

2020-01-01

A Survey Of The Current State Of The Practice In Bridging The Gap Between Engineering Ontologies And Modeling Profiles For Engineering Applications

John Artus
University of Texas at El Paso

Follow this and additional works at: https://scholarworks.utep.edu/open_etd



Part of the [Industrial Engineering Commons](#)

Recommended Citation

Artus, John, "A Survey Of The Current State Of The Practice In Bridging The Gap Between Engineering Ontologies And Modeling Profiles For Engineering Applications" (2020). *Open Access Theses & Dissertations*. 2923.

https://scholarworks.utep.edu/open_etd/2923

This is brought to you for free and open access by ScholarWorks@UTEP. It has been accepted for inclusion in Open Access Theses & Dissertations by an authorized administrator of ScholarWorks@UTEP. For more information, please contact lweber@utep.edu.

A SURVEY OF THE CURRENT STATE OF THE PRACTICE IN BRIDGING THE GAP
BETWEEN ENGINEERING ONTOLOGIES AND MODELING PROFILES FOR
ENGINEERING APPLICATIONS

JOHN GERARD ARTUS

Master's Program in Systems Engineering

APPROVED:

Jose F. Espiritu, Ph.D., Chair

Heidi A. Taboada, Ph.D.

Tzu-Liang (Bill) Tseng, Ph.D.

Virgilio Gonzalez, Ph.D.

Stephen Crites, Ph.D.
Dean of the Graduate School

A SURVEY OF THE CURRENT STATE OF THE PRACTICE IN BRIDGING THE GAP
BETWEEN ENGINEERING ONTOLOGIES AND MODELING PROFILES FOR
ENGINEERING APPLICATIONS

by

JOHN GERARD ARTUS, BSEE

THESIS

Presented to the Faculty of the Graduate School of
The University of Texas at El Paso
in Partial Fulfillment
of the Requirements
for the Degree of

MASTER OF SCIENCE

Department of Industrial Manufacturing & Systems Engineering

THE UNIVERSITY OF TEXAS AT EL PASO

May 2020

Abstract

Model-Based Systems Engineering holds the promise of enhancing systems engineering tasks and product quality through better, more efficient exchange of data across tools, personnel, and departments, a higher quality of analysis resulting from greater access to cross-discipline data, better coordination between engineering activities and those of program management, and many other benefits. Yet, there is still no standardized method for achieving commonality of architecture vocabulary across projects, across companies, across industries such that entities operating in the same domain produce products that speak the same technical language. Some specialized domains, such as the Department of Defense (DoD), have developed Architecture Frameworks (AF), such as the DoDAF, to establish a level of standardization through enforcement of a profile on the architecture development activities. Still, for other industries, there is no such standardization. Ontologies offer a possible solution to this problem by allowing domain interests to collaborate to construct a domain ontology that can then be transformed into a modeling profile to constrain architectural development activities to use a more common vocabulary. This thesis examines the current state of the practice in industry towards developing a standard methodology for transforming such standardized domain ontologies into modeling profiles.

Table of Contents

	Page
Abstract	iii
Table of Contents	iv
List of Tables	vii
List of Figures	viii
1 Introduction	1
1.1 What is an Ontology?	3
1.2 Ontologies As the Source of this Knowledge in Engineering Architectures	4
1.3 Benefits of Using Ontologies for Architecture Development	4
1.4 What is Preventing the Use of Ontologies in Architectural Development?	5
1.5 Thesis Roadmap	6
2 Historical Background of Ontology Development	7
2.1 Vocabulary	9
2.2 Taxonomy	9
2.3 Ontology	10
2.4 State of Readiness in Ontology Development	12
3 Historical Background on Use of Ontologies in Engineering	18
3.1 Use of Ontologies in Engineering in General	18

3.2	Use of Ontologies in Systems Engineering and Manufacturing	19
3.3	Use of Ontologies Specifically in Software Engineering.....	21
3.4	Use of Ontologies Specifically in Systems Engineering.....	27
4	Constructing Ontologies	38
4.1	Design Criteria	38
4.2	Ontological Formalisms	40
4.3	Methods for Modeling Ontologies	40
4.4	Types of Ontologies	42
4.5	Languages for Building Ontologies	44
4.6	Ontology Development Tools	45
4.7	Ontology Development Methodologies	45
4.8	How Ontology Development Differs from Object-Oriented Design	47
4.9	Important Ontological Terms	47
4.10	Understanding Classes and Class Hierarchies	49
4.11	A Simple Knowledge-Engineering Methodology	61
4.12	Ontology Maintenance	70
5	Bridging the Gap Between Ontologies and Modeling Profiles	71
5.1	Modeling	71
5.2	Meta-Object Facility	73
5.3	UML Profile Extension Mechanism	75

5.4	Model Driven Architecture	77
5.5	Ontology Definition Metamodel	80
6	Current State of the Practice	84
6.1	NASA JPL Integrated Model-Centric Engineering Initiative	84
6.2	NASA JPL View of Systems Engineering Landscape in the 2010 Timeframe ..	85
6.3	NASA JPL Use of Models as Information Structures	89
6.4	NASA JPL Use of Semantic Technologies	91
6.5	Embedding Ontologies in SysML Profiles	96
6.6	Embedding Ontologies in SysML Profiles	101
6.7	NASA JPL Conclusions and Future Work.....	105
7	Summary.....	107
8	Conclusion.....	110
	References	113
	Curriculum Vita	123

List of Tables

Table 1: Core Skills Required of a Professional Ontologist	15
Table 2: Core Knowledge Required of a Professional Ontologist.....	16
Table 3: Elective Skills of a Professional Ontologist	16
Table 4: Elective Knowledge of a Professional Ontologist	17
Table 5: Ontology Standard Used at JPL.....	92
Table 6: Ontology Technologies Used at JPL	92
Table 7: Relative Utility of Semantic Web Technologies and UML/SysML.....	93
Table 8: OWL Ontologies for Systems Engineering	95

List of Figures

Figure 1: Thesis Roadmap	6
Figure 2: CAEX Plant Models Validation Process Using OWL	33
Figure 3: Relation Among Modeler-Model-Interpreter	71
Figure 4: Representation of Figure 3 in Modeling Language	72
Figure 5: Syntax and Semantics of Metamodel	73
Figure 6: Meta-Object Facility Metamodel Hierarchy	74
Figure 7: SE Lifecycle Phases Mapped to MDA	80
Figure 8: Semantic Web Architecture	81
Figure 9: Ontology Modeling in the Context of MDA and the Semantic Web	83
Figure 10: Example Type Classification Hierarchy	90
Figure 11: Relationships are Also Properties	91
Figure 12: Transformation of UML/SysML Models to Ontologies	97
Figure 13: Reification in UML/SysML	99
Figure 14: OWL2 Property Chain Example	100
Figure 15: Example UML/SysML Reification	103
Figure 16: Example OWL2 Reification	103

1 Introduction

Ontologies are built for the purpose of capturing knowledge about objects and their relations to each other so that this knowledge can be reused in multiple activities. The primary emphasis on application of ontologies today is in the development of the Semantic Web. The Semantic Web “has the potential for semantically richer representations of things ... and should provide us with more intelligent services.” (Gasevic, Djuric, & Devedzic, 2006)

The study and use of ontologies in modern engineering practices has only recently come to the attention of systems engineers. An ontology defines the basic terms and relations comprising the vocabulary of a topic area as well as the rules for combining terms and relations to define extensions to the vocabulary. (Neches, et al., 1991) The most effective use of ontologies in systems engineering thus far has been in the area of requirements engineering. (Bernardi, Rabello, & Cervi, 2016) Most of the progress has been made in the areas of 1) mining domain knowledge from Natural Language requirements text documents to construct ontologies that support other requirements engineering activities, 2) requirements analysis using a domain ontology to reduce requirements ambiguity and promote completeness, and 3) requirements specification development using the domain ontology to layout the structure of the document. (Siegemund, 2014) The activity identified in 1) above holds promise for contributing to a domain-specific ontology that could also serve the purpose of architecture development.

To deliver a quality architectural description of a solution, the architect faces a task that requires substantial knowledge about the domain of the problem space that the solution is intended to address. This thesis examines the state of the practice of using the domain knowledge captured in an ontology that is then transferred to a modeling profile. The modeling profile can then be used to represent the captured knowledge about the given domain in the development of

system architectures, which form the basis for the conceptual description of engineered systems. The knowledge transferred from an ontology to a modeling profile includes primarily a description of the individual domain objects and their relationships to each other. To derive maximum usability from an ontology selected to support architecture modeling activities, the ontology should meet certain basic criteria:

- Expressive Power – Does the ontology communicate the domain knowledge effectively to the modeling profile?
- Understandability – Can the architect understand the contents and meaning of the ontology as represented in the profile?
- Accessibility – Can the needed knowledge be easily extracted from the ontology for use in the modeling profile? (Fikes & Tom, 1985)

Satisfaction of some of these criteria are influenced by the translation mechanism going from the selected ontology to the modeling profile. But having a quality ontology selected at the start solves much of the problem.

However, the problem is not restricted to selecting an ontology and applying it to the task of architectural development. Two additional problems exist in the landscape of ontology applications to architecture. One problem is the lack of sufficient domain ontologies of concern to architects of complex engineered systems. In order for more engineering-related ontologies to become available, engineering organizations involved in design activities should begin contributing to the development of an open repository of general-purpose ontologies in various engineering fields of study. However, this requires that engineers who understand the complex nature of the subsystems involved in these specialized engineering domains become involved in the actual construction of ontologies. This leads to the second problem to be dealt with.

According to Boyce & Pahl, the tools currently available require a degree of expertise that does not favor the generation of ontologies by people who are experts in a particular subject area but not practiced in ontological engineering. Currently, a joint effort by domain experts and ontology engineers is necessary for ontology development. To see the widespread development of domain ontologies would require availability of ontological tools for creating ontologies from scratch, or to enrich pre-existing ontologies with minimal human intervention. (Boyce & Pahl, 2007)

1.1 What is an Ontology?

There are many interpretations of the meaning of ontology depending on the perspective of the user (philosophical, conceptual, logical, etc), the degree of formality required, and whether the need is for a domain-specific application or something more generalized. For the purpose of this thesis, which focuses on the practical engineering use of an ontology for developing architectures, the following definition (Neches, et al., 1991) suits the need well: “An ontology defines the basic terms and relations comprising the vocabulary of a topic area as well as the rules for combining terms and relations to define extensions to the vocabulary.” It is the identification of key architectural elements and the defining of relationships among those elements that interest the architect the most. The primary relationships of interest include classification (type casting), generalization-specialization (is-a-kind-of), and whole-part (is-a-part-of). (Graves, Integrating Reasoning with SysML, 2012) Together, the defined elements and their relationships allow the architect to establish the principle features of the architecture, including the components, subcomponents, assemblies, roles, functions, interfaces, ports, connectors, data exchange items, etc. This task is enhanced by the availability of a modeling profile that represents community knowledge of a particular domain, thereby promoting efficient

use of pre-existing knowledge, and consistency of the product architecture with other architectures within the domain community.

1.2 Ontologies as the Source of this Knowledge in Engineering Architectures

While the interest in using ontologies to support the development of modeling profiles in system architecture practice is growing, the actual state of current practice is that most architecture development efforts do not take advantage of ontologies, nor of the domain profiles built from recognized domain ontologies. As a result, the identification of architectural elements and their relationships end up being parochial activities sponsored by individual organizations such as companies or departments, with little-to-no knowledge sharing or commonality with other organizations. Users of the architectures discover that terminology normally accepted as common, in the end has multiple interpretations. The architectures thus produced have limited transference outside of the organization without the user of the architectural description having to inquire about the definition of the fundamental terms used to describe the various elements of the architecture. This situation produces misinterpretations of the architecture and is an especially important consideration with system architectures because they represent a conceptualization of a product solution, and thus, to some degree, represent a product of the mind on the part of architects constructing the architecture. These conceptualizations can be difficult to precisely define if not well documented, and thus easily misinterpreted by users of the architecture not familiar with the frame of mind of the architects constructing the architecture. The only solution is to include definitions of terms as a part of the architectural description.

1.3 Benefits of Using Ontologies for Architecture Development

The use of modeling profiles generated from ontologies that have been built and accepted by a domain community is that the terms become standardized within the community, the

definitions of all the elements of the architecture no longer need to be embedded within the architecture, and chance of misinterpretation of the architecture is vastly reduced.

With almost every architectural development activity undertaken, a new architectural database must be built from scratch. The cost of duplicating the effort of previous architecture development projects without reuse remains one of the major costs in system development activities. The cost of this duplication of effort will become prohibitive as larger architectural projects are undertaken. To overcome this waste of effort, ways of preserving existing knowledge bases and of sharing, reusing, and building on them must be developed. Ontologies provide the basis for building, storing, and sharing reusable knowledge for a variety of uses, not just for architectural development. Thus, ontologies will provide sources of information that serve the same functions as traditional text-based databases, such as books, and reports. Yet, they are more flexible, easier to update, and easier to query. Ontologies will make it possible for end users to tailor large systems to their needs by assembling knowledge bases and services rather than developing architectures from scratch. (Neches, et al., 1991)

1.4 What is Preventing the Use of Ontologies in Architectural Development?

The availability of ontologies for engineering use in particular domains is extremely low due to 1) the nascency of this specific application of ontologies, 2) the fluid nature of standards development in this area, and 3) the lack of tools to facility the transformation of ontologies into modeling profiles. This thesis examines the current state of practice in the construction of modeling profiles from existing ontologies and recommends steps to be taken to improve the process such that it becomes more readily available for architects to employ.

1.5 Thesis Roadmap

The roadmap for this thesis, illustrated in Figure 1, is intended to summarize the development of the theme of this thesis and walk the reader down the path of understanding the value and potential of the use of ontologies to understanding how ontologies can be of value to architects when developing system architecture, to finally understanding the current state of the practice in transforming ontologies into useful profiles for modeling systems architectures.

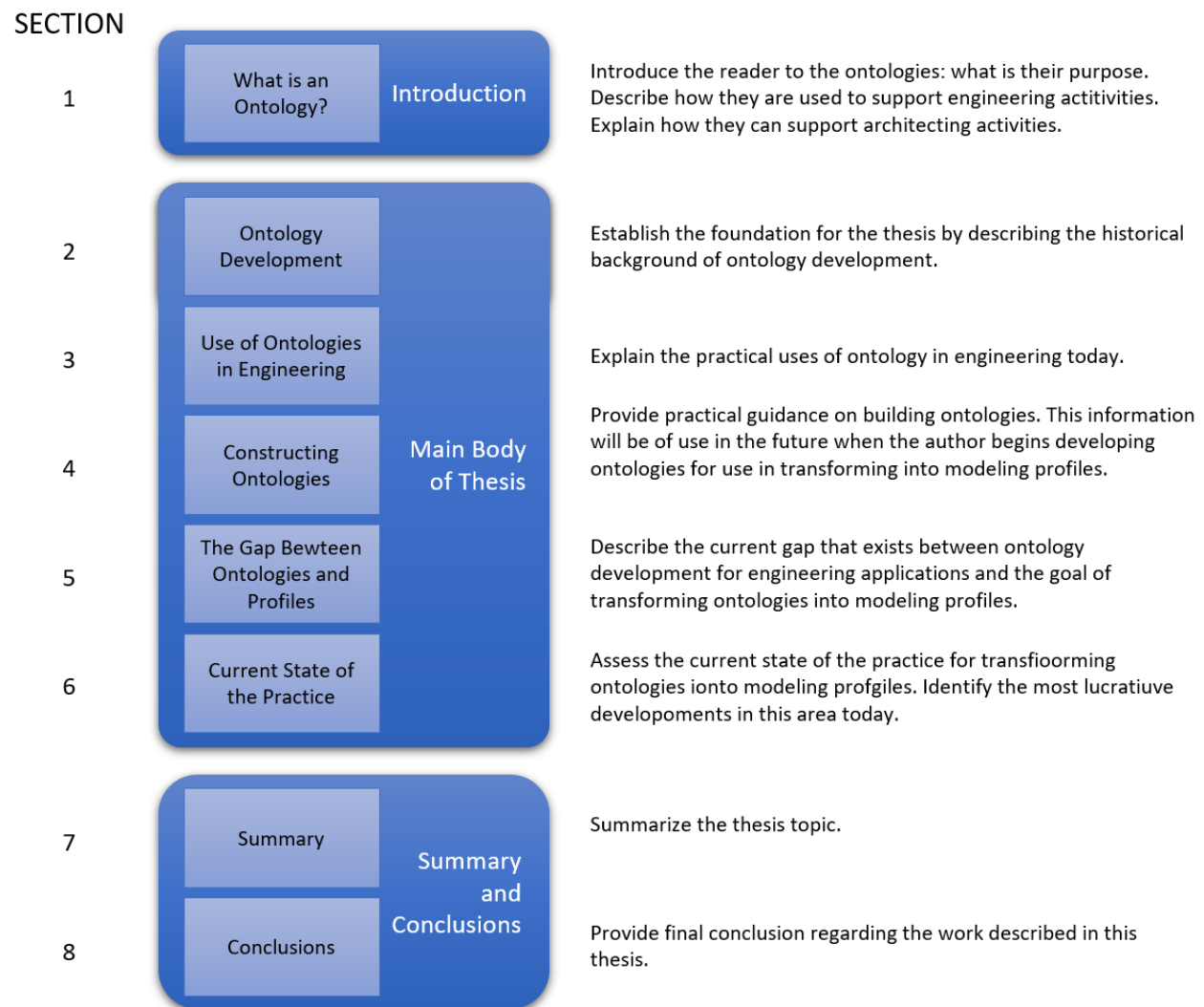


Figure 1: Thesis Roadmap

2 Historical Background of Ontology Development

Ontology development spawns from the human desire to provide structure and meaning to our universe. Ontology development has its origins with the Ancient Greeks with their study of Philosophy. Parmenides is generally recognized as the first to discuss the ontological categorization of existence. The etymology of the term ontology points to Greek origins that refer to ontology as “the study or theory of being or that which is.” (Roe, 2012) However, it is not the ancient purpose and use of ontology that interests us in engineering today.

Since the times of the Ancient Greeks, the modern use of ontology in the sciences has its origins with the research activities into Artificial Intelligence (AI) in the 1970s and 1980s. During the 1990s, interest in ontology moved from the AI laboratories to the desks of domain experts who saw the potential of the organized classification of information to help solve practical real-world problems. (Noy & McGuinness, 2001) The modern application of ontology is better defined as “a technical term denoting an artifact that is designed for a purpose, which is to enable the modeling of knowledge about some domain, real or imagined.” (Gruber, Ontology, 2009) The overall concept of ontology and its application in a diverse set of fields of study has sparked a debate over its precise meaning for different applications.

Developing an ontology is not an end in itself. Noy and McGuinness identified five reasons why people would have interest in developing ontologies. These are 1) to share a common understanding of the structure of information among people or software agents, 2) to enable reuse of domain knowledge, 3) to make domain assumptions explicit, 4) to separate domain knowledge from the operational knowledge, and 5) to analyze domain knowledge. (Noy & McGuinness, 2001) The interest of this thesis lies in the second reason – to enable and foster greater sharing of common knowledge among domain practitioners in the area of system

architecture, so that the definition of system elements, interrelationships, behavior, etc., are reused from other domain contributors and not developed afresh for each undertaking.

In (Pahl & Holohan, 2004), Pahl and Holohan identified four purposes for developing knowledge spaces. These are:

- Vocabulary – To collect terms along with their definitions with no specific defined relationships among the elements of the vocabulary.
- Taxonomy – To establish terminology definition and classification as the central issues. It supports browsing and retrieval of educational resources.
- Thesaurus – To identify relationships between terms are the central issues. It constrains the use of a vocabulary.
- Conceptual Model – As a formal model of some domain that supports modeling of the subject area and technical aspects, and often uses more than simple classification-oriented relationship types.
- Logical Theory – To reason and infer on a given problem. It combines knowledge representation with logic and, thus, supports reasoning within a knowledge domain.

One objective of developing an ontology is to establish a firm understanding of the terminology used within a domain. Thus, it is appropriate when studying ontologies to be familiar with terms surrounding the development of ontologies. Here, we examine the difference between the terms vocabulary, taxonomy, and ontology.

2.1 Vocabulary

Simple vocabularies are human-oriented, as opposed to having structure that a machine could easily interpret. However, there can be vocabularies that have some organization and structure that aids in their interpretation, although the structure may not be optimized for machine interpretation and the interpretations are often subjective and ambiguous. Types of vocabularies include 1) controlled vocabulary, such as a catalog, which provides a finite list of terms together with an unambiguous interpretation of those terms, 2) glossary which provides a list of terms and their meanings in a natural language, and 3) thesaurus which provides some semantics in the form of synonym relationships between terms that greatly reduces ambiguity. However, none of these vocabularies provide explicit term hierarchies. (Gasevic, Djuric, & Devedzic, 2006)

2.2 Taxonomy

A taxonomy is a hierarchical categorization or classification of entities within a domain. (Gasevic, Djuric, & Devedzic, 2006) A taxonomy is used to classify or categorize a collection of concepts within a hierarchical structure. This is a treelike structure that places the most general concept as the root of the tree. Each node of the tree represents some object in the real world that the designer has decided should be modeled based on the stated purpose of the taxonomy. Each link between two nodes in the taxonomy represents a “subclassification-of” relation or a “superclassification-of” relationship. (Boyce & Pahl, 2007)

Taxonomies illustrate more structure than do vocabularies in that they describe supertype/subtype relationships among entities where a child only has a single parent and a parent can contain one or more children. Taxonomies allow the classification of the members of a population into groups and subgroups within subgroups, where every sibling set under a parent

node (class) enables the division of the parent population into mutually exclusive and collectively exhaustive subsets. However, they suffer from not having sufficient range of entity relationship characterization to fully describe a domain. According to David Hay, a major issue with taxonomies for organizing knowledge is that “most of our knowledge is not hierarchical. To cram a body of knowledge into a hierarchical structure leads to all kinds of problems.”

(Hoberman, 2008) A properly structured taxonomy separates the entities into mutually exclusive, unambiguous groups and subgroups that, taken together, include all possibilities. (Gasevic, Djuric, & Devedzic, 2006)

2.3 Ontology

Ontologies are a formal way of organizing information using categories, and relating one category to another. Ontologies include a taxonomy along with additional data that provides a full specification of the domain of interest. (Gasevic, Djuric, & Devedzic, 2006) They offer a simplification of something complex in our environment described by using a standard set of symbols. Ontologies identify a variety of types of relationships among elements, not just hierarchical classification of types of entities as in a taxonomy. (Hoberman, 2008)

Ontologies are used by people, databases, and applications to share terms used to describe an area of knowledge in a given domain of interest. Thus, ontologies assist in resolving a prevalent problem in the data-centric world today – that of data and information that is heavily siloed, having been collected to service very specific and narrowly-focused local needs within the context of specific applications. This poorly managed data capture/location scheme makes it very difficult to reuse data. (Neuhaus, et al., 2011) Ontologies can be used in applications requiring computer-usable definitions of basic concepts in a domain and the relationships among those concepts. Certain applications need ontologies with a significant degree of structure. This

applies to an architectural description of a complex engineered system. These architectures need to specify descriptions for the following kinds of concepts:

- Classes (general things) in the many domains of interest.
- The relationships that can exist among things.
- The properties (or attributes) those things may have.

Ontologies are expressed in a logic-based language, so that accurate and meaningful distinctions can be made among the classes, properties, and relations. (Boyce & Pahl, 2007)

The view that humans take of the world around them is affected by the natural order of things, and by man's impulse to organize the world around him. Whether studying the natural or man-made order of things, the complex interweaving of dependence connections and forms of independence among the many items of which systems are composed becomes apparent to the observer and can become the subject of the attempt to record the discovered elements and relations among them. Through examination of natural and man-made systems, a list of objects can be identified. These objects can be categorized generally as independent items and dependent items. Independent items are those that exist naturally on their own. They are not the result of any intervention on the part of humans. The independent items can be further categorized as real and ideal. The real items being real physical elements we see around us, such as mountains. The ideal elements being abstract objects that are a result of human representation of relations among real concepts by various formalisms in the various sciences, such as sets. Further, the dependent items can also be categorized by real items and ideal items. The real dependent items do not necessarily exist naturally in the world, but are created through human activity, such as a handshake. The ideal dependent elements being abstract objects that are a result of human representation of relative concepts among the real dependent items, such as color. (Poli, 2003)

2.4 State of Readiness in Ontology Development

This section addresses the current state of readiness of institutions and industry to train the workforce needed to take on ontology development tasks. The information provided here is derived from the Joint Communiqué of the Ontology Summit 2010. The 2010 Ontology Summit was devoted to the education of ontologists under the heading “Creating the Ontologists of the Future”. (Neuhaus, et al., 2011)

2.4.1 Current State of Need for Trained Ontologists

There already exists a great demand for trained ontologists in research and industry. This demand is expected to increase over time as new uses for ontologies are identified and their successful applications proven. As the world becomes more data-centric, the need to characterize and process that data becomes more urgent. Ontologies can assist in efficiently processing this data. Ontologist skilled in these methods will need to become available to meet the growing demand. These ontologists will be employed in research to develop new ontological theories, methods, and tools that are then used by ontologists in industry to create ontologies, use them to manipulate data, and evaluate the results. (Neuhaus, et al., 2011)

2.4.2 Current State of Ontologist Training

The communique reported the following findings regarding the educational opportunities available. (Neuhaus, et al., 2011)

2.4.2.1 Demand for Ontologists Increasing

The demand for ontologists is expected to rise considerably. It is expected that 5% of information system and software engineering professionals will be required to have some degree of ontology education or training.

2.4.2.2 Gap Between Educational Needs and Education Availability

There exists a large gap between educational needs and education availability. Unfortunately, our educational system and industry training mechanism are not suited for delivering trained ontologists at the rate needed in research and industry. Institutions are finding it difficult to locate skilled ontologists. There exists no professional organization chartered to certify skilled ontologists. There are few educational institutions that offer courses in ontology, let alone degrees in the field. As of 2011, only 21 educational programs were identified that offered courses in ontology. And of those, only one was identified as being devoted to education in applied ontology. As a result of the lack of training opportunities for aspiring ontologists, graduates often do not meet the needs of organizations seeking skilled professionals. New educational organizations and new educational methods need to be identified and developed to train the ontologists needed today and into the future. Due to the lack of educational opportunities in this field, those that consider a career in a field involving ontology development or use must rely on becoming self-taught or seek out on-the-job training.

2.4.2.3 Demand for Training Opportunities Increasing

Significant demand for training opportunities for working professionals exists. The communique reported finding that most training opportunities were as a result of formal educational opportunities, as few as there are. Thus, training opportunities outside of formal education are essentially non-existent. People that indicated an interest in ontology education were found to have developed that interest from exposure to the topic through work assignments. While these subjects are often part of the typical college curriculum, they often are not found in industry training programs. Thus, there is a significant demand for training at work; and not just to become familiar with the subject area, but rather to develop significant technical competence.

2.4.2.4 Important Subjects are Absent

Important subjects are absent from existing curricula. Workforce professionals who have to staff and manage those entering career ontologist positions have identified technical subject areas related to ontology development that are not covered by the educational curriculum taken by candidates filling the positions.

2.4.2.5 Ontology is Interdisciplinary

The research, development, and application of ontologies is seen by those practicing the skill as being a very interdisciplinary occupation. Since ontology engineering is seen much as a service that supports multiple domains, practitioners suggest a curriculum that includes contributions from a variety of fields that can benefit from the product of ontological engineering. Educational programs should be designed to attract students with varied backgrounds and interests.

2.4.2.6 Qualified Ontologists not Recognizable by Industry

Employers cannot easily recognize qualified ontologists. Due to the fact that educational programs have such few opportunities for those interested in fields directly or indirectly related ontological engineering to engage in courses that relate to the field, organizations requiring that skill find it difficult to identify qualified candidates. Further, there exists no professional certification or qualification organization established to certify professionals who have the requisite skills for performing the tasks in an industrial setting. This makes it very difficult for employers to identify candidates that have proven specialty skills in ontology.

2.4.3 Required Knowledge and Skills for Development and Application of Ontologies

In order to be an effective professional in the field of ontology development and application, the ontologist must command a specific set of skills. The Communiqué identified

three sets of knowledge and skills that a candidate would need to display to qualify for performing tasks associated with the development and application of ontologies in research and industry. The three knowledge and skill sets identified in the Communiqué are 1) Core skills, 2) Core knowledge, and 3) Elective knowledge and skills. The Communiqué recommends that educational institutions interested in training the next generation of ontologist should consider providing course content and career path opportunities that emphasize these skills and knowledge areas. According to the Communiqué, in order to be prepared for a career involving ontology development or application, a student should be required to gain competence in all of the core areas and some of the elective skills and knowledge. The three skill sets that the Communiqué developed are reproduced in Tables 1-4, with an assessment by the author of this thesis as to which of these skills would be of particular use in developing ontologies that serve the purpose of transferring knowledge to modeling profiles used to develop system architectures. The Communiqué indicates that many of these skills are not developed through course lecture alone. Practical, hands-on experience with developing ontologies that help solve real-world issues is important to developing the requisite knowledge and skills. (Neuhaus, et al., 2011)

Table 1: Core Skills Required of a Professional Ontologist (Neches, et al., 1991)

Required Core Skill	Service to Architectural Development
Clarify the purpose of an ontology	High
Analyze data for relevancy to a project	High
Judge the kinds of ontologies useful to a project	High
Managing ontologies across the lifecycle	Medium
Using software tools for ontology development	High
Choosing a representation language	Low
Selecting the appropriate level of detail	Medium
Identify existing content resources	High
Assemble an ontology from reusable modules	High
Using different representation languages	Low
Identify ontological entities and relationships	High
Evaluate and improve ontologies	Medium

Document ontologies	Medium
Support distributed development of ontologies	Medium
Use one or more modern programming language	Low

Table 2: Core Knowledge Required of a Professional Ontologist (Neuhaus, et al., 2011)

Required Core Knowledge	Service to Architectural Development
Basic terminology of ontology	High
Theoretical foundations of ontology	
First-order logic	Low
Set theory	Low
Basic notions of philosophical ontology	Medium
Philosophy of language	Low
Conceptual modeling	High
Representation languages (RDF, OWL, Common Logic)	Low
Building/editing ontologies	
Application of classification principles	High
Software tools	High
Addressing interoperability issues	High
Ontology evaluation strategies	High
Ontology methodologies	
Upper-level ontologies	Medium
Mid-level domain-spanning ontologies	Medium
Domain-specific ontologies	High
Applications of ontologies	
As controlled vocabulary	High
To solve interoperability problems	Low
For reasoning	Low (at this time)
To improve search and retrieval	Low
For natural language processing	Low
For decision support	Low
Web Applications	
General foundations (URIs, XML, etc.)	Low
Semantic Web initiatives	Low
Publishing, annotation, curation	Low

Table 3: Elective Skills of a Professional Ontologist (Neuhaus, et al., 2011)

Elective Skills	Service to Architectural Development
Coordinate ontology development efforts	High
Creating visualizations of ontologies	Medium

Training people in the use of ontologies	Low
--	-----

Table 4: Elective Knowledge of a Professional Ontologist (Neuhaus, et al., 2011)

Elective Knowledge	Service to Architectural Development
Advanced logic	Low
Advanced philosophical ontology	Low
Computer science	
Formal languages	Low
Automated reasoning	Low
Database theory	Low
Artificial intelligence	Low
Logic programming	Low
Linguistic/cognitive sciences	
Syntax, semantics, pragmatics	Medium
Natural language processing	Low
Cognitive theories of categorization	High
Representation languages (SWRL, RIF, SKOS, OBO Format, UML, IKRIS)	High (for UML, SysML)
Ontology content acquisition (data mining)	High
Ontology interoperability	Medium
Building ontology repositories	Medium
Usability and user interface issues	Low
Knowledge of application domain	High

As can be seen by the large variety of topics that the ontologist must have either have some familiarity with or develop significant knowledge of, there remains a good deal of work required of educational institutions to identify and develop a quality curriculum for those seeking careers as ontologists. The field of ontology research, and application is still a very young discipline. There is, as the Communiqué points out, no widely agreed upon body of shared knowledge, established methodologies or common terminology. Instead, multiple terminologies are used in the different subfields of ontology, for example, deriving from specific programming environments, from database design and the conceptual modeling community, or from traditional philosophical ontology. (Neuhaus, et al., 2011)

3 Historical Background on Use of Ontologies in Engineering

3.1 Use of Ontologies in Engineering in General

Globa, et al. provide a set of considerations for usage of ontologies in engineering applications. The objective of their work is to help establish answers to the following questions in order to properly scope the ontology development effort:

- Which domain will be the subject of the ontology?
- What questions should the knowledge representation in the ontology address?
- Who will use and maintain the ontology?

They suggest the development of four separate ontologies to support engineering activities. These are 1) an engineering activity ontology, 2) an engineering knowledge ontology, 3) an engineering computations ontology, and 4) a subject domain ontology. The purpose of the engineering activity ontology is to capture concepts related to the business organization of engineering activities, such as the people, organizations, tasks, etc related to accomplishing the engineering objectives within the business. The purpose of the engineering knowledge ontology is to capture the meta-concepts that specify structures to describe the problem, such as the methods, objects, results, and equipment used in research activities that provide the knowledge to support the engineering activities. The purpose of the engineering computations ontology is to capture the classes that describe calculation abilities needed to support the conversion of data to knowledge, such as the kinds of calculations, services, service parameters, interfaces, etc., needed. (Globa, Novogrudska, Koval, & Senchenko, 2018)

3.2 Use of Ontologies in Systems Engineering and Manufacturing

Systems engineering and system manufacturing have both seen an increase in the importance and popularity of the use of ontologies to solve critical problems. Ontology and semantic technologies have been adopted by the engineering community as a promising approach to solve several of these issues such as information modeling, data integration, data analysis, data exchange, system interoperability, etc. For example, in product design, ontologies are used 1) for modeling the product structure and taxonomy, 2) for design automation using existing engineering knowledge, and 3) for requirements engineering. In manufacturing, ontologies are used 1) for the control of production processes for dynamic orchestration, 2) for factory automation, and 3) for the mapping of data sources to Manufacturing Execution Systems functions. (El Kadiri, et al., 2015) El Kadiri, et.al. describe three specific FP7¹ European projects that exemplify the use of ontologies in engineering applications.

3.2.1 LinkedDesign Project

The goal of the LinkedDesign project is to collect product manufacturing data from factory floor work stations to feed operational efficiency analysis, future product design Life Cycle Cost (LCC) analysis, and to respond to changing customer requirements with speed. In order to collect and process manufacturing data from a variety of work stations reporting such data in a variety of formats, locations, and times, etc, the team defined a common semantic model that enables common interpretations of data and information exchanged between people and systems that have no common recognition of data type or relationships. Analysis of the LCC across the enterprise allows the factory configuration to be selectively optimized to meet LCC

¹ FP7 refers to the Seventh Framework Programme of the European Union

requirements. Thus, various LCC options can be presented to the customer to enhance the available selection. To enable advanced control of products design and maintenance, three groups of rules were created: 1) rules for enforcing customer requests that select workstation configurations that meet customer LCC requirements, 2) rules for inheritance of properties from part to product such that if the configuration of a part drives LCC, that property is inherited by the workstation processing the part, 3) rules to alert service teams when the production line is not functioning at optimal performance. (El Kadiri, et al., 2015)

3.2.2 VFF Project

The goal of the Virtual Factory Framework (VFF) project was to develop an integrated collaborative virtual environment intended to synchronize factory floor production operations with various simulations of those operations for near-real time optimization of factory operations. Distribution, modeling integration, and reasoning of data was accomplished using Semantic Web technologies, in particular, an ontology-based data model, named Virtual Factory Data Model (VFDM). The VFDM model allows seamless data exchange between disparate software tools provided they employ a software connector that transforms data from the ontology format to the proprietary data structures of the tools, and vice-versa. (El Kadiri, et al., 2015)

3.2.3 FLEXINET Project

The goal of this project is to “provide decision support on how to best design and facilitate Global Production Networks (GPN)”. GPNs consist of a set of diverse and divergent facilities, personnel, and organizations over vast geographical areas. FLEXINET is intended to provide the ability to reconfigure the configuration and operation of these networks in order to accommodate the introduction of new manufacturing technologies thereby reducing costs, risk, and/or improving production rates. To accomplish this, FLEXINET employs a reference

ontology that provides a consistent data-set of production information and knowledge from across the entire span of facilities that support the GPN. The resulting process helps identify the optimal arrangement of in-sourcing, out-sourcing, partnerships, logistics, etc. to achieve manufacturing and LCC goals. (El Kadiri, et al., 2015)

3.3 Use of Ontologies Specifically in Software Engineering

Software Engineering shares a common legacy with Knowledge Engineering from which the current interpretation of what ontology means emanates. However, despite this common legacy, both communities have developed along different paths and mostly live in their own worlds. The aim of Software Engineering has been toward achieving a higher degree of abstraction through 1) modeling that places greater emphasis on development activities based on the modeling of objects and procedures, and 2) higher-level programming languages. Meanwhile, Knowledge Engineering has been focused on realizing the vision of the Semantic Web, which has spawned the development of new technologies and tools for ontology representation, machine-processing, and ontology sharing. This makes their adoption in real-world applications much easier placing ontologies in the position to enter mainstream use. While there are movements to build commonality among the two disciplines, little work is being done to develop specific guidelines for practicing engineers to employ. As a result, each discipline continues to develop their own core concepts, thus making it increasingly difficult for one community to engage with the other. Nevertheless, there are opportunities for ontologies to bridge the gap between the two communities. (Happel & Seedorf, 2006)

Happel and Seedorf have defined a set of concrete approaches for using ontologies in the context of Software Engineering, presented here in the order of appearance in the Software Engineering lifecycle.

3.3.1 Analysis and Design

3.3.1.1 Requirements Engineering

In this phase of the lifecycle, the objective is to gather the desired system functionality from customers. It is important for all participants in the process to have a shared understanding of the problem domain. Ontologies can be used to describe the requirements specification documents and to formally represent requirements knowledge. Requirements are normally stated in terms of natural language. However, ontologies can play a role here through the use of formal specification languages which are generally more precise than natural language. This higher level of precision can lead more directly and more effectively towards the production of formal system specifications. The use of ontologies offers several improvements to traditional Requirements Engineering: 1) requirements ontologies, if properly architected, support automated requirements consistency checking and validation, 2) serve as prerequisites to realize model-driven approaches in the design and implementation phases. (Happel & Seedorf, 2006)

3.3.1.2 Component Reuse

Reuse implies the use of previously designed and developed components when implementing functionality in order to reduce costs by avoiding rework. Most reuse repositories rely on plain syntactical key-word-based search which suffers from low precision (due to homonyms) and low recall (due to synonyms). Ontologies can help due to their more convenient and powerful querying capability made possible by a knowledge representation formalism for describing the functionality of components sought for reuse. Thus, ontologies can help to combine information isolated in several separate component description repositories. Ontologies can also provide background information that allows non-experts to query the repository in

search of reuse components from their point of view, using terminology that may not be exactly aligned with the terminology used in the components sought after. (Happel & Seedorf, 2006)

3.3.2 Implementation

3.3.2.1 Integration with Software Modeling Languages

Modern software development practices follow the Model-Driven Architecture approach which provides an architecture for creating models based on metamodels, and which defines the transformations between those models, and managing metadata. MDA-based languages do not yet have a knowledge-based foundation to enable reasoning. So, there exists interest in integrating MDA-based information representation languages, such as UML and SysML, with ontology languages, such as RDF/OWL. These two language bases are regarded as two distinct technological spaces. However, it is possible to discover synergies between them that can be realized by defining bridges between them, such as the Ontology Definition Metamodel (ODM). ODM is an effort to standardize the mappings between knowledge representation and conceptual modeling languages. (Happel & Seedorf, 2006)

3.3.2.2 Ontology as Domain Object Model

In order to promote broad acceptance and use of ontologies in software development projects, it is imperative that automated means for object-oriented software developers to access ontologies be developed to avoid the need for building special knowledge by the developers to gain that access. This is accomplished by automating the mapping of the domain model to code in order to enable the dynamic use by other components and applications. This can be achieved by ontology tools that generate an API from the ontology, by mapping concepts of the ontology to classes in an object oriented language. The generated domain object model can then be used to manage models, and for inferencing and querying. The automated end-to-end use of ontologies

in analysis and design, as well as implementation, is highly desirable for rapid application development. (Happel & Seedorf, 2006)

3.3.2.3 Coding Support

Some Integrated Development Environments (IDEs) like Eclipse use the documentation of Application Programming Interfaces (APIs) to enhance developer productivity by providing autocompletion of method calls. New approaches to IDE environment programming suggest enriching APIs with semantic information provided by ontologies. The needed annotations could be stored in a public web service to enable collaborative knowledge acquisition. This approach could also be used to automatically generate a suitable sequence of method calls to achieve a desired goal state (like getting a database result set). The main advantage of ontologies is that they provide a globally unique identifier for concepts. An ontology enables developers to annotate API elements with an unambiguous concept. (Happel & Seedorf, 2006)

3.3.2.4 Code Documentation

Programming languages are poorly suited for software maintenance tasks such as documentation. They describe knowledge in a procedural way and are not well suited for the querying of knowledge required to pull knowledge to support documentation activities. The use of applied description logics provides a data environment that consists of programming-language independent descriptions of software structures and an ontology that describes the problem domain of the software. Both can be manually connected to allow the querying of code features dealing with a certain domain object. (Happel & Seedorf, 2006)

3.3.3 Deployment and Run-Time

3.3.3.1 Semantic Middleware

In modern three-tier architectures for software systems, the middleware layer lies in the focus of attention. Sophisticated middleware infrastructures shield a lot of complexity from the application developer, but creates challenging tasks for other tasks. Ontologies can be used to support the formal description of concepts from component-based and service-oriented development. The ontology provides a precise, formal definition of some ambiguous terms from Software Engineering as well as structures supporting the formalization of middleware knowledge by modeling the dependencies of libraries, licenses etc. (Happel & Seedorf, 2006)

3.3.3.2 Business Rules

Today's business environment requires that companies react to rapidly-changing market conditions necessitating frequent adjustments to business rules. Often, the business logic of a company is hard-coded in programming languages. Thus, changes to the business logic of a software system require modifications to the source code, triggering the normal compilation and deployment cycle. As a result, companies are looking for solutions that support a quick propagation of new business rules into the core software systems by disconnecting business logic from processing logic. Rule engines are a possible solution to this problem. The business logic is modeled declaratively with logical statements and processed by a rule engine. Similar to a reasoner, the rule engine applies inference algorithms to derive new facts on a knowledge base. Business rule engines can be regarded as "ontology-based" approaches since they run declarative knowledge on a special middleware. Business rules can be changed more easily, because they are explicitly stated in a formal language that can be presented in a user friendly way for editing. (Happel & Seedorf, 2006)

3.3.3.3 Semantic Web Services

Web services enable developers to combine information from different sources to new services. Offering data and services via well-defined interface descriptions in the web is the core idea of web services. However, it is often difficult for developers to find appropriate services, since most industry standards are purely syntactical, lacking semantical meaning. Thus, an algorithm cannot find out whether the output of one service is appropriate as an input to another service. Semantic web services add a semantic layer on top of the existing web service infrastructure. Input parameters, functionality and return values are annotated semantically, allowing automatic discovery, matching, and composition of service-based workflows. Ontologies can ensure discovery and interoperability in cases that were not anticipated by the initial developer, since semantic descriptions can be extended over the course of time. Even mediation among services that have been developed independently and annotated with different ontologies could interoperate by defining mappings between the services that is then interpreted by the ontology language. (Happel & Seedorf, 2006)

3.3.4 Maintenance

3.3.4.1 Project Support

In software maintenance workflows such as bug fixing, several kinds of related information exist without an explicit connection. This is problematic, since a unified view could avoid redundant work and speed up problem solving. Ontologies help to connect the electronic communication (via forums and mailing lists) of the developers with bug-reports and the affected areas in the source code. Central concepts are the community (e.g. developers), their interactions, and content (e.g. emails). The knowledge is codified in three kinds of ontologies: 1) content ontologies that describe the structure of artefacts, 2) an ontology of interactions that describes the

communication flow among the developers, and 3) a community ontology that defines the roles that are involved in the problem solving process. Ontologies thus provide a layer to integrate data from different sources into a unified semantic model. The combined data can then be used to derive additional information that was not stated explicitly in any one of the single sources before. (Happel & Seedorf, 2006)

3.3.4.2 Testing

Software testing is an important part of quality assurance. However, the writing of test cases is an expensive endeavor that does not directly yield business value. Furthermore, the derivation of suitable test cases demands a certain amount of domain knowledge. Ontologies could help to generate basic test cases since they encode domain knowledge in a machine processable format. Ontologies may not be the first candidate for such a scenario, since there are formalisms like Object Constraint Language (OCL) that are specialized for such tasks. However, once domain knowledge is available in an ontology format, it might be feasible to reuse that knowledge. (Happel & Seedorf, 2006)

3.4 Use of Ontologies Specifically in Systems Engineering

There exist many possible applications of ontologies in systems engineering activities. The trend is growing to investigate newer such applications. Hennig, et al. in 2011 surveyed a set of reported applications to assess their type and usefulness as exemplars of the application of ontologies in systems engineering projects. While somewhat dated, the survey illustrates the types of applications that organizations see as having solutions by using ontologies. Nine of the surveyed projects are summarized here to describe the application of the ontologies that the organization implementing them had in mind.

3.4.1 Domain Knowledge Acquisition Process

In (Sarder & Ferreira, 2007), Sarder and Ferreira (2006) describe their Domain Knowledge Acquisition Process (DKAP) to capture a systems engineering functional domain ontology, with plans to use the developed systems engineering ontology to further develop a system of systems (SoS) engineering ontology. In order to serve the interest of SoS projects, they acknowledge that it is important to resolve the differing semantics and standards used by the many and varied system types that make up a SoS and the varied disciplines and backgrounds of the engineers performing the SE tasks on such projects. The authors see as a solution the development of a SoS ontology that consolidates and resolves differences among the individual system ontologies. The authors surveyed several techniques and tools for developing ontologies and selected the IDEF5 elaboration language as the means for developing ontologies in their project. The authors also surveyed methodologies for developing ontologies and selected the DKAP method for their project. The authors described the DKAP process and used the process to identify the major entities of the systems engineering domain, which were then presented as a taxonomy. It should be noted that the entities shown in the resulting taxonomy are of the systems engineering “process”, not of any given systems engineering “project.” The work is concluded by indicating that the authors intend to also apply the DKAP methodology to develop a System of Systems Engineering (SoSE) ontology. (Sarder & Ferreira, 2007)

3.4.2 Knowledge Modeling Framework

In (Chourabi, Pollet, & Ben Ahmed, 2008), Chourabi, et al. describe a layered set of ontologies intended to capture knowledge items used in the systems engineering process in order to record engineers’ ideas and reasoning processes, and facilitate their reuse. They propose a Knowledge Modeling Framework for systems engineering projects consisting of a SE General

Ontology and an ontological framework organized into four semantic layers used to capture knowledge. The SE General Ontology three description facets: 1) Domain Facet - contains a set of ontologies that capture basic concepts and relations used to describe the content of engineered systems on a high semantic level, 2) Product Facet – contains concepts and relations representing a system by formally relating modeling elements to domain concepts to provide a systematic and semantic description of an engineering solution, 3) Process Facet - contains concepts and relations that formally describe engineering activities, tasks, actors, and design rationale. The Multi-layered ontologies for SE knowledge modeling are subdivided into several levels of abstraction, thus separating general knowledge from knowledge about particular domains, organizations and projects. These four layers are 1) General Layer - to describe super-concepts that are the same across all domains, it corresponds to the SE General Ontology, 2) Domain Layer - defines specializing concepts and semantic relations for a specific systems engineering domain , 3) Application Layer - presents specialized concepts that act as a systematized representation for annotating engineering knowledge on a particular project, 4) Instance Layer – defines all instances of engineering ontology concepts, defining a conceptual vocabulary from the application layer.

3.4.3 Combining Metamodel-Based Models with Ontology-Oriented Implementation

In (Ernadote, 2015), Ernadote proposes the use of ontologies to fulfill several objectives: to enhance the communications between domain specialists and modelers, to enhance the communications among specialist in different domains, facilitate the collection of system information to be used in modeling, to create new perspectives on existing models, and to generate documentation using those perspectives. Ernadote suggests a new modeling approach which is a combination of metamodel-based models with ontology-oriented implementation. In

Erandote's view, metamodel-modeling fails to fully address the communication problems project that spans multiple domains since modelers have to agree in advance on the meaning of the data they are creating in the models. And while ontology-oriented approaches are seen as properly addressing multi-domain projects, Erandote nevertheless feels there remain several disadvantages of ontology-oriented modeling. These are: without the benefit of a metamodeling tool (and modelers) stakeholders now have the responsibility of binding an ontology to existing system data, the visualization in ontology authoring tools is difficult for end-user to understand, domain-specific languages and tool are time-consuming to use and lack flexibility. The solution Erandote proposes is a combined metamodel-ontology, or "mixed" approach. The advantages of this approach over the others is that all the advantages of metamodeling apply while only modelers need to know the particulars of constructing the metamodels. Erandote proceeds to describe the use of Category Theory as a means for mapping the ontology to the metamodel.

3.4.4 Decision Support System

In (Thakker, Dimitrova, Cohn, & Valdes, 2015) Thakker, et al. describe a prototype application of ontologies in a systems engineering Decision Support System (DSS) project to capture and preserve tacit knowledge from domain experts involved in the inspection of a railway tunnel network in France. The project turned to knowledge systems for assistance due to the complexity of the inspection process which is prone to subjectivity and scales poorly across cases and domains. The Pathology Assessment and Diagnosis of Tunnels (PADTUN) project assist tunnel experts in "making decisions about a tunnel's condition with respect to its disorders and diagnosis influencing factors." The system consists of two main components: the Pathology Assessment and Diagnosis component, and the Ontology component. The Pathology Assessment and Diagnosis component is designed using a three-tier architecture of Presentation (User

Interface) Layer, a Processing (Application) Layer, and a Data Layer. The Data Layer contains a relational database for storing inspection data supplied by the domain experts. The Data Layer also contains a semantic repository (triple store) that stores the domain knowledge in the form of an ontology, and performs reasoning on the inspection data. The Processing Layer consists of three subcomponents: 1) the Pathology Inferencing component that uses the stored ontologies to infer a list of pathologies when provided with observed tunnel inspection disorders, 2) a Regions Of Interest (ROI) component uses the output of the Pathology Inferencing component together with stored ontologies to infer aggregate tunnel portions that are susceptible to the same types of pathologies – a process which traditionally has been done by experts in an intuitive fashion, and 3) a Data Management component that stores inspection data as per the schema dictated by the ontologies. The conceptualization of the domain by experts was converted into OWL ontologies. The PADTUN ontologies were designed based on the knowledge of domain experts and were developed using the METHONTOLOGY methodology. The ontologies were designed for the purpose of capturing the existing decision process used in diagnosing tunnel pathologies, and to provide a context for automated decision support on the part of inspectors so as to result in a more consistent and reliable pathology assessment. In a comparison of ROI inferencing between the new ontology-based system and traditional methods the new system produced results which were in “almost perfect agreement.” (Thakker, Dimitrova, Cohn, & Valdes, 2015)

3.4.5 Knowledge Base from SysML Block Definition Diagrams

In (Graves, Integrating SysML and OWL, 2009), Graves describes a method for constructing a system design Knowledge Base (KB) based on information transformed from SysML Block Definition Diagrams (BDD). Such a KB could represent detailed information of a system design, such as the number of occurrences of a part and interconnections between parts.

The objective would be to take advantage of ontological reasoning tools to analyze the system design. Graves argues that SysML BDDs have sufficient expressiveness to represent these detailed designs. Accordingly, if the SysML BDDs are restricted to include only associations and no operations, then these diagrams can be translated into OWL2 to provide the degree of system description being sought. In this approach, design KBs can be developed in engineering design tools using SysML and then exported to OWL tools for analysis while preserving the intended semantics of the SysML BDD. A larger goal would be to use formal reasoning tools in product development that takes full advantage of the expressivity provided in the SysML. However, this would require a formal semantics for a much richer subset of SysML, for example, including ports with their interfaces, and including SysML operations. (Graves, Integrating SysML and OWL, 2009)

3.4.6 Computer Aided Engineering Exchange

In (Abele, Legat, Grimm, & Muller, 2013), Abele, et al. a solution is sought for the problem of exchanging and validating manufacturing plant engineering models. In particular, a data exchange mechanism is needed to transform data among models using the XML-based data format called Computer Aided Engineering Exchange (CAEX) which is part of the AutomationML (AML) language specification. CAEX was specially developed to meet the requirements of the manufacturing engineering domain. It is currently the most recognized standard data exchange tool for plant engineering data. A proposed solution to these data exchange issues is presented which includes the automated validation of CAEX plant models by means of their transformation into Web Ontology Language (OWL) ontologies, and subsequent application of reasoning mechanisms to perform the validation process. The engineering process

using the CAEX standard consists of three major steps, as illustrated in Figure 2. (Abele, Legat, Grimm, & Muller, 2013)

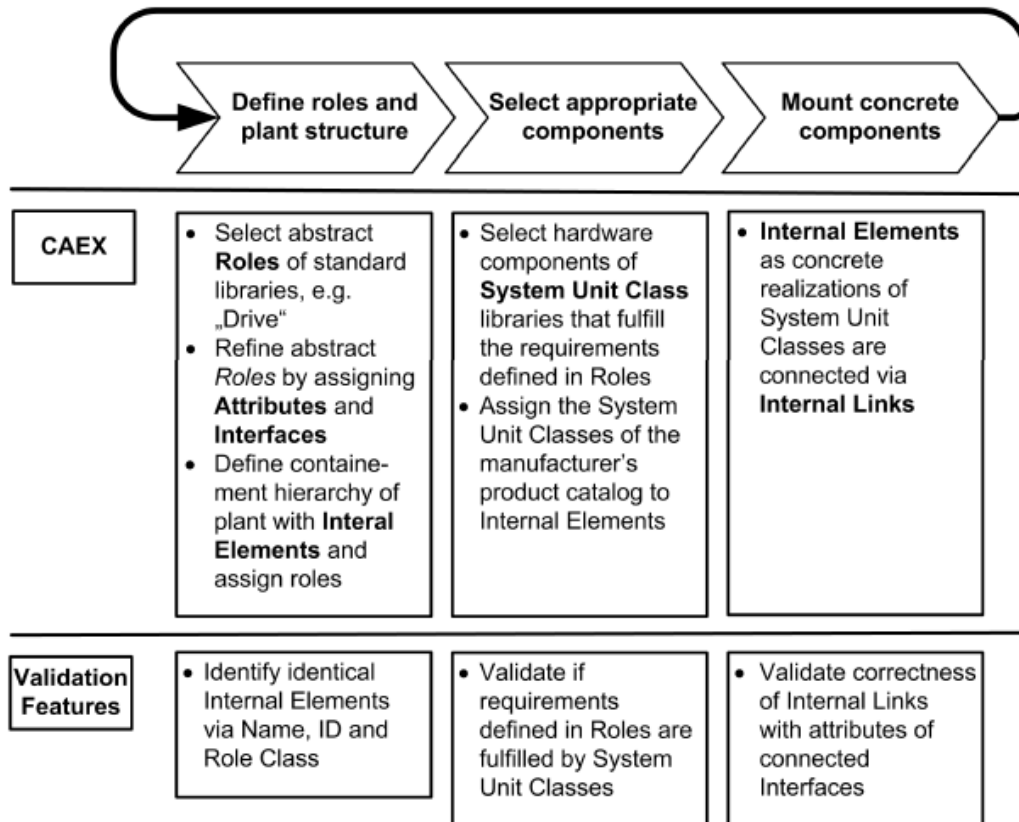


Figure 2: CAEX Plant Models Validation Process Using OWL (Abele, Legat, Grimm, & Muller, 2013)

In the first step of the process, the roles to be used in the respective domain are defined. This is accomplished by defining the user-specific roles for system elements as specializations of standard roles defined in the AML standard libraries. Since multiple domain experts may be working on the engineering model concurrently, different system elements representing the same physical component might possibly be created. To support the consistency of the model, a validation activity must be developed to identify multiple instances of the same physical element in the model. In the second step, the defined roles are used for selecting suitable components

from vendor-specific product catalogues. After manually selecting a suitable component from the catalogues, it is assigned to the previously defined system elements. Due to the manual component selection process, it is possible that a component not fully meeting the role requirements might be assigned. Therefore, another validation activity must be developed to check for improper assignment of catalogue component to system elements in the model. In the third step, the system elements are connected by interfaces representing the plant-specific inter-component connections. Due to the complexity of a plant model, interfaces might be incorrectly place between the wrong system elements. Therefore, another validation activity must be developed to check for such inconsistencies. The transformation from CAEX to OWL captures the basic design decisions of representing CAEX plant models in OWL ontologies. The Semantic Web querying and reasoning technologies incorporated into OWL are used to perform three validation consistency checks of the CAEX process. These are 1) performing a query to identify all system elements with the same name to determine whether they were intentionally assigned the same name, 2) performing a query to ensure that components selected from vendor-specific product catalogues match the defined roles to which they are being assigned, and 3) performing a query to check that all interfaces are properly aligned according to the standard definition of interfaces provided in the AML standard libraries. (Abele, Legat, Grimm, & Muller, 2013)

3.4.7 State Analysis Methodology

In (Wagner, et al., 2012), Wagner, et al. present the State Analysis methodology as a means for architecting, designing and documenting complex control systems. In this project, State Analysis is performed using the Systems Modeling Language (SysML). To make use of the SysML capabilities, it is necessary to provide ontological definitions of the concepts and relations in State Analysis. This is accomplished through a mapping of State Analysis into a

practical extension of SysML. The ontology provides the formal basis for verifying compliance of the system model developed in SysML with State Analysis semantics including architectural constraints. This is accomplished by first applying stereotyped relations in the SysML model so that it can be analyzed to compare the semantics and constraints expressed in the stereotype definitions with the details of the model, and thereby verify that the model conforms to the semantics of the domain expressed in the ontology. The State Analysis domain is constructed as an ontology in OWL2 using an ontology editing tool. Thus, by using a model transformation from OWL2, meaningful domain-specific stereotypes are defined and applied in a SysML modeling tool to construct a system model. Then, the system model is exported to OWL in order to enforce semantic consistency rules established by the principles of State Analysis and verify the correctness properties in the model. While the focus of this work was to illustrate the use of State Analysis in the design of control systems for large, complex enterprises, the value of this work to the author of this thesis is the process used for mapping of the ontologies into SysML to define the ontological concepts and relationships as SysML stereotypes that can be applied to appropriate modeling entities. Wagner, et al. only say that success in this endeavor is due to “some advanced model transformations developed by JPL’s Integrated Model-Centric Engineering team.” This hint prompted the author of this thesis to investigate activities at JPL further to discover the nature of the JPL advanced model transformations. Of all the applications of the ontologies to systems engineering, this project showed the most promise for describing a practical approach to populating profiles for use in architectural development, which is the concern of this thesis. The further works of NASA JPL toward this goal are described in Section 6 of this thesis. (Wagner, et al., 2012)

3.4.8 Integrating Reasoning with SysML

In (Graves, Integrating Reasoning with SysML, 2014), Graves addresses the need to perform in-depth reasoning on engineering tasks by embedding a model of the system under analysis as an axiom set within a suitable logic. By taking this approach, engineering questions translate into questions about axiom sets. Automated reasoning can then be used to answer these questions. Graves illustrates techniques for embedding the class diagram fragment of SysML into OWL, and then extending that approach to cover other SysML constructs. Graves then illustrates how reasoning can be integrated with SysML to answer engineering questions, with three examples. These examples relate a variety of engineering questions to axiom set questions that are then formulated as model queries. The first example illustrates how an advertised system capability can be verified using reasoning. The second two examples illustrate design consistency can be maintained by verifying the consistency of design changes. Examples are given to, and illustrate how formal reasoning can be exploited to answer these questions. The examples presented illustrate the semantic embedding of a Block Definition Diagram (BDD) fragment of SysML into a type theory logic. Other important SysML language constructions, such as the Internal Block Diagram (IBD) cannot be embedded within OWL. To overcome this issue, Graves suggests that SysML be reengineered to use an engineered version of type theory as its foundation. Graves states that “Type theory provides the language extensions suggested by the examples with a formal semantics well adapted for use with inference engines.” (Graves, Integrating Reasoning with SysML, 2012)

3.4.9 Managing Inconsistencies in Models

In (Feldmann, et al., 2015), Feldmann et al. address the challenges related to managing inconsistencies in models of systems from the domain of automated production systems. These

inconsistencies arise out of the collaborative nature of a variety of stakeholders from different disciplines employing a variety of modeling languages, formalisms, and tools. Three challenges to consistency management are identified as needing to be resolved: 1) heterogeneity of models causes issues such as misinterpretation of parameters among those that specify a required attribute, and those that reveal the current state of an attribute for analysis, as well as fundamentally different formalisms, varying abstraction levels, and terminology relevant to a particular application domain, 2) semantically overlapping models marked by the presence of either duplicate, or related information, referred to as semantic overlaps, 3) lack of automated inconsistency management techniques. The proposed solution to manage inconsistencies is the use of a knowledge-based system composed of two parts: a knowledge base and an inference mechanism. The Resource Description Framework (RDF) is proposed for use as a knowledge representation formalism. RDF allows for statements to be made about entities the form of subject-predicate-object triples and therefore is similar to conceptual modeling approaches such as class or entity relationship diagrams. Use of the SPARQL Protocol and RDF Query Language are proposed as the means to retrieve and manipulate information represented in RDF. The process of using these tools first involves an expert identifying a-priori the specific types of inconsistencies anticipated to be encountered. The application of this approach is illustrated with two examples of inconsistency queries that result in successful identification of inconsistencies, while passing on valid consistency checks. A technology demonstrator was then exercised to evaluate the technical feasibility and viability of the conceptual approach. (Feldmann, et al., 2015)

The process of building or engineering ontologies for use in information systems remains an arcane art form, which must be transformed into a rigorous engineering discipline in order to be viewed as a useful and reliable resource for engineering applications, particularly for developing architectural descriptions of complex engineered systems. (Guarino & Welty, 2002) This section provides guidance on best practices for constructing ontologies.

4.1 Design Criteria

In the words of Tom Gruber, “an ontology is an explicit specification of a conceptualization.” That conceptualization consists of the entities that exist in the domain being described as well as the relationships among those entities. It is said that an ontology is “committed” to the conceptualization, meaning that the design of the ontology accurately represents the conceptualized view of the domain. The set of entities represented in such an ontology is called the “universe of discourse” for that domain. These are the classes, functions, relations, and other objects declared to represent the domain. The ontology includes definitions associated with the names of all the entities in the universe of discourse. The definitions include human-readable text describing what the names mean as well as formal axioms that constrain the possible interpretations of the defined terms. (Gruber, A Translation Approach to Portable Ontology Specifications, 1993)

In order for an ontology to be an accurate description of the conceptualized domain, it needs to be designed as such. This implies that the process for designing ontologies comes with design criteria. Tom Gruber defined five design criteria for constructing ontologies. The first of these is clarity. Definitions of terms should be complete, objective, and written in natural language. Definitions should be independent of social or computational context. Formalism

promotes this independence. To achieve formalism in the definition, logical axioms should be used to define the terms. Completeness implies the use of a predicate defined by necessary and sufficient conditions. This is preferred over a partial definition which is defined only by necessary or sufficient conditions. (Gruber, A Translation Approach to Portable Ontology Specifications, 1993)

The second criterion is coherence. This applies to both the formal and informal elements of the definition. At the least, the defining axioms should be logically consistent. If the axioms infer a sentence that contradicts an informal definition, then the ontology is incoherent. (Gruber, A Translation Approach to Portable Ontology Specifications, 1993)

The ontology should be extendable monotonically in order to be reusable for multiple purposes or tasks without requiring revision of the existing definitions. (Gruber, A Translation Approach to Portable Ontology Specifications, 1993)

The ontology should exhibit minimal encoding bias. An encoding bias results when design choices are made purely for the convenience of notation or implementation of the encoding. Minimization of such bias is necessary since knowledge-sharing agents may be implemented in different representation systems. (Gruber, A Translation Approach to Portable Ontology Specifications, 1993)

Finally, the ontology should require minimal ontological commitment sufficient to support the intended knowledge-sharing activities. This allows parties who are committed to using the ontology the freedom to specialize and instantiate the ontology as needed. Such minimization can be achieved by defining only the terms that are essential to the communication

of knowledge consistent with the weakest theory of the domain. (Gruber, A Translation Approach to Portable Ontology Specifications, 1993)

4.2 Ontological Formalisms

Ontologies are often categorized according the degree of restriction on the semantics used to express the ontological terms. As such, ontologies are broken into two major groups: 1) lightweight ontologies, which are mainly taxonomies, and 2) heavyweight ontologies, which provide more restrictions on domain semantics in order to model the domain in a deeper way. Within these groups, ontologies are also categorized according to the level of formality incorporated into their design and definition. The classifications according to formalism are: 1) 1) highly informal - if expressed in natural language; 2) semi-informal - if expressed in a restricted and structured form of natural language; 3) semi-formal - if expressed in an artificial and formally defined language; and 4) rigorously formal - if they provide meticulously defined terms with formal semantics, theorems and proofs of properties. (Gómez-Pérez, Fernández-López, & Corcho, 2004)

4.3 Methods for Modeling Ontologies

This section describes several popular methods employed to develop ontology models. It is important to note that the selection of the formalisms used to model domain knowledge and the languages that implement the modeling techniques limit the kind of knowledge that can be modeled and implemented. For example, to model formal axioms either as independent components in the ontology or embedded in other components, the use of Artificial Intelligence (AI) formalisms are required. AI-based languages and ontology markup languages are better candidates for representing and implementing ontologies than other non AI approaches. Another

important note is that simply because an ontology is written using a language specifically designed for constructing ontologies does not mean that the result constitutes an ontology.

4.3.1 Frames and First Order Logic

In (Gruber, A Translation Approach to Portable Ontology Specifications, 1993), Gruber suggested modeling heavyweight ontologies by using frames and first order logic. In this approach, Gruber used five kinds of modeling components: classes, relations, functions, formal axioms, and instances. (Gómez-Pérez, Fernández-López, & Corcho, 2004)

4.3.2 Description Logics

Description Logics (DL) is a kind of logical formalism theory which is divided into two parts: the TBox and the ABox. The TBox contains the definitions of concepts and roles built through declarations that describe general properties of domain concepts. These are expressed as intensional (terminological) knowledge in the form of a terminology. The ABox contains the definitions of individuals (instances) which is specific to the individuals of the discourse domain. These contain extensional (assertional) knowledge. (Gómez-Pérez, Fernández-López, & Corcho, 2004)

4.3.3 Ontology Modeling Using UML/SysML

The UML (Unified Modeling Language) and SysML (Systems Modeling Language) can both be used for modeling ontologies. UML is commonly used in the software engineering community, and SysML in the systems engineering community, and therefore modeling of lightweight ontologies is a task easily picked up by engineers using either of these two methods. Resulting models can be enriched by adding Object Constraint Language (OCL) expressions to add axioms to these models. (Gómez-Pérez, Fernández-López, & Corcho, 2004) These are the

methods of interest in this thesis, and will be explored further in Sections 5 and 6. (Gómez-Pérez, Fernández-López, & Corcho, 2004)

4.3.4 Ontology Modeling Using Database Technology

This modeling technique primarily involves the use of Entity/Relationship (ER) diagrams and their extensions, as well as other types of databases, such as object-oriented database models or deductive database models. Though, it is not possible to model heavyweight ontologies with the extended ER diagrams commonly used. Other extended ER notations or complementary notations would be needed. Only those ER diagrams that have been agreed upon could be considered ontologies. It is highly desirable that ontologies be machine-readable since many Computer-Aided Software Engineering (CASE) tools are set up for this purpose. (Gómez-Pérez, Fernández-López, & Corcho, 2004)

4.4 Types of Ontologies

Gómez-Pérez, et al. assembled a type characterization of developed ontologies according to the subject of their conceptualization. The result is captured in the following subsections. These are not meant to be exhaustive lists.

4.4.1 Knowledge Representation Ontologies

The most well-known of these Knowledge Representation (KR) ontologies are the Frame Ontology (Gruber, A Translation Approach to Portable Ontology Specifications, 1993) and the Open Knowledge Base Connectivity (OKBC) Ontology. They provide formal definitions of the representation primitives used mainly in frame-based languages and thus permit building other ontologies by means of frame-based conventions. Other KR ontologies include the RDF KR Ontology, RDF Schema KR Ontology, OIL KR Ontology, DAML+OIL KR Ontology and OWL KR Ontology. (Gómez-Pérez, Fernández-López, & Corcho, 2004)

4.4.2 General or Common Ontologies

These are used to represent common sense knowledge that can be reused across all domains. These ontologies capture very general vocabularies related to subjects common to all ontologies, such as things, events, time, space, causality, behavior, function, mereology, etc. The Mereology Ontology is a good example of a general ontology. It defines the Part-Of relation that can be used to state how devices are formed by the assembly of components, each of which might also be decomposed into subcomponents. This ontology defines the principle properties that any decomposition should have. (Gómez-Pérez, Fernández-López, & Corcho, 2004)

4.4.3 Top-level Ontologies Or Upper-level Ontologies

These ontologies describe very general concepts to which all root terms in existing ontologies should be linked. There exist several top-level ontologies that differ on the criteria followed to classify the most general concepts and therefore create some confusion about the manner in which domain ontologies should link to them. To solve work is being performed to develop a Standard Upper Ontology (SUO) that is intended to give a structure and a set of general concepts from which domain ontologies could be constructed. (Gómez-Pérez, Fernández-López, & Corcho, 2004)

4.4.4 Domain Ontologies

These kinds of ontologies may be reusable in a given specific domain, such as medical, pharmaceutical, engineering, etc. They provide vocabularies that describe the concepts within a domain and their relationships, as well as the activities that take place in the domain. There is a clear boundary that separates the domain from the upper-level ontologies. The domain concepts are established by specializing off of concepts defined in top-level ontologies. (Gómez-Pérez, Fernández-López, & Corcho, 2004)

4.4.5 Task Ontologies

Task ontologies describe the vocabulary related to a generic task or activity that can be found in most modern organizations today. They provide a vocabulary of terms used with tasks that may or may not belong to the same domain. (Gómez-Pérez, Fernández-López, & Corcho, 2004)

4.4.6 Domain-Task Ontologies

These ontologies are reusable in a given domain, but not across domains, and therefore are application-independent. (Gómez-Pérez, Fernández-López, & Corcho, 2004)

4.4.7 Method Ontologies

These ontologies define the concepts and relations that can be used to specify a reasoning process that is designed to achieve a particular task, for example. (Gómez-Pérez, Fernández-López, & Corcho, 2004)

4.4.8 Application Ontologies

These are application-dependent ontologies that contain all the definitions needed to model the knowledge required for a particular application. They extend and specialize the vocabulary of the domain and of task ontologies for a given application. (Gómez-Pérez, Fernández-López, & Corcho, 2004)

4.5 Languages for Building Ontologies

In (Gómez-Pérez, Fernández-López, & Corcho, 2004), Gómez-Pérez, et al. provide a comprehensive overview of the languages used by ontologists to construct ontologies. This section will quickly summarize those languages identified, simply for reference purposes. The authors break the grouping of languages into two types: traditional languages and markup

languages. For traditional languages, the authors identified KIT, LOOM, OKBC, OCML, and FLogic. For ontology markup languages, the authors identified SHOE, XOL, RDF and RDF Schema, OIL, DAML-OIL, and OWL. (Gómez-Pérez, Fernández-López, & Corcho, 2004) Any further discussion of the particular languages is beyond the scope of this thesis. Further research is needed to identify the pros and cons of each language and to determine which type of language and which language in particular might be used to develop ontologies that establish the basis for modeling profiles used to build system architectures. (Gómez-Pérez, Fernández-López, & Corcho, 2004)

4.6 Ontology Development Tools

Likewise, with ontology development tools, Gómez-Pérez, et al. provide a listing of tools commonly used in the ontology engineering field. The language-dependent tools identified are these tools are characterized by their tight association with an ontology language. These are the Ontolingua Server, OntoSaurus, WebOnto, and OilEd. The extensible language-independent tools are easily extensible and can easily be integrated with other applications. These are Protégé-2000, WebODE, OntoEdit, and KAON. The ontology merging tool identified is PROMPT. The ontology-based annotation tools are COHSE, MnM, OntoMat-Annotizer and OntoAnnotate, SHOE Knowledge Annotator, and UBOT AeroDAML. (Gómez-Pérez, Fernández-López, & Corcho, 2004)

4.7 Ontology Development Methodologies

Gómez-Pérez, et al. also provide a comprehensive listing of ontology development methodologies and the pros and cons of each, focussing on the following methodologies: the Cyc method, the Uschold and King's method, the Grüninger and Fox's methodology, the KACTUS approach, METHONTOLOGY, the SENSUS method, and the On-To-Knowledge

methodology. (Gómez-Pérez, Fernández-López, & Corcho, 2004) However, in this thesis the author would like to focus on a recent methodology outlined by Noy and McGuinness in (Noy & McGuinness, 2001) which provides a concise, step-by-step description of their recommended approach, summarized here.

Before beginning an ontology development effort, the designer needs to carefully consider the various aspects of the development process that will impact the final product. These considerations include deciding what the ontology is going to be used for, deciding how important is it for the ontology to be intuitive, extensible, maintainable, etc. The developer must also keep in mind that the ontology is a model of the real world, and the concepts in the ontology must reflect that reality. Afterall, the goal of building the ontology is not the ontology itself, but the best use of the ontology in a particular application or practice. (Noy & McGuinness, 2001)

The ontology engineer should not seek to identify all the possible information about the domain. There is often no value added to specializing (or generalizing) more than is needed for the intended application of the ontology. The farthest extent that an ontology development activity should go is at most one extra level each way (towards specialization and generalization.) Similarly, the ontology engineer should not try to capture all the possible properties of and distinctions among classes in the hierarchy. (Noy & McGuinness, 2001)

As far as methodologies for developing ontologies are concerned, there are many proposed methodologies, and they all have their pros and cons depending on the purpose of the ontology and the way in which the ontology will be used. This thesis describes a general approach to ontology development as suggested by Noy and McGuinness.

4.8 How Ontology Development Differs from Object-Oriented Design

Since ontologies are closely related to software products such as editors, readers, processors, interpreters, etc., software developers may be involved in some aspect of the ontology development or use. It is important to emphasize the difference between ontology development and the design of classes and objects in object-oriented programming. When developing object-oriented programming, a software developer normally gives primary consideration to the operational properties of a class, whereas with ontology development, the primary consideration is that of the structural properties of a class. As a result, the class structure in an ontology and the relations among the various classes of the ontology are different from the structure designed in an object-oriented program, for the same or similar domain of interest. (Noy & McGuinness, 2001)

4.9 Important Ontological Terms

The following terms defined by Noy and McGuinness and elsewhere are used in the development of frame-based ontologies and will be used throughout this discussion.

- Ontology – A formal explicit description of concepts (aka classes) in a domain of discourse
- Class – Represents a concept in a domain of discourse
- Superclass – Represents a concept that is more general than the subclass that is derived from it
- Subclass – Represents a concept that is more specific than the superclass from which it is derived
- Is-A Relation (aka Is-A-Kind-Of Relation) – A taxonomic relation in which a subclass is related to a superclass

- Direct Subclass – A subclass that is directly subordinate to its superclass
- Direct Superclass – A superclass that is directly superior to its subclass
- Indirect Subclass – A subclass that has an intervening class between it and a superior superclass
- Indirect Superclass – A superclass that has an intervening class between it and an inferior subclass
- Instance – Individual implementation of a concept (a class)
- Disjoint Classes – Two classes that cannot have any instances in common
- Slots (aka Roles) – Properties of each concept describing various features and attributes of the concept
- Slot Value – A slot value that is fixed for all instances and cannot be changed
- Range of a Slot – The classes of the Instances to which a slot is attached (that a slot describes)
- Domain of a Slot - The classes to which a slot is attached (that a slot describes)
- Facets (aka Role Restrictions) – Restrictions on slots, such as cardinality
- Inverse Relation – A situation in which the value of one slot depends on the value of another slot (example: “produces” versus “produced by”)
- Knowledge Base – An ontology together with a set of instances of classes

There exists a fine line between the point at which an ontology ends and a knowledge base begins. This can be equated with the idea of a database structural template that has no actual data loaded (no practical use other than a template) and that of a fully populated database that can be used to load, process, analyze, and report database results. In the case of an ontology, it

begins to serve practical use as a knowledge base when individual instances are defined with associated slot and facet information. (Noy & McGuinness, 2001)

4.10 Understanding Classes and Class Hierarchies

Before taking on the task of constructing an ontology, it is important to ensure that the authoring engineer has a good understanding of the concept of class and of class hierarchies. A hierarchy of classes is established by what is termed an “is-a” or “is-a-kind-of” relation among two classes. Formally, these relations are known as hyperonymy and hyponymy. Hyperonymy is the semantic relation between a more general word and a more specific word. Example: “tree” is a hyperonym of “oak.” Hyponymy is the semantic relation between a more specific word and a more general word. Example: “oak” is a hyponym of “tree.” This process is also known as subsumption. Example: “A canoe is a kind of boat.” Here, the Canoe class is a subclass of class Boat. So, in a hierarchy, the Boat class would exist at a higher level than the Canoe class. “Canoe” is subsumed by “Boat.”

4.10.1 Is-A Overloading in Subsumption

Guarino warns against overloading of the “is-a” mechanism for subsumption. Many ontology development efforts suffer from “is-a overloading” by using the subsumption relationship for many different kinds of associations. To help avoid some of these issues with the “Is-A” relationship, Boyce and Pahl suggest using an its inverse relation, which they called the ‘HasSubtype’ relation. The use of the ‘HasSubtype’ relationship makes it easier to avoid the pitfalls associated with the ‘Is-A’ relation, while remaining analogous to it. (Boyce & Pahl, 2007)

4.10.1.1 Confusion of Senses

This is a case in which a subclass is identified in an ontology as a child of two or more different superclasses. For example, “crane” is a kind of “bird”, while “crane” is also a kind of “lifting device.” While the two uses of “crane” are phonetically the same, it is not appropriate in an ontology to make them equivalent.

4.10.1.2 Reduction of Sense

In this usage, the superclass does not represent a sufficiently complete aspect of the child. For example, it would be inappropriate to place “computer” as a kind of “calculator.” While computers can certainly perform calculations, their primary functionality provide much more capability than mere calculation.

4.10.1.3 Overgeneralization

In this usage, the superclass is many levels above the child, such that, while true, the specialization of the child seems too far removed from the parent. For example, “computer” is a kind of “physical object.” While true, several levels of specialization have been skipped to go from “physical object” to “computer.”

4.10.1.4 Suspect Type-to-Role Link

This is a case where there exists confusion whether the child class is actually as a concept or a role. For example, “apple” is a kind of “fruit.” This is a proper subsumption of “apple” by “fruit.” But, were we to say the “apple” is a kind of “food,” then this suggests a role for the apple to play (as food) and not a classification of the apple (as a subclass.)

4.10.1.5 Confusion of Taxonomic Roles

In this case, ontological engineers tend to express all the unary properties of a certain class of entities in terms of superclasses to inherit from. For example, with a general list of quality attributes, such as accessibility, adaptability, flexibility, testability, etc., there is no distinction between the classes representing a major organizational role in the taxonomy, and those that simply express a particular property. (Guarino, Formal Ontology and Information Systems, 1998)

4.10.2 Concept Metaproperties

Guarino and Welty developed the OntoClean methodology to provide guidance on the kinds of ontological decisions that need to be made by an ontological engineer when developing the structure of an ontology based on rules of subsumption. OntoClean also describes approaches that can be taken to evaluate the decisions made when choosing a construct for representing a concept. Guarino and Welty identified several formal notions to define a set of metaproperties used to characterize relevant aspects of the intended meaning of the properties, classes, and relations that make up an ontology. These metaproperties are used to impose several constraints on the taxonomic structure of an ontology, which help in evaluating the structural choices made when constructing the ontology. (Guarino & Welty, 2002)

4.10.2.1 Essence

A property of an entity is essential to that entity (has essence) if the property must hold for the entity to be properly characterized. This is a stronger notion than one of permanence. Whether an entity has a property that is permanent or not, does not make that property essential to its characterization. For example, magnets have the property that they are magnetic. This a property which is essential to magnets when used in application such as electric motors.

However, if a common nail is magnetized and therefore takes on that property whether permanently or not, does not make it a property which is essential to its characterization as a nail.

4.10.2.2 Rigidity

Rigidity is a special form of essence that describes the strictness with which the property applies to all the instances of the class having the property. Guarino and Welty identify three types of rigidity. These definitions are restricted to meaningful properties (not necessarily true nor necessarily false), so trivial cases are excluded. (Guarino & Welty, 2002)

- Rigid – a property that is essential to all instances of a class. Example: all magnets are magnetic; therefore, magnetism is a rigid property of magnets.
- Non-Rigid (or Semi-Rigid) – a property that is not essential to all instances of a class. Example: common nails could possibly be magnetic; therefore, magnetism is a non-rigid property of common nails.
- Anti-Rigid – a property that is not essential to any instances of a class. Example: brass nails could never be magnetic; therefore, magnetism is an anti-rigid property of brass nails.

Rigidity is an important notion, every property in an ontology should be labeled as rigid, non-rigid, or anti-rigid. In addition to providing more information about what a property is intended to mean, these metaproperties impose constraints on the subsumption relation, which can be used to check the ontological consistency of taxonomic links. One of these constraints is that class with anti-rigid properties cannot subsume classes with rigid properties. (Guarino & Welty, 2002)

4.10.3 Identity and Unity

Identity and unity are the most important philosophical notions used in the OntoClean methodology. They are different notions, although strictly related and often confused with each other. (Guarino & Welty, 2002)

4.10.3.1 Identity

Identity refers to one of the most common decisions that must be made in ontological analysis, that of being able to recognize individual entities (concepts) in the world as being the same or different. This concerns circumstances in which something that is seen as one entity is actually two or more. Examining situations involving time provides a way of interpreting identity. Is a person the same person even if their appearance has changed over time? The problem can be evaluated also by considering the identity criteria at a single point in time. How can a time interval (from a start time to an end time) be related to a time duration (a measured length of time?) One approach is to make time interval a kind of (subclass of) time duration, since all time intervals could be seen as time durations. While this makes intuitive sense, since two durations of the same length are the same duration, two intervals occurring at the same time are the same, but two intervals occurring at different times, even if they are the same length, are different. Therefore, the two example intervals given would be different intervals, with the same duration. This creates a contradiction in which two time intervals that have the same duration, even if they occur at different times, are the same kind of (subsumed by) time duration, while two intervals that have the same duration, but do not occur at the same time, cannot be identical because they occur at two different times. This situation is brought on through common confusions of natural language and can be avoided by realizing that duration is a component

(property) of an interval, but it is not the interval itself. Therefore, the relationship cannot be modeled as a subclass. (Guarino & Welty, 2002)

4.10.3.2 Unity

Unity refers to property that identifies and describes all the parts that form an individual entity and the way that parts of an object are bound together, such that we know in general what is part of the object, what is not, and under what conditions the object is a whole. Unity can tell us a lot about the intended meaning of properties or classes based on whether class instances are parts or wholes. (Guarino & Welty, 2002)

4.10.3.3 Whole Entities

For some classes, all their instances are wholes, for others, none of their instances are wholes. For example, “water” cannot conveniently be identified as an isolated entity as can “ocean;” therefore, “water” is not commonly represented as a “whole” entity. On the other hand, “ocean” for which “Atlantic Ocean” can be identified as an instance, is an identifiable whole entity. This leads to another problem with subsumption in that “ocean” might be established as a subclass of “water,” since all oceans are made up of water. But this raises an inconsistency since instances of “water” are never wholes, yet instances of “ocean” always are. This presents a contradiction since oceans are not “kinds of” water; they are instead composed of water. This is a distinction that must be carefully thought through when constructing an ontology. (Guarino & Welty, 2002)

4.10.3.4 Part Entities

It is also important to analyze the conditions that must hold among the parts of an entity in order to consider it a whole. These conditions are called unity criteria. With suitable

metaproperties, these criteria distinguish the classes that carry a common unity criterion for all their instances (such as “ocean”) from those that do not (like “water”). (Guarino & Welty, 2002)

4.10.4 Subsumption

The subsumption relation that is the most commonly used and the most commonly misused structuring primitive used in constructing ontologies. Guarino and Welty have established a set of heuristics (below) which can be used to guide the ontological engineer in making the correct decision regarding subsumption of classes into an ontological hierarchy. Deciding whether one property should subsume another is one of the most important ontological decisions a modeler must make in building an ontology, and providing a formal foundation for evaluating these decisions has proved an important milestone in the practice of conceptual modeling. (Guarino & Welty, 2002)

4.10.4.1 Subsumption is not Instantiation

Subsumption is not the same as instantiation. The subsumption relationship is often used when instantiation was actually intended.

4.10.4.2 Subsumption is not a Meta Principle

“Rigidity” is considered a metaproperty in that rigidity is a property of properties, and not a property of objects in the world. It may be tempting to create a class called “rigid class” and have it subsume all classes that are rigid, such as Human. But, instances of “rigid class” are classes and these identity criteria cannot be applied to the instances of Human, so being rigid is a metaproperty of the class Human. Therefore, it is improper to establish Human as a subclass of “rigid class.”

4.10.4.3 Subsumption is not a Part Property

Confusion here is due to the fact that subclass is analogous to subset, and a subset of a set is a part of it. This confusion can be overcome when it is realized that the difference between the parts of a set and the parts of its members. For example, while “engine” is a part of a “car”, “engine” is not a kind of “car.”

4.10.4.4 Subsumption is not Disjunction

An often-used “work-around” to the part property problem is creating artificial classes representing different levels of decomposition, such as a class for “car parts” of which “engine” would be a subclass along with a restriction or axiom requiring that all the parts of cars be subclasses of “car parts.” This work-around amounts to using subsumption to create a disjunction of classes in order to accommodate a type restriction. Rigidity analysis can be used to expose the difficulty. There is no instance of a car part that is of necessity by itself always a car part. For example, the engine could be removed and used in another application such as in a power boat. Therefore, the car part class would be anti-rigid. The class engine is rigid, since an engine is and always will be characteristically an engine. This violates the rule that an anti-rigid class cannot subsume a rigid one. Since most modeling systems do not provide for disjunction, modelers believe they are justified in using this kind of work around.

4.10.4.5 Subsumption is not Polysemy

The most common misuse of subsumption in linguistics is to represent the multiple meanings (polysemy) of a term. This may have some linguistic motivations, but is incorrect from the ontological point of view. To see how this is incorrect, we can usefully employ identity or unity analysis. The term “book” can refer to a physical item that has weight, size, position in space, etc. “Book” can also refer to abstract notion of a work written by an author that has a title,

etc. Bound volumes are identified by their location in space/time, so that two bound volumes cannot occupy the same space at the same time. The abstract notion of book is independent of space and time, being identified by other criteria. No instance can meet both of these identity criteria; they belong to two different classes of entity, though there is a close relationship between them. No “book” is both a bound volume and an abstract entity.

4.10.4.6 Subsumption is not Constitution

Another common misuse of subsumption is to use it to represent the fact that one thing is constituted of another. It is important to understand that one class of entities may be constituted by entities in the other class, but it may not be subsumed by it. For example, a company might be constituted by a group of people, but a group of people are not (necessarily) a kind of company.

4.10.5 Choosing Classes and Class Names

When constructing ontology hierarchies in this fashion, it is important to avoid making the mistake of including both a singular and a plural version of the same concept in the hierarchy making the former a subclass of the latter. For example, avoid creating a class Boat that is a subclass of Boats. To avoid this issue, remain consistent throughout the hierarchy by using only either singular class names or plural class names.

It is also important to recall that classes represent concepts. No matter what name is chosen for the class, the concept remains the same. The name of a class in a hierarchy might change depending on the use of the ontology, but the concept, and its relation to other concepts (other classes) must remain the same. An example is the use of the same idea (concept) in different languages, or different applications, such as different services of the military. Do not create two classes for the same concept simply because two similar terms (synonyms) exist in the

common vocabulary of the domain. If it is important to identify synonyms, then include a list of synonyms in the ontology documentation.

Avoid creating class cycles in the ontology hierarchy in which one class (A) is a subclass of another class (B), while class B is also a subclass of class A. This is the same as saying that class A and class B are equivalent.

Sibling classes should be at the same level of generality compared to the parent class. For examples, classes Canoe, Skiff, Yacht, Schooner are all siblings of class Boat, and are all at the same level of generality. The exception to this rule is at the highest possible level of an ontology where the immediate children of the most general class may represent major divisions of the domain and therefore may not be similar concepts.

A superclass should not have only one subclass. Such a situation would indicate that further development of the superclass is warranted. To maintain a good structure of the hierarchy it is recommended that a given superclass have between two and a dozen direct subclasses. However, to best reflect the natural world it is better to not force a specific number of subclasses. If a large number of subclasses exist in the natural world, then the ontology should reflect that natural order. The rule of between two and a dozen subclasses is to be used when additional ontology development can be afforded without violating the natural order of the reality within the domain.

Guarino and Welty recommend beginning the class hierarchy construction with a “backbone taxonomy” consisting of all the rigid properties in the ontology, organized according to their subsumption relationships. It represents a view of the ontology showing all the most important properties—those that cover the entire universe of discourse. Every entity in the

backbone taxonomy must have identity criteria and must have a rigid property that describes those criteria. Backbone properties are the most important to analyze first—those that represent the invariant, essential aspects of the domain. Guarino and Welty identify three benefits to constructing a backbone taxonomy: 1) it jump starts to the integration process since every entity in the resulting ontology must instantiate at least one property in the backbone taxonomy, 2) it allows for the discovery of inconsistencies in the use of subsumption among the classes of the backbone taxonomy, 3) it can serve as the common backbone when comparing the rigid properties for two different ontologies that must be merged together, trying to establish a basic set of stable properties within the merged domain. (Guarino & Welty, 2002)

4.10.6 Whether to Introduce A New Class

When developing an ontology, it is not uncommon to come across a situation in which it is difficult to decide whether a concept should be established in the ontology as a new class or as a property value of an existing class. Noy and McGuinness identify a few rules of thumb for helping to determine which approach to take.

4.10.6.1 Subclasses Have Additional Properties

Subclasses of a class usually (1) have additional properties that the superclass does not have, or (2) have restrictions different from those of the superclass, or (3) participate in different relationships than the superclasses. So, only introduce a new class in the hierarchy when there is something that can be said about this class that cannot be said about the superclass. In practical terms, each subclass should either have new slots added to it, or have new slot values defined, or override some facets for the inherited slots.

4.10.6.2 Subclasses in Terminological Hierarchies

Sometimes it may be useful to create new classes even if they do not introduce any new properties. Classes in terminological hierarchies do not have to introduce new properties. This type of classification may be just a hierarchy of terms, without properties (or with the same set of properties). In that case, it is still useful to organize the terms in a hierarchy rather than a flat list because it will (1) allow easier exploration and navigation and (2) enable a user to choose easily a level of generality of the term that is appropriate for the situation.

4.10.6.3 Concepts which have Specific Distinction

Another reason to introduce new classes without any new properties is to model concepts among which domain experts commonly make a distinction even though it may have been decided not to model the distinction itself. Since ontologies are used to facilitate communication among domain experts and between domain experts and knowledge-based systems it would be good to reflect the expert's view of the domain in the ontology.

4.10.6.4 Importance of the Concept within the Domain

Whether to establish the concept as a class or a property value of an existing class depends on the scope of the domain and the task at hand. It depends on how important is the concept within the domain. If the concepts with different slot values become restrictions for different slots in other classes, then a new class should be created to emphasize the distinction. Otherwise, represent the distinction in a slot value. For example, if it appears that whether a type of boat is powered or not is becoming an important distinction in the ontology, then perhaps this requires that two subclasses be established under the Boat class; one for Powered Boats and one for Unpowered Boats.

4.10.6.5 Importance of a Distinction within the Domain

If a distinction is important in the domain and if the objects with different values for the distinction are viewed as different kinds of objects, then a new class should be created for the distinction. For example, if it is important to distinguish unpowered boats that can hold no more than two people, such as dinghies, canoes, and pirogues, then perhaps a new class should be established for these types of objects.

4.10.6.6 Consideration of Individual Instances

Considering the potential individual instances of a class may also be helpful in deciding whether or not to introduce a new class. A class to which an individual instance belongs should not change often.

4.11 A Simple Knowledge-Engineering Methodology

Noy and McGuinness offer a methodology for ontology development that addresses the general concerns that apply to most ontology development activities. They emphasize the importance of observing a few fundamental rules to ontology development:

- There is no correct way to model a domain. The approach taken depends strongly on the ultimate application of the ontology and any extensions that are anticipated to be added to the ontology through lessons learned as a result of use of the ontology.
- Ontology development is necessarily an iterative process. They start the process with a rough first pass, followed by practical application, and review by experts, after which subsequent passes are made to continually refine the ontology.
- Concepts in the ontology should be closely related to objects (nouns) and their relationships (verbs) as observed in the domain of interest.

4.11.1 Step 1 – Determine the Domain and Scope of the Ontology

It is understood that anyone constructing an ontology would already have determined the domain of interest for which the ontology is being built. The question should really be whether the domain is fully understood in relation to the intended use of the ontology. Noy and McGuinness suggest that the following topics be addressed to narrow and focus the scope of the ontology to be built. (Noy & McGuinness, 2001)

- Competency of the ontology – The ontology should be competent with regard to the issues that the user intends to address. In order to determine whether the ontology is competent enough, the kinds of questions that the target user would ask should be posed against the ontology to determine whether the ontology is sufficiently suitable to address those questions.
- Use of the ontology – No matter what the domain of interest, users in any given domain will have some particular interest in using the ontology to address some concern. Depending on that concern, an ontology in any given domain could be suitable to address the concerns or not. It is important to understand those concerns to ensure that the ontology addresses the user's issues.
- Queries the ontology is intended to address – With the understanding of who is going to use the ontology and for what purpose, it is now important to focus on the specific questions those users will ask of the ontology to ensure that the ontology will be capable of providing the answers to those questions.
- Maintenance of the ontology – No ontology will be able to achieve competency over the long term without maintenance, since the kinds of problems to address will likely change over time. It is important to anticipate the kinds of changes that

are likely to occur to ensure that the ontology will be designed in a way to allow for maintenance of the ontology that will preserve its competence.

4.11.2 Step 2 – Consider reusing existing ontologies

Depending on the objective, consider whether any previous development efforts would either serve as a starting point for the new ontology or would contribute in some way to its development. This might be a consideration existing sources can be refined or extended for a particular domain or task, or if the system for which the ontology is being built needs to interact with other applications that have already committed to particular ontologies. Most modern knowledge-representation systems have extensive import and export facilities, and therefore the formalism in which an ontology is expressed often does not matter since the task of translating an ontology from one formalism to another is usually not a difficult one. (Noy & McGuinness, 2001)

4.11.3 Step 3 – Enumerate important terms in the ontology

Since an ontology is first and foremost a domain vocabulary, it is important to identify and capture the terms that the user operating in that domain will be interested in formulating statements about or will be in need of an explanation. These terms will be used to formulate the concepts that become classes in the class hierarchy of the ontology. It is important at this point to consider not only the primary terms concepts that make up that hierarchy, but also related terms that help fill out the domain of discourse the user will expect to require in usage of the ontology. (Noy & McGuinness, 2001)

4.11.4 Step 4 – Define the class and the class hierarchy

The class hierarchy for a particular project within a particular domain will depend greatly on the ultimate application of the ontology. One class hierarchy in a given domain can appear

quite different from another in the same domain. There is no single correct class hierarchy for any given domain. The hierarchy depends on the possible uses of the ontology, the level of the detail that is necessary for the application, personal preferences, and sometimes requirements for compatibility with other models. (Noy & McGuinness, 2001) Noy and McGuinness identify three approaches to defining the class hierarchy.

Top-Down – In this approach, the engineer starts by first identifying and defining the most general concepts in the domain. This could be one single concept at the top of the domain hierarchy, or several concepts under the domain title. From this point the engineer identifies subsequent specialization of the principle concepts. By taking this approach, the engineer is creating subclasses at increasingly lower levels of the hierarchy. In the process, the engineer is identifying “is-a” type relations between levels of the hierarchy.

Bottom-up – This is the antithesis of the top-down approach in which the engineer starts by first defining the most specific classes, those being the leaves of the hierarchy, and develops grouping of the leaf-level concepts into higher-level groupings. The higher-level groupings would be generalizations of the more specific lower-level concepts. This process is repeated for each level until no higher generalizations can be identified, or until sufficient leaf-level identification of concepts has been accomplished.

Combination – This approach is a combination of the top-down and bottom-up approaches. Here, the engineer defines the more salient concepts first – those that represent the mid-level concepts that best represent the more visible and identifiable concepts of the domain. These concepts are then generalized (going higher into the hierarchy) and specialized (going lower into the hierarchy) until the hierarchy is populated to the degree desired.

It is important to consider the structure of the ontology as it can be difficult to navigate a poorly structured hierarchy. Ontologies that are either extremely nested with many extraneous classes, or very flat with too few classes and too much information encoded in slots, are very difficult to navigate. Finding the appropriate balance though is not easy.

Noy and McGuinness point out that the selection of which approach to take when constructing an ontology depends on the personal perspective that the ontology engineer has of the domain. If the engineer has a systematic, organizational view of the domain, then it may make best sense to use the top-down approach. If the engineer normally operates at the low-level of detail, is able to identify the majority of the leaf-level concepts, and is not fully cognizant of how these lower-level elements roll-up into higher-level organization, then it may be best to start at the bottom of the hierarchy. Most engineers are more aware of the mid-level concepts which tend to be the more descriptive concepts in the domain. In this case, it is best to start where most of the knowledge and experience exists, and work up/down from that point developing the upper and lower levels of the hierarchy. (Noy & McGuinness, 2001)

Whichever approach is taken, the ontology engineer starts by defining the classes at the chosen level starting with the list that was created in Step 3. It's best to start by select the terms from the list that describe objects having independent existence rather than terms that are descriptive of other objects (whose identification is tied to other objects and are therefore dependent on those other objects for meaning). The chosen terms will then be identified as classes in the ontology. These first selected terms are key elements of the ontology and will serve as anchors in the class hierarchy from which others elements will be supported. Once this initial identification of related concepts is established, the classes are then organized into a hierarchical taxonomy. This is accomplished by identifying super-class/sub-class relationships. One way of

accomplishing this task is by posing the following question: if by being an instance of one class, will the object necessarily also be an instance of some other class (its superclass)? This would be so for a valid super-class/sub-class relationship because if a class A is a superclass of class B, then every instance of B is also an instance of A. In other words, class B represents a concept that is a “kind of” A. In such a case, an instance of class B is also, by definition, an instance of class A. (Noy & McGuinness, 2001)

4.11.5 Step 5 – Define the Properties of a Class (Slots)

With classes identified and located in the hierarchical structure, it is necessary to then elaborate on the internal structure of the concepts represented by the class. This is done by describing the class properties, known in ontological engineering as “slots”. This step can be performed by either considering what are the properties of a class individually, or considering which are domain properties and then assigning each of those properties to a particular class.

There are several types of object properties that can become slots in an ontology:

- Intrinsic properties – those which are natural or essential properties of the class
- Extrinsic properties – those which are not directly attributable to the class, but are nonetheless closely related to it
- Part properties – if the class represents a structured object, then its parts are defined as part properties; these can be both actual physical parts as well as abstract (non-physical) parts
- Relationship properties – These are the relationships between individual members of the class and other items

In order to promote the hierarchical concept of inheritance, a slot should be attached at the most general class that can have that property. Thus, when a subclass for the superclass is identified, it inherits all the properties of the superclass. (Noy & McGuinness, 2001)

When establishing relations between classes, avoid establishing relations among strikingly different branches of the ontology simply because they make literal sense. While these relations may be literally correct, they cause confusion within the ontology and the user of the ontology may not understand the purpose for their existence.

Avoid storing the information for inverse slots “in both directions”. This constitutes redundant information. An application using the knowledge base can always infer the value for the inverse relation. Decide on which direction to keep. If this is a pattern throughout the ontology, and if appropriate, choose one direction to describe the inverse relationship and maintain that direction throughout the ontology.

4.11.6 Step 6 – Define the Facets of the Slots

With slots defined for the classes, it is time to identify the slot value features, known in ontological engineering as facets. These include such items as the type of the values that the slot can assume, the allowed values, the number of the values (the cardinality), and other features of the values the slot can take on.

4.11.7 Slot Value Type

This describes the type of the values that can occupy a slot. Examples of the most common value types are: string, number (integer and float), Boolean, enumerated, and instance-type. Instance-type slots allow the definition of relationships between individuals. That is, which

other class instances can have a relation with this class instance. Slots with value type Instance must also define a list of allowed classes from which the instances can come.

4.11.8 Slot Cardinality

Slot cardinality defines how many values a slot can have. Some systems distinguish only between single cardinality (allowing at most one value) and multiple cardinality (allowing any number of values). Some systems allow specification of a minimum and maximum cardinality to describe the number of slot values more precisely. Minimum cardinality of N means that a slot must have at least N values. Maximum cardinality of M means that a slot can have at most M values. Sometimes it may be useful to set the maximum cardinality to 0. This setting would indicate that the slot cannot have any values for a particular subclass.

4.11.9 Slot Domain and Range

The domain of a slot is the class to which a slot is attached or the class with the property that the slot describes. The range of a slot identifies the allowed classes for slots of type Instance. In the phrase “Wineries produce wines”, “produce” is the slot, “wineries” is the domain, and “wine” is the range.

Noy and McGuinness identify several basic rules for determining a domain and a range of a slot.

- When defining a domain or a range for a slot, find the most general classes or class that can be respectively the domain or the range for the slots.
- On the other hand, do not define a domain and range that is overly general.
- All the classes in the domain of a slot should be described by the slot.

- Instances of all the classes in the range of a slot should be potential fillers for the slot.
- Avoid choosing an overly general class for the range, but rather choose a class that will cover all fillers. For example, avoid choosing “THING” for the range of a slot. (THING is generally accepted as the uppermost possible element in any ontology.) Instead of listing all possible wines that a winery can produce, simply choose “wine” for the range. “THING” would be too general.
- More specifically, if a list of classes defining a range or a domain of a slot includes a class and its subclass, remove the subclass.
- If a list of classes defining a range or a domain of a slot contains all subclasses of a class A, but not the class A itself, the range or domain should contain only the class A and not the subclasses.
- If a list of classes defining a range or a domain of a slot contains all but a few subclasses of a class A, consider if the class A would make a more appropriate range definition.

4.11.10 Step 7 – Create Instances

The last step is creating individual instances of the classes defined in the hierarchy. To do so, perform the following activities: 1) choose a class, 2) create an individual instance of that class, and 3) fill in the slot values for that instance.

At times it can become difficult to decide whether a particular concept is a class in an ontology or an individual instance. The answer often depends on what the potential applications of the ontology are. Noy and McGuinness suggest taking the approach of deciding what is the lowest level of granularity in the representation. This is determined by the intended application

of the ontology. Ask what are the most specific items that are going to be represented in the knowledge base. The most specific concepts that constitute answers to competency questions are very good candidates for individual instances in the knowledge base. (Noy & McGuinness, 2001)

For architectural development activities, the instances are more reasonably developed in the actual architecture of the system and not in the ontology. In order for ontologies to remain generally applicable to multiple projects within a domain, the identification of instances of concepts should be left to the actual architectural description of the designed solution.

4.12 Ontology Maintenance

It is important to maintain the ontology over time. Concepts in a given domain can change over time. Depending on the use of the ontology it may be necessary to periodically review the ontology to ensure that it is up-to-date with the current vocabulary usage within the given domain.

5 Bridging the Gap Between Ontologies and Modeling Profiles

Several technologies have been under development in the last couple of decades that have accelerated the potential for bridging the gap for the transfer of knowledge between ontologies and system modeling profiles, and thus into the system model architectures themselves. These technologies are discussed in the following sections.

5.1 Modeling

Humans have been using models to describe the world around them for as long as the need to convey information from one to another has existed. A model is simply a conceptual representation of some entity in the real world, whether that entity actually exists at the point in time that it is modeled, or simply exists as a vision of something that has existed in the past or could exist in the future. By definition, a model is “a description of (part of) a system written in a well-defined language. A well-defined language is a language with well-defined form (syntax) and meaning (semantics), which is suitable for automated interpretation by a computer ”.

(Kleppe, Warmer, & Bast, 2003) In the realm of systems engineering, models are representations of systems to be built to provide some capability that satisfies the needs of a stakeholder. This arrangement of a modeler expressing an idea that a stakeholder would interpret is illustrated in Figure 3. (Overbeek, 2006)

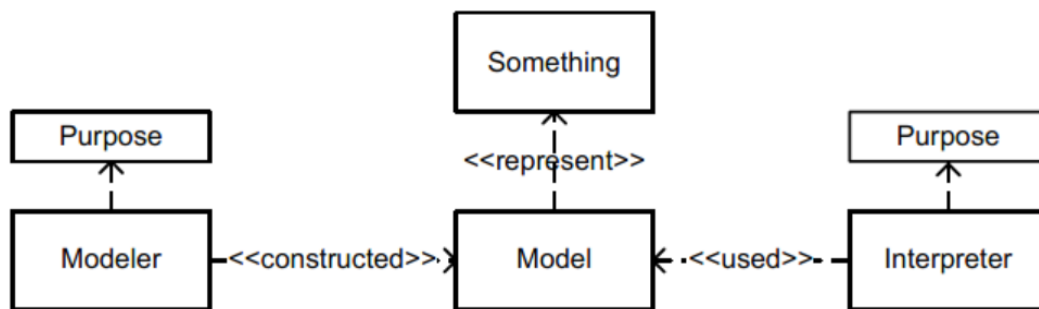


Figure 3: Relation Among Modeler-Model-Interpreter (Overbeek, 2006)

For our purposes, we are addressing system and software models as capturing descriptions of system elements, their characteristics, behaviors, interfaces, etc. These modeling elements are captured in a form that can be interpreted by stakeholders using industry standards that define the syntax and semantics of the languages used to model the system. The idea of a modeling language as being the means by which the expression of the model is captured is illustrated in Figure 4. (Overbeek, 2006)

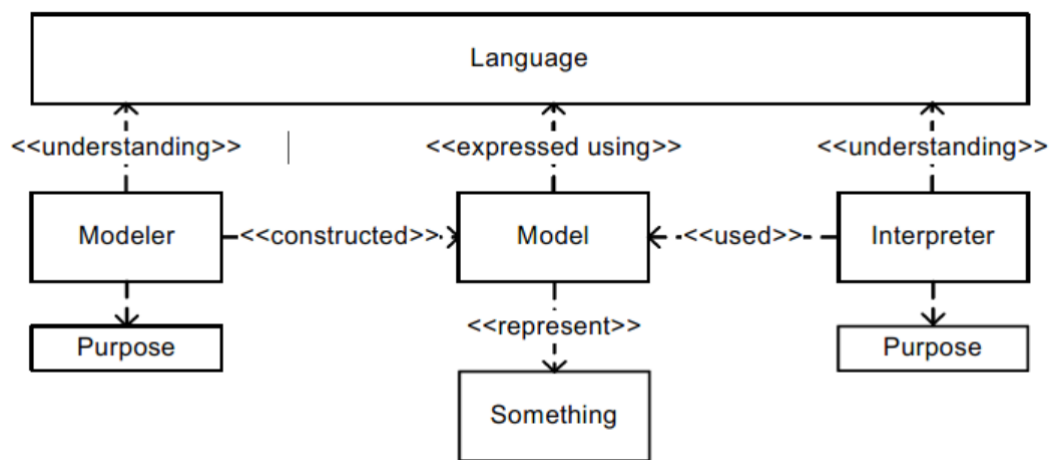


Figure 4: Representation of Figure 3 in Modeling Language (Overbeek, 2006)

A modeling language includes the syntax (the part of the language that defines the notation) and the semantics (the part of the language that describes the meaning of the notation). The syntax is further divided into a concrete syntax, which defines the physical notation of the language observed by the user, and the abstract syntax, which describes the concepts in the language, their characteristics, and interrelationships. The semantics describe the meaning of the language in terms of concepts that are well-defined and understood. These concepts are contained in the semantic domain which envelopes the whole of the concepts included in the selected language used for modeling. The language used for describing models is called a modeling language. (Overbeek, 2006)

The concepts of concrete syntax, abstract syntax, semantic domain and the mapping of the elements to each other are represented in Figure 5, which describes a model of a language used for modeling. This is also known as a language metamodel. (Overbeek, 2006)

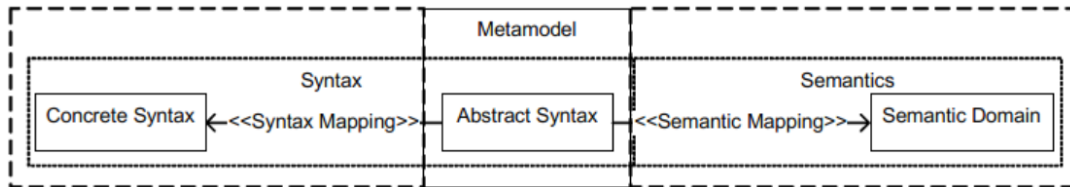


Figure 5: Syntax and Semantics of Metamodel (Overbeek, 2006)

Recent advances in the practice of system and software modeling have led to the popularity of object-oriented (OO) modeling, in which modeling elements are treated as objects in an OO modeling paradigm. The popularity of OO modeling has led to the use of graphical languages to describe the systems being modeled. The modeling solution for textual modeling languages is well-developed in the form of the Backus Naur Form (BNF) notation. (Fuentes-Fernández & Vallecillo-Moreno, 2004) However, for graphical modeling, as is commonly used in software and systems engineering, a different mechanism is needed. The desire to use graphical languages in modeling gave rise to the need to formalize the graphical modeling process. As a result, the Object Management Group (OMG) has developed the Meta-Object Facility (MOF) as one solution which has now become an accepted standard in the industry.

5.2 Meta-Object Facility

As suggested in the previous section a metamodel is a model of a model. In the case of a modeling language, the language metamodel is a model that describes a language used for modeling. Taking this concept a step further, a meta-metamodel is a specialized metamodel that describes other metamodels. (Overbeek, 2006)

The purpose of the MOF is to create, store, and manipulate object schemas into the form of a meta metamodel used for defining metamodels, like the Unified Modeling Language (UML). The OMG has established a four-layer construct known as the OMG metamodel hierarchy, in which the MOF is designed to occupy the top layer of the structure, as illustrated in Figure 6. All the layers in this structure (M3 down to M0) employ a strict instance-of relationship with layer above, down to the M0 layer. (Overbeek, 2006)

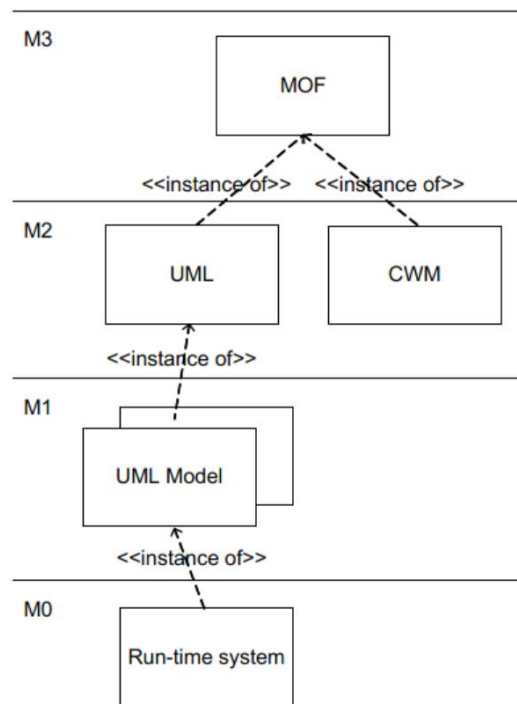


Figure 6: Meta-Object Facility Metamodel Hierarchy (Overbeek, 2006)

The four layers of the metamodel hierarchy are, namely:

M3 – Meta-metamodel layer – This layer represents the MOF, which is a language specification layer. The purpose of this layer is to specify the language of the metamodel at the M2 layer. This layer contains only one metamodel, which is the MOF. The MOF is what is known as a recursive layer. In addition to specifying the language at the next lower level, a

recursive layer defines a representation of its own behavior and structure, so no additional language is needed at a higher layer to describe the MOF. (Overbeek, 2006)

M2 – Metamodel layer – This layer is a language specification layer that specifies the languages used to define those models. It is also a metamodel layer in that it is used to specify models. The metamodels in this layer are more specific as compared with the meta-metamodel layer. This layer can contain multiple metamodels. (Overbeek, 2006) It is within this layer that modeling languages such as the Unified Modeling Language (UML) and the Systems Modeling Language (SysML) are specified.

M1 – Model layer – The model layer is a specification layer available to the modeler to develop models of the object of interest. This layer will contain a concrete definition of the data created by the modeler to represent the system being modeled. This is the layer in which the modeler uses modeling tools to create an architectural description of some system of interest to stakeholders, for example. (Overbeek, 2006)

M0 – Run-time layer – In software engineering terms, the run-time layer contains the objects instantiated out of the model which will be executed during run-time, and thus represent the final products of the software engineering effort. (Overbeek, 2006) In systems engineering terms, this might be termed the Real-world layer, in which the modeled systems are actually produced and delivered to customers, and used in the real world to deliver value to stakeholders.

5.3 UML Profile Extension Mechanism

The Unified Modeling Language (UML) was established as a standardized modeling language by the Object Management Group (OMG) in the mid-1990s, and has since enjoyed widespread acceptance and usage. The UML is a general-purpose graphical and visual modeling

language used initially to specify the design of software engineering projects and products, irrespective of the domain of the problem solution. The drawback of the general-purpose nature of the UML is that there exists a lack of features that can be directly used to represent specific characteristics of the domain of the problem space. The OMG accommodates this need for additional features through two available mechanisms. (Overbeek, 2006)

The first of these involves using the MOF to create a new meta-model at the M2 layer to describe a modeling language that provides the domain-specific features not found in a general-purpose language like the UML. By taking this approach, the desired modeling characteristics of the domain are defined into the syntax and semantics of the elements of the new language. However, the result is a modeling language that is quite limited to applications within the domain of discourse covered by the language syntax and semantics. Furthermore, the new language will not observe UML semantics, and therefore the language will not be compatible with commercial UML tools for drawing diagrams, generating code, etc. (Overbeek, 2006)

The second approach is that of the usage of a language extension profiling mechanism. With language profiles, some elements of the language are specialized by imposing constraints which more closely represent the characteristics of the elements in the domain of interest. However, in order to retain the general-purpose nature of the UML, the profiling extension mechanism continues to conform with the UML metamodel by leaving the original semantics of the UML elements unchanged. (Overbeek, 2006)

The advantages of using the UML profile extension mechanism are 1) extend the modeling terminology to cover domain-specific terminology, 2) extend the syntax of the modeling language to include modeling concepts specific to the domain, 3) display a customized set of graphical symbology more appropriate to the target application domain, 4) add semantics

that were left unspecified in the metamodel that defined the UML, 5) add semantics that do not exist in the metamodel that defined the UML, 6) add constraints to the way the metamodel can be used. (Overbeek, 2006) However, above all, the most important advantage to be gained by the profile extension mechanism is that a domain profile can be developed that can then in turn be reused on other projects within the same domain in order to establish a consistent domain-specific modeling approach within an organization and across an industry.

5.4 Model Driven Architecture

Model Driven Architecture (MDA) was conceived out of the need to separate the elements of software engineering activities that were driven by the desired functionality of the system from those that were affected by the constraints of the computing hardware of the system. (Truyen, 2006) This approach allows us to focus on model definition, leaving implementation details until the end. Doing so makes the models more portable, more adaptable to new technologies, and more interoperable with other systems, regardless of the technology they use. (Fuentes-Fernández & Vallecillo-Moreno, 2004)

The MDA specification identifies three distinct viewpoints intended to emphasize this separation of concerns. These are 1) the computation independent viewpoint, 2) the platform independent viewpoint, and 3) the platform specific viewpoint. The computation independent viewpoint considers the problem seeking a solution from the stakeholder perspective in which the method of achieving the solution (the way the problem is solved) is independent from the problem statement (what the problem is). The platform independent viewpoint focuses on the functional and physical characteristics of the solution that allow it to meet its operational objectives independent of how the solution will actually be implemented. The platform dependent viewpoint provides the detailed information that describes how the platform

independent viewpoint will be implemented in a specific hardware configuration. The platform is a set of software and hardware subsystems and technologies that provide a coherent set of functionalities to provide the complete deliverable functionality which serves as a solution to address the stakeholders' problem space. Examples of platforms include operating systems, programming languages, databases, user interfaces, middleware solutions, processors, interfaces, etc. that service the platform independent elements of the MDA. (Truyen, 2006)

In order to realize these viewpoints, MDA defines three models of a system that corresponding to the three MDA viewpoints: 1) Computation Independent Model (CIM), 2) the Platform Independent Model (PIM), and 3) the Platform Specific Model (PSM). (Truyen, 2006)

The CIM is also referred to as the business or domain model since it uses a vocabulary that is familiar to subject matter experts (SMEs) operating in the domain of discourse. It describes the operational functionality and performance that the system is expected to deliver in order to meet the stakeholders' objectives. In the process of doing so, the CIM hides technology related details to maintain independence from the system solution description. This independence is critical so that the specification of the desires of the stakeholder are not influenced by any particular technology solution. However, as the development proceeds, the CIM requirements should be made traceable to the PIM and PSM constructs that implement them. (Truyen, 2006)

The independence of the PIM is an intentional characteristic in order for the PIM to be developed such that it can be easily mapped to one or more platforms without impacting the PIM. The mapping from PIM to PSM is then performed by defining a set of services in a way that abstracts out the technical details of the mapping. Other models on the PSM side of the mapping then realize these services in a manner specific to the platform on which the PIM will be implemented. (Truyen, 2006) UML Profiles can be used to describe the platform model and

the transformation rules between models. Doing so guarantees that the transformed models will be consistent with the UML. (Fuentes-Fernández & Vallecillo-Moreno, 2004) Note, that while this application of UML Profiles is valuable, it is not the use of profiles that is being sought after as a solution to the problem of incorporating ontologies into the systems modeling process.

The PSM then combines the specifications in the PIM with the details that describe how the PSM is implemented on a particular platform. (Truyen, 2006)

Figure 7 (Alhir, 2003) illustrates the foundational concepts that constitutes the MDA. In this figure, the Requirements Gathering process produces the Requirements Model (CIM) that feeds the requirements specifications to the Analysis process. The Analysis process is actually the beginning of the conceptualization of a solution to the problem through the architectural design tasks that are included in the Analysis process in this figure. During the Analysis process, the architecture that describes the platform independent functionality and performance of the system is defined to produce the architectural description (PIM). The Analysis process is performed as part of the architectural development by analyzing whether the architecture developed will provide the needed functionality and performance to satisfy the stakeholder requirements. The architectural description (PIM) is then passed to the Design process where the detailed implementation plan is developed. From this point, the Design process produces the bill of materials (PSM) which is then fed to the Implementation process. It is in the Implementation process where the actual, realized product is constructed (System).

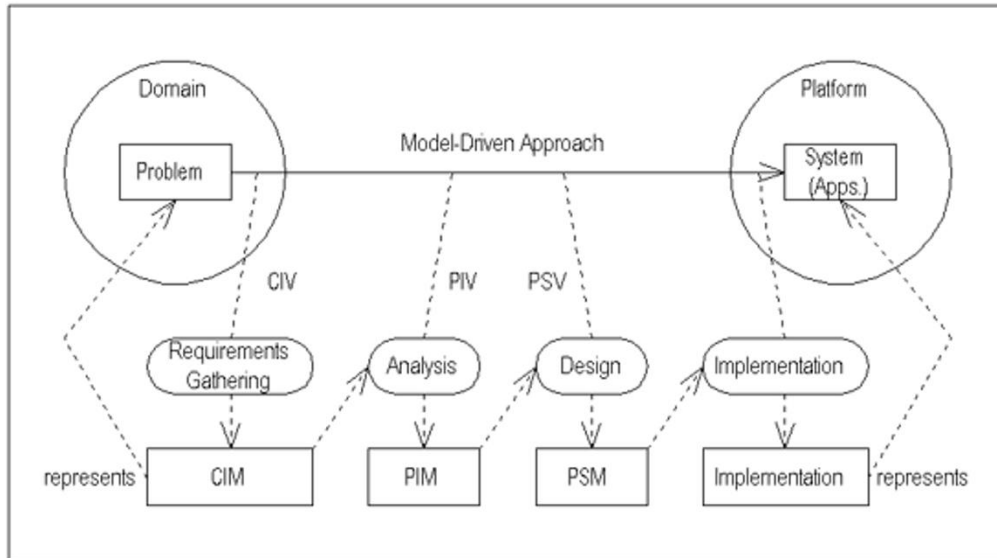


Figure 7: SE Lifecycle Phases Mapped to MDA (Alhir, 2003)

The most important advantage of this approach, and the main purpose for developing this architectural approach, is so that software engineers are then able to define transformations that automatically convert the PIM to a PSM. The PIM is supplied as an input to this process, along with a description of the PSM to be used to implement the system. A set of transformation rules are then used to implement the system in the most automated way possible. (Truyen, 2006)

5.5 Ontology Definition Metamodel

The Semantic Web represents the next logical step beyond the World Wide Web, and is intended to enable machine-understandable data to be shared across the Net. Ontologies will give the Semantic Web machine-understandable meaning to its data. These interoperable ontologies will facilitate Web with the ability to “know” something. The Semantic Web architecture defines three levels that incrementally introduce expressive primitives: *metadata* layer, *schema* layer and *logical* layer. The Semantic Web ontology languages that support this architecture are depicted in Figure 8. (Djuric, Gašević, Devedžic, & Damjanovic, 2004)

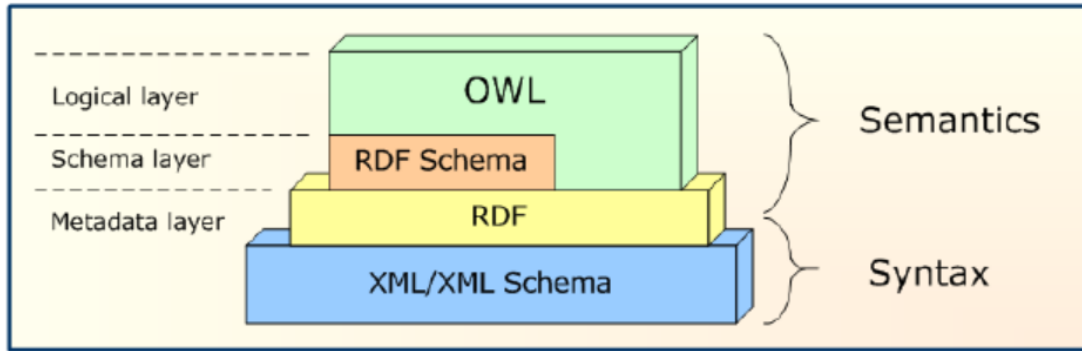


Figure 8: Semantic Web Architecture (Djuric, Gašević, Devedžic, & Damjanovic, 2004)

The Resource Description Framework (RDF) and the RDF Schema are used as general languages for the description of metadata on the Web. OWL has been developed as a vocabulary extension of RDF. OWL is a semantic markup language for publishing and sharing ontologies on the WWW. OWL is designed to advance beyond simply presenting information to humans. It is designed to provide the ability for applications to process information content. OWL facilitates greater machine interpretability of Web content by providing additional vocabulary along with a formal semantics. This capability goes beyond that which is supported by XML, RDF and RDFS alone. To achieve common data interoperability in applications, XML is the preferred choice as it supports syntax, while semantics is provided by RDF, RDF Schema, and mainly by OWL. Through the use of OWL, developers can achieve unconstrained representation of the Web knowledge and, at the same time, support calculations and reasoning. However, AI techniques needed for ontology creation are relatively unknown to the wider software engineering population. In order to overcome this gap, several proposals have been offered that suggest using UML in ontology development. The drawback of some of these proposals is that UML does not by itself satisfy the needs for representation of ontological concepts borrowed from description logics, and included in Semantic Web ontology languages. (Djuric, Gašević, Devedžic, & Damjanovic, 2004)

Development activities have been underway, focused on to move ontology development techniques toward taking advantage of the metamodeling approach offered by the OMG's Model Driven Architecture (MDA) technology. Toward this end, several metamodels and UML profiles have been developed which are based on ontology representation languages such as RDF(S), DAML+OIL, etc. However, none of these solutions use OWL. As a result, the Object Management Group (OMG) has established an initiative aimed at defining a suitable language for modeling Semantic Web ontology languages in the context of MDA. This initiative is known as the Ontology Definition Metamodel (ODM). This initiative has been established in large part due to the recognition that the Semantic Web and its XML-based languages are the main enablers of future Web development. (Djuric, Gašević, Devedžić, & Damjanovic, 2004)

Djuric, et al. propose to take advantage of the OMG's Model Driven Architecture (MDA) concept to create a language that is defined in a similar way that the UML is defined, using metamodeling. Accordingly, they have developed a metamodel for an ontology modeling language which is defined using the OMG Meta-Object Facility (MOF), and is based on the Web Ontology Language (OWL). To facilitate use by the wider engineering community, they developed a profile that supports ontology design, called the Ontology UML Profile (OUP). This profile is a standard extension of UML, and is also based on MOF. To provide a usable ontology development environment, several data mappings are required. Three two-way mappings are required: 1) between OWL and ODM, 2) between ODM and the OUP, and 3) from the OUP to other UML profiles. These mappings are impacted by the fact that they involve traversing ontology languages based on different platforms (i.e. Semantic Web and MDA), and therefore several tools are required to provide those mappings. One approach to this issue is to apply the concept of technical spaces. The authors implemented an XSLT that transforms OUP ontologies

into OWL in order to provide suitable tool support. The needed transformations are illustrated in Figure 8. (Djuric, Gašević, Devedžic, & Damjanovic, 2004)

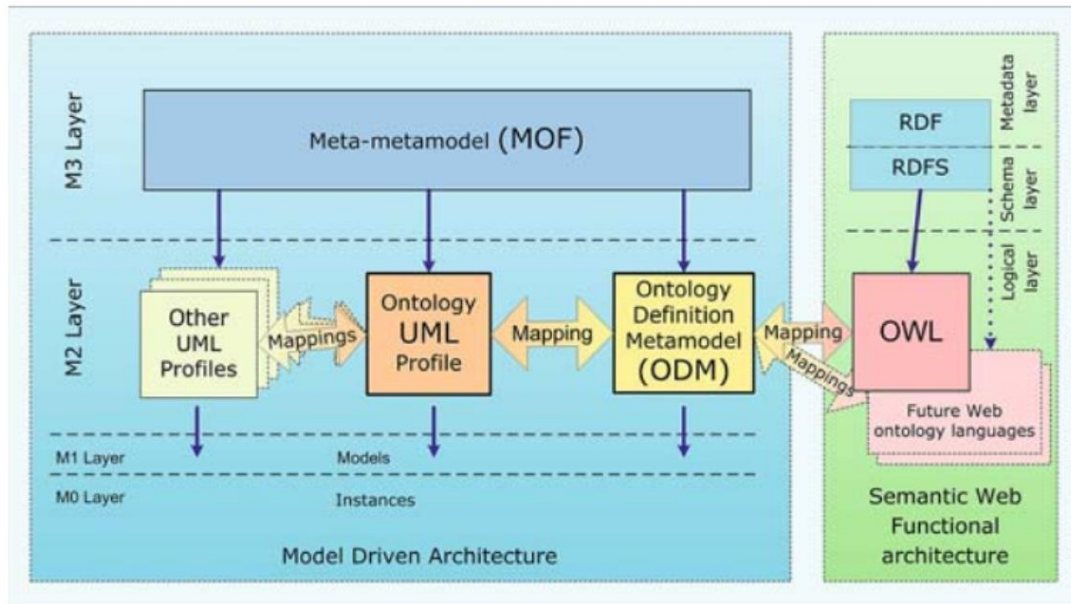


Figure 9: Ontology Modeling in the Context of MDA and the Semantic Web (Djuric, Gašević, Devedžic, & Damjanovic, 2004)

In the approach proposed by Djuric, et al., ODM encloses common ontology concepts by using OWL, since it is the result of the evolution of existing ontology representation languages. The position of OWL at the Logical layer of the Semantic Web architecture, on top of RDF Schema (Schema layer) allows it to make use of graphical modeling capabilities of the UML. Thus, ODM should have a corresponding UML Profile to enable the graphical editing of ontologies using UML diagrams. The required two-way transformations between UML and ODM can be accomplished using XSLT, since both models are serialized in the XMI format. Another pair of XSLTs should be provided for the two-way mapping between ODM and OWL since OWL also has representation in the XML format. Additional transformations can be added to support the use of ontologies in the design of other domains and vice versa. This would allow for the mapping of the Ontology UML Profile into other, technology-specific UML Profiles.

6 Current State of the Practice

As described in Section 3.4.7, NASA JPL has reported on a program that the institution has been involved with to transform domain ontologies into system modeling profiles for use in systems architecture development. This section delves deeper into the NASA JPL activities in this area as reported during the period 2010-2019 to examine more closely the approach taken to provide this capability. Towards the end of this period, a consortium of interests launched an initiative known as the Semantics Technologies for Systems Engineering (ST4SE). This initiative is taking the work of NASA JPL, as well as the results of other research activities into the application of ontologies to solve systems engineering problems, to advance the state of the art in this area. The ST4SE seeks to “promote and champion the development and utilization of ontologies and semantic technologies to support system engineering practice, education, and research.” (Jenkins, 2018) The author of this thesis intends to follow the activities of the ST4SE group to keep abreast of advances made in this area of research.

6.1 NASA JPL Integrated Model-Centric Engineering Initiative

The National Aeronautics and Space Administration (NASA) Jet Propulsion Laboratory (JPL) is a national research facility that designs, develops robotic sensors, spacecraft, and surface vehicles to perform Earth and interplanetary science missions. (About JPL, n.d.) JPL launched the Integrated Model-Centric Engineering (IMCE) initiative for the purpose of advancing enterprises practices from “the current document-centric engineering practices to one in which structural, behavioral, physics and simulation-based models representing the technical designs are integrated and evolve throughout the life-cycle, supporting trade studies, design verification and system verification and validation.” The objective of the IMCE initiative is to “advance engineering practice to a state in which descriptive and analytical models representing technical

designs and relating them to stakeholder concerns are developed and integrated throughout the mission life cycle, from early concept through operations.” (Bayer, et al., 2011)

6.2 NASA JPL View of Systems Engineering Landscape in the 2010 Timeframe

The organization was experiencing the same issues with traditional system engineering practices as has been reported in this thesis as being experienced by other organizations, those being: managing growing system complexity, dealing with emergent system behavior, and inability to fully test systems using traditional test methods, among others. The IMCE identified four specific challenges to address: 1) JPL products were being designed around “off-the-shelf” components, rather than through a mission-oriented architectural development activity, 2) there was no effective mechanism to transfer knowledge from one project to the next, 3) the programmatic activities and technical activities were managed separately resulting in poor decision-making and increasing risk, 4) It is of value to examine these issues more closely as they are more closely aligned with the architecture modeling issues this thesis is intending to address. (Bayer, et al., 2011)

6.2.1 System Design Emerges from the Pieces

The issues raised here have to do with the tendency of an organization to pull system components “off-the-shelf”, or in this case, to use equipment designed by laboratories to deliver a particular capability, irrespective of the ability of those components to properly integrate into the aggregate system. The IMCE identified the following challenges. (Bayer, et al., 2011)

- The role of the system architect is not an influential element of the engineering process.
- The architecture is disproportionately driven by the design process of functional decomposition.

- The management of ad-hoc, point-to-point interfaces becomes overwhelming.
- Extensive decomposition of models into simpler submodels can result in conflicting conclusions from the submodels.
- The tendency to delineate fault protection from nominal functionality results in systems that are brittle, difficult to operate, and less reliable.
- The abandonment of architectural principles to solve technical problems of the day, whether those principles are spelled out in policy or not, make the system brittle, difficult to operate, and increases risk.
- System designs are spread across many disconnected architectural description artifacts requiring many meetings, emails, and conversations to resolve design changes over months of effort.
- Weakly architected systems results in aspects of the design itself scattered over system elements resulting in the execution of functionality with little high-level oversight and coordination.
- The physics-based models of subsystem performance are not connected to each other, resulting in “stove-piped” analysis (performing analysis separately for each subsystem), and manually integrating the results. This extends the time necessary to conduct an analysis or trade study and hides significant system-level interactions which might later be exposed during testing, or during operations.
- Insufficient consideration for verification and validation during requirements development can render aspects of the design untestable.
- The primary mission objective requirements are not adequately coordinated with the practical infrastructure system requirements resulting in conflicts between

basic system operations and fulfilling the primary mission objectives. A side result is that opportunities to reduce risk/cost/schedule or even enhance performance are missed because of the disconnect between the two.

- Some desired system behaviors are difficult to express in textual specification format, resulting in miscommunication between systems engineers and software developers, and incorrect system behavior.

6.2.2 Knowledge and Investment are Lost Across Phases

There is no effective mechanism to transfer knowledge from one project to the next or between phases within the same project. (Bayer, et al., 2011)

- The system modeling efforts performed during the conceptual phase are abandoned when transitioning to the implementation phase. The new modeling work is essentially started from scratch using non-model-based artifacts to kick-start the activity.
- Inadequate configuration management (CM) during one phase results in incomplete, or non-existent reuse of artifacts from one phase to another.
- Essential attributes of the system design, such as architectural principles, assumptions, rationale, and explanatory narrative are not properly captured or made available to engineers to take advantage of.
- Because the system design is so poorly captured in available artifacts, training new team members requires locating key documents, and having lengthy conversations with them in order to bring new personnel up to speed on the system design, resulting in new engineers continuing to discover key attributes of the design over a very extended period of time.

6.2.3 Technical and Programmatic are Poorly Coupled

This topic area addresses the fact that programmatic activities and technical activities are managed separately resulting in poor decision-making and increasing risk. (Bayer, et al., 2011)

- Very little coupling exists between the technical aspects of the system design and the programmatic aspects resulting in the inability to correctly determine the cost, schedule, scope, and risk implications of a given set of requirements, science objectives, components, and functions. This is due to the difficulty in transferring information between disciplines and between the various tool types used.
- Systems engineers are often insufficiently knowledgeable about the programmatic realities of a project and the impact of engineering decisions on programmatics. The tools typically used by systems engineers do not support an integrated view that includes consideration of programmatics. Trade studies seldom fully incorporate programmatic considerations.

6.2.4 System Design Re-Use is Lacking

The lack of facilities to document and integrate the broad experience and knowledge of engineers across a project makes it difficult, if not impossible, to train new systems engineers who will need to absorb this broad knowledge quickly and deeply, and to make this knowledge available as a legacy to future projects. Re-using system architectures and designs on subsequent projects seldom happens because they are not well-captured. Institutional guidance documents often do provide useful heuristics and lessons learned, but these resources often are not sufficient to enable architecture re-use. (Bayer, et al., 2011)

6.3 NASA JPL Use of Models as Information Structures

In modern complex engineering systems, models can take various forms, such as differential equations, simulations, or SysML drawings. The purpose of such models is to organize concepts and properties into meaningful relationships. These concepts, properties, and relationships can be unique to an individual model, inasmuch as it concerns the description of the elements of that particular model and their interrelationships, or they can be common to a family of models. For models that share a common format or purpose, they can more easily be compared, contrasted, and reused. This enables engineers to more effectively understand the content of a model and what is intended to be communicated by a model without the need for extensive explanation. Standardized model formats allow engineers to focus on understanding and creating, not on explaining and cross-training. Having standardized formats does not restrict an engineering team to only use those common formats. Unique situations can be handled by model extensions. Ontologies can be used to support the definitions of system modeling concepts, properties, and relationships by providing these definitions as inputs that are digested by models in the form of modeling profiles. Ontologies can be used to make explicit the knowledge about system elements that is often hidden, implied, or non-existent in a system description, such as a modeling diagram. Part of the challenge to improving the approach to designing modern systems is to devise a method by which model element information (the description of some “thing” in the model) can be brought forth for use by the engineers architecting a system design. This can be more easily solved by separating what the thing is called (its assigned identity) from what kind of a thing it is (where it can be found in a controlled vocabulary of concepts). This is the value that an ontology brings to system modeling. (Jenkins, *Ontologies And Model Based Systems Engineering*, 2010) An example of organization of

concepts, properties, and relationships is shown in Figure 10, as developed by Jenkins in (Jenkins S. , Ontologies And Model Based Systems Engineering, 2010).

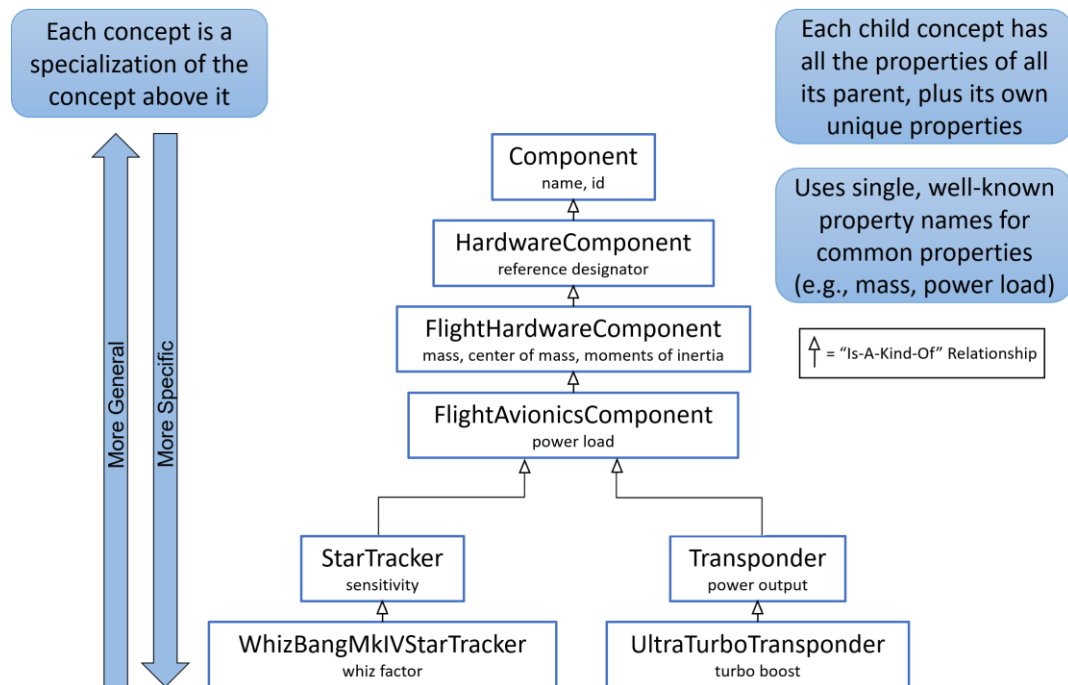


Figure 10: Example Type Classification Hierarchy (Jenkins, Ontologies And Model Based Systems Engineering, 2010)

Facts are expressed in “triples” of the form (subject, predicate, object). Facts such as these that describe and relate system elements can then be expressed using these terms (as shown in the above figures) and stored in a repository called a “triple store.” An example of a set of triples for the NASA JPL project is shown in Figure 11. Here three triples are shown. One triple can be stated as “The Component Performs a Function”.

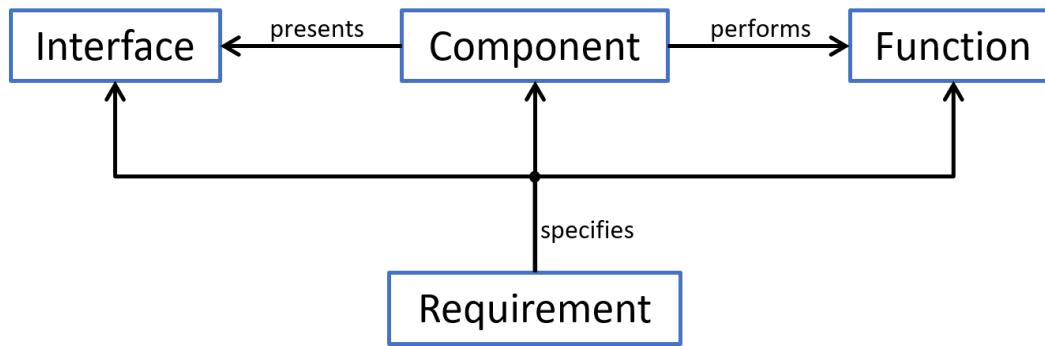


Figure 11: Relationships are Also Properties (Jenkins, Ontologies And Model Based Systems Engineering, 2010)

The stored facts can then be used to address simple questions like “What is the sensitivity of the WhizBangMkIVStarTracker named ‘Star Tracker A’?” Further, if the repository can draw inferences by using an inference engine, then we can ask things like “What is the sensitivity of the StarTracker named ‘Star Tracker A’?” Also, to produce the master equipment list is a simple matter of submitting a query to the database in the form: “Find all FlightHardwareComponents and print their names and masses.” If the database is strategically designed, then these queries become mission-independent procedures and can thus be reused from one project to another.

(Jenkins, Ontologies And Model Based Systems Engineering, 2010)

6.4 NASA JPL Use of Semantic Technologies

Many of the issues identified above by the IMCE initiative revolve around the interrelatedness of all the elements of the systems engineering process employed within an organization. According to JPL, achieving a high-level of interrelatedness requires standards for naming and classification of model elements and properties (using ontologies) and the expression of those standards in SysML-specific terms (modeling profiles). This is the focus of the current activities at JPL and what is of interest to this thesis. The integration of Semantic Technologies

with SysML modeling is the approach that JPL is currently pursuing to accomplish their near-term objectives. (Bayer, et al., 2011)

The concept of the Semantic Web was introduced in Section 5.5. NASA JPL uses the term Semantic Technologies to refer to the theories, technologies, and practice of the Semantic Web. In the 2010 timeframe, NASA JPL began examining the use of Semantic Web technologies. These include the standards indicated in Table 5, and the technologies indicated in Table 6. Since this is an active field of research, the technology and tools advance rapidly. (Bayer, et al., 2010) Therefore, it is advisable to periodically check on the currency of these technologies to determine whether they have rolled over into a new set over time. For example, OpenRDF Sesame is now known as RDF4J.

Table 5: Ontology Standard Used at JPL (Jenkins, Ontologies And Model Based Systems Engineering, 2010)

Ontology Standards	Description
Resource Description Framework (RDF)	Statements of the form (subject, predicate, object)
	Simple class hierarchies
Web Ontology Language (OWL)	RDF vocabulary for formal logic
SPARQL Query Language for RDF	Powerful language for querying RDF/OWL databases

Table 6: Ontology Technologies Used at JPL (Jenkins, Ontologies And Model Based Systems Engineering, 2010)

Ontology Technology Types	Example Technologies
Ontology Editors	Protégé, TopBraid Composer, etc.
Knowledge Repositories	Sesame, Oracle Semantic Database, Mulgara, etc.
Application Frameworks	Sesame, Jena, TopBraid Suite, OpenRDF Sesame, RDF4J, etc.

The SysML specification (Object Management Group, 2019) includes definitions of concepts that form a kind ontology, including concepts such as Block, Interface, Activity,

Requirement, etc. In order to build SysML models capable of a higher degree of interchangeability, it is necessary to build additional ontological structure beneath these high-level concepts. This includes concepts such as Work Breakdown Structure, Hardware, Software, Stakeholder, Concern, etc., plus any specialized associations, such as: authorizes, represents, specifies, etc. JPL is developing its ontologies using OWL2, which is more fundamental than SysML, in terms of the interoperability it implies for the system model (more general, and therefore applicable across multiple models.) These ontologies are then translated into SysML conceptual models and profiles. The relative utility of Semantic Web technologies and UML/SysML as seen by JPL are described in Table 7. (Jenkins, Ontologies And Model Based Systems Engineering, 2010)

Table 7: Relative Utility of Semantic Web Technologies and UML/SysML (Jenkins, Ontologies And Model Based Systems Engineering, 2010)

Topic	Semantic Web	UML/SysML
Graphics		++
Tooling	+	++
Querying	++	+
Inferences	++	
Interchange	++	+
Expression	++	
Description	+	++

As is shown in the table above, and as noted in (Jenkins, Ontologies And Model Based Systems Engineering, 2010), the emphasis of SysML is on notation, whereas OWL was founded on formal logical principles. Consequently, OWL provides strong support for verification of consistency and satisfiability, extraction of entailments, conjunctive query answering, etc. SysML inherits a semantic foundation that provides for only limited reasoning and analysis,

which is a substantial impediment to developing high confidence in the soundness of any conclusions drawn therefrom. For example, a number of foundation concepts from systems engineering, such as work package, objective, environment, etc., do not explicitly appear in SysML. (Jenkins & Rouquette, 2012)

OWL has had only limited adoption in systems engineering due to the absence of any graphical notation conventions in the OWL standards. But, the complementary strengths and weaknesses of SysML and OWL invite the possibility of combining strengths to provide a capability that provides the easily editable graphical notation of SysML and the formal reasoning of OWL. If the systems engineering ontologies are expressed in OWL, this makes them amenable to formal validation. Formal reasoning techniques can then be used to ensure that model syntax and semantics are consistent and satisfiable, and that reasoning operations remain tractable since they are constrained within the bounds of Description Logic. (Jenkins & Rouquette, 2012)

An additional IMCE objective is to develop the systems engineering ontologies to reflect common systems engineering conventions such that they provide the formal unifying framework for all systems engineering information in any language, in any tool, in any repository. These systems engineering ontologies provide a common controlled vocabulary that can be used to address a wide range of assertions about complex systems throughout their life cycles. Some of the advantages of using controlled vocabularies in modeling, and enforcing rules for well-formedness are that it enables durable information storage, lossless information interchange, interdisciplinary information integration, and automated analysis and product generation. (Jenkins & Rouquette, 2012)

The ontologies developed by JPL for these purposes are partitioned into three categories: Foundation, Discipline, and Application. These categories are intended to group concerns according to differing foci and objectives. The Foundation ontologies define concepts, and properties that apply generally across all projects to establish an overall framework for systems engineering. The Discipline ontologies define those concepts and properties that are pertinent to a particular engineering discipline. This is accomplished primarily through the use of specialization from the Foundation ontologies. The primary objective of using the discipline ontologies is to provide for information interchange across all disciplines. In this way, all systems engineering models, regardless of discipline, use a common vocabulary. This makes it a simple matter of using the common vocabulary in a query to extract common properties of any modeled component across all disciplines. Application ontologies define the concepts and properties pertinent to a particular class of engineered system irrespective of discipline. A certain subsystem ontology, for example, would draw from multiple discipline and foundation ontologies to characterize components particular to that subsystem application. Multiple individual ontologies have been developed within the Foundation and Discipline categories as described in Table 8. (Jenkins & Rouquette, 2012)

Table 8: OWL Ontologies for Systems Engineering (Jenkins & Rouquette, 2012)

Ontology Category	Ontology Name	Description
Foundation	Base	The base ontology defines a small number of general concepts (e.g., container) and properties (e.g., contains) that are refined in other ontologies.
	Mission	The mission ontology defines concepts and properties used to describe the execution of a mission and its context: objectives, performing elements, functions, interfaces, requirements, etc.

	Analysis	The analysis ontology defines concepts and properties used for qualitative and quantitative characterization of individuals of any time.
	Project	The project ontology defines concepts and properties used to describe the entities and endeavors involved in designing, analyzing, acquiring, integrating, and testing the elements of a mission: projects, programs, work packages, deliverables, etc.
Discipline	Electrical	Defines concepts and properties for current sources and loads, signal types, conditioning and distribution equipment, etc.
	Mechanical	Defines mass properties, mechanical interface types, etc.
	Verification and Validation	Defines process and analysis specializations to capture V&V activities and results.

6.5 Embedding Ontologies in SysML Profiles

In order to build SysML profiles from domain ontologies, it is necessary to establish formal relationships between the elements of the ontologies and their counterparts in SysML/UML. These relationships cannot be established until the UML/SysML concepts and properties are transformed into ontologies which can be reasoned over. Once these transformations are performed and both sets of ontologies are available, the SysML/UML concepts and properties can then be specified and reasoned about, providing the ability to express relationships between domain ontologies and SysML using OWL axioms. (Jenkins & Rouquette, 2012) The use of QVTo to perform the transformation from UML/SysML form to ontological format in order to perform reasoning on the ULM/SysML elements is illustrated in Figure 12.



Figure 12: Transformation of UML/SysML Models to Ontologies (Jenkins & Rouquette, Progress on Integrating OWL and SysML, 2012)

6.5.1 Relate OWL Concepts to SysML Classes

Concept (class) relationships are established by declaring that some class defined in a domain ontology is a subclass of some corresponding element in SysML, or vice versa, on a concept-by-concept basis. This process is covered in detail in Step 2 of the Step-By-Step process defined further below. (Jenkins & Rouquette, 2012)

6.5.2 Relate OWL Relationships to SysML Properties

Likewise, some properties defined in a domain ontology as relationships can be declared as subproperties of some corresponding element in SysML, or vice versa. This process is covered in detail in Step 3 defined further below. (Jenkins & Rouquette, 2012)

Unfortunately, the process of embedding of OWL relationships in SysML/UML relationships is not as direct as with OWL classes. This is because there is no direct mechanism to reify occurrences of object properties in OWL. To explain this requires some background of the concept of reification. (Jenkins & Rouquette, 2012)

6.5.2.1 RDF Triples

RDF is intended to provide a simple way to make statements about the world. RDF is based on the idea that the things being described have properties which have values, and that

resources can be described by making statements that specify those properties and values. RDF uses a particular terminology for talking about the various parts of statements. Specifically, the part that identifies the thing the statement is about is called the subject. The part that identifies the property or characteristic of the subject that the statement specifies is called the predicate, and the part that identifies the value of that property is called the object. (World Wide Web Consortium, 2004) A statement is an object, predicate, subject triple. (World Wide Web Consortium, 2006) The subject-predicate-object triple form a relationship described by the predicate, between the source (subject) and the target (object.)

6.5.2.2 Reification in General

Reification is widely used in conceptual modeling primarily for the purpose of viewing a relationship (such as an RDF triple) as an entity (a concept or class). The purpose of reifying a relationship is to make it explicit (to create a class to represent the relationship explicitly), so that additional information can be added to it. (Wikipedia, 2019)

6.5.2.3 Reification in UML/SysML

In UML/SysML, a relationship between two entities cannot be specified in an RDF statement without reifying the relationship (creating a class to represent the relationship as a concept or class.) In the example illustrated in Figure 13, the statement “The task is allocated to a resource” cannot be mapped to an RDF triplet without creating a class called Allocation to represent the relationship between the task and the resource. (Arlow & Neustadt, 2005)

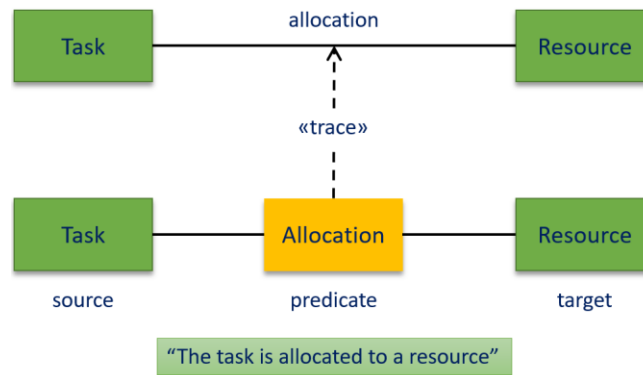


Figure 13: Reification in UML/SysML (Arlow & Neustadt, 2005)

6.5.2.4 Reification in RDF

The RDF reification vocabulary is designed to talk about statements. (World Wide Web Consortium, 2006) RDF applications sometimes need to describe other RDF statements using RDF, for instance, to record information about when statements were made, who made them, or other similar information (referred to as "provenance" information). (World Wide Web Consortium, 2004) For example, consider a particular camping tent product

`exproducts:item10245`, offered for sale. A triple that describes the weight of the tent, is:

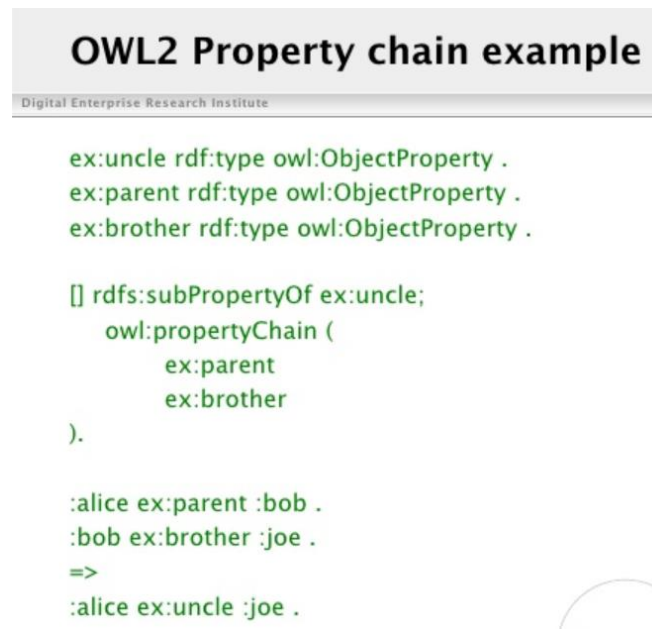
```
exproducts:item10245    exterms:weight    "2.4"^^xsd:decimal .
```

(World Wide Web Consortium, 2004)

It might be useful to record who provided that particular piece of information. RDF provides a built-in vocabulary intended for describing RDF statements. A description of a statement using this vocabulary is called a reification of the statement. The RDF reification vocabulary consists of the type `rdf:Statement`, and the properties `rdf:subject`, `rdf:predicate`, and `rdf:object`. However, while RDF provides this reification vocabulary, care is needed in using it, because it is easy to imagine that the vocabulary defines some things that are not actually defined. (World Wide Web Consortium, 2004)

6.5.2.5 Reification in OWL

There exists no mechanism in OWL1 to define properties as a composition of other properties. So, the concept of an uncle as a brother of a father cannot be described in OWL1. However, in OWL2, the construct `ObjectPropertyChain` in a `SubObjectPropertyOf` axiom allows a property to be defined as the composition of several properties. (World Wide Web Consortium, 2012) An example code set for performing reification to define the “uncle” relationship is illustrated in Figure 14.



```
ex:uncle rdf:type owl:ObjectProperty .
ex:parent rdf:type owl:ObjectProperty .
ex:brother rdf:type owl:ObjectProperty .

[] rdfs:subPropertyOf ex:uncle;
   owl:propertyChain (
     ex:parent
     ex:brother
   ).

:alice ex:parent :bob .
:bob ex:brother :joe .
=>
:alice ex:uncle :joe .
```

Figure 14: OWL2 Property Chain Example (Passant, 2009)

As OWL2 provides mechanisms to define arbitrary classes and properties, there is no difficulty with creating, for every object property p in some ontology, a corresponding class P to represent occurrences of that property, as well as for the source and target properties that connect the reified occurrence to the model elements that it relates. Having done so, then the OWL2 property chain mechanism that can be used to declare that the existence of the reified object property occurrence of class P with source A and target B implies that A - p - B (a triple). The next

step, then, would be to supplement the system engineering ontologies with axioms that implement this reification pattern for every object property in the ontologies. (Jenkins & Rouquette, 2012)

According to Jenkins and Rouquette in (Jenkins & Rouquette, 2012) “reified relationships are the key to a semantics-preserving mapping between UML and OWL. Without reification, there are many possible combinations for mapping OWL classes and object properties to UML classes, associations, association classes, properties and other relationships (e.g., dependencies). The Object Management Group’s Ontology Definition Metamodel (ODM) specification explains some of these possibilities but does not recommend a particular one. More importantly, the ODM lacks a unifying pattern for handling the various ways in which conceptual relationships are modeled as associations, dependencies, generalizations, ports, etc. A generic reification pattern simplifies the UML/OWL mapping because it separates the problem of modeling a conceptual relationship in OWL in terms of classes, object properties and property chain axioms from the problem of choosing an adequate embedding of this conceptual relationship in UML or in a profile extension of UML.” (Jenkins & Rouquette, 2012)

6.6 Embedding Ontologies in SysML Profiles

The steps identified in (Jenkins & Rouquette, 2012) and (Jenkins & Rouquette, Semantically-Rigorous Systems Engineering, 2012) for performing this procedure are outlined below.

6.6.1 Step 1 - Create OWL ontologies for SysML

Create OWL ontologies for SysML by 1) transforming the UML metamodel into a UML ontology, and 2) transforming the SysML (as a profile of UML) into a SysML ontology. These transformations are accomplished using the Operational Query/View/Transformation Language

(QVTo). The resulting transformed ontologies express certain features of SysML/UML in OWL, including the UML/SysML class taxonomy.

6.6.2 Step 2 - Relate Domain Concepts to SysML

Relate domain concepts to the best match in SysML by writing embedding axioms that define those relations. Embedding domain classes into SysML in this way is straightforward. In the following examples, a system Component is defined as a subclass of a SysML Block, and a system Requirement is defined as a subclass of a SysML Requirement.

- mission:Component owl:subClassOf SysML:Block
- mission:Requirement owl:subClassOf SysML:Requirement

6.6.3 Step 3 - Relate Domain Properties to SysML

Relate domain properties to the best match in SysML by writing embedding axioms that define those relations. This is more complex than the same process for concepts (classes) the uses the OWL2 Property Chain mechanism as described in Section 6.5.2.5.

- To use the *owl:inverseOf* relationship requires Extended MOF semantics (not explained further in (Jenkins & Rouquette, Semantically-Rigorous Systems Engineering, 2012).)
- As described in Section 6.5.2.5, occurrences of object properties are not reified in OWL, so there is no way to represent “this requirement specifies the ‘performs’ relationship between this component and this function” because the particular occurrence of ‘performs’ has no class defined, and therefore no identity
- Therefore (as described in Section 6.5.2.5) for a given object property, e.g., ‘performs,’ create a corresponding reification class ‘Performs,’ corresponding

object properties 'hasPerformsSource' and 'hasPerformsTarget', and OWL property chain axiom

- An instance of this reification class appears in Figure 15, as:

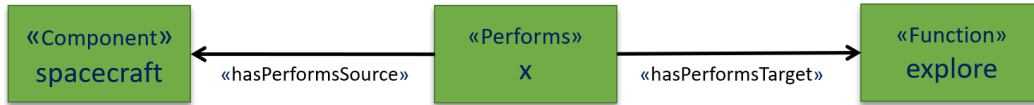


Figure 15: Example UML/SysML Reification (Jenkins & Rouquette, Semantically-Rigorous Systems Engineering, 2012)

- By the effect of the OWL2 property chain axiom, illustrated in Figure 16, this implies:

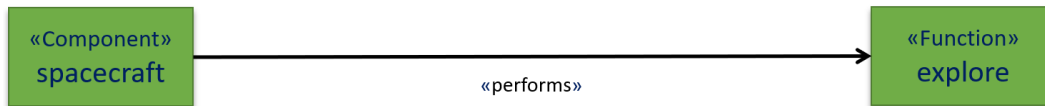


Figure 16: Example OWL2 Reification (Jenkins & Rouquette, Semantically-Rigorous Systems Engineering, 2012)

- Which is what is needed for the SysML-to-OWL transformation

6.6.4 Step 4 – Test the Ontologies

Next, subject the ontologies (including embedding axioms) to a battery of tests.

- For Consistency
 - Ensure that no axioms contradict other axioms
- For Satisfiability
 - Ensure that every class can be nonempty
- For Well-Formedness
 - Ensure that every class is correctly embedded in SysML

- Ensure that every property is correctly embedded in SysML
- Ensure that the domain and range of super/subproperty pairs are consistent
- Ensure that every object property has a reification apparatus
- Ensure consistent embedding of super/subclass pairs

6.6.5 Step 5 – Use a Continuous Integration System

Run these tests run under a continuous integration system such as Jenkins whenever one of the ontologies changes

6.6.6 Step 6 – Load the Ontologies into a Repository

Load the ontologies into a Sesame repository and use SPARQL queries to generate bundle digests that simplify profile construction by offload reasoning that's much easier to do in SPARQL than QVTo.

- Query for object property ranges after applying a range restriction
- Query for valid predicates for each subject class
- Query for valid object classes for each subject/predicate pair

6.6.7 Step 7 – Produce SysML Profiles

Perform a transformation in Operational Query/View/Transform (QVTo) to produce SysML profiles

6.6.8 Step 8 – Produce User Interface Customizations

The QVTo transforms can also produce architecture-tool-specific user interface customizations

- To assist the modeler in complying with profile rules

6.6.9 Step 9 –Transform SysML Models Back into OWL

Models with profiles applied can then be transformed from SysML back into OWL using QVTo to extract the ontological commitments from the profiled model. The OWL representation is then suitable for

- Validation of well-formedness
- Validation of adherence to local business rules, e.g.,
 - Validate that every Component performs at least one Function
 - Validate that every Function is performed by exactly one Component
 - Validate that every ‘presents’ relationship is specified by at least one Requirement
- The OWL representation is also suitable for performing feature extraction and transformation for specialized analysis tools, e.g.,
 - Maple, Mathematica
- The OWL representation is also suitable for long-term archival and data warehousing

6.7 NASA JPL Conclusions and Future Work

NASA JPL has come to the following conclusions regarding their work to transform domain ontologies into SysML profiles.

- Transforming SysML/UML specifications to OWL and then embedding ontologies back into SysML profiles has proven to be a flexible process
- Pre-processing ontologies with SPARQL simplifies the profile generation code
- QVTo has proven to be powerful once some performance issues were addressed
- SPARQL and Sesame are powerful for analyzing and transforming SysML models with the transformed profiles applied

NASA JPL has identified the following future work activities they plan to undertake as a result of success with this ontology transformation approach.

- Add support for datatype properties
- Enhance the SysML-to-OWL transformation
- Develop analysis tooling in the OWL domain
- Develop discipline and application ontologies that extend foundation concepts, such as electrical, mechanical, verification, etc.

7 Summary

Ontologies have been and are proving to be of value to engineering activities. This thesis has reported on several fields in which ontologies have been developed and employed to enhance the quality and effectiveness of engineering activities. These include 1) for modeling the product structure and taxonomy, 2) for design automation using existing engineering knowledge, 3) for requirements engineering, 4) for the control of production processes for dynamic orchestration, 5) for factory automation, and 6) for the mapping of data sources to Manufacturing Execution Systems functions. (El Kadiri, et al., 2015) There are many possible uses on ontology, and as more industries and organizations learn of the benefits of using ontologies, this field of application will grow. These benefits include 1) more effective engineering knowledge openness and diffusion, 2) faster sharing of product-related information and knowledge across the entire value-chain, 3) more innovative mechanisms to enable new feedback, 4) provisioning of new feed-forward mechanisms to deliver information to actors in downstream lifecycle phases, 5) better decision-support tools, 6) innovative designs, and 6) realization of product-service capability to support quick reaction to changing user requirements, among many other possible benefits. (El Kadiri, et al., 2015)

El Kadiri, et.al. describe three specific projects that exemplify the use of ontologies in general engineering applications. These were 1) for collecting product manufacturing data from factory floor work stations to feed operational efficiency analysis, future product design Life Cycle Cost (LCC) analysis, and to respond to changing customer requirements with speed, 2) to develop an integrated collaborative virtual environment intended to synchronize factory floor production operations with various simulations of those operations for near-real time optimization of factory operations, 3) to provide decision support on how to best design and

implement facilities, personnel, and organizations over vast geographical areas. (El Kadiri, et al., 2015)

Happel and Seedorf described several approaches for using ontologies in the context of the Software Engineering Life Cycle in the areas of requirements engineering, component reuse, integration with software modeling languages, ontology as domain object model, coding support, code documentation, semantic middleware, business rules, semantic web services, project support, and testing. (Happel & Seedorf, 2006)

In addition, various researchers have identified uses of ontologies specifically in Systems Engineering, including 1) a process to capture a systems engineering functional domain ontologies (Sarder & Ferreira, 2007), 2) to capture knowledge items used in the systems engineering process in order to record engineers' ideas and reasoning processes, and facilitate their reuse (Chourabi, Pollet, & Ben Ahmed, 2008), 3) to enhance the communications between domain specialists and modelers, to enhance the communications among specialist in different domains, facilitate the collection of system information to be used in modeling, to create new perspectives on existing models, and to generate documentation using those perspectives (Ernadote, 2015), 4) to capture and preserve tacit knowledge from domain experts involved in the inspection of a railway tunnel network (Thakker, Dimitrova, Cohn, & Valdes, 2015), 5) to analyze the system design (Graves, Integrating SysML and OWL, 2009), 6) to transform data among models using an XML-based data format (Abele, Legat, Grimm, & Muller, 2013), 7) to provide the formal basis for verifying compliance of the system model developed in SysML with State Analysis semantics (Wagner, et al., 2012), 8) to perform in-depth reasoning on engineering tasks by embedding a model of the system under analysis as an axiom set within a suitable logic

(Graves, Integrating Reasoning with SysML, 2012), and 9) to managing inconsistencies in models of systems from the domain of automated production systems (Feldmann, et al., 2015).

The work of greatest promise to the objectives of the author of this thesis is embodied in the work of NASA JPL as described in (Wagner, et al., 2012) and other related works. The author intends to follow closely the progress of the Semantics Technologies for Systems Engineering (ST4SE) group introduced in Section 6 to determine the applicability of their work towards the construction of ontologies that can be used to develop modeling profiles.

8 Conclusion

The author of this thesis set out to identify the state of the practice in bridging the gap between engineering ontologies and modeling profiles for engineering applications. This subject has been of interest to the author for the last 15 years of involvement in the architecting of complex systems (military aircraft), which provided exposure to the subject, without the opportunity to investigate the subject adequately. The work invested in the development of this thesis has provided the author with an overview of the various themes and threads involved in the process of developing ontologies for engineered systems, and converting those ontologies into system modeling profiles. The work has shown that there exists a complex network of researchers working on advancing the state of the art of ontological engineering, albeit for a large variety of end-purposes.

The specific purpose of interest here is to support the development of profiles for system modeling which capture the domain-specific concepts, properties, and relationships, etc useful to the architect developing the system architecture of some particular product in some particular domain. The detailed process of going from “some” ontology to a useable profile has not yet been sufficiently examined by the author. This thesis only addresses the mechanics of efforts previously performed for activities of interest to the information sources accessed. The material documented here in this thesis does not yet focus on the steps needed to be taken to develop a profile for a specific purpose in a specific domain. It is the author’s intention to take the next logical step of narrowing down the scope from the broad-brush survey presented in this thesis to examine a particular engineering application.

The application of interest to the author is the development of system models that capture the essence of Systems-of-Systems (SoS) architectures wherein constituent systems with a set of

mission-relevant capabilities participate in the SoS on a capability-defined basis to occupy mission roles requiring capabilities that the particular platform can fulfill. Thus, role-filling by the SoS becomes a constant activity intended to make maximum use of available resources, and in order to meet the minimum mission needs at the least cost. These are valid goals for complex systems participating in complex SoSes, no matter the domain of discourse involved. Towards this goal, it is desirable to define a consistent modeling approach that focuses on role-filling using the most cost-effective assets available. To control the model development activities, modeling profile(s) are desirable to provide consistency via constraints that force the hand of the modeler to observe accepted standards in architecture modeling relevant to complex SoSes. The availability of profiles that are consistent with the accepted terminology of the domain requires that such profiles be developed off of consistent source material, and the most logical choices for this source material are domain ontologies related to the particular domain of discourse involved. It is the author's opinion that, in large part, such ontologies do not yet exist to support development of role-filling SoSes in any domain of discourse.

The next step for the author is to begin focusing on specific technologies introduced in this thesis that demonstrate high potential for use in the application described above. Toward that objective, the author has identified the work performed by Djuric, et al. in (Djuric, Gašević, Devedžić, & Damjanovic, 2004) and other of their related works, as well as that of NASA JPL as described in (Jenkins S. , *Ontologies And Model Based Systems Engineering*, 2010), (Jenkins & Rouquette, 2012), (Jenkins & Rouquette, *Semantically-Rigorous Systems Engineering*, 2012), (Jenkins S. , *Semantic Technologies*, 2018), and (Wagner, et al., 2012), as holding the most promise for paying dividend on investment of time in researching further their activities. Toward

that objective, the author has taken keen interest in the activities of the Semantics Technologies for Systems Engineering (ST4SE) introduced in Section 6.

With regard to roles and their importance in the architecture of a SoS and presequently their usage and declaration in ontologies, the author intends to pursue the work of Kouji Kozaki, Yoshinobu Kitamura, Mitsuru Ikeda, Riichiro Mizoguchi, Eiichi Sunagawa, Matteo Baldon , Guido Boella, Leendert van der Torre, and others who have addressed the concept of roles (or perhaps better said as the “role” of roles) in ontologies.

References

- Abele, L., Legat, C., Grimm, S., & Muller, A. W. (2013). Ontology-based Validation of Plant Models. *11th IEEE International Conference on Industrial Informatics*. Bochum, Germany: IEEE. Retrieved from <https://ieeexplore.ieee.org/document/6622888>
- About JPL. (n.d.). Retrieved January 3, 2020, from NASA Jet Propulsion Laboratory: <https://www.jpl.nasa.gov/about/>
- Alhir, S. S. (2003). *Understanding the Model Driven Architecture (MDA)*. Retrieved January 3, 2020, from Methods & Tools: <https://www.methodsandtools.com/archive/archive.php?id=5>
- Antoniou, G., Groth, P., van Harmelen, F., & Hoekstra, R. (2012). *A Semantic Web Primer*. Retrieved from Islamic Azad University Mobarakeh Branch: [http://prof.mau.ac.ir/images/Uploaded_files/A%20Semantic%20Web%20Primer-The%20MIT%20Press%20\(2012\)\[7460174\].PDF](http://prof.mau.ac.ir/images/Uploaded_files/A%20Semantic%20Web%20Primer-The%20MIT%20Press%20(2012)[7460174].PDF)
- Arlow, J., & Neustadt, I. (2005). *UML 2 and the Unified Process*. Upper Saddle River, New Jersey, USA: Addison-Wesley.
- Bayer, T. J., Bennett, M., Delp, C. L., Dvorak, D., Jenkins, S. J., & Mandutianu, S. (2011). Update – Concept of operations for integrated model- Centric Engineering at JPL. *IEEE Aerospace Conference*. Big Sky, Montana: IEEE. Retrieved January 3, 2020, from <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.667.4638&rep=rep1&type=pdf>
- Bayer, T. J., Cooney, L. A., Delp, C. L., Dutenhoffer, C. A., Gostelow, R. D., Ingham, M. D., . . . Smith, B. S. (2010). An Operations Concept for Integrated Model-Centric Engineering at JPL. *IEEE Aerospace Conference 2010* (pp. 1-14). Big Sky, Montana: IEEE. Retrieved

January 3, 2020, from https://trs.jpl.nasa.gov/bitstream/handle/2014/45289/09-5046_A1b.pdf?sequence=1

Bernardi, T. L., Rabello, R. D., & Cervi, C. R. (2016). An Ontology-Based Approach to Use Requirements Engineering in Portals of Transparency. *Proceedings of the IX ONTOBRAS Brazilian Ontology Research Seminar*. Curitiba, Brazil: CEUR-WS.org. Retrieved December 27, 2019, from <http://ceur-ws.org/Vol-1862/paper-13.pdf>

Boyce, S., & Pahl, C. (2007). Developing Domain Ontologies for Course Content. *Educational Technology & Society*, 275-288. Retrieved January 3, 2020, from <https://core.ac.uk/download/pdf/11310019.pdf>

Chourabi, O., Pollet, Y., & Ben Ahmed, M. (2008). Ontology Based Knowledge Modeling for System Engineering Projects. *2008 Second International Conference on Research Challenges in Information Science*. Marrakech, Morocco: IEEE. Retrieved January 3, 2020, from <https://ieeexplore.ieee.org/document/4632138?arnumber=4632138>

Djuric, D., Gašević, D., Devedžić, V., & Damjanovic, V. (2004). A UML profile for OWL ontologies. *Model Driven Architecture, European Workshop on Model Driven Architecture* (pp. 204-219). Twente, The Netherlands: Springer. Retrieved January 3, 2020, from https://www.researchgate.net/publication/221233499_A_UML_profile_for_OWL_ontologies

El Kadiri, S., Terkaj, W., Urwin, E. N., Palmer, C., Kiritsis, D., & Young, R. (2015). Ontology in Engineering Applications. In R. Cuel, & R. Young, *Formal Ontologies Meet Industry: 7th International Workshop* (pp. 91-98). Berlin, Germany: Formal Ontologies Meet

- Industry. Retrieved January 3, 2020, from
https://www.researchgate.net/publication/220438605_Creating_the_ontologists_of_the_future
- Ernadote, D. (2015). An ontology mindset for system engineering. *2015 IEEE International Symposium on Systems Engineering*. Rome, Italy: IEEE. Retrieved January 3, 2020, from
https://www.researchgate.net/profile/Dominique_Ernadote/publication/308842690_An_ontology_mindset_for_system_engineering/links/5c8b47b345851564fade5d4b/An-ontology-mindset-for-system-engineering.pdf
- Feldmann, S., Herzig, S. J., Kernschmidt, K., Wolfenstetter, T., Kammerl, D., Qamar, A., . . . Vogel-Heuser, B. (2015). Towards Effective Management of Inconsistencies in Model-Based Engineering of Automated Production Systems. *15th IFAC Symposium on Information Control Problems in Manufacturing*, (pp. 916-923). Ontario, Canada. Retrieved January 3, 2020, from
<https://reader.elsevier.com/reader/sd/pii/S2405896315004395?token=41FCAFBB3B918119BCA7BDCCCD92AD5F5456F8489B11FD99B1DB756E45BE564D02D40E322F8C8A1AD13A3A475725A805>
- Fensel, D., & Federico, F. (2010). *Web Ontology Language (OWL)*. Retrieved January 3, 2020, from Innsbruck University Department of Computer Science: https://www.sti-innsbruck.at/sites/default/files/courses/fileadmin/documents/semweb14-15/07_SW-OWL.pdf
- Fikes, R., & Tom, K. (1985). The Role of Frame-Based Representation in Reasoning. *Communications of the ACM*, 904-920. Retrieved December 27, 2019, from

https://www.researchgate.net/publication/220426864_The_Role_of_Frame-Based_Representation_in_Reasoning

Fuentes-Fernández, L., & Vallecillo-Moreno, A. (2004, April). An Introduction to UML Profiles. *Upgrade - The European Journal for the Informatics Professional*, 6-13. Retrieved January 3, 2020, from https://www.researchgate.net/publication/245346983_An_introduction_to_UML_profiles/link/02e7e537c492be345d000000/download

Gasevic, D., Djuric, D., & Devedzic, V. (2006). *Model Driven Architecture and Ontology Development*. Berlin, Heidelberg: Springer-Verlag.

Globa, L., Novograduska, R., Koval, A., & Senchenko, V. (2018, February 20). *Examples of Ontology Model Usage in Engineering Fields*. Retrieved from IntechOpen: <https://www.intechopen.com/books/ontology-in-information-science/examples-of-ontology-model-usage-in-engineering-fields>

Graves, H. (2009). Integrating SysML and OWL. *OWLED'09: Proceedings of the 6th International Conference on OWL: Experiences and Directions* (pp. 117–124). Chantilly, Virginia: CEUR Workshop Proceedings. Retrieved January 3, 2020, from http://ceur-ws.org/Vol-529/owled2009_submission_8.pdf

Graves, H. (2012). Integrating Reasoning with SysML. *INCOSE International Symposium 2012*. Rome, Italy: International Council on Systems Engineering. Retrieved January 3, 2020, from <http://ontolog.cim3.net/forum/ontology-summit/2012-02/pdfrV7esTvCnf.pdf>

- Graves, H. (2014). Integrating Reasoning with SysML. *INCOSE International Symposium 2014*. Las Vegas, Nevada: INCOSE. Retrieved January 3, 2020, from <http://ontolog.cim3.net/forum/ontology-summit/2012-02/pdfrV7esTvCnf.pdf>
- Gruber, T. R. (1993). A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, 199-220. Retrieved January 3, 2020, from <https://tomgruber.org/writing/ontolingua-kaj-1993.pdf>
- Gruber, T. R. (2009). Ontology. In *Encyclopedia of Database Systems* (pp. 1963-1965). Switzerland: Springer-Verlag. Retrieved January 3, 2020, from <https://tomgruber.org/writing/ontology-in-encyclopedia-of-dbs.pdf>
- Guarino, N. (1998). Formal Ontology and Information Systems. *Proceedings of FOIS'98* (pp. 3-15). Trento, Italy: Formal Ontology in Information Systems. Retrieved January 3, 2020, from <https://klevas.mif.vu.lt/~donatas/Vadovavimas/Temos/OntologiskaiTeisingasKonceptinisModeliavimas/papildoma/Guarino98-Formal%20Ontology%20and%20Information%20Systems.pdf>
- Guarino, N., & Welty, C. (2002, February). Evaluating ontological decisions with ontoclean. *Communications of the ACM*, 61-65. Retrieved January 3, 2020, from https://www.researchgate.net/publication/297428382_Evaluating_ontological_decisions_with_ontoclean/link/59f1b061aca272cdc7ce21db/download
- Happel, H.-J., & Seedorf, S. (2006, January 1). Applications of Ontologies in Software Engineering. *Computer Science*, p. Unknown. Retrieved January 3, 2020, from

https://pdfs.semanticscholar.org/11e2/c3bfe1dd68446180f17e476addc947dad095.pdf?_ga=2.13634355.564213146.1578065933-477270357.1575778655

Harold, E. R., & Means, W. S. (2004). *XML in a Nutshell, Third Edition*. Sebastopol, California: O'Reilly Media, Inc.

Hennig, C., Viehl, A., Kämpgen, B., & Eisenmann, H. (2016). Ontology-Based Design of Space Systems. *International Semantic Web Conference* (pp. 308-324). Kobe, Japan: Springer International Publishing. Retrieved January 3, 2020, from http://www-kasm.nii.ac.jp/iswc2016/papers/paper_A6_.pdf

Hoberman, S. (2008, January 14). *Taxonomy and Ontology*. Retrieved January 3, 2020, from Information Management: <https://www.information-management.com/news/ontology-and-taxonomy>

Jenkins, J. S., & Rouquette, N. F. (2012). Semantically-Rigorous Systems Engineering Modeling Using SysML and OWL. *5th International Workshop on System & Concurrent Engineering for Space Applications*. Lisbon, Portugal. Retrieved January 3, 2020, from <https://pdfs.semanticscholar.org/1c94/f934eb1fd5033c019260470b071346ac91e6.pdf>

Jenkins, S. (2010). Ontologies And Model Based Systems Engineering. INCOSE. Retrieved January 3, 2020, from <https://www.omgwiki.org/MBSE/lib/exe/fetch.php?media=mbse:ontologies-jenkins.pptx>

Jenkins, S. (2018). Semantic Technologies. *Semantic Technologies for Systems Engineering*. NASA JPL. Retrieved January 3, 2020, from http://www.omgwiki.org/MBSE/lib/exe/fetch.php?media=mbse:incose_mbse_iw_2018:st4se_incose_mbse_2018-01-20.pptx

- Jenkins, S., & Rouquette, N. (2012). Progress on Integrating OWL and SysML. *INCOSE International Workshop 2012*. Jacksonville, Florida. Retrieved January 3, 2020, from <https://pdfs.semanticscholar.org/d3bb/3e2b98e16e0f93834bf44ba17feb4327b5c1.pdf>
- Jenkins, S., & Rouquette, N. (2012). Semantically-Rigorous Systems Engineering. Retrieved January 3, 2020, from <https://pdfs.semanticscholar.org/6e2c/dbb556015380f447ce455c6756ae5844fcd6.pdf>
- Kleppe, A., Warmer, J., & Bast, W. (2003). *MDA Explained*. Boston, Massachusetts: Addison-Wesley.
- Neches, R., Fikes, R., Finin, T., Gruber, T., Patil, R., Senator, T., & Swartout, W. R. (1991, Spetember). Enabling Technology for Knowledge Sharing. *AI Magazine*, pp. 36-56.
- Neuhaus, F., Florescu, E., Galton, A., Gruninger, M., Guarino, N., Obrst, L., . . . Smith, B. (2011). Creating the ontologists of the future. *Applied Ontology*, 91-98. Retrieved January 3, 2020, from https://www.academia.edu/2824018/Creating_the_ontologists_of_the_future
- Noy, N., & McGuinness, D. (2001). *Ontology Development 101: A Guide to Creating Your First Ontology*. Knowledge Systems Laboratory. Stanford, California: Stanford University. Retrieved January 3, 2020, from https://protege.stanford.edu/publications/ontology_development/ontology101.pdf
- Object Management Group. (2019, December). *OMG System Modeling Language*. Retrieved January 3, 2020, from OMG: <https://www.omg.org/spec/SysML/1.6>

- Overbeek, J. (2006). *Meta Object Facility (MOF): Investigation of the State of the Art*. University of Twente, Computer Science. Enschede, The Netherlands: University of Twente. Retrieved January 3, 2020, from https://essay.utwente.nl/57286/1/scriptie_Overbeek.pdf
- Pahl, C., & Holohan, E. (2004). Ontology Technology for the Development and Deployment of Learning Technology Systems - a Survey. *Proceedings of ED-MEDIA 2004--World Conference on Educational Multimedia, Hypermedia & Telecommunications* (pp. 2077-2084). Lugano, Switzerland: Association for the Advancement of Computing in Education. Retrieved January 3, 2020, from <https://core.ac.uk/download/pdf/147601228.pdf>
- Passant, A. (2009, November 3). *OWL2 Property Chain Example*. (D. E. Institute, Producer) Retrieved January 3, 2020, from SlideShare: https://www.slideshare.net/apassant/introduction-to-the-semantic-web-2410632/21-OWL2_Property_chain_exampleDigital_Enterprise
- Poli, R. (2003). Descriptive, Formal and Formalized Ontologies. In D. Fisette, *Husserl's Logical Investigations Reconsidered* (pp. 183-210). Unknown: Springer. Retrieved January 3, 2020, from <https://www.ontology.co/essays/descriptive-ontologies.pdf>
- Roe, C. (2012, June 7). *A Short History of Ontology: It's not just a Matter of Philosophy Anymore*. Retrieved January 3, 2020, from Dataversity: <https://www.dataversity.net/a-short-history-of-ontology-its-not-just-a-matter-of-philosophy-anymore/>

- Sarder, B., & Ferreira, S. (2007). Developing Systems Engineering Ontologies. *2007 IEEE International Conference on System of Systems Engineering*. San Antonio, Texas: IEEE. Retrieved January 3, 2020, from <https://ieeexplore.ieee.org/document/4304237>
- Shadbolt, N., Berners-Lee, T., & Hall, W. (2006, July). The Semantic Web Revisited. *IEEE Intelligent Systems*, pp. 96-101. Retrieved January 3, 2020, from https://eprints.soton.ac.uk/262614/1/Semantic_Web_Revisted.pdf
- Siegemund, K. (2014). *Contributions to Ontology-Driven Requirements Engineering*. Dresden, Germany: Technische Universität Dresden. Retrieved December 27, 2019, from <https://d-nb.info/1069096571/34>
- Thakker, D., Dimitrova, V., Cohn, A. G., & Valdes, J. (2015). PADTUN - Using Semantic Technologies in Tunnel Diagnosis and Maintenance Domain. *The Semantic Web. Latest Advances and New Domains: 12th European Semantic Web Conference*. Portoroz, Slovenia: Springer. Retrieved January 3, 2020, from https://www.researchgate.net/publication/277714675_PADTUN_-_Using_Semantic_Technologies_in_Tunnel_Diagnosis_and_Maintenance_Domain
- Truyen, F. (2006). *The Fast Guide to Model Driven Architecture*. Tustin, CA: Cephass Consulting Corp. Retrieved January 3, 2020, from https://www.omg.org/mda/mda_files/Cephass_MDA_Fast_Guide.pdf
- Wagner, D. A., Bennett, M. B., Karban, R., Rouquette, N., Jenkins, S., & Ingham, M. (2012). An Ontology for State Analysis: Formalizing the Mapping to SysML. *2012 IEEE Aerospace Conference*. Big Sky, Montana: IEEE. Retrieved January 3, 2020, from <https://ieeexplore.ieee.org/document/6187335>

Wikipedia. (2019, December 3). *RDF Schema*. Retrieved January 3, 2020, from Wikipedia:

https://en.wikipedia.org/wiki/RDF_Schema

Wikipedia. (2019, December 2). *Reification (computer science)*. Retrieved January 3, 2020, from

Wikipedia: [https://en.wikipedia.org/wiki/Reification_\(computer_science\)](https://en.wikipedia.org/wiki/Reification_(computer_science))

World Wide Web Consortium. (2004, February 10). *RDF Primer*. Retrieved from W3C:

<https://www.w3.org/TR/rdf-primer/>

World Wide Web Consortium. (2006, April 12). *Defining N-ary Relations on the Semantic Web*.

Retrieved from W3C: <https://www.w3.org/TR/swbp-n-aryRelations>

World Wide Web Consortium. (2012, December 9). *OWL 2 New Features and Rationale*.

Retrieved January 3, 2020, from W3C:

https://www.w3.org/2007/OWL/wiki/New_Features_and_Rationale#F8:_Property_Chain_Inclusion

World Wide Web Consortium. (2013, June 19). *W3C Semantic Web Activity*. Retrieved January

3, 2020, from W3C: <https://www.w3.org/2001/sw/>

World Wide Web Consortium. (2014, February 25). *RDF Schema 1.1*. Retrieved from W3C:

<https://www.w3.org/TR/rdf-schema/>

World Wide Web Consortium. (2014, March 15). *Resource Description Framework (RDF)*.

Retrieved from RDF: <https://www.w3.org/RDF/>

Curriculum Vita

Mr. John G. Artus (txartus@gmail.com) graduated in 1978 from Louisiana State University in Baton Rouge with a Bachelor of Science Degree in Electrical Engineering. Mr. Artus took employment in the Defense Industry with General Dynamics, Fort Worth Division in Fort Worth, Texas as a Test Director in the Air Force – Electronic Warfare and Evaluation Simulator (AF-EWES). Mr. Artus advanced his career into Operations Research as a Senior Operations Analyst, performing on many Independent Research And Development (IRAD) projects over a period of 18 years to assess the operational effectiveness of the company's military aircraft products using industry-standard Modeling and Simulation (M&S) tools.

In 2006, Mr. Artus transitioned to Systems Engineering at Lockheed Martin Aeronautics Company as a Senior Staff Systems Engineer. In this position, Mr. Artus provided expertise in Requirements, Architecture, and Design to develop Organizational Standard Practices (OSP) in these technical areas in preparation for CMMI assessment, successfully achieving a Level 3 assessment for the company as part of the assessment support team.

In 2014, Mr. Artus transitioned to the Advanced Development Programs (ADP, aka “Skunk Works”) as Senior Staff Architect, to perform architecture development on a number of IRAD and CRAD programs. Mr. Artus will retire in the summer of 2020, after a successful 42-year career in engineering, having achieved several certifications during his career: International Council On Systems Engineering (INCOSE) – Certified Systems Engineering Professional since 2009, and Expert Systems Engineering Professional since 2012, Federal Enterprise Architect Certification (FEAC) Institute – Certified Enterprise Architect in DoDAF since 2008, Object Management Group (OMG) – Certified Systems Modeling Professional II since 2010, Zachman International Inc. – Zachman Certified Enterprise Architect Associate since 2012.