProperty, daml: UnambiguousProperty.)

Whereas, the property in DAML+OIL, relates only in very high-level terms the properties of classes, the “*property pattern*”, “*virtual object pattern*” typically used in design pattern are more specific and low code level in detail. However, DAML+OIL can augment these low-level properties (e.g. property pattern and the virtual object pattern) by expressing them in high level (abstract) terms. DAML+OIL can thus provide an interface to pattern properties to Artificial Intelligence (AI) and Knowledge Base Systems.

An example of Design Pattern of a “*real-world object* and “*virtual objects*” was submitted at Conference on Object-Oriented Programming, Systems, Languages, and Applications 1993 (OOPSLA 93), by Stanfanie Brueninghaus and Michael Weiss.

[REF: APPENDIX 8: VIRTUAL OBJECTS]

The properties of patterns maybe expressible in hierarchical abstraction in DAML+OIL and may be diagrammatically represented as shown below:

Figure 4.8.1-2: DAML+OIL abstraction of properties of patterns

4.8.2. Example of Real, virtual, view and agent pattern

[REF: APPENDIX 8: VIRTUAL OBJECTS]

```
class car {                // real-world object
float theInitialCost;

string theNumberPlate;
carView viewRequest( taxationAgent )

    { return carView( theInitialCost ); }
... }

class carView {            // view definition
float theInitialCost;
void carView( float someInitialCost )
    { theInitialCost = someInitialCost; }
... }

class carInBalanceSheet {  // virtual object
float theResidualValue;
integer thePredefinedServiceLife;
... }

class taxationAgent{       // agent
float Depreciation( carView );
... }
```

Listing: 4.8-1: Real, virtual, view and agent pattern

4.8.3. Design pattern properties in terms of DAML+OIL

The example of “real-world”, “view-definition”, “virtual-objects” and “agent” may be defined in terms of DAML+OIL properties in the following manner:

```

<daml :ObjectProperty rdf:ID = "real-world-object"/>
<rdfs: domain rdfs: resource= "legal-domain" >
<rdfs: range rdfs: resource= "taxation" >
</daml :ObjectProperty>

```

```

<daml :ObjectProperty rdf:ID = "view-definition"/>
<rdfs: domain rdfs: resource= "legal-domain" >
<rdfs: range rdfs: resource= "taxation" >
</daml :ObjectProperty>

```

```

<daml :ObjectProperty rdf:ID = "virtual-objects"/>
<rdfs: domain rdfs: resource= "legal-domain" />
<rdfs: range rdfs: resource= "taxation" />
</daml :ObjectProperty>

```

```

<daml :ObjectProperty rdf:ID = "agent"/>
<rdfs: domain rdfs: resource= "legal-domain" />
<rdfs: range rdfs: resource= "taxation" />
</daml :ObjectProperty>

```

It may be noted that these properties may be assigned to appropriate classes and the domains and range of these properties may be adjusted as per its applicability.

4.8.4. Pattern in terms of DAML+OIL

The classes defined may be written in DAML+OIL as shown below:

```

<daml :class rdf:ID = "carView"/>
<rdfs: label> view </rdfs: label/>
<rdfs: comment> <rdfs: comment/>
</daml :class>

```

```
<daml :class rdf:ID = "carInBalanceSheet"/>
<rdfs: label> balance </rdfs: label/>
<rdfs: comment> <rdfs: comment>
</daml :class
```

```
<daml :class rdf:ID = "taxationAgent"/>
<rdfs: label> agent </rdfs: label/>
<rdfs: comment> <rdfs: comment/>
</daml :class>
```

It may be noted that the above design pattern [APPENDIX 8: VIRTUAL] may be made applicable to “trucks” or “buses” or other vehicles by the use of daml: sameClassAs

Observation 4.8-1:

It can be observed from the above example, that DAML+OIL compliments the various pattern properties due to the features of being able to define classes and properties separately and later assign one to the other.

4.9 UML, Semantic Web and the layer cake model

With respect to the Semantic Web, UML can be used for modeling services at the different levels of the layer cake model [Figure 1.10.3-1] with classes, objects, charts and diagrams required to model each level. As for example, the electronic component catalog [Ref: section 0] that was modeled in UML can be considered to be at the XML and XML-Schema level of the layer cake model.

4.10 DAML+OIL and ASN.1

There is no direct relationship between DAML+OIL and ASN.1. DAML+OIL uses underlying support of RDF Schema and XML Schema. There is similarity between ASN.1 and XML Schema [6]. Both ASN.1 and XML Schema deal with data formats. ASN.1 was developed to describe the format of the data i.e. bit, integer, string, Boolean, etc. (and not the meaning). ASN.1 is ‘communication systems orientated’, rather than ‘semantic system orientated’. The goal of ASN.1 is to facilitate the reconstruction of the original data format at the destination location after the data is transmitted. The ISO [44] standards 8824 (ASN.1) defines the formatting and ISO 8825 (BER), encoding and decoding rules.

For comparison, ASN.1 is similar to XML Schema. However, XML Schema provides much richer format creation and checking of not only data but also the entire document. There is currently work in progress on encoding ASN.1 defined structures to XML thus producing a specification for XML Encoding Rules (XER) [43].

In XML Schema, the destination needs to know in advance the format of the incoming document via DTD or XML Schema. The XML Schema between a particular pair of sender and receiver may be unique. In ASN.1, the data format is fixed by the standard and the destination can decode without prior knowledge of the sender’s format.

The goal of the XML family of Recommendations and DAML+OIL is beyond the goal of ASN.1 and is intended to conveying meaning rather than being limited to data

representation as in ASN.1. ASN.1 can be considered a technology for the lower implementation level rather than for high level modeling of services in the Semantic Web.

5. CONCLUSIONS AND CLAIMS

The objective of the research that was stated in section 1.3 was only *partially achieved* because the ontologies for the devices could not be fully developed. Electronic devices, such as integrated circuits and programmable devices were found to be highly complex and required extensive amount of details. The laser device ontology [<http://www.sce.carleton.ca/~siahmed/laser>] that was developed in this thesis is insufficiently detailed for use in the envisioned electronic product design services in the Semantic Web. However, this research has demonstrated the principles and techniques that can be used for the development of electronic device ontology in DAML+OIL.

5.1 Summary of the conclusion.

It can be summarized that DAML+OIL appear to be suited for inscribing electronic device ontologies at a high level. UML is a useful ontology-modeling tool, but requires extension to model the ontologies written in DAML+OIL.

5.2 Summary of contributions

The research undertaken in this thesis has resulted in the following *new knowledge or observations* with regard to the envisioned electronic product design services in the Semantic Web, the development of device ontologies and the use of UML for modeling.

- 1 *Architecture for services:* A high-level architecture for engineering services in the Semantic Web consisting of three virtual planes was illustrated that may be used as a prototype service architecture of the Semantic Web.
- 2 *Device ontologies:* Development of device ontologies in DAML+OIL for the static features, such as DC characteristics is straightforward, provided the complex interrelationship between the device parameters is not specified. It was discovered that each parameter and signals of the electronic device can be declared with a unique ID using `daml: ObjectProperty` for the name and `daml: ObjecttypeProperty` for the values of those parameters.
- 3 *Limitations:* (a). It was also realized from the examination of the device specification documents that a complete conversion of electronic device data sheets and specifications into machine interpretable ontological form is extremely difficult and remains a major unresolved problem. (b). The ability of the current DAML+OIL to ontologically express the AC characteristics of the device is weak because it requires the notion of time to be established in machine comprehensible form. DAML time ontology is currently in research stage. However, the AC signal names can be declared in the same manner as the DC signals.
- 4 *Interface to KBS:* Unlike some of the current standards [Section 2.10.4] that promote machine readability of the electronic device specifications, the device specification in ontological form using DAML+OIL can provide several extra benefits. One of those

benefits is that it can facilitate the interface of the device specifications to Knowledge Base Systems (KBS).

- 5 *Device classes and properties:* In DAML+OIL, classes and properties are first class modeling elements. Therefore, electrical properties could be declared independently of device classification. This feature of DAML+OIL provides an excellent means to apply electrical properties to the appropriate classes very selectively. In DAML+OIL, it is also possible to define device classes with the properties integrated.
- 6 *Careful use of range:* It was discovered that careful use of rdf: range assignment was important in the development of device ontology. As for example, it was undesirable to assign the range of “*hasGroundPins*” property to all types of electronic devices, such as, diodes with the premise that all electronic devices must have voltage and ground pins.
- 7 *Use of XML tags vs. daml: ObjectProperty for device properties:* It was seen that each property of device can be represented either with a specific XML tag or with a unique daml: ObjectProperty ID. The disadvantage of using XML tags is that the semantics of the signal becomes too localized and additional sub-properties cannot be appended to that device signal easily. Whereas, defining device signals using daml: ObjectProperty provides the flexibility to add additional properties to the signal (from any part of the world using the URI) with the use of daml: subPropertyOf, or with other appropriate DAML+OIL language elements.

- 8 *Signal names of the devices:* Comparison of the various device vendor datasheets showed that manufacturers often use dissimilar terminology even for the same type of signals and properties of devices belonging to the same class. From the Semantic Web perspective, a common industry standard needs to be established for naming the device signals and properties to facilitate the development of automated design services in the web. However, dissimilar signal names may be transformed from one form to another, using XSLT provided the semantics of signal properties are kept identical.
- 9 *Electronic device classification:* There were no serious problems to initially start the device ontology development with a tree diagram. However, since DAML+OIL classes can be sub-classes of several super classes and a device can inherit properties of several super classes of devices, the tree diagram can become inadequate for representing all the properties of the device.
- 10 *Applicability of UML in device ontology:* UML can be used to model ontology written in DAML+OIL to some extent. The main drawback is that UML does not have a concept that is exactly equivalent to the concept of DAML+OIL *property*, which is a first class modeling element. The closest similarity to DAML+OIL property in UML is *Association*. However, if the property feature of the DAML+OIL was added to UML, with the semantics that was identical to DAML+OIL property, then UML could be more readily used for modeling ontologies written in DAML+OIL.

11 XML Schema and ASN. 1 XML Schema is user definable and provides more varieties of document structure and data type that is more flexible than ASN.1

12 Property pattern: The semantics of DAML+OIL property elements are different from property pattern used in the domain of Design Patterns. Property pattern normally describes object characteristics and recurring pattern at the code or almost codes level, whereas the property language elements in DAML+OIL relate characteristics of entities at a higher abstract level.

5.3 Claim of original contribution

No published or informal literature was found that considered DAML+OIL from electronic device ontology perspective. Prior to the research undertaken in this thesis, it was not known how DAML+OIL could be used to develop ontologies for electronic devices. This thesis has experimentally shown with several prototype ontology segment examples how the device ontologies can be developed in DAML+OIL.

5.4 Future work

For the envisioned product design services in the Semantic Web, further research is required to extend the findings of this thesis. Some of these are:

1. Determination of how the dynamic (AC characteristics) of the electronic devices can be ontologically represented in DAML+OIL.
2. How the interrelationships between device signals can be represented
3. Determination of how the device and signal names could be globally standardized and mapped to well-defined properties of the device signals to facilitate recognition of the signal properties by software agents in the Semantic Web.

6. APPENDIX 1: WHAT IS ONTOLOGY?

Many definitions of ontology exist in literature [31] that ranges from a very abstract philosophical view to concrete computer database specific concept. Initially, ontology was used to describe the philosophical study of the nature and organization of reality.

Ontology is a model of reality of the world and the relationships between objects written in a language. In very simple terms, ontology is defined as a common vocabulary that can be used to share meaning and associated concepts for describing a domain. A few examples of different domains are Biology, Fiber Optics, Wireless, Semiconductor, Medicine, Thermodynamics, Geography, etc.

Domains may be subdivided into sub-domains, mini and micro domains and to any suitable size or granularity. For example, electronic circuit design domain may be described ontologically and sub divided into, analog and digital sub domains. Digital design may be subdivided into discrete and integrated circuit domains.

6.1 Various definitions of Ontology

Several definitions of ontology also exist in the Artificial Intelligence (AI) literature. McGuinness [22] identifies that some definitions of ontology contradict with each other. Guarino and Giaretta [15] indicated that the term 'ontology' is often vague.

The term 'ontology' has been adopted by the AI community to refer to a set of concepts or terms that can be used to describe some area of knowledge or to build a representation of it. In the Artificial Intelligence (AI) area, the most referred definition is the one from Tom Gruber [29]

"An ontology is an explicit specification of a conceptualization." In this definition, a conceptualization is an abstract view of the world, along the lines of Genesereth and Nilsson [17].

The American Heritage Dictionary defines ontology as *"the branch of metaphysics that deals with the nature of beings"*.

Michel Klein, Dieter Fensel, Frank van Harmelen, and Ian Horrocks [18] define ontology in terms of heterogeneous systems.

"Ontology provides a shared and common understanding of domain that can be communicated between people and heterogeneous and distributed application systems"

The main idea of ontology according to Michael Klein et al is domain knowledge, sharing and commonality, and that ontology use is between people and computers and between computers. However, the mention of 'heterogeneous' in their definition appear to be unnecessary, since it could imply that homogeneous system might be excluded in

ontologically constructed systems. Therefore, their definition may be improved by deleting the word 'heterogeneous' and reworded as follows:

“Ontology provides a shared and common understanding of domain that can be communicated between people and distributed application systems”

The IEEE Communication Society [51] defines “*ONTOLOGIES are hierarchical (tree-like) classifications of all the articles and people in a collection*”.

Chandrasekaran et al [26] describes ontology as a specialization of a domain

“Ontology is a representation vocabulary, often specialized to some domain or subject matter”

and further identifies that “*...it is not the vocabulary as such that qualifies as an ontology, but the conceptualizations that the terms in the vocabulary are intended to capture. Thus, translating the terms in an ontology from one language to another, for example from English to French, does not change the ontology conceptually*”.

This observation implies that the core ontology for developing intelligence for the Semantic Web services should be domain specific and that the same ontology can be used to provide services in multiple human languages. Although, in some domains (e.g. in French Literature), it may be impossible to translate semantically the exact meaning from

one human language to another, this semantic problem is not likely to be severe in the science and technology domains. This is because unlike social sciences, where concepts can be contentious, most concepts in engineering are non-controversial.

William Swartout and Austin Tate [27] emphasis on the distinction between an ontology and knowledge as follows:

“An ontology provides a set of concepts and terms for describing some domain, while a knowledge base uses those terms to represent what is true about some real or hypothetical world”. They further clarify that, *“...thus, a medical ontology might contain definitions for terms such as ‘leukemia’ or ‘terminal illness,’ but it would not contain assertions that a particular patient had some disease, although a knowledge base might”*

6.1.1. *Ontology as Controlled Vocabulary*

McGuinness [28] describes that: *“One of the simplest notions of a possible ontology may be a controlled vocabulary – i.e., a finite list of terms”*.

In telecommunication standards, ontology based on “controlled vocabulary” [57] offers some prospects in converting technical standards into machine-readable form. A finite “controlled vocabulary” within a technical standard or a part of a standard can be created by the technical domain authority or organization, such as for example, the ATM Forum [59], Frame Relay Forum [58] etc. The meaning of some of the words can be precisely

defined, while other words can only be abstractly defined in machine-readable form. As an example, the acronym, ABR or 'Available Bit Rate' in ATM Forum can be exactly defined but the word "Performance" can only be abstractly defined since the meaning of "performance" is extremely difficult to encode in machine.

6.1.2. *Ontology as a glossary*

McGuinness [28] considers glossary as a potential ontology specification (but considers it unsuitable for machine agents). A list of terms and meanings, such as a glossary [57] in Frame Relay can potentially be used for ontology development. However, the problem is that the specification written in natural language can be ambiguous and depend on the author' or the reader 's interpretation. Therefore, between software agents, it is extremely difficulty or impossible to derive meaning from a glossary written in natural language. However, in the Semantic Web, at the human interface end of the service, ontology written as a glossary may be presented to the human user.

6.1.3. *Ontology as a Thesauri*

The advantage of thesauri over glossary is that it provides some additional semantics in their relations between terms, such as synonym relationships. Software agents may be able to interpret their relationships unambiguously. However, the difficulty in treating thesauri is that most thesauri do not provide a hierarchy and the consensus in the research

community is that an explicit hierarchy must be included before something is considered ontology.

6.1.4. *Test of a well developed ontology*

If device ontology is properly constructed, then that ontology can be used for the same class of devices. With properly constructed ontology, the software agent will be able to program, for example, the SC26C562 CDUSCC [Section 8.1] and PC16550 [Section 8.2] chip and other similar devices in the same class. In fact, the ability to successfully program any device in the same class or subclass by an agent can be “litmus” test of how successfully the functionality of devices can be represented ontologically in machine-readable and machine-comprehensible form.

6.1.5. *Ontology as data structure*

The development of ontology is often akin to defining a set of data and their structure for other programs to use. Such an ontology design originated from the literature on object-oriented design.

Some definitions of ontology are more appropriate for human interface (such as simple vocabulary) while others (that use classes, properties and slots) are more suitable for semantic communication between machine agents.

6.1.6. *Ontology as classes*

Ontology is a formal explicit description of concepts in a domain of discourse. It may be represented by classes, which is sometimes called concept and by properties represented by slots and sometimes called roles of each concept. It also describes various features and restrictions called facets or sometimes called role restrictions on slot. Ontology with a set of individual instances of classes constitutes a knowledge base.

6.1.7. *Ontology vs. classes in Object Oriented Design*

Ontology development is different from designing classes and relations in object-oriented programming. Object-oriented programming activity is primarily about methods on classes where a programmer makes design decisions based on the operational properties of a class, whereas an ontology designer makes these decisions based on the structural properties of a class. Therefore, a class structure in an object-oriented program and relations among classes in ontology are different from the structure for a similar domain.

6.1.8. *Ontology as a subclass hierarchies*

Ontologies can include strict subclass hierarchies. In these systems if x is a super-class of y, then if an object is an instance of y, it necessarily implies that the object is an instance of x. Strict subclass hierarchies are necessary for exploitation of inheritance.

6.1.9. Expressiveness of ontology

Ontologies may need to express more information. As an example, there may be a need to fill in the value of one property based on equation using values from other properties. Examples of very expressive ontology languages are seen Ontolingua [[35] or CycL [89] that allow ontologists to specify first order logic constraints between terms such as part-whole relationships, disjoint classes, inverse relationships etc.

6.2 Ontology and the Semantic Web

At this point, it is useful to compare the definition of Semantic Web and ontology and to note the overlapping ideas between the two. The table below is an attempt to show the similarity between the two.

Semantic Web (Berners-Lee et al)	Ontology (Michel Klein et al)
Information, meaning co-operation people current web Extension Meaning and information	Common understanding distributed application people common understanding domain heterogeneous Application

Table 6.2-1 : Comparision Ontology and Semantic Web

In ontology, 'common understanding of the domain' can be equated to idea of 'meaning and information'

6.3 Reasons for developing ontology for engineering services

Some of the reasons are:

- To share common understanding of the structure of engineering information among people or software agents, such as common standards and vocabulary in component data sheets
- To enable reuse of domain knowledge
- To make domain assumptions
- To separate domain knowledge from the operational knowledge
- To analyze domain knowledge (e.g. estimate accuracy in data compression)

Musen [36] and Gruber [29] has identified that sharing of common understanding of the structure of information among people or software agents is one of the dominant motives for developing ontologies. For example, suppose several different Web sites offer engineering design services. If these Web sites use the same underlying ontology of the terms, say to describe a silicon chip of a certain class, then software agents in the Internet can extract and aggregate information from these different sites. The agents can use this aggregated information to provide design services in the Semantic Web.

The successful development of the various domain specific ontologies for the next generation Internet will be critical for the construction of the Semantic Web services.

6.3.1. Separating the domain knowledge from the operational knowledge

McGuinness and Wright [38] explain that another important use of ontologies is in the task of configuring a product from its components according to a required specification. It can implement a program that does this configuration independent of the products and components themselves.

6.4 Conclusion

Ontology has many definitions that vary from highly philosophical abstract concept to exact descriptions of concrete physical items. There are many ways and flavors of building and using ontologies appropriate to a particular application.

7. APPENDIX 2: XML IN TELECOMMUNICATION

1. "Contents Listing for XML Applications and Industry Initiatives"
<http://www.oasis-open.org/cover/xml.html#contents>
2. Telecommunications Markup Language (tML).
<http://www.oasis-open.org/cover/atis-tML.html>
3. Telecommunications Interchange Markup (TCIF/IPI)
[http://www.oasis-open.org/cover/xml.html#xml-tim.](http://www.oasis-open.org/cover/xml.html#xml-tim)
4. Information and Content Exchange (ICE) Protocol.
<http://www.oasis-open.org/cover/ice.html>
5. Product Definition Exchange (PDX).
<http://www.oasis-open.org/cover/pdx.html>
6. Timing Diagram Markup Language (TDML).
<http://www.oasis-open.org/cover/tdml.html>
7. ECIX Component Information Dictionary Standard (CIDS).
<http://www.oasis-open.org/cover/xml.html#cids>
8. XML and Electronic Design Automation (EDA).
<http://www.oasis-open.org/cover/xmlAndEDA.html>
9. ECIX QuickData Specifications.
<http://www.oasis-open.org/cover/ecix-qd.html>

10. Electronic Component Information Exchange (ECIX) and Pinnacles Component Information Standard (PCIS).

<http://www.oasis-open.org/cover/xml.html#ecixXML>

11. Mathematical Markup Language (MathML).

<http://www.oasis-open.org/cover/xml.html#xml-mml>

12. XML-Based Provisioning Services.

<http://www.oasis-open.org/cover/provisioningServices.html>

13. WAP Wireless Markup Language Specification (WML).

<http://www.oasis-open.org/cover/wap-wml.html>

14. Synchronized Multimedia Integration Language (SMIL).

<http://www.oasis-open.org/cover/smil.html>

8. APPENDIX 3 : DATA SHEETS OF ELECTRONIC DEVICES

8.1 **Phillips Semiconductor.**

http://www.semiconductors.philips.com/acrobat/datasheets/SC26C562_2.pdf

8.2 **National Semiconductor Universal Asynchronous Receiver Transmitter.**

<http://www.national.com/ds/NS/NS16C552.pdf>

8.3 **Texas Instrument Chip 16C550 data sheet.**

<http://www-s.ti.com/sc/ds/tl16c550b.pdf>

8.4 **Zarlink Semiconductor.**

http://assets.zarlink.com/products/data/datasheets/zarlink_ZL60005_Jul_2002.pdf

8.5 **Digikey Catalog.**

<http://dkc3.digikey.com/T023/V5P2.html>

8.6 **Texas Instrument Mixed Signal Processors.**

http://focus.ti.com/docs/search/paramsearch.jhtml?familyId=342&tfsection=param_table&templateId=1

9. APPENDIX 4 KNOWLEDGE REPRESENTATION LABS

1. Massachusetts Institute of Technology, Artificial Intelligence Lab.
<http://medg.lcs.mit.edu/ftp/psz/k-rep.html>
2. Knowledge Systems Laboratory, Stanford University, Department of Computer Science, Stanford, CA 94305-9020.
<http://www-ksl.stanford.edu/>
3. Knowledge Representation Laboratory, Computer Science and Information Management Program at Asian Institute of Technology.
<http://kr.cs.ait.ac.th/>
4. The Laboratory for Applied Ontology, (ISTC-CNR)
c/o ISIB-CNR (formerly LADSEB-CNR), 35127 Padova, ITALY.
<http://ontology.ip.rm.cnr.it/>
5. KR, Inc.
<http://www.kr.org/kr/>
6. Classic.
<http://www.bell-labs.com/project/classic/>
7. Ontology-based Knowledge Representation for Bioinformatics, Department of Computer Science and School of Biological Sciences
University of Manchester
<http://img.cs.man.ac.uk/stevens/onto/>

10. APPENDIX 5: DAML+OIL ONTOLOGY LISTING FOR LASER

(This is experimental ontology has been placed in Carleton University web site

<http://www.sce.carleton.ca/~siahmed/laser>) 15 January 2003.

```
<?xml version="1.0" ?>
<rdf:RDF      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
  xmlns:xsd="http://www.w3.org/2000/10/XMLSchema#"
  xmlns="http://carleton8.tripod.com/laser.daml#">
- <daml:Ontology about="">
  <daml:versionInfo>July 2002</daml:versionInfo>
  <rdfs:comment>Ontology for Laser diodes</rdfs:comment>
  </daml:Ontology>
  <daml:Class rdf:ID="LaserDevices" />
- <daml:Class rdf:ID="VCSEL_Devices">
  <rdf:subClassOf rdf:resource="#LaserDevices" />
  </daml:Class>
- <daml:ObjectProperty rdf:ID="hasOpticalPower">
  <rdf:domain rdf:resource="#LaserDevices" />
  <rdf:range rdf:resource="#VCSEL-Devices" />
  </daml:ObjectProperty>
- <daml:ObjectProperty rdf:ID="hasSlope-Efficiency">
  <rdf:domain rdf:resource="#LaserDevices" />
  <rdf:range rdf:resource="#VCSEL-Lasers" />
  </daml:ObjectProperty>
- <daml:ObjectProperty rdf:ID="hasBeam-Divergence">
  <rdf:domain rdf:resource="#LaserDevices" />
  <rdf:range rdf:resource="#VCSEL-Lasers" />
  </daml:ObjectProperty>
- <daml:ObjectProperty rdf:ID="hasBandwidth">
  <rdf:domain rdf:resource="#LaserDevices" />
  <rdf:range rdf:resource="#VCSEL-Lasers" />
  </daml:ObjectProperty>
- <daml:ObjectProperty rdf:ID="hasPeakWavelength">
  <rdf:domain rdf:resource="#LaserDevices" />
  <rdf:range rdf:resource="#VCSEL-Lasers" />
  </daml:ObjectProperty>
- <daml:ObjectProperty rdf:ID="hasSpectralWidth">
  <rdf:domain rdf:resource="#LaserDevices" />
  <rdf:range rdf:resource="#VCSEL-Lasers" />
  </daml:ObjectProperty>
- <daml:ObjectProperty rdf:ID="hasForwardVoltage">
  <rdf:domain rdf:resource="#LaserDevices" />
```

```

<rdf:range rdf:resource="#VCSEL-Lasers" />
</daml:ObjectProperty>
- <xsd:simpleType name="range-forward-voltage">
- <!-- range-forward-voltage is an XMLS union datatype
-->
- <!-- values of may be integers for some manufacturers or decimal for others
-->
- <xsd:union>
- <xsd:simpleType>
  <xsd:restriction base="integer" />
</xsd:simpleType>
- <xsd:simpleType>
  <xsd:restriction base="decimal" />
</xsd:simpleType>
</xsd:union>
</xsd:simpleType>
- <daml:DatatypeProperty rdf:ID="Forward-Voltage">
  <rdfs:comment>Forward-Voltage is a DatatypeProperty whose range is range-forward-voltage.</rdfs:comment>
  <rdf:type rdf:resource="http://www.daml.org/2001/03/daml+oil#UniqueProperty" />
  <rdfs:range rdf:resource="http://www.daml.org/2001/03/daml+oil-ex-dt# range-forward-voltage" />
</daml:DatatypeProperty>
- <daml:DatatypeProperty rdf:ID="BeamDivergence">
  <rdfs:comment>BeamDivergence is a DatatypeProperty whose range is xsd:non-negative integer BeamDivergence is also a UniqueProperty (can only have one BeamDivergence)</rdfs:comment>
  <rdf:type rdf:resource="http://www.daml.org/2001/03/daml+oil#UniqueProperty" />
  <rdfs:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#nonNegativeInteger" />
</daml:DatatypeProperty>
- <daml:DatatypeProperty rdf:ID="BandWidth">
  <rdfs:comment>BandWidth s a DatatypeProperty whose range is xsd:decimal. BandWidth is also a UniqueProperty (can only have one BandWidth)</rdfs:comment>
  <rdf:type rdf:resource="http://www.daml.org/2001/03/daml+oil#UniqueProperty" />
  <rdfs:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#decimal" />
</daml:DatatypeProperty>
- <VCSEL-Diode rdf:ID="ZL6005">
  <Forward-Voltage>2.5</Forward-Voltage>
  <BandWidth>1</BandWidth>
  <BeamDivergence>11</BeamDivergence>
</VCSEL-Diode>
</rdf:RDF>

```

11. APPENDIX 6 : SUMMARY OF RDF

(Derived from XML/RDF web site)

rdf:about identifies the resource to which the following statements apply. The value of the attribute is a URI reference.

rdf:Description The RDF XML syntax groups multiple statements for the same resource into this **rdf:Description** element. The **rdf:Description** element references, in the **rdf:about** attribute, the resource to which each of the statements applies. If the resource does not yet exist (i.e., does not yet have a resource identifier), then a **rdf:Description** element can supply the identifier for the resource using an **rdf:ID** attribute.

rdf:ID **rdf:ID** also identifies the resource to which the following statements apply. The resource will be created in the default namespace of the declaring document. The resulting URI is made by concatenating the namespace with the value of the attribute.

rdf:li is used by RDF/XML as a convenience element to avoid having to explicitly number each member of a **rdf:Bag**, **rdf:Seq**, or **rdf:Alt**. The **rdf:li** element assigns the properties **_1**, **_2**, and so on as necessary. The element name "li" was chosen as a mnemonic with the term "list item" from HTML.

rdf:parseType Attribute. **rdf:ParseType** is an attribute whose values are either **Resource** or **Literal**. The value is case-sensitive. DAML+OIL introduced the **daml:collection** value, see **daml:collection**

`rdf:RDF` is a simple wrapper that marks the boundaries in an XML document between which the content is explicitly intended to be mapped into an RDF data model instance, including RDF, RDFS, and DAML+OIL.

`rdf:resource` is used in a property declaration to specify that some other resource, not a literal, is the value of the property.

Literal parsetype means that the content of the element is a literal and any markup will not be interpreted by RDF.

Resource parsetype specifies that the element content must be treated as if it were the content of an `rdf:Description` element.

`xml:lang` attribute may be used as defined by XML to associate a language with the property value.

`daml:collection` Attribute value. DAML+OIL needs to represent unordered collections of items (also known as bags or multisets) in a number of constructions, such as `daml:intersectionOf`, `daml:unionOf`, `daml:oneOf`, `daml:disjointUnionOf`. DAML+OIL exploits the `rdf:parseType` attribute to extend the syntax of RDF with a convenient notation for such collections. Whenever an element has the `rdf:parseType` attribute with value "`daml:collection`", the enclosed elements must be interpreted as elements in a list structure, constructed using the elements `daml:List`, `daml:first`, `daml:rest` and `daml:nil`.

`rdf:Alt` represents an instance of an Alternative container resource type. It represents alternatives for the (single) value of a property. The members of the collection are denoted by `rdf:li` in the XML syntax and by `_1`, `_2`, `_3` etc. in the model.

`rdf:Bag` represents an instance of a Bag container resource type. A Bag is an unordered list of resources or literal. The members of the collection are denoted by `rdf:li` in the XML syntax and by `_1`, `_2`, `_3` etc. in the model.

`rdf:Property` represents the subset of RDF resources that are properties. A property is a specific aspect, characteristic, attribute, or relation used to describe a resource.

`rdf:Seq` represents an instance of a Sequence container resource type. A Sequence is an ordered list of resources or literal. The members of the collection are denoted by `rdf:li` in the XML syntax and by `_1`, `_2`, `_3` etc. in the model.

`rdf:Statement` When a resource represents a reified statement and has an `rdf:type` of `rdf:Statement`, then that resource must have a `rdf:subject` property, one `rdf:object` property, and one `rdf:predicate` property. A reified statement is a statement about another statement.

`rdf:object` identifies the value of the property in the modeled statement. The value of the object property is the object in the original `rdf:Statement`.

`rdf:predicate` identifies the original property itself in the modeled statement. In our example, the resource `"#member"` represents the member property.

`rdf:subject` identifies the resource being described by the modeled statement; that is, the value of the subject property is the resource about which the original `rdf:Statement` was made. Therefore, `"#jsmith"` was our `rdf:subject`.

`rdf:type` indicates that a resource is a member of a class, and thus has all the characteristics that are to be expected of a member of that class. Note that a resource can be an instance of more than one class.

`rdf:value` specifies the value of a property: it can be a literal or a resource defined by the attribute `rdf:resource`.

`rdfs:Class` corresponds to the generic concept of a Type or Category, similar to the notion of a Class in object-oriented programming languages. When a schema defines a new class, the resource representing that class must have an `rdf:type` property whose value is the resource `rdfs:Class`.

`rdfs:Resource` Anything described by RDF expressions is called a resource and is considered to be an instance of the class `rdfs:Resource`. The RDF class `rdfs:Resource` represents the set called "Resources" in the formal model for RDF presented in section

`rdfs:comment` is a human-readable description of a resource.

`rdfs:domain` indicates the classes on whose member a property can be used. (each containing a class-expression). If there are multiple domains, then according to DAML+OIL semantics, the

domain is the union of all domain specifications. Warning: this is different from the semantics in RDFS.

`rdfs:isDefinedBy` is a subproperty of `rdfs:seeAlso` and indicates the resource defining the subject resource. Its most common use is to identify an RDF schema that is not otherwise identified, i.e. the URI in the resource attribute does not indicate the URI of its schema.

`rdfs:label` is a human-readable version of a resource name and it can only be a string literal.

`rdfs:range` indicates the classes that the values of a property must be members of. If there are multiple ranges, then according to DAML+OIL semantics, the range is the union of all range specifications. Warning: this is different from the semantics in RDFS.

`rdfs:seeAlso` specifies a resource that might provide additional information about the subject resource.

`rdfs:subClassOf` is a transitive property that specifies a subset-superset relation between classes. Both the subject and object of the `rdfs:subClassOf` property must be class-expression.

`rdfs:subPropertyOf` is an instance of `rdf:Property` used to specify that one property is a specialization of another. A property can never be declared to be a subproperty of itself, nor of any of its own subproperties.

12. APPENDIX 7: SUMMARY OF DAML+OIL LANGUAGE ELEMENTS

(Language Elements listed below are hyper-linked to DAML web site in the soft copy of this document and can be accessed with mouse click)

cardinality cardinalityQ Class complementOf Datatype DatatypeProperty
DatatypeRestriction Datatype value differentIndividualFrom Disjoint disjointUnionOf
disjointWith domain equivalentTo hasClass hasClassQ hasValue imports intersectionOf
inverseOf maxCardinality maxCardinalityQ minCardinality minCardinalityQ ObjectClass
ObjectProperty ObjectRestriction oneOf onProperty Ontology Property range Restriction
sameClassAs sameIndividualAs samePropertyAs subClassOf subPropertyOf toClass
TransitiveProperty UnambiguousProperty unionOf versionInfo

1. `daml:Class` can be a sub class of itself. It contains part of the definition of an object class and refers to a class name using either `rdf:about` or `rdf:ID` attributes.

2. `daml:DatatypeProperty`

Instances of `daml:Datatype` are datatypes instead of object classes like instances of `daml:Class`. A datatype is a simple type such as those defined in the W3C XML Schema; it includes string, numbers, date, etc. The `daml:Datatype` class is disjoint from the `daml:Class` class, therefore no instance can be a datatype and an object class.

The `rdfs:range` value of a `daml:DatatypeProperty` is an instance of `daml:Datatype`. An example of a `daml:DatatypeProperty` is the `#productNumber` property, its range is a `xsd:nonNegativeInteger` datatype as defined in the XML Schema specification.

3. `daml:List` is a set of statements which recursively breaks the list down into the first element, `daml:first`, and the sublist consisting of the rest of the elements `daml:rest`. When the last element is reached, the sublist of remaining elements is a special DAML resource `daml:nil`.
4. `daml:Nothing` The class with no things in it.
5. `daml:Thing`
6. `daml:ObjectProperty`

The `rdfs:range` value of a `daml:ObjectProperty` is of `rdf:type`, `daml:Class`, or any of its subclasses. Any instance of any class, except `daml:Datatype`, can be the value of the range of an instance of a `daml:ObjectProperty`.

7. `daml:Ontology` is the DAML+OIL header. This provides certain metadata for the ontology itself. It contains zero or more `daml:versionInfo` and `daml:imports` properties.
8. `daml:Restriction` is a property restriction, which is a special kind of class-expressions. It implicitly defines an anonymous class, namely, the class of all objects that satisfy the restriction on their properties. `daml:Restriction` contains:

9. daml:onProperty property, which refers to a property name (a URI)

10. daml:toClassproperty Properties indicating the type of restriction:

(which contains a class-expression).

11. daml:hasValue Properties indicating the type of restriction:

property (which contains a reference to an individual object or a datatype value).

12. daml:hasClass Properties indicating the type of restriction:

property (which contains a class-expression or a datatype references).

13. daml:minCardinality property.

Properties containing a non-negative integer indicating a qualified cardinality restriction, and containing a daml:hasClassQ property, containing a class-expressions or a datatype references.

14. daml:hasClassQ property

The most general (object) class in DAML. This is equal to the union of any class and its complement.

15. daml:TransitiveProperty is a daml:ObjectProperty that is transitively applied. If Mr. John Smith is a member of the Boulder Nordic Club (BNC) and the BNC is a member of the American Cross Country Skiers (ACCS), then Mr. Smith is also a member of the ACCS.
16. daml:UnambiguousProperty is a property that has only one possible subject per object; it is an injective property. For example, if Mr. Smith's position in the race is first, and there was no tie in the race, then the person who came in first place in the race is Mr. Smith and nobody else.
17. daml:UniqueProperty is a property that has only one possible value, it is a short-cut for a property with a daml:maxCardinality of 1. Example: Mount Everest is the highest peak on Earth. No other mountain is the highest peak.
18. daml:cardinality determines the exact number of possible values for a given property. For instance, a daml:UniqueProperty is like a property with a daml:cardinality equal to 1.
19. daml:cardinalityQ is like daml:cardinality except that it specifies the rdf:type of the possible values of the property. Example, not only is there a single highest peak on Earth, but that peak must be natural, i.e. of rdf:type Mountain. Something of another type would be considered the highest structure and not the highest peak.
20. daml:complementOf is a property that specifies that if something is an instance of X, and Y is the daml:complement of X, then Y cannot be an instance of X. daml:complementOf contains a single class-expression and defines the class that consists of exactly all objects

that do not belong to the class-expressions. It is analogous to logical negation but restricted to objects only.

21. daml:differentIndividualFrom property means that an instance of a class is specifically not the same as some other instance.
22. daml:disjointUnionOf property means that a class contains all the disjoint instances of the specified classes.
23. daml:disjointWith property states that all the instances of the class will not be instances of the class it is disjoint with. daml:disjointWith contains a class-expression, it means that no instances of the class-expression are common to the subject.
24. daml:equivalentTo has the same semantics as the daml:sameClassAs property when it is applied to a class.
25. daml:first represents the first element of a daml:List.
26. daml:hasClass property specifies that at least one value of the property must be of the rdf:type in order to satisfy a restriction. daml:hasClass property contains a class-expression or a datatype reference and defines the class of all objects for which at least one value of the property is a member of the class-expressions or datatype.

27. daml:hasClassQ property is used for specifying class restriction with daml:cardinalityQ constraints. Which means that not only must there be at least one of the values of the property with the given class (as with daml:hasClass), but the number of such properties must also meet the given cardinality rules. So, for instance, if we wish to make a rule that a BasketballTeam has 5 StartingPlayers, we can do so using a daml:hasClassQ of StartingPlayer and a daml:cardinalityQ of 5.
28. daml:hasValue property is used to specify that a property must have at least one value equivalent to the specified value. daml:hasValue property contains a reference to an individual object or a datatype value and defines the class of all objects for whom the property has at least one value equal to the named object or datatype value.
29. daml:imports statement references another DAML+OIL ontology containing definitions that apply to the current DAML+OIL resource. Each reference consists of a URI specifying where the ontology is to be imported from. daml:imports are transitive.
30. daml:intersectionOf property means that the defined class contains all the instances that belong to all the classes in the list. In our example, the #HikingShoes class instances must be instances of both #HikingGear and #Footwear. daml:intersectionOf contains a list of class-expressions and defines the class that consists of exactly all the objects that are common to all class-expressions from the list. It is analogous to logical conjunction.
31. daml:inverseOf property means that it is the inverse of another property. For example, a product is used for a specific activity, an ice axe for mountaineering. The inverse relation is

- that one instance of mountaineering gear is an ice axe. Here #gear is the daml:inverseOf #usedFor.
32. daml:item property represents an item in a daml:List, it is either a daml:first element or a part of the daml:rest.
 33. daml:maxCardinality determine the maximum number of possible values for a given property.
 34. daml:maxCardinalityQ determine the maximum number of possible values of a certain rdf:type for a given property.
 35. daml:minCardinality determine the minimum number of possible values for a given property.
 36. daml:minCardinalityQ determine the minimum number of possible values of a certain rdf:type for a given property.
 37. daml:nil represents the empty list, used as the value of daml:rest when no more items are in the daml:List.
 38. daml:onProperty property is used on a daml:Restriction class to specify the property the restriction will act on. daml:onProperty property refers to a property name (a URI) by the attribute rdf:resource.

39. daml:oneOf property lists all the things that are instances of the domain. This lets us define classes by enumerating the members. daml:oneOf contains a list of the objects that are its instances. The class defined by the daml:oneOf property contains exactly the enumerated objects, declared using daml:Thing.

40. daml:rest represents all parts of daml:List except daml:first.

41. daml:sameClassAs property means that the value of the property is an equivalent class to its subject. daml:sameClassAs property asserts that the subject is equivalent to the class-expression (i.e. the subject and all the class-expression must have the same instances).

42. daml:sameIndividualAs property states that the value of the property is the same individual as its subject. daml:samePropertyAs defines that the value is an equivalent property to its subject. daml:samePropertyAs asserts that the subject is equivalent to the named property (i.e. they must have the same instances).

daml:toClass property means that only instances of the specified class can be values of the property. daml:toClass property contains a class-expressions and defines the class of all objects for whom the values of the property all belong to the class-expressions.

43. daml:unionOf property means that a class contains all the instances of all the classes it is the unionOf. For example, all camping and hiking products are considered family products. daml:unionOf contains a list of class-expressions and defines the class that consists of all the objects that belong to at least one of the class-expressions from the list. It is analogous to logical disjunction.

44. daml:versionInfo is an arbitrary value that describes versioning information for the document. It can be a simple string, or a complex RDF or literal XML structure.

13. APPENDIX 8: VIRTUAL OBJECTS

Excerpt from *Conference on Object-Oriented Programming, Systems, Languages, and Applications 1993 (OOPSLA 93)*

<http://wuarchive.wustl.edu/languages/smalltalk/patterns/patterns.93>

(last accessed January 14, 2003)

+++++

* Basic information

- Name

Context

- Short description

When the same object is used in multiple areas of discourse (contexts) at the same time, its context-specific properties should be factored out into separate objects.

- Keywords

Contexts, virtual objects, agents, views, environments, domain-specific knowledge, information sharing

- Submitted by

Stefanie Brueninghaus and Michael Weiss
Lehrstuhl fuer Praktische Informatik I, Universitaet Mannheim, Germany

- Version date & time

August 6, 1993

* The problem

- Description

In some application domains, objects are used in multiple areas of discourse concurrently. In each of these domains of discourse, objects are represented by domain-specific properties and methods. There must be a way to represent these in domain ontologies and to model domain-specific knowledge about the objects in the most appropriate place (we will elaborate below which one).

- Example

Consider how to model the depreciation of a car in a tax application. Modeling fiscal depreciation within the car class will incur serious problems whenever legislation concerning depreciation is modified: a change in legislation does not (1) change the identity of the car. Further, the properties required for the depreciation method cannot be understood outside the taxation domain. For example, the residual value of a car (i.e. its value after depreciation) in the balance sheet does not reflect the true value of the car at all: it can only be given an interpretation when extensive knowledge about the application domain is applied.

- Comments

It is a principle of object-oriented design to collocate data and methods as much as possible, thereby ensuring maximal encapsulation and enforcing the location of methods with the respective objects. When some of the data are context-specific, however, it is more sensible to model these data and the context-specific methods applied to them separately from the object proper. Yet, subclassing would not solve this problem, because contexts correspond to different perspectives on the same (!) object. Instead we argue that the context-specific counterpart of a real-world state of affairs should be modeled separately (i.e. *by a virtual object*) from the methods applied to it under the outlined conditions (i.e. *by an agent*). Therefore, knowledge that is not originally part of an object should be modeled separately.

* The pattern

- Description

For each objects used in multiple contexts, the pattern creates

- a real-world object that represents the context-independent state of affairs,
- a virtual object that contains the context-specific representation of the corresponding real-world object,
- and an agent, which is an object that actively applies the context knowledge. An agent can also be thought of as a personifying the knowledge from a certain context.

The borderline between what is context-independent and what is context-dependent knowledge cannot be drawn exactly and depends on the application domain.

Further elements of the pattern are

- views that implement access restrictions to real-world and virtual objects, and
- environments, which embrace a real-world object, a virtual object and the corresponding agent for a particular context. They can be said to perform this context.

The interaction of these object categories can be expressed by the following illustration:

INTERACTION VIA
LINK TO A CERTAIN CONTEXT

V

```

.....
: Real-World :
: Object    :
:.....:

```

- Example

In the car depreciation example above, the car is itself represented by a real-world object. Its fiscal properties, however, like its residual value or its (legally predefined) service life are captured within a virtual object. The knowledge from the statutes is modeled by an agent and can be structured independently from the balance sheet.

```

class car {                                // real-world object
float  theInitialCost;
string theNumberPlate;
carView viewRequest( taxationAgent )

    { return carView( theInitialCost ); }
... }

class carView {                            // view definition
float  theInitialCost;
void   carView( float someInitialCost )
    { theInitialCost = someInitialCost; }
... }

class carInBalanceSheet {                 // virtual object
float  theResidualValue;
integer thePredefinedServiceLife;
... }

class taxationAgent{                      // agent
float  Depreciation( carView );
... }

```


Comments

The pattern proposed can be applied with advantage to domains in which

- one and the same object is involved in multiple specialized contexts (the contexts are said to share this object),
- the domain-specific knowledge is subject to frequent changes and changes independently from the objects it applies to,
- this knowledge is typically ill-structured and needs to be structured into conceptual categories in order to arrive at objects (this further motivates the separation into virtual objects and agents), and
- a special terminology is used within each context that collides (is either inconsistent with or irrelevant to) with the shared language.

An important effect of this approach is the possibility of structuring the domain knowledge independently from the objects to which it is applied. The guiding principle is that of a task-structure that associates each of the agents with a task or subtask.

The pattern was developed for the legal domain (in particular, taxation), which is reflected by our example. However, the idea of separating the knowledge such that as much knowledge as possible is factored out into context-dependent objects, is not restricted to this domain. For example, we have been looking at the domain of computer supported cooperative work, in which the goal is to share enough information between people from different perspectives to allow them to collaborate effectively.

+++++

14. BIBLIOGRAPHY

- [1] T. Berners-Lee, J. Hendler, O. Lasilla, "The Semantic Web", *Scientific American*, May 2001.
<http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21>

- [2] J. D. Heflin, "Towards the Semantic Web: Knowledge Representation in a Dynamic, Distributed Environment", Ph D. Thesis, University of Maryland, 2001, pp1.
<http://www.cse.lehigh.edu/~heflin/pubs/heflin-thesis-orig.pdf>.

- [3] The Unicode Standard. Last updated: - December 27, 2002.
<http://www.unicode.org/unicode/standard/standard.html>

- [4] T. Bray, J. Paoli, C. M. Sperberg-McQueen, Eve Maler, "Extensible Markup Language (XML) 1.0, (Second Edition)", W3C Recommendation 6 October 2000.
<http://www.w3.org/TR/REC-xml>.

- [5] H. D. Thompson, Beech, M. Maloney, N. Mendelsohn, "XML Schema Part 1: Structures", W3C Recommendation 2 May 2001.
<http://www.w3.org/TR/2001/REC-xmlschema-1-20010502>.

- [6] P. V. Biron, A. Malhotra, "XML Schema Part 2: Datatypes", W3C Recommendation 02 May 2001.
<http://www.w3.org/TR/xmlschema-2/>

- [7] O. Lassila and R. Swick, "Resource Description Framework (RDF) Model and Syntax Specification", World Wide Web Consortium (W3C), February 1999.
<http://www.w3.org/TR/1999/REC-rdf-syntax-19990222>
<http://www.w3.org/TR/2002/WD-rdf-primer-20020319/>
- [8] D. Brickley and R.V. Guha, "Resource RDF Vocabulary Description Language 1.0: RDF Schema", W3C Working Draft 12 November 2002.
<http://www.w3.org/TR/rdf-schema/>
- [9] Uniform Resource Identifiers (URI), RFC 2396.
<http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc2396.html>
- [10] XML Linking Language (XLink) Version 1.0, W3C Recommendation 27 June 2001.
<http://www.w3.org/TR/xlink/>
- [11] XML Pointer Language (XPointer) Version 1.0, W3C Candidate Recommendation 11 September 2001.
<http://www.w3.org/TR/xptr/>
- [12] XML Path Language (XPath) Version 1.0, W3C Recommendation 16 November 1999.
<http://www.w3.org/TR/xpath>
- [13] B. Chillovskii, "Using Regular Tree Automata as XML schemas", *IEEE Conference Proceedings*, Research Centre Europe Grenoble Laboratory 6 Chemin de Maupertuis F 38240 Meylan.
<http://dlib2.computer.org/conferen/adl/0659/pdf/06590089.pdf?>

- [14] S. A. McIlraith, T. C. Son and H. Zeng, "Semantic Web Services", *IEEE Intelligent Systems*, March/April 2001.
- [15] N. Guarino and Giaretta, "Ontologies and Knowledge Bases: Towards a Terminological Clarification", *Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing*, IOS Press, Amsterdam: 25-32, 1995.
- [16] XSL Transformations (XSLT), Version 1.0, W3C Recommendation 16 November 1999.
<http://www.w3c.org/TR/xslt>.
- [17] Genesereth and N. Nilsson, "Logical Foundations of Artificial Intelligence", Morgan Kaufmann, San Mateo, CA, 1987.
- [18] M. Klein, D. Fensel, F. Harmelen, I Horrocks, "The Relation between Ontologies and Schema-languages: Translating OIL-specification in XML Schema", Department of Computer Science, Vrije Universiteit Amsterdam, the Netherlands, Department of Computer Science, University of Manchester, Oxford, UK, 2001.
<http://www.cs.vu.nl/~mcaklein/papers/oil-xmils.pdf>
- [19] S. Decker, F. Harmelen, J. Broekstra, M. Erdmann, D. Fensel, I. Horrocks, M. Klein, S. Melnik, "The Semantic Web - on the respective Roles of XML and RDF".
<http://www.ontoknowledge.org/oil/download/IEEE00.pdf>
- [20] J. Hendler, "Agents and the Semantic Web", *IEEE Intelligent Systems*, March/April 2001.

- [21] D. McGuinness, J. Hendler, "The DARPA Agent Markup Language", *IEEE Intelligent Systems*, vol 15, no 6, Nov/Dec. pp. 72-73, 2000.
- [22] N. F. Noy, D. McGuinness, "Ontology Development 101: A Guide to Creating Your First Ontology", Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report SMI-2001-0880, March 2001.
<http://www.ksl.stanford.edu/people/dlm/papers/ontology-tutorial-noy-mcguinness.htm>
- [23] DAML+OIL (March 2001) Reference Description, W3C Note 18 December 2001.
<http://www.w3.org/TR/daml+oil-reference>
- [24] T. Bray, D. Hollander, and A. Layman, "Namespaces in XML", W3C, January 1999.
<http://www.w3.org/TR/1999/REC-xml-names-19990114/>
- [25] S. Russell and P. Norvig, "Artificial Intelligence: A Modern Approach", Prentice Hall, Upper Saddle River, N.J., 1995.
- [26] B. Chandrasekaran, J. R. Josephson, V. Richard Benjamins, "What are Ontologies and why do we need them", *IEEE Intelligent Systems*, January/February pp 20, 1999.
- [27] William Swartout, Austin Tate, "Ontologies", *IEEE Intelligent Systems*, January/February 1999.

- [28] D. L. McGuinness, "Ontologies Come of Age", Knowledge Systems Laboratory, Stanford University, Stanford, CA MIT Press, 2002.
[http://www.ksl.stanford.edu/people/dlm/papers/ontologies-come-of-age-mit-press-\(with-citation\).htm](http://www.ksl.stanford.edu/people/dlm/papers/ontologies-come-of-age-mit-press-(with-citation).htm)
- [29] T. Gruber, "A translation approach to portable ontologies", *Knowledge Acquisition*, 5(2):199-220, 1993.
http://ksl-web.stanford.edu/KSL_Abstracts/KSL-92-71.html
- [30] Karp et al, "XOL: An XML-Based Ontology Exchange Language", 1999.
<http://www.ai.sri.com/pkarp/xol/xol.html>
- [31] Various Definitions of Ontology, DAI Web Pages, University of Edinburgh, Infomatics, U.K.
<http://www.dai.ed.ac.uk/homes/yannisk/node1.html>
- [32] Y. Shoham, "What we talk about, when we talk about software agents", *IEEE Intelligent Systems*, March/April 1999.
- [33] M. Uschold and M. Gruninger, "Ontologies: Principles, Methods and Applications", *Knowledge Engineering Review*, 11(2), pp 93—155, 1996.
- [34] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W. "Object-oriented modeling and design", Englewood Cliffs, New Jersey: Prentice Hall. 1991.

- [35] R. Fikes, A. Farquhar, and J. Rice, "Tools for Assembling Modular Ontologies in Ontolingua", Knowledge Systems Laboratory, April 1997.
http://www-ksl.stanford.edu/KSL_Abstracts/KSL-97-03.html
- [36] M.A. Musen, "Dimensions of knowledge sharing and reuse", *Computers and Biomedical Research* 25: 435-467, 1992.
- [37] T.R. Rothenfluh, J.H. Gennari, H. Eriksson, A.R. Puert, S.W. Tu, and M.A. Musen, "Reusable ontologies, knowledge-acquisition tools, and performance systems: PROTÉGÉ-II solutions to Sisyphus-2", *International Journal of Human-Computer Studies* 44: 303-332, 1996.
- [38] D.L. McGuinness and J. Wright, "Conceptual Modeling for Configuration: A Description Logic-based Approach" *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* - special issue on Configuration, 1998.
- [39] J. Bradshaw, S. Dutfield, et al, "Towards an Industrial-Strength Open Distributed Agent Architecture", *Software Agents*, AAAI/MIT Press, 375-418, 1997.
- [40] D. Deugo and M. Weiss, "A Case for Mobile Agent Patterns", Workshop on Mobile Agents in the Context of Competition and Cooperation (MAC3), *Autonomous Agents*, 19-23, 1999.
- [41] S. Cranefield, "UML and the Semantic Web", The Information Science Discussion Paper Series Number 2001/04 February 2001 ISSN 1172-6024, 2001.
- [42] Unified Modeling Language (UML), Object Management Group (OMG).
<http://www.omg.org/uml/>

- [43] XML Encoding Rules for ASN.1 December 16, 2002.
<http://xml.coverpages.org/xer.html>
- [44] International Organization for Standardization (ISO) Recommendation ISO 8824.
- [45] P. Clark and B. Porter, "Building Concept Representation from Reusable Components", Proc. 14th Nat'l Conf Artificial Intelligence (AAAI-97), MIT Press, Cambridge Mass., pp.369-376, 1997.
- [46] M. Hauben, "History of ARPANET" Homepage.
<http://www.dei.isep.ipp.pt/docs/arpa.html>
- [47] The DARPA Agent Markup Language Homepage.
<http://www.daml.org/>
- [48] DAML+OIL Index of all language elements.
<http://www.daml.org/2001/03/reference.html#Index>
- [49] International Telecommunication Union Homepage.
<http://www.itu.int/publications/index.html>
- [50] IEEE Computer Society Digital Library Homepage.
<http://computer.org/publications/dlib/>
- [51] IEEE Communication Society Digital Library Homepage.
<http://dl.comsoc.org/dl/>
- [52] The World Wide Web Consortium (W3C) Homepage.
<http://www.w3.org/>

- [53] The Defence Advanced Research Projects Agency (DARPA) Homepage.
<http://www.darpa.mil/>

- [54] DAML Tools Homepage, December 2002.
<http://www.daml.org/tools/>

- [55] DAML Homepage, June 2002.
<http://www.daml.org/about.html>

- [56] DAML Web Service Homepage, October 2002.
<http://www.daml.org/services/>

- [57] Frame Relay Glossary, January 2003.
<http://www.frforum.com/basicguide/glossary.html>

- [58] Frame Relay Implementation Agreements Homepage, January, 2003.
<http://www.frforum.com/5000/5000index.html>

- [59] ATM Forum Homepage, January 2003.
<http://www.atmforum.com/>

- [60] I. Horrocks, D. Fensel, J. Broekstra, S. Decker, M. Erdmann, C. Goble¹, F. Harmelen, M. Klein, S. Staab, R. Studer, and E. Motta
“The Ontology Inference Layer OIL”, August 2000.
<http://www.ontoknowledge.org/oil/papers.shtml>

- [61] Ora Lassila & Deborah McGuinness: “The Role of Frame-Based Representation on the Semantic Web”, Knowledge Systems Laboratory, Stanford University.
<http://www.ksl.stanford.edu/people/dlm/etai/lassila-mcguinness-fbr-sw.html>

- [62] M. Minsky, "A Framework for Representing Knowledge", in Patrick Henry Winston (ed.), *The Psychology of Computer Vision*, McGraw-Hill, New York, 1975.
- [63] R. Fikes & T. Kehler, "The Role of Frame-Based Representation in Reasoning", *CACM* 28(9): 904-920, 1985.
- [64] P. D. Karp, "The design space of frame knowledge representation systems", Technical Report 520, SRI International AI Center, October, 1992.
<http://www.ai.sri.com/pubs/search.php>
- [65] V. Chaudhri, A. Farquhar, R. Fikes, P. Karp, and J. Rice; "OKBC: A Programmatic Foundation for Knowledge Base Interoperability", American Association for Artificial Intelligence (AAAI) 1998.
- [66] DigiKeyTM Component Catalog Homepage.
<http://www.digikey.com>
- [67] J. Heflin, J. Hendler, and S. Luke. "Coping with changing ontologies in a distributed environment", American Association for Artificial Intelligence AAAI-99 Workshop on Ontology Management. MIT Press, 1999.
- [68] J. Heflin, J. Hendler, and S. Luke. "SHOE: A knowledge representation language for Internet applications", Technical Report Institute for Advanced Studies, University of Maryland, 2000.
www.cs.umd.edu/projects/plus/SHOE

- [69] D. L. McGuinness, R. Fikes, J. Rice, and S. Wilde. "An environment for merging and testing large ontologies", in Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR2000), 2000.
- [70] Brown, L. Shona, M. Kathleen and Eisenhardt, "Product Development: Past research, present findings and future directions", Academy of Management Review, 20(2), 343-378, 1995.
- [71] Rosettanet Homepage, January, 2003.
<http://www.rosettanet.org/RosettaNet/Rooms/DisplayPages/LayoutInitial>
- [72] Silicon Integration Initiative Homepage, January, 2003.
<http://www.si2.org/>
- [73] DAML Time Ontology Homepage, Oct 2002.
<http://www.cs.rochester.edu/~ferguson/daml/>
- [74] XML Metadata Interchange (XMI), Version 1.2, January 2002.
<http://cgi.omg.org/docs/formal/02-01-01.pdf>
- [75] D. Carlson, "Modeling XML Applications with UML: Practical e-Business Applications", Addison-Wesley, 2001.
- [76] F. Baader, H.-J. Heinsohn, B. Hollunder, J. Muller, B. Nebel, W. Nutt, and H.-J. Profitlich: "Terminological knowledge representation: A proposal for a terminological logic", Technical Memo TM- 90-04, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH (DFKI), 1991.

- [77] R. J. Brachman and J. G. Schmolze: "An overview of the KL-ONE knowledge representation system", *Cognitive Science*, 9(2):171–216, 1985.
- [78] S. Cranefield and M. Purvis, "UML as an ontology modelling language"
In Proceedings of the Workshop on Intelligent Information Integration,
16th. *International Joint Conference on Artificial Intelligence (IJCAI-*
99), 1999.
http://nzdis.otago.ac.nz/download/papers/UML_Ontology_99.pdf
- [79] Object Management Group (OMG) Homepage. January 13, 2003.
<http://www.omg.org/>
- [80] Naiburg and Maksimchuk, "*UML for Database Design*". Addison-Wesley 2001.
- [81] David R. Wallace, Shaun Abrahamson, Nicola Senin, Peter Sferro,
"Integrated Design in a Service Marketplace", *Computer-aided Design*, volume
32, number 2, pp. 97-107, 2000.
http://web.mit.edu/cipd/documents/workingpaper_files/DOME_IntegrDesign.pdf
- [82] G. Schreiber, "Knowledge Engineering and Management", MIT Press, 1999.
- [83]. S. Hawke, "Using Reification To Extend RDF", W3C Web site.
January 2003.
<http://www.w3.org/2002/Talks/0910-rdf-reification/>
<http://www.w3.org/TR/rdf-mt/#ReifAndCont>
- [84] ISO 8879, Text and office systems -- Standard Generalized Markup Language
(SGML), International Standards Organization.
<http://www.iso.org/iso/en/CombinedQueryResult.CombinedQueryResult?queryString=8879>

- [85] John F. Sowa, "Knowledge Representation: Logical, Philosophical, and Computational Foundations", Brooks Cole Publishing Co., Pacific Grove, CA,. 594 + xiv pages; ISBN 0-534-94965-7, 1999.
<http://users.bestweb.net/~sowa/krbook/index.htm>

- [86] H. Nwana and M. Wooldridge "Software Agent Technologies", BT Technology Journal, 14(4). (British Telecom) 1996.
<http://www.btexact.com/docimages/71153/71153.pdf>

- [87] N. Guarino, "Semantic Matching: Formal Ontological Distinctions for Information Organization, Extraction, and Integration", presented at Information Extraction: A Multidisciplinary Approach to an Emerging Information Technology, International Summer School, SCIE-97, Frascati, Italy, 1997.

- [88] F. Fonseca, M. Egenhofer, C. Davis and G. Câmara, "Semantic Granularity in Ontology-Driven Geographic Information Systems", *AMAI Annals of Mathematics and Artificial Intelligence - Special Issue on Spatial and Temporal Granularity* 2001.
http://www.spatial.maine.edu/~fred/fonseca_amai.pdf

- [89] D.B. Lenat "CYC: A Large-Scale Investment in Knowledge Infrastructure", *Communications of the ACM*, 38 , No 11, pp 33—38 1995.

- [90] The Electronic Design Interchange Format (EDIF) Homepage.
<http://www.edif.org/about.html>

- [91] Electronics Design Automation Industry (EDA) Homepage.
<http://www.edac.org/>

[92] Uniform Code Council (UCC) Homepage.

<http://www.uc-council.org/>