ABSTRACT

AN ADAPTATION METHODOLOGY FOR REUSING ONTOLOGIES

By Vishal Bathija

Ontologies are an emerging means of knowledge representation that can improve information organization and management in an application. They have demonstrated their value in numerous application areas such as intelligent information integration or information brokering since they offer the technical support for sharing and exchanging information between human and/or software agents. Despite their successes, their time-consuming and expensive development process deters the prevalent use of ontologies. Thus, research is focusing on ways to improve the process of ontology construction which involves recognizing, representing, and recording concept definitions and their relationships. This thesis research investigates existing methods for ontology learning and develops ontology adaptation software architecture for transforming an ontology from one domain to a related or similar domain. Using this software, the SEURAT’s Argument Ontology for the domain of software engineering is adapted to create an initial ontology that supports engineering design for the specific problem domain of spacecraft design.
AN ADAPTATION METHODOLOGY FOR REUSING ONTOLOGIES

A Thesis

Submitted to the Faculty of Miami University
in partial fulfillment of
the requirements for the degree of
Master of Computer Science

Department of Computer Science & Systems Analysis

by

Vishal Bathija

Miami University

Oxford, Ohio

2006

Advisor______________________________________

Dr. Valerie Cross

Reader_______________________________________

Dr. Janet Burge

Reader_______________________________________

Dr. James Kiper
# TABLE OF CONTENTS

1. **Introduction** .......................................................................................................................... 1
2. **Ontologies** .............................................................................................................................. 2
   2.1 Definition of an Ontology ........................................................................................................ 3
   2.2 Types of Ontologies ............................................................................................................... 4
3. **Learning Ontologies** ............................................................................................................. 6
4. **Introduction to Design Rationale** .......................................................................................... 8
   4.1 SEURAT Overview ............................................................................................................... 10
   4.2 SEURAT’s Argument Ontology ............................................................................................ 11
5. **Related Work for Developing Domain Ontologies** ............................................................ 16
   5.1 GATE (Generalised Architecture for Text Engineering) .................................................... 17
   5.2 Learning Web Service Ontologies using GATE ................................................................. 18
   5.3 SHUMI (Support for Human Machine Interaction) ........................................................... 21
      5.3.1 Overview of Learning the Mission Ontology .............................................................. 21
      5.3.2 SHUMI Term Extraction ............................................................................................ 22
      5.3.3 SHUMI Relation Extraction ....................................................................................... 24
   5.4 OntoLT ................................................................................................................................ 25
6. **Adaptation Methodology and Software Architecture** .......................................................... 26
   6.1 Input Resources .................................................................................................................... 27
      6.1.1 Argument Ontology ..................................................................................................... 27
      6.1.2 Domain Corpora .......................................................................................................... 29
      6.1.3 General Corpus .......................................................................................................... 30
   6.2 Software Resources .............................................................................................................. 30
      6.2.1 Apache Lucene ............................................................................................................. 30
      6.2.2 JWordNet .................................................................................................................... 32
      6.2.3 SCHUG ...................................................................................................................... 33
      6.2.4 OntoLT ....................................................................................................................... 34
   6.3 Algorithms ............................................................................................................................. 34
      6.3.1 Determining Domain Relevance .................................................................................. 35
      6.3.2 Pruning ....................................................................................................................... 36
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.3.3</td>
<td>Adapting and Specializing</td>
<td>41</td>
</tr>
<tr>
<td>6.4</td>
<td>Evaluation</td>
<td>46</td>
</tr>
<tr>
<td>6.4.1</td>
<td>Coverage</td>
<td>47</td>
</tr>
<tr>
<td>6.4.2</td>
<td>Accuracy</td>
<td>48</td>
</tr>
<tr>
<td>6.4.3</td>
<td>Precision</td>
<td>49</td>
</tr>
<tr>
<td>6.4.4</td>
<td>Recall</td>
<td>50</td>
</tr>
<tr>
<td>7</td>
<td>Experiments and Results</td>
<td>50</td>
</tr>
<tr>
<td>7.1</td>
<td>Format of Experiments</td>
<td>50</td>
</tr>
<tr>
<td>7.2</td>
<td>Experimental Parameters</td>
<td>51</td>
</tr>
<tr>
<td>7.3</td>
<td>Comparison and Analysis of Experimental Results</td>
<td>52</td>
</tr>
<tr>
<td>8</td>
<td>Ontology Adaptation User Interface</td>
<td>62</td>
</tr>
<tr>
<td>9</td>
<td>Research Significance and Future Directions</td>
<td>65</td>
</tr>
<tr>
<td>References</td>
<td></td>
<td>68</td>
</tr>
</tbody>
</table>
**LIST OF FIGURES**

Figure 1  Related Kinds of Ontologies ................................................................. 5
Figure 2 Ontology learning layer cake................................................................. 7
Figure 3 SEURAT System Architecture.............................................................. 10
Figure 4 Top level of the Argument Ontology ................................................... 12
Figure 5 Classes for Argument Ontology and Frame for DRargOntology Class ... 13
Figure 6 subEntry Class ..................................................................................... 13
Figure 7 ontEntry Class ..................................................................................... 14
Figure 8 Instance of Root for Argument Ontology ............................................ 14
Figure 9 Alternating subEntry with ontEntry ..................................................... 15
Figure 10 Learning Web Service Ontology ......................................................... 18
Figure 11 Site Concept ...................................................................................... 19
Figure 12 SHUMI Architecture ....................................................................... 22
Figure 13 Terminology Extractor ...................................................................... 23
Figure 14 Relation Extractor ........................................................................... 24
Figure 15 Ontology Adaptation Architecture .................................................... 28
Figure 16 Mapping Domain Concept Trees ..................................................... 41
Figure 17 Sense 1 Airplane Semantic Net ......................................................... 42
Figure 18 Top level Concepts for Adapted Ontology (Point 9 test case) ............ 59
Figure 19 Independent Domain Concept Tree added to Adapted Ontology ...... 60
Figure 20 Merged Domain Concept Trees ....................................................... 61
Figure 21 Adaptation Methodology User Interface ......................................... 63
Figure 22 Evaluation Results from Adaptation Process ................................... 64
LIST OF TABLES/GRAPHS

Table 1 Performance Measures for Argument Ontology.................................................. 51
Table 2 Word Pruning Results using Z value for Relevance............................................. 52
Table 3 Performance Results for Selected parameters ..................................................... 53
Table 4 Before/After Adaptation –Change in Measures................................................... 58
Graph 1 Coverage vs Accuracy ........................................................................................ 54
Graph 2 Precision vs Recall ............................................................................................ 55
Graph 3 Precision vs #concepts ..................................................................................... 55
Graph 4 Recall vs #concepts ......................................................................................... 56
Graph 5 Accuracy vs #concepts ..................................................................................... 56
DEDICATION/ACKNOWLEDGEMENT

This thesis has been an inspiring, sometimes challenging, but always interesting experience. It has been made possible by many people, who have supported me. First of all, I would like to express my deepest sense of gratitude to my supervisor Dr Valerie Cross for her patient guidance, encouragement and for proofreading hundreds of thesis drafts without which this thesis would not have been possible. I am very grateful to my thesis committee members, Dr Janet Burge and Dr James Kiper for their valuable advice and suggestions. I am grateful to the other members of the ontology research group here at Miami University for their support; especially to Anindita Pal, Ramanathan Somasundaram, Janani Venkatachalam, and Yi Sun. I would also like to thank the OntoLT team for converting the text documents required for this thesis into the appropriate format. My two years at Miami University have been the best years as a student. I have learned a great deal from my professors and my classmates and have made friends for life. Finally, I wish to thank my parents for their continuous support and encouragement.
1. Introduction

Ontologies are an emerging means of knowledge representation that can improve information organization and management in an application. They capture the structure and semantics of an application (Modica 2002). They are used to capture domain knowledge in order to facilitate interoperability among different applications within a domain. They reduce terminological confusion by representing a common understanding of the concepts in the domain. Ontologies have also demonstrated their value in numerous application areas such as intelligent information integration or information brokering since they offer the technical support for sharing and exchanging information between human and/or software agents.

Despite their successes, their time-consuming and expensive development process hampers the wide-spread use of ontologies. Thus, a crucial concern is ontology construction. Ontology construction involves recognizing, representing, and recording concept definitions and the relationships among the domain concepts. In (Maedche and Staab 2001) two basic approaches for constructing ontologies are described. The first approach, appropriate for manual ontology engineering, incorporates natural language processing tools and ontology import tools into ontology editors such as Protégé (Noy et al. 2001) and WebOnto (Domingue et al. 1999). Machine learning and automated language-processing techniques are used in the second approach to extract concepts and ontological relations from structured data such as databases and unstructured data such as text documents. The second approach is implemented in some ontology construction tools; for example, OntoLearn (Navigli et al. 2003) uses machine learning and natural language processing techniques for automated ontology learning from domain text.

This thesis investigates existing methods for ontology learning that can be applied to the process of adapting ontologies from one domain to a related or similar domain. Existing software tools for natural language processing and ontology editing are integrated with thesis developed software to complete an adaptation software architecture. This software provides the user with a friendly interface that informs the user of the parameters that may be set for the major steps of the adaptation process and displays numerous reports necessary for the user to understand the results. This software tool may
be used to ease the creation of domain ontologies, which are essential to the success of the Semantic Web.

In addition to the developed software, this thesis produces an initial ontology to support design criteria for engineering design for the problem domain of spacecraft, i.e., the Engineering Design (ED) Criteria ontology. The starting point for the Engineering Design (ED) Criteria ontology is SEURAT’s Argument ontology (Burge 2005). The Argument ontology was developed to represent nonfunctional attributes for the domain of software engineering. The adaptation software is used to transform the Argument ontology into the ED Criteria ontology with a focus specifically on spacecraft design. The development of the ED Criteria ontology from the Argument ontology serves as the applied component of the thesis to examine how well the adaptation methodology and integrated software perform and to evaluate the resulting adapted ontology.

First a brief introduction to ontologies is provided and followed by an overview of approaches to constructing ontologies in Section 3. Next a brief introduction to the area of design rationale and the details of the Argument Ontology are presented in Chapter 4. Chapter 5 describes related work for developing domain ontologies. Chapter 6 describes the adaptation methodology and the components of its software architecture. Various design decisions in developing the software are also discussed. Ontology evaluation results using the various parameters are presented in Chapter 7. The user interface for the ontology adaptation software is discussed in Chapter 8. Chapter 9 presents conclusions and future research possibilities for improving the adaptation methodology and software.

2. Ontologies

In the field of philosophy (Bunge 1977), ontology is the theory of objects and their relationships. The field of computer science began using the word ontology after John McCarthy (1980) used it when suggesting that intelligent logic based systems need ontologies to record what exists within the system. Later the term ontology was used to mean a formal specification of a knowledge domain (Alexander et al. 1986). With the advent of the World Wide Web and the increased emphasis on its evolution into the Semantic Web, the use of ontologies is becoming increasingly popular for specifying a framework for common understanding of a particular domain, thereby enabling better
communication among users and software. Ontologies also help provide interoperability within an organization by providing a common set of terminologies. They are widely used in areas such as artificial intelligence, electronic commerce, medical domains, and information retrieval to provide a thorough and formal specification of a particular problem domain.

2.1 Definition of an Ontology

Gruber’s (1993) definition of an ontology is one of most often used definitions: “an ontology is an explicit specification of a conceptualization.” It is a formal specification of concepts pertaining to a domain, properties of these concepts that define features of the concepts also referred to as slots, and facets that restrict the values of the properties.

Maedche and Staab (2001) provide a more formal mathematical definition for an ontology. An ontology can be defined as a 5-tuple $O=\{C, R, H, f, A\}$. $C$ denotes Classes or Concepts. $R$ represents relationships. $C$ and $R$ are disjoint sets. $H$ is the class hierarchy or taxonomy where $H(c1, c2)$ means that $c1$ is a subclass of class $c2$. The $f$ function relates classes non taxonomically. The function $f$ takes in a relation and returns the pair of concepts linked by that relation, $f: R \rightarrow C \times C$. The first concept in the pair represents the domain of the relation, and the second represents the range. $A$ is the set of axioms for the ontology.

An ontology contains many kinds of relations among the concepts in the domain. The primary relation in an ontology is the subsumption relation or is-a relation. This kind of relation creates the hierarchical structure of the ontology and corresponds to $H$. An ontology may also contain other semantic relations $R$ among concepts, for example, the part-of relation. The ontology is taxonomic in nature if it only contains is-a relations and is a hierarchical structure where each child has only one parent class. If each child can have multiple parent classes, the structure becomes a directed acyclic graph. Associative relations specify relations across the sub–super concept structure and are domain specific. For example, a car “uses” gasoline or a penguin “lives-in” Antarctica. Axioms are the constraints placed on the instances or concepts. For instance, an axiom in molecular
biology could specify that nucleic acids shorter than 20 residues are oligonucleotides (Stevens et al. 2001).

The above-described components of an ontology make up the intentional ontology, or the definition of the ontology. The extensional ontology consists of the instances of the concepts and relationships. Instances are actual occurrences of the concepts and relationships. For example Miami University is an instance of the concept ‘university’. Dr. Cross is an instance of the concept ‘faculty member’. Dr. Cross teaches the course Artificial Intelligence. This example represents an instance of the ‘teaches’ relationship. The intentional ontology along with its associated extensional ontology is referred to as a knowledge base.

2.2 Types of Ontologies

There are many different approaches to classifying ontologies. One simple categorization described by Wikipedia is based on how easily an ontology can be understood by the computer. Using this approach, the two major categories are weak and strong. A paragraph of text describing an ontology is a weak ontology, as the computer cannot comprehend the semantics behind it. An ontology defined using a ontology language such as OWL is classified as a strong ontology as it is machine readable.

Guarino (1997) classifies ontologies according to their level of dependence: top level ontologies, domain ontologies, task ontologies and application ontologies. Figure 1 illustrates the relationships among these kinds of ontologies.

- Top level ontologies: They describe concepts that are not specific to any particular domain and may be used in other ontologies. For example, the concepts of space and time are applicable to many domains, tasks and applications.
- Domain ontologies: These ontologies describe concepts specific to a domain. These concepts may be the result of applying specialization to concepts in the top-level ontologies.
- Task ontologies: They describe a vocabulary related to a task by specialization of the vocabulary of the upper level ontologies.
- Application ontologies: Application ontologies are the most specific ontologies since they may be viewed as a specialization of domain and task ontologies.
Concepts defined in application ontologies typically represent roles performed by domain entities in accomplishing a particular task.

![Diagram of ontology hierarchy](image)

**Figure 1: Related Kinds of Ontologies (Guarino 1997)**

Another approach to ontology classification (van Heijst et al. 1997) is based on the amount and type of structure of the conceptualization and the subject of the conceptualization. The classification is determined by the level of complexity used to describe the concepts and the relations between them. The following categories are presented in increasing order of complexity:

- **Terminological ontologies** specify the terms used to describe the knowledge about a domain.
- **Information ontologies** specify the structure of concepts by identifying terms for the concepts but also associating properties with the concepts.
- **Knowledge modeling ontologies** represent the conceptualization of knowledge with a more complex format for the internal organization of the knowledge.

With respect to the subject of the conceptualization, the categories are similar to those of Guarino (1997): application, domain and generic which corresponds to the top level ontology. In addition there are representation ontologies, also referred to as meta-ontologies, that explain the conceptualizations underlying knowledge representation formalisms (Davis et al. 1993).
3. Learning Ontologies

The last two decades have witnessed widespread growth in the use of ontologies in different fields such as e-commerce and medical domains. Ontologies serve as the central component for the agent-based services over the WWW (Brewster et al. 2001). They help represent domain knowledge and facilitate a common understanding of and provide structure to information shared among people or agents over the WWW. These advantages have led to the more efforts to develop ontologies. One problem of building an ontology is that automated ontology construction tools provide limited support for knowledge acquisition (Celjuska et al. 2004). As a result many of the current ontologies that are in use have been created manually. Due to the enormous amount of time and effort to manually develop an ontology, research has been focused on integrating automated methods for ontology construction along with manual validation.

Ontology learning is a complex field that combines the fields of natural language processing, machine learning, and knowledge representation. Alexander Maedche and Steffen Staab (2001) classify ontology-learning approaches based on the source used to construct the ontology. They are broadly classified as ontology learning from 1) text, 2) machine readable dictionaries, 3) knowledge bases, 4) semi structured schemata, and 5) relation schemata. Although the approaches use different sources of input, they all construct a conceptualization of the domain. This thesis focuses on ontology learning from text corpura since that is one of the predominant approaches in the research literature (Gomez-Perez et al 2003).

Ontology learning consists of many subtasks that can be organized in the form of a layer cake (Buitelaar et al. 2005) shown in Figure 2. Ontology development mainly consists of identification of concepts and relations among those concepts in a domain. Identification of concepts requires the acquisition of linguistic information about the terms that refer to a particular concept. Relations are either hierarchical (is-a) or non hierarchical in nature. The ontology development process may also include methods by which rules may be defined. Rules are used to derive information that is not explicitly encoded in an ontology.
Several common approaches have been used to construct ontologies from text corpura (Gómez-Pérez et al. 2003). In **pattern-based extraction**, relations are recognized when a sequence of words in the text matches a particular pattern (Morin 1999). **Association rules** have been used in the field of data mining to discover information stored in databases. Given an existing concept hierarchy as background knowledge, association rules have also been used in order to learn non-taxonomic relations (Maedche and Staab, 2001).

The **conceptual clustering** (Bisson et al. 2000) approach groups concepts based on their semantic distance from other concepts in the domain. A variety of semantic distance measures exist (Cross et al. 2005) for use in conceptual clustering. **Concept learning** augments a given taxonomy as and when new concepts are acquired from the real world (Hahn et al., 2000). In **ontology pruning** (Kietz et al. 2000), a generic core ontology is used as the starting ontology and serves to create the top-level structure. Domain specific concepts are classified into this ontology by extracting relevant concepts from a dictionary specific to the domain of interest. Finally, using domain specific and general corpura, concepts judged not domain specific are removed from the ontology. The heuristic that specific concepts should be more frequent in a domain specific corpus than in general texts is used to eliminate the concepts not relevant to the domain.

Numerous examples using different approaches to constructing ontologies can be found in the research literature. SVETLAN (Chalendar and Grau 2000) exploits the contextual use of words regardless of their domain to learn categories of nouns from texts.
The Text To Onto environment (Maedche and al 2001) is an ontology learning framework including ontology import, extraction, pruning, refinement and evaluation techniques. ASIUM (Faure et al. 1998, Faure et al. 2000) uses a conceptual clustering approach but the clusters are not developed based on semantic distance. Instead, a syntactic parser examines technical texts and constructs an acyclic conceptual graph of clusters. The clustering is based on grouping words associated with the same verb after the same preposition. Formal Concept Analysis (Ciamiano 2005) defines the linguistic context of a term by using syntactic dependencies where the term is the head of a subject, of an object or of a complement with a verb. With Mo’K (Bisson et al. 2000) conceptual clustering methods can be designed, compared and evaluated to assist users in building an ontology.

The adaptation methodology developed for this thesis combines the ontology pruning techniques with the pattern-based extraction approaches. The adaptation approach is based on the reuse of ontologies, thereby, taking advantage of already existing knowledge resources. Ontology reuse can be defined as “the process in which available (ontological) knowledge is used as input to generate new ontologies” (Bontas et al. 2005). The existing SEURAT Argument Ontology serves as the starting top-level ontology. However, the first step used is pruning concepts that are not generic to engineering design. Then the resulting pruned ontology is enhanced with domain specific concepts for the design of spacecrafts using both a domain specific and general corpus. Since the starting point for the adaptation process is SEURAT’s Argument Ontology, the next chapter provides an introduction to design rationale, a brief description of the SEURAT system and the structure of the Argument ontology.

4. Introduction to Design Rationale

Design rational (DR) is not a new area of study. Researchers in the field of artificial intelligence have been investigating design rationale (DR) for some time now because of its usefulness in documenting the underlying justifications for decisions made during the design process. DR includes the decisions, the reasons behind each decision with their justification, other alternatives considered, and argumentation leading to the decision (Lee 1997). According to Stumpf (2005), DR is intended to capture the reasons why a
certain artifact is designed the way it is. In addition to providing support for future reflection upon the artifact, DR documents the process of designing and the choices made in the design context.

Designing any product or system is a complex process with numerous decisions to be made by the engineers and designers. These decisions must meet the requirements specified for the system. Requirements can be defined by many different people involved in the development of the system from users to those financially responsible for the development of the system. Because requirements often change and evolve over time, re-evaluation of the design is essential to determine the impact of the requirement changes on the design. This re-evaluation process may take considerable time and effort in order to determine the specific components of the design that are affected. Standard design documentation is useful in that it provides a clear description of the final system or product but it does not capture the thought processes and the decision making steps that led to the final design. When requirements change, the designer cannot use the final design document to help him realize, for example, that a particular alternative was rejected because the former requirements made that alternative not feasible. By recording the DR, the designer can easily re-evaluate the impact of the changed requirements and consider the previously rejected alternative.

Although DR provides many advantages, its use is not widespread primarily because many view the process of documenting all the decisions made with their justifications as well as rejected alternatives with reasons for rejection as too time consuming and expensive. As tools are developed to ease the burden of such documentation, more designers will begin to incorporate DR in the design process in order to take advantage of its benefits. The following section provides a brief overview of SEURAT, one such DR tool developed for the domain of software engineering. An objective of this thesis is to experiment with ontology learning techniques to be adapt SEURAT’s Argument ontology for the domain of software engineering to an Engineering Design Criteria ontology for the domain of spacecraft design.
4.1 SEURAT Overview

SEURAT (Burge 2005) was developed as a plug-in for Eclipse, an open source Java Integrated Development Environment (IDE) because of the ability to integrate the display of the rationale within the software development environment of Eclipse. Figure 3 below shows the primary components of the SEURAT system architecture: the Argument Editor and Analyzer, the Inference Engine, the Rationale Repository, and the Argument Ontology. The Argument Editor and Analyzer is the main controller and provides the user interface for the system. It accesses the rationale repository that stores the rationale documented for various software design projects. The Argument Editor and Analyzer uses the inferencing engine to perform inferencing capabilities over this rationale. The Argument Ontology provides a common vocabulary for specifying the rationale used in the design decision-making process.

![Figure 3 SEURAT System Architecture (Burge 2005)](image)

Early on researchers suggested defining ontologies for design rationale (Gruber et al. 1991, Bradshaw et al. 1992). More recently tools have begun using ontologies (Bevo et al. 2003, Nkambou 2004, Pereira et. 2005) to support the design and reuse of software. SEURAT references the Argument ontology for high level non-functional qualities of a system such as usability, scalability, flexibility and maintainability. This ontology
represents relationships between qualities at different levels of detail. These non-functional qualities specify the overall qualities of a system; whereas, the functional qualities refer to a specific functionality. For example, research at Carnegie Mellon University (1998) developed a quality measures taxonomy with the following high level categories: needs satisfaction measures, performance measures, maintenance measures, adaptive measures and organizational measures. Others such as Bruegge and Dutoit (2000) classify them into five major groups: performance, dependability, cost, maintenance and end user criteria. The SEURAT system (Burge & Brown, 2004) uses an Argument Ontology that gives common arguments behind software design and development at different levels of abstraction. Since a primary component of the thesis research is to transform SEURAT’S Argument Ontology into the ED Criteria Ontology, the following section provides a more detailed description of the SEURAT Argument Ontology.

4.2 SEURAT’s Argument Ontology

The Argument Ontology is a hierarchy of reasons or motivations that can be used to support a decision (“arguments” for the decision) over several choices in the software design of a system. At the top of the hierarchy are the more general or abstract reasons. As one proceeds down the hierarchy, these reasons are refined with more specific reasons. The leaves of the hierarchy represent the most detailed reasons (Burge, 2005). The hierarchy contains a vocabulary of the non-functional characteristics relevant to the domain of software engineering.

At the top level the non-functional criteria categories are Adaptability, Affordability, Dependability, End User, Needs Satisfaction, Maintainability, and Performance. At the next level down, for example, under Affordability are more specific kinds of costs such as development cost, deployment cost etc. The adaptability criterion refers to how easy it is to modify the software to deal with new circumstances. Examples of more specific adaptability criteria include portability and scalability.

Although the Argument Ontology is hierarchical, entries may fall under more than one category. For example, throughput criteria is associated both with affordability criteria and performance criteria. The argument ontology is extensible so that new
argument types or criteria may be easily added. Figure 4 below shows a snapshot of the Argument Ontology from SEURAT’s RationaleExplorer view.

The SEURAT’s Argument Ontology has an XML schema representation. This XML schema is imported into Protégé, an ontology editing tool and then stored as a Protégé project. The following describes the structure of the Argument Ontology using the Protégé frame-based knowledge representation format. The three kinds of classes that are pertinent to this research are the argOntology, the subEntry, and the ontEntry which are listed in Figure 5 on the left hand side. Figure 5 also shows the frame structure for the argOntology class. The subsequent Figures 6 and 7 show the frame structure for the subEntry and the ontEntry classes, respectively. Note that the argOntology class allows for multiple instances of ontEntry for the ontEntrySlot. One can view an instance of the argOntology class as the root of the Argument extensional ontology from which the top level instances for Adaptability, Affordability, Dependability, End User, Needs Satisfaction, Maintainability, and Performance descend. Figure 8 shows the root instance of the Argument extensional ontology with 7 subEntry pointers for these top level criteria.
Figure 5 Classes for Argument Ontology and Frame for DRargOntology Class

Figure 6 subEntry Class
The design of the Argument Ontology has an alternation of ontEntry instance which points to one or more subEntry instances. Then each subEntry instance points to an ontEntry instance which point to one or more subEntry instances, and so on. Figure 8 shows that ontEntry instances may point to multiple occurrences of a subEntry but that a subEntry can point to only one instance of an ontEntry. Figure 9 shows an example of the alternation for the one of top-level criteria and one second level criteria.
Figure 9  Alternating subEntry with ontEntry

From Figure 8, the first subEntry points to subEntry_131. The top left corner of Figure 9 shows subEntry_131 points to ontEntry_243 whose name is Affordability Criteria which has several subcategories as seen in the list contained in the subEntrySlot. The first one in this list points to subEntry_65 which then points to ontEntry_96 whose name is Development Cost. Although this XML schema works for the Argument Ontology, it adds an extra layer of complexity that is not necessary. The class definition for an ontEntry could be modified to include a slot parentOntEntry with type of multiple instances of ontEntry to be used to point to its parent(s). The subEntry instances could be eliminated by having a slot childOntEntry with type of multiple instances of ontEntry to be used to point to its children. This design would be easier to work with during the specialization step of positioning concept trees into the Argument Ontology.

The XML representation format and its associated data can be converted into a series of database tables. The relational database schemas for the Argument Ontology only use two tables, one is the ontEntries table and the other is the ontRelationships table. This
thesis research does not use the database representation but instead converts the XML representation into the Protégé frame-based representation before performing the pruning and specialization operations on the Argument Ontology.

Beginning with the high level types of criteria, the Argument ontology refines the criteria to those that are quite specific to the software engineering domain. The design criteria at the higher levels such as Affordability and Performance are applicable to other design domains such as engineering and can also be applied to even more specific domains such as aerospace and in particular, spacecraft design. The ontology adaptation architecture extends the Argument Ontology by adding criteria specific to the spacecraft design domain. These domain specific concepts and simple taxonomic relations are extracted from text documents using ontology learning techniques. The following section describes related work in ontology learning, some of which has been used to motivate the development of the Ontology Adaptation Architecture

5 Developing Domain Ontologies Using Linguistic Techniques

The importance of domain ontologies has been increasing with the advent of the Semantic Web. Building domain specific ontologies is challenging because of the large number of concepts within a domain, their highly specific semantics, and the large number of relationships between them. Identifying the relevant concepts and their relationships manually takes considerable time and effort.

The aim of this research is to develop an ontology adaptation software architecture which takes an existing ontology in one domain and transforms it into one for a related domain. The standard ontology learning techniques have been predominantly focused on learning from text documents and use linguistic techniques; therefore, the ontology adaptation software architecture relies on integrating existing ontology learning techniques. In the following sections several existing software tools either that are used for linguistic analysis or that learn from text using linguistics techniques are described. The ones selected had the most easily available and readable documentation and articles. Several of these provided ideas and approaches used in developing the ontology adaptation software architecture.
5.1 GATE

One of the most widely used tools in the field of Language Engineering is GATE (Generalised Architecture for Text Engineering). GATE is a framework for developing and deploying software components that process human language (Cunningham et al 2002). It is used in several different systems for ontology learning.

GATE follows a modular architecture that allows plugging in different NLP software such as POS (part of speech) taggers, sentence splitters, Named Entity Recognizers, etc. The components are organized as different resources. Two of its main resources are:

- Language resources: Language resources are the data only resources. They include text corpora, lexicons, thesauri and ontologies. GATE supports different document formats such as HTML, SGML, XML and plain text.
- Processing resources: The processing resources include the different parsers and generators, which constitute the algorithmic components of GATE. For example, the POS tagger is characterized by the process it performs on the text. Processing resources usually contain a language resource. For instance a word sense disambiguator contains a dictionary.

The processing resources can be embedded into applications and the sequence of execution can be controlled through pipelines. GATE supports various types of pipelines. A simple pipeline groups a set of processing resources and executes them. Corpus pipelines are specific to the processing resources that are applied to documents. The pipeline opens each document in the corpus and runs all the processing resources on the corpus and closes the document.

The information extraction component of GATE is called ANNIE (A Nearly New Information Extraction system) (Cunningham et al. 2002). It is made up of different processing resources such as an English tokenizer. The tokenizer splits the text into tokens such as words, punctuations, symbols and numbers. Sentence splitters are a cascade of finite state transducers that segment the text into sentences. It tags the break of a sentence such as a full stop with a split. GATE also has a processing resource called JAPE (Java Annotation Pattern Engine) (Cunningham et al. 2000) that allows the user to specify regular expressions over the annotations. The following section describes an application of the GATE software.
5.2 Learning Web Service Ontologies using GATE

For successful use of Semantic Web services, domain ontologies must be of high quality since descriptions of these services are created using domain ontologies. The approach suggested by Martha Sabou (2005) describes an automatic extraction method to learn domain ontologies from textual documentation attached to web services.

The motivation behind this approach is that textual information regarding web services contains useful information in order to build an ontology. The approach followed here leverages the sublanguage nature of the texts. A sublanguage is a specialized form of natural language which is used within a particular domain or subject matter and is characterized by a specialized vocabulary, semantic relations and syntax (e.g., weather reports, real estate advertisements) (Sabou 2005). The extraction process exploits the syntactic regularities that are present in the web service documentation. The overall procedure followed is shown below in Figure 10 with each of the following steps described below.

![Figure 10 Learning Web Service Ontology (Sabou 2005)](image)

1. The first step is POS tagging. Here the text corpus is annotated with Parts of Speech information. The entire preprocessing phase is performed using the GATE framework.
2. Surface patterns are then applied on the processed corpus to extract linguistic information that may be useful to the ontology development process. These surface patterns rely on surface information such as the position of words relative to other words to extract meaningful information. Surface patterns are implemented using the
JAPE preprocessor. It is a regular expression based rule mechanism that identifies patterns based on a set of rules. For instance, a surface pattern that extracts concepts looks for noun phrases as concepts which are usually represented by nouns in sentences. The following JAPE rule
\[(\text{DET})*(\text{ADJ} | \text{NOUN} | \text{POS})* \text{NOUN} \]:np-- >:np.NP={}
annotates a phrase that starts with 0 or more determinators, adjectives, nouns or possession indicators ending with one noun as a noun phrase.

Functionalities offered by a web service are usually represented by verbs. A noun that immediately follows this verb is the data structure that is involved with that functionality. A JAPE rule that identifies functionalities is
\[(\text{VB})\{\text{NP}\} \]:funct -->:funct.Functionality = {}

3. The next phase is the ontology building phase where the information obtained from these patterns are converted to ontology constructs such as concepts and relations, and the hierarchy is created. The terms extracted from the previous phase are used to build the Data Structure hierarchy and the Functionality Hierarchy. The hierarchy building algorithm is based on the idea of compositionality. If a term A is a proper substring of term B then A is more generic than B. In this case, a subsumption relationship is created. The ontology building phase may add concepts that do not appear in the corpus. For example, if concepts B and C have a common substring, it is assumed that this string represents a valid domain concept even if it does not appear as a stand alone term in the corpus. The functionality hierarchy is constructed by either including the data element along with the functionality (e.g. BuyTicket) or by just including the functionality (e.g. Aligning) without specifying the noun concept. An example of a learned concept and its subconcepts is the site concept displayed in Figure 11.

![Figure 11 Site Concept (Sabou 2005).](image-url)
4. These concepts are pruned in the next stage to make the ontology more specific to the domain. One of the heuristics used here is concepts below the average frequency of terms are considered irrelevant and are pruned. Another heuristic used is if a low frequency data structure concept appears in the functionality hierarchy, it is considered important and will not be pruned as long as the functionality concept is not pruned.

The term extraction phase is then evaluated using qualitative and quantitative measures. Evaluation is done at the term level using quantitative measures such as precision and recall (Sabou 2005). Recall is the ratio of relevant terms that are extracted from the analyzed corpus (Correctextracted, which was determined manually) over all the terms that should be extracted from the corpus (allcorpus which was determined manually) Precision is the ratio of correctly extracted terms over all extracted terms (allextracted).

\[
\text{TRecall} = \frac{\text{Correctextracted}}{\text{allcorpus}} \\
\text{TPrecision} = \frac{\text{Correctextracted}}{\text{allextracted}}
\]

Expert evaluation is performed to determine the overall goodness of the constructed ontology. The expert performs a concept per concept evaluation of the learned ontology. The goodness is measured using a metric known as ontology precision defined as

\[
\text{OPrecision} = \frac{\text{correct} + \text{new}}{\text{correct} + \text{new} + \text{spurious}}
\]

The expert rates a concept as correct if it is included in the gold standard ontology manually built by a domain expert and hence considered relevant. Concepts that are considered relevant by the domain expert but were not included in the gold Standard ontology originally are rated as new. All irrelevant concepts are rated as spurious.

Domain coverage on the other hand is more subjective. It is done by comparing the ontology to a gold standard ontology. Ontology comparison can be done at the lexical and the conceptual level. This comparison is done at the lexical level by comparing the terms that represent the concepts. The lexical overlap metric is used to evaluate the domain coverage. It is defined as the ratio of number of concepts shared by both ontologies and the number of all gold standard ontology concepts.

\[
\text{LO}(O1,O2) = \frac{|L_{O1} \cap L_{O2}|}{|L_{O2}|} = \frac{\text{correct}}{\text{all}}
\]

The other metric used is known as ontological improvement. This metric is basically used to measure any important additions to the manual ontology by analyzing concepts that
were ignored during the manual creation but added during the learning process. It is defined as the ratio between all domain relevant extracted concepts that are not in the gold standard ontology and all the concepts of the gold standard ontology.

\[ OI(O1,O2) = |L_{o1} - L_{o2}| / |L_{o2}| = \text{new} / \text{all} \]

A central component of this approach is the pattern based extraction technique that worked well with respect to the kind of documents being used to describe web services such as javadoc documentation of all methods offered by APIs or a detailed service description consisting of a short description, detailed information about command line arguments, and its relations to other services in the collection.

5. 3 SHUMI

SHUMI (Support for Human Machine Interaction) (Pazienza et al 2004) is a tool that acts as an “intelligent advisor” to spacecraft designers. A large amount of information that may be useful to designers is represented in text documents in the form of natural language. The intelligent advisor interprets the user’s requests and searches knowledge repositories to extract the information. The system stores the knowledge relevant to the design project in a Mission ontology. Hence the mission ontology can represent how the project contributed to the systematic representation of knowledge about the space missions. It is also used to index documentation gathered during the mission. Documents that describe the mission details are used as source documents to extract the conceptualization of the mission.

5.3.1 Overview of Learning the Mission Ontology

The approach uses both semantic resources such as Word Net as well as a set of domain specific corpus. The ontology extraction process consists of two phases: Term Extraction and Relation Extraction. The ontology building starts by identifying significant terms using natural language tools. The results of the natural language processing tools are validated by a domain expert. The terms deemed relevant by the domain experts are used to build or increment the domain concept hierarchy.

The next step in enriching the ontology is extracting relational concepts. Relational Concepts are semantic relations between terms. They usually represent events
such as “satellite reaching orbit”. These are extracted by looking for surface forms which are a form of verb phrase. A surface form that expresses the relational concept may be “The satellite gets close to the orbit”. As done with term extraction, these surface forms are validated by a human expert and then added to the domain concept hierarchy along with the associated terms and concepts. The overall architecture is shown below.

![Figure 12 SHUMI Architecture (Pazienza et al 2004)](image)

5.3.2 SHUMI Term Extraction
Terminology extraction consists of 4 modules:

1. Preprocessing modules: The preprocessing module takes in a set of documents which are in ASCII text format and converts them to XML which is understandable by the syntactic parser. There may be a phase before this where text documents are in different formats such as Html and PDF and converted to common ASCII text format. The preprocessing module is also responsible for syntactic analysis such as identification of paragraphs and other entities.

2. Parsing modules: The parsing module uses the CHAOS parser (Pazienza 2004) to perform syntactic analysis on the XML files. This parser has modules such as POS tagging, tokenizer, morpho-analyser and Name Entity (NE) recognizer.

3. Terminology extraction module: The extraction module extracts surface forms based on a sequence of rules. These syntactic rules are expressed using the Brown Corpus Tag Set (Green and Rubin 1981). For example, a surface form of the type JJ NN(Adjective followed by common noun) is considered a possible
term. The term “lunar mission” satisfies the above rule and is added as a concept in the ontology.

4. Terminology sorting module: This module sorts the surface terms based on the frequency with which they appear in the corpus.

The figure below shows the architecture for the Terminology extractor (Pazienza 2004)

![Terminology Extractor](image)

**Figure 13 Terminology Extractor (Pazienza et al 2004)**

The extractor has the ability to extract complex terms along with generalizations of these terms through the use of named entities. Local hierarchies among terms can be constructed by using this ability and can be integrated into the mission ontology. For instance the term “entity#ne#_orbit” can be structured as a generalization of terms “planet#ne#_orbit” and “spacecraft#ne#_orbit” in the hierarchy. The development of such hierarchies can be aided using semantic measures with WordNet.

The terms extracted are validated by an expert. The interface provides the term along with all its occurrences within documents to the expert. The expert then decides whether to accept or reject the term. The term that is shown to the expert is either a simple word or a semantically generalized form known as Named Entity. A named entity is a generalization of a place or a person that represents important information in the domain.
5.3.3 SHUMI Relation Extraction

The architecture for the relation extractor (Pazienza 2004) is shown below:

![Figure 14 Relation Extractor (Pazienza et al 2004)]

The relation extraction module consists of 3 sub models:

1. The Parsing Module invokes the CHAOS modules using as input the text which consists of the corpus documents in XML formatted files produced by the pre-processing modules of the Term Extractor. The verb shallow analyzer VSA is the primary CHAOS submodule used since it finds the syntactic arguments of the verbs.

2. The Surface form extraction module analyzes the parsed text and extracts the verb phrases. These verb phrases are known as surface forms. Sentences are represented as verbs and arguments. The arguments denote the lexical form in the sentence.

   approach((SUBJ, the spacecraft), (OBJ, the orbit), (IN, ten minutes))

All sentences in the corpus are represented in this format and are passed into the sorting module.

3. The surface form sorting module generalizes the verbs with the arguments to generate surface forms. These surface forms are then ranked based on the frequency of their appearance in the corpus. This set of surface forms is then validated by an expert.
5.4 OntoLT

OntoLT (Buitelaar and Olejnik 2004) is a plug-in developed for Protégé that supports extraction of ontologies from text corpora. It differs from other software tools for ontology construction in that an ontology engineer is able to bootstrap or extend domain-specific ontology from a relevant text collection or corpus. OntoLT provides a more direct connection between linguistic structure and ontological knowledge. OntoLT provides mapping rules through which concepts and attributes are extracted from text corpura.

The mapping rules are defined using a precondition language which permits a mapping between linguistic entities in text and class/slot candidates in Protégé. The first phase of ontology extraction is linguistic annotation of the text corpus. This phase is not integrated into OntoLT. It is provided by SCHUG (Declerck 2002), a rule based system for English and German that annotates parts of speech, morphological inflections, and phrase and dependency structure.

OntoLT then maps these linguistic structures to concepts and relations in an ontology. This correspondence is established by using a set of predefined mapping rules that maps linguistic structure to the ontological knowledge, thereby determining how the ontology is created. The mapping rule is defined using a precondition language such as XPATH (Clark et al. 1999). The constraints that the mapping rule specifies are represented using XPATH expressions. If all the constraints specified by a rule are satisfied, the operators that extend the ontology get invoked. OntoLT has a few predefined rules such as:

- **HeadNounToClass_ModToSubClass**: This rule maps the head noun of a sentence to a class of the ontology and the modifiers of the head noun to sub classes in the ontology.

- **SubjToClass_PredToSlot_DObjToRange**: This rule maps a linguistic subject to a class, predicate to the slot of the class and direct object to range of the slot

In addition to these existing rules the user can define their own rules. OntoLT creates the ontology by using a set of operators such as:

- CreateCls: creates a new class
- AddSlot: adds a slot to a class
- CreateInstance: introduces an instance for a class
FillSlot: sets the value of a slot of an instance
If a rule’s condition is satisfied, the corresponding operator in the conclusion of the rule is invoked.

Developing an ontology from a text corpus requires a methodology to extract the most significant words from the text corpus. OntoLT uses a statistical preprocessing step to select the most relevant words in the domain. This uses a chi square function (Agirre 2001) to select words that are considered domain relevant. The function computes a relevance score by comparing the frequencies in a domain corpus with the frequencies in a reference corpus. Experiments showed that absolute frequency is also an important factor in relevance. Hence the chi square score is multiplied by the absolute frequency to get a combined measure of frequency and relevance.

6. Adaptation Methodology:

There is a growing interest in ontologies for the Semantic Web. However, construction of a domain specific ontology from scratch is a bottleneck as this process requires a great deal of time and effort. This problem gives rise to the need to use and adapt already existing ontologies to the needs of the application domain. The available ontologies may not represent the specific domain of interest but cover related domains. This thesis develops a framework for adapting an ontology from one domain to a related domain. Specifically, the Argument ontology of SEURAT for the software engineering domain is used as the basis for adaptation to an Engineering Design (ED) Criteria ontology for the domain of spacecraft design. This research assumes that many important concepts related to this domain are described in documents describing the engineering design of spacecraft. The proposed approach requires several steps:

1) Pruning: Removing from the existing Argument ontology the concepts and relations that are not relevant to engineering design.
2) Adapting and Specializing: Adding new concepts to transform the existing ontology into the ED criteria ontology. Concepts specific to the engineering domain are obtained from a design document corpus by using an existing ontology learning software OntoLT (Buitelaar et al. 2004) to produce concept trees. Additional
software developed as part of this thesis positions the concept trees appropriately within the pruned Argument ontology.

3) Evaluating the adapted ontology for its usefulness to the domain of engineering design for spacecrafts. Several approaches to ontology evaluation are described below in Section 6.4. Their use in evaluating the ED criteria ontology is also discussed.

The adaptation architecture shown in Figure 10 uses several resources and existing software as well as the integration of software developed for this thesis which is implemented as a Protégé plug-in. The various components of the methodology are discussed in detail below.

6.1 Input Resources

Numerous data are needed as input to the adaptation process. The base input to the adaptation process is the ontology which is to be transformed from one domain to a related domain. A training corpus and general corpus are needed for first pruning the base ontology and then extracting domain specific concepts to add to the pruned ontology. The general corpus is needed to apply the heuristic that specific concepts should be more frequent in a domain specific corpus than in a general corpus. The test corpus is used in the evaluation of the resulting adapted ontology. The following sections describe these input resources in more detail.

6.1.1 Argument Ontology

The Argument ontology stores the hierarchy of common argument types used in the design rationale as instances of various concept schemas. The ontology has 280 instances which represent non-functional attributes such as Affordability Cost, Usability, Scalability, Flexibility and Maintainability. These attributes are instances of the DR:ontEntry concept. The value of the name slot for instances of DR:ontEntry specifies the non functional attribute. The most general terms are maintained at the root and the most specific terms are maintained at the leaves. The ontology can be traversed by looking up the DR:subEntrySlot value of the DR:ontEntry class which stores a link to an instance of the DR:subentry class. The DR:ontoEntrySlot of the DR:subentry class points
to the child of that instance. As previously discussed, this introduces extra levels within the extensional ontology.
6.1.2 Domain Corpura

Two domain corpura are required, the domain training corpus and the domain test corpus. The adaptation methodology uses the same domain training corpus for both pruning and merging in order to produce the adapted ED ontology. The domain test corpus is used to evaluate the performance of the adaptation methodology.

The training corpus was created by selecting documents from the NASA Technical Reports Server (NTRS). NTRS (http://ntrs.nasa.gov/) is an experimental service that provides users with the ability to search many different kinds of documents such as research reports, journal articles, conference and meeting papers, mission-related operational documents, and even preliminary data. NTRS documents are unlimited, unclassified, and publicly available.

Although several efforts were made to secure domain experts from NASA to help create the training corpus, these were unsuccessful. For purposes of this research, documents were selected from those available through NTRS. The first selection criteria used a simple search on the keywords ‘engineering’, ‘design’, and ‘criteria’. This search returned over 250 abstracts with the keywords highlighted. These abstracts were next screened as to whether a digital version was electronically available. If a viewable pdf version existed, then the document’s abstract was first carefully read to determine if the document appeared to contain engineering design information. If so, then the pdf was viewed to see more details and search for references to design criteria. Based on this screening process, 25 documents were selected for the training corpus with a size of 1.05 MB. These documents were then manually converted to text files with section headings removed and chunked at 1000 words each so that they could later be used as input to the Lucene and SCHUG software described below.

The test corpus was created in a manner similar to the training corpus only it was created about three months after the initial training corpus documents were selected. Updates to the NTRS provided new documents that fit the selection criteria so that 20 additional documents were selected for the test corpus that did not overlap with those used for the training corpus. The size of the test corpus was kept close to the size of the training corpus as recommended in the research literature. The test corpus consists of 22 files with a size of 1.30 MB for text only.
6.1.3 General Corpus

Ontology evaluation approaches often use a generic or “neutral” corpus representing general language not specific to the domain of interest. This corpus is used in the adaptation methodology for determining whether a term in the training or test corpus is domain specific. For this thesis, the Reuters-21578 text categorization test collection (http://kdd.ics.uci.edu/databases/reuters21578/reuters21578.html) is used for the general corpus. It is a resource for research in information retrieval, machine learning, and other corpus-based research and is one of the most widely used test collection for text categorization research. For the task of text categorization, a text is classified as belonging to any of a set of pre-specified categories or possibly none. This task may be used as a generic step in NLP systems, for example, to index documents for later retrieval. This collection consists of 22 files with a size of 15 MB. Each of the first 21 files (reut2-000.sgm through reut2-020.sgm) contain 1000 documents, while the last (reut2-021.sgm) contains 578 documents.

The general corpus is used in the adaptation methodology to determine the list of significant words from both the domain training corpus and the domain test corpus. The general corpus is used to calculate the degree of specificity of domain relevant terms with respect to the general purpose terms. The details of how the significant word lists is used in the adaptation methodology are provided in the descriptions of the algorithms found in Section 6.3 and the discussion of the evaluation of the adapted ontology in Section 6.4.

6.2 Software Resources

The adaptation architecture is composed of several independent software packages that have been integrated with software written for this thesis to accomplish the pruning and specialization steps of the adaptation process. Additional software was developed for the evaluation process.

6.2.1 Apache Lucene

The vector-based pruner approach (Volz et al. 2003) was investigated to identify the different lexicalizations that represent a particular concept, thereby calculating the frequency of the concept in the corpus. However, the vector based approach has its
disadvantages as it cannot identify compound words or phrases. In order to recognize phrases it was necessary to investigate other software packages. In (Volz et al. 2003) the TRIE based pruner is described. The TRIE based approach is based on the usage of the TRIE data structure (Fredkin 1960) to store information. The disadvantage of this approach is inefficiency in storage space.

The GATE (General Architecture for Text Engineering) software package (Cunningham et al. 2002) was investigated for identifying lexicalizations in the text corpus. GATE is a framework for developing and deploying software components that process human language. This system is quite complex and provides the capability of embedding its processing resources into applications and controlling the sequence of execution through pipelines. After more study, it was determined that GATE’s sophistication and complexity made it difficult to understand and integrate into existing and developed software for the adaptation architecture. Instead, another package simpler to understand and to use, Apache’s search engine library Lucene (http://lucene.apache.org/java/docs/index.html) was selected for the process of identifying compound words.

The text training and test corpus are parsed using Apache Lucene. This text search engine library is Apache’s open source project written in Java. Since the plug-in software for Protégé is written in Java, Lucene could be easily integrated into the adaptation methodology plug-in for the Protégé editor. Lucene has powerful search features that can be used through simple API’s.

A Lucene index is first created and the list of documents to be parsed is added to this index. Next an enumeration of all the terms present in this index is constructed by using the terms() method of the IndexReader class. This enumeration may be restricted to include only terms that belong to the “contents” portion of the documents and not any other section such as title, etc. by passing “contents” as a parameter to the terms() method. Another interface called TermDocs provides an the enumeration <document , frequency> pair for each term. Here a document id denotes the document in which the term appears. The frequency is the number of times that term appears in the document. Using the frequency from the <document, frequency> pair for the term frequency of term j in a document d_i, that is, tf(i, j) and counting the number of documents in which term j
appears, \( df(j) \), the TFIDF value for that term is computed. The TFIDF measure as described above is

\[
\text{TFIDF}(i,j) = \text{tf}(i,j) \times \log \frac{N}{df(j)}
\]

The term and document frequency of each term in the corpus is calculated using the API’s of Lucene using the index. In word pruning an enumeration of all terms present in the index is used to iterate through each term in the corpus. This enumeration may be restricted to include only terms that belong to the “contents” portion of the documents and not any other section such as title, etc. The term frequency is calculated along with frequency of its corresponding synonyms using JWordNet to access its synset. A total frequency is determined for a concept by adding up the frequencies for all the terms in the synset associated with the concept. In phrase pruning a slightly different approach is used since it is not possible to know what phrases to compute the frequencies for. First all the instances of the Argument ontology are examined and the phrase and document frequencies of only these phrases are determined by querying the corpus with the API’s of Lucene. A more detailed description of estimating the term frequency/inverted document frequency is described in section 6.3.2 on pruning.

Lucene supports enhanced searching capabilities using wildcards. The symbol for single character wildcard is \( ? \). For example, to search for ‘text’ or ‘test’ we could use \( te?t \). The symbol for multi character wild card is \( * \). Lucene also has to ability to perform fuzzy searches based on the edit distance algorithm. The \( ~ \) symbol is used for this. For example while searching for terms with similar spelling to “tame” we could use tame~. This would also return the word ‘lame’. Boolean operators such as OR, AND, NOT can also be used in queries to look for multiple terms.

### 6.2.2 JWordnet

JWordnet is a java interface to the Wordnet library. It is used to retrieve information from Wordnet, an English Language electronic dictionary accessible from the internet ([http://www.seas.gwu.edu/~simhaweb/software/jwordnet/](http://www.seas.gwu.edu/~simhaweb/software/jwordnet/)). WordNet (Miller et al. 1990) was developed by the faculty and students at Princeton University. WordNet forms a taxonomy or hierarchical ordering of the English language with more general concepts higher in the hierarchy and more specific ones lower in hierarchy. The primary
relationships are the hypernym (parent) and hyponym (child) relationships. For example, flower is the hypernym of rose and rose is the hyponym of flower. Instances of these relationships form the taxonomic structure of WordNet. Another important WordNet component is the synset. The synset class represents a concept and contains words which have a sense that names that concept (synonyms). For example, deadlock has different synonyms such as standstill, stalemate, etc. The related words are stored in a structure called the synonym set.

The DictionaryDatabase interface is used to access the WordNet ontology. Relevant synonyms of a term are obtained using the IndexWord class. The lookupIndexWord() method of the DictionaryDatabase interface is used to look up a word in the database. This method takes in a POS, i.e. noun or verb, as an argument to filter words only to that particular part of speech and a lemma parameter which is an orthographic representation of a word, i.e. the lexical occurrence in the corpus. For example if we are looking for all words which are synonyms of the word ‘Data’ and are nouns, then noun would be the parts of speech parameter and the term ‘Data’ would be the lemma parameter.

JWordnet is used along with Apache Lucene to estimate the term and document frequency of a term and its synonyms appearing in the text corpus. For example if the frequency of the term “avoids” is 10 and the frequency of the term “prevents” is 4 then the overall frequency for the concept avoids-prevents should be 14 as both these terms represent the same concept.

6.2.3 SCHUG

SCHUG (Shallow and CHunk-based Unification Grammar tools) (Declerck 2002) is a rule based system for English and German that provides annotation of parts of speech, morphological inflections, and phrase and dependency structure for a text file. SCHUG assigns phrasal categories and grammatical functions to the textual material it processes. For example, a noun phrase (NP) represents a phrasal category. A NP has several grammatical functions such as subject, direct object, and indirect object.

SCHUG takes plain text documents as input and outputs the annotated file in XML format. It does not support PDF files. Also the plain text input to SCHUG has to be of small size as it only processes small text files with around 1000 tokens.
6.2.4 OntoLT

OntoLT has been described in section 5.4. Here its use in the Ontology Adaptation Architecture is discussed. OntoLT accepts the annotated files from SCHUG as input and based on the defined rules produces concept trees. The default rule \textbf{HeadNounToClass} is used to produce the concept trees from OntoLT.

Extending an ontology requires an open existing ontology within Protege to which OntoLT may add additional concept trees. OntoLT adds these as new concept trees within the existing ontology and does not merge them within the existing trees. Additional software was written (see section 6.3.3 that describes the algorithm) to allow merging of these concept trees into the Argument extensional ontology. One of the difficulties encountered is the concept trees produced by OntoLT represent classes for an intensional ontology. The Argument Ontology, however, is extensional and contains instances of subentry and ontEntry classes as described in section 4.2. The instances of ontEntry actually name and describe the design criteria. The merging solution had to convert the intensional concepts in the OntoLT concept trees into instances of ontEntry in order to merge them into the Argument Ontology.

Although OntoLT has a statistical preprocessing step for determining word relevancy as previously described, it required different formatting for the corpus to use this step and the adaptation process already employed statistical significance testing between two proportions in the pruning phase. This approach is also used in the merging process of the specializing phase. Calculation of the statistical Z value for comparing proportions is explained in section 6.3.1. A tree’s Z value is the average of the Z values of all its nodes. Trees that satisfy a particular threshold Z value are only used in the adaptation process. Typical thresholds for the Z value are 1.64 for 5% or 2.32 for 1% significance.

6.3 Algorithms

The ontology adaptation architecture consists of various existing software packages integrated with software developed as part of this thesis. This section describes the various algorithms used to implement the ontology adaptation methodology.
6.3.1 Determining Domain Relevance

In the following algorithms, methods to determine whether a word and/or concept are relevant to the domain are needed. Two approaches have been implemented for determining domain relevance: the significant words list and the term frequency/inverted document frequency measure.

The method used to determine a set of words that differentiate the domain corpora with respect to the general corpus is based on the approach taken in (Spyns et al. 2005). The assumption made is that with respect to technical texts, their specialized vocabulary makes up the majority of the distinguishing vocabulary, especially, if the general corpus is a collection of broad-spectrum newspaper articles.

First, the relative frequency $f_{rel}$ of a word with respect to the corpus is determined as:

$$f_{rel} = (f / N) * 100$$

with $f$ being the absolute frequency of a word in the corpus and $N$ being the total number of words in the corpus. Then the $Z$ value is calculated using the difference between the word’s relative frequency in the domain corpus $f_{relD}$ and in the general corpus $f_{relG}$:

$$z = (f_{relD} - f_{relG}) / \sqrt{f_{relD} *(100-f_{relD}) / N_D + f_{relG} *(100-f_{relG}) / N_G}$$

The above equation is based on the formula for testing the differences of two proportions. Basically the null hypothesis is that $f_{relD} = f_{relG}$. The alternative hypothesis is that $f_{relD} > f_{relG}$. A word is considered not significant if the value of $Z$ is less than 1.64 (for 5%) or 2.32 for 1% .

The list of significant words is determined for both the training domain corpus and the test domain corpus. The training domain significant words list is used in the pruning algorithms as described in the following section and the test domain significant words list is used in the evaluation of the adapted ontology as described in section 6.4.

The term frequency-inverted document frequency measure (Song et al. 2005) is used to determine relevant concepts. Term frequency refers to the number of times a word occurs in a document. The term frequency/inverted document frequency also penalizes words which occur too often in different documents. It is specified as

$$TFIDF(i,j) = tf(i,j) * \log (N/df(j))$$

where $tf(i,j)$ is the term frequency of term $j$ in a document $d_i$, $i = 1, 2, ..., N$. $df(j)$ is the document frequency of term $j$ which is the number of documents in which term $j$ appears.
TFIDF weights the frequency of a term in a document based on the how often it appears in all documents. Terms that appear too frequently and too rarely are ranked lower than terms that appear moderately (Jones 1972).

The user may select between two different methods TFIDF ratio or the Z-value for determining significance. The threshold value for each method is settable by the user. The default for TFIDF ratio is 5 to 1 and for the Z-value, the default threshold is 1.64.

6.3.2 Pruning

The Argument ontology maintains a vocabulary of the non functional criteria. Examples of non functional criteria include Usability, Scalability, Flexibility and Maintainability. Most of these are not specific to software engineering and can also be used to describe the criteria for designing other systems such as spacecraft. However, the Argument ontology may also contains concepts specific to software engineering which may not be relevant to the domain of engineering design for spacecraft. These concepts and relations with these concepts need to be eliminated from the ontology by applying pruning techniques. The aim of pruning is to remove concepts that are not relevant to the domain. Concepts that are relevant to the domain are obtained by analyzing text corpura specific to the domain. Hence creation of the domain specific corpus is important and should be done by a domain expert. As previously explained, domain experts were not available for creating the domain specific corpus, but careful and conscientious approach as described in Section 6.1.2 was used to obtain pertinent documents to be incorporated into the domain-specific corpura.

6.3.2.1 Pruning Overview

The approach for pruning is motivated by (Maedche and Studer 2003, Caralt 2004). Statistical processing is used to identify concepts that are relevant to the domain. Concepts are expressed as multiple lexicalizations in a document, i.e., synonym occurrences for the same concept, for example, ‘deadlock’ and ‘stalemate’. JWordnet is used to identify such lexicalizations. Hence, whenever a lexicalization of the same concept found in the ontology is identified in the document, the concept frequency is incremented. The heuristic analyses the frequency of words that represent a particular
concept. All frequencies of concepts are aggregated upward through the taxonomy to ensure that concepts higher in the taxonomy are not considered irrelevant.

The significance of a node in the ontology is determined by the presence in the corpus of either the individual words used in the label of the node or the entire phrase used to label the node. In the first case we look to see if all the words or a percentage of words that form the phrase are domain specific. Specificity is determined either using the absolute term frequency or the term frequency-inverted document frequency measure or the Z value of the word. In this approach the words could occur anywhere in the corpus and the phrase would still be considered significant. For example the node “Minimizes Equipment Cost” has 3 terms. This node is significant if all the 3 terms or a percent of the number of terms are significant. They need not occur besides each other in the same sentence in the corpus. In the other approach these terms are to occur besides each other. With the second approach, there are two ways to match; the first looks for exact matches of the phrase. In the other approach, a slop value is set for the phrase. The slop is an edit distance that permits other words within the query. In order to understand the impact on the pruned ontology of both approaches, a parameter can be set by the user to perform either word-based pruning or phrase-based pruning. These two approaches are described in the following sections. A bottom up approach starting with the leaves is used for both:

\[
\begin{align*}
\text{Already checked-leaves-list} & \leftarrow \text{null} \\
\text{Leaves-list} & \leftarrow \text{all leaves of ontology} \\
\text{Diff-list} & \leftarrow (\text{leaves-list DIFF Already-checked-leaves-list}) \\
\text{While Diff-list is not empty} & \text{ do} \\
& \text{For each leaf node in Leaves-list} \\
& \quad \text{If leaf node not relevant, then} \\
& \quad \quad \text{prune leaf} \\
& \quad \quad \text{remove leaf from Leaves-list} \\
& \quad \text{Already-checked-leaves-list} \leftarrow \text{Leaves-list} \\
& \quad \text{Leaves-list} \leftarrow \text{all leaves of ontology} \\
& \quad \text{Diff-list} \leftarrow (\text{Leaves-list DIFF Already-checked-leaves-list}) \\
\end{align*}
\]

End While

First, a list of leaf nodes of the ontology is created. These nodes are checked to see if they are relevant to the domain based on the relevancy approach selected by the
user. If the leaf node is not considered relevant, that node is pruned from the ontology and deleted from the list of leaf nodes. After all leaf nodes in the list have been examined, the remaining leaves are transferred to the list that keeps track of already checked leaf nodes. A node in an ontology may have multiple children so all the children of a node must be pruned before the parent itself becomes a leaf node and eligible for pruning. An ontology node is only considered for pruning if it is a leaf node since a node representing a more general concept than its children is automatically considered significant.

Each phrase in the ontology will have stop words such as ‘an’, ‘the’, ‘in’, etc. In the word pruning approach, these words are not considered in determining the significance of an instance as they do not hold much meaning. The Z value measure, absolute frequency and TFIDF measure are used to determine significance of a node in the word pruning approach. On the other hand, in the phrase prune approach, a phrase may not make much sense without these stop words, so they are included in this approach. In the phrase pruning approach we experiment only with the absolute frequency and TFIDF measure. The Z measure is not used in phrase pruning as the stop words in a phrase is are not going to be present in the significant list.

### 6.3.2.2 Word-based pruning

With word-based pruning, the ontology node is considered relevant to the domain if all or a user settable percentage of the terms in the node’s label are considered relevant. Stop words such as ‘a’, ‘the’, ‘and’, ‘or’ and ‘of’ are not considered. For example, if an ontology node is labeled “Avoids a Deadlock”, the node is considered relevant to the domain if the words “Avoids” or any of its synonyms such as “prevents” are determined to be relevant and the word “deadlock” or any of its synonyms such as “stalemate” are determined to be relevant. However, the words “avoids” and “deadlock” and their synonyms need not be adjacent to each other in the text. The phrase-based pruning approach described below requires adjacency of the terms.

First the text in the training corpus is parsed in order to calculate the term frequencies and document frequencies of all the terms present in the corpus. The Apache Lucene software described in section 6.2.1 is used to accomplish this task. The frequency used in
this approach is actually a total term frequency. For a given term, the frequency of all its synonyms is added to its frequency to determine the total frequency for a document. For example, if an ontology node includes the term “deadlock” and “deadlock” occurs twice and “stalemate” occurs four times in the document, then the term frequency of the concept “deadlock” is six for that document. The same approach is followed while calculating the number of documents that have the term “deadlock” or any of its synonyms in them. JWordnet is used to determine all the synonyms for a particular term. These terms along with their frequencies are stored in a list used to determine the significance of a node.

6.3.2.3 Phrase Pruning

This approach checks for an exact phrase match or synonym variations of the phrase in the corpus, thereby improving the accuracy of the pruning. Since it is unclear from just examining the text in the corpus which phrase frequencies need to be estimated, the processing starts by examining the phrases that make up each node in the ontology and then estimating the phrase frequency and document frequency for each of those phrases. Once again the bottom up approach of iterating through the ontology is followed since nodes that are at the lower most level are more specific than nodes higher up.

Lucene’s PhraseQuery API is used to construct a query to look for the particular phrase within the corpus. An instance of the IndexSearcher class is created that references the IndexReader that was created previously. The search() method of the IndexSearcher takes the PhraseQuery as a parameter and returns the number of hits for that phrase in the corpus, i.e., the document frequency for that phrase for each document. The phrase count for these phrases in a document is obtained by using the SpanNearQuery class. This class matches spans which are near one and other. It also allows you to set a slop value which is the maximum number of intervening unmatched positions. The getspans() method returns the spans for the phrase in each document. The spans.doc() method returns the document id for each document in the spans list, i.e., the document id for each occurrence of the phrase in the entire corpus thereby giving us the phrase count in each document.

Some of the concepts in the argument ontology are of the form “{supports | provides} consistency checking”. The label for this node is specified as either “supports
consistency checking” or “provides consistency checking”. The occurrences of either of these phrases in the corpus suggest that this node is significant. This is taken into consideration by using the BooleanQuery class. An instance of this class takes in the individual sub queries as parameters along with another parameter that mentions if all the sub queries must return hits or if any one of the sub queries returns a match. The Boolean query for the phrase “{supports | provides} consistency checking” will have “supports consistency checking” as one of the Boolean clauses and “provides consistency checking” as the other Boolean clause.

An initial implementation of phrase pruning attempted to create a cross-product of all the synonyms for the words in a phrase by using the synset associated with each term in a node’s label. For each term in the label, the synset is retrieved and then the cross product of the synsets for each term in the label is used to create all variations for phrase used to label the node. For example, if the ontology node is labeled “avoids deadlock” and the synset for “avoids” consists of {avoids, prevents} and the synset for “deadlock” consists of {deadlock, stalemate}, then the following set of phrases would be searched for: “avoids deadlock”, “avoids stalemate”, “prevents deadlock”, and “prevents stalemate”.

After an initial experiment with this approach it was determined that too much execution time was needed to develop all the combinations of equivalent phrases. The current implementation selects the term in the phrase with the highest significance value and its synonyms are used to create equivalent phrases for the node only using replacement on that most significant term.

Initial experiments with Phrase Pruning using absolute frequencies bore out the research literature that just using absolute frequencies was not a good approach. The initial plans of using different levels of granularity ALL and ONE was implemented but no thorough experimentation was done since it was felt absolute frequencies for comparison purposes between the domain corpus and general corpus were not adequate. The granularity level ALL compares the absolute phrase frequencies in all documents in the domain specific corpus with respect to the generic corpus. A phrase is considered significant if its phrase frequency is higher in the domain corpus than in the generic corpus for all documents. On the other hand granularity of one means we consider a phrase to be significant if its absolute frequency is higher in at least one document in the
domain corpus. The other method is by determining the relative frequency of phrases in the domain and general corpura. A minimum ratio $r$ determines how specific domain terms must be when compared to those in a general corpus.

6.3.3 Adapting and Specializing

The ontology obtained from the previous phase may not contain concepts specific to the ED domain for spacecraft. Concepts and the taxonomic relations relevant to this domain are added to the pruned ontology using the domain training corpus and the OntoLT Protégé plugin. OntoLT provides a framework to develop domain specific ontologies using relevant text corpuralinguistically annotated by SCHUG.

The approach used in the adaptation methodology is based on Navigli’s research (Navigli 2002). He develops domain ontologies by automatic enrichment and reorganization of the WordNet hierarchy. Domain knowledge in the form of domain concept trees are carefully attached as children to existing WordNet concept nodes. The domain concept trees can be created using different approaches to extracting concepts from text corpura and filtering using natural language processing and statistical techniques. Concepts are semantically interpreted using WordNet and structured according to the is-a taxonomic relations. These resulting domain concept trees make up the Domain concept forest. These domain concept forests are then mapped to the WordNet hierarchy. Figure 11 below shows the mapping of the concept trees to the WordNet hierarchy. The contribution in (Navigli 2002) is the algorithm used to automatically map each domain tree root node shown in white to the ‘correct’ WordNet concept node shown in gray.

![Figure 16 Mapping domain concept trees (Navigli 2002)](image)
In WordNet a word sense is identified by a term called a synset, which is also equivalent to a concept. WordNet also contains different relations such as hypernym (is-a), hyponym (its inverse), meronymy (has-a), holonymy (its inverse), pertainymy (pertains-to) and several more. Given a domain tree T and the root of that tree r, attaching the domain tree T to the WordNet hierarchy disambiguates the root r of the domain tree with respect to WordNet. The algorithm used to disambiguate the root is fairly complex.

The context $C_r$ of root r in T is the set of all other domain roots. For each term t in $C_r$ and for each sense S of t in WordNet, a semantic net is built using relations such as hyperonymy, hyponymy, meronym, holonymy, pertainymy, attribute, gloss and topic. Concepts within a distance of 3 from sense S of t are only included to restrict the size of the semantic net. Figure 12 illustrates the semantic net for sense 1 of airplane in Wordnet.

![Figure 17 Sense 1 Airplane Semantic Net (Navigli 2002)](image)

A score vector measures the intersection between the two semantic nets. The score is incremented based on 11 different heuristics used to match one semantic net with another semantic net. The procedure followed to enrich Wordnet with the domain trees is shown below:

```
for each sense $R$ of r do
    $score_R := 0$ (initially, $score_R$ is the null vector)
for each term t in $C_r$ do
    for each sense $S$ of t in WordNet do
        for each sense $R$ of t in WordNet do
            Calculate the score vector $\gamma$
            for $SN(S) \cap SN(R)$
            $score_R := score_R + \gamma$
```
A similar but simpler approach is used in the ontology adaptation process. Instead of attaching the domain concept trees produced by OntoLT to a WordNet concept as done in (Navigli 2002), the OntoLT domain concept trees are being attached to the pruned ontology. The algorithm is not trying to disambiguate the sense of the root of the domain concept tree but instead looks for the best match of the domain concept tree to the concepts in the pruned ontology in order to attach domain concept tree.

The pruned Argument ontology contains concepts for non functional design criteria such as affordability criteria, security, etc. that are relevant to engineering design. Instead of using heuristic rules to score the degree of intersection between semantic nets, a similarity scoring function between a root of a domain concept tree and a concept in pruned ontology is based on the overlap between terms present in an OntoLT concept tree and the terms present in the label of a concept in the pruned ontology. This overlap is measured using the Jaccard index value which is a measure used to compare the similarity between two different sets A and B. The Jaccard coefficient is defined as the cardinality of their intersection divided by the cardinality of their union. (Jaccard 1903)

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Using the terminology in (Navigli 2002), let T be a domain concept tree produced by OntoLT. Each domain concept tree consists of at least a root node. In addition it may have multiple descendents nodes. Let terms(T) specify the set of terms used to label each of these nodes in T. For instance consider the domain concept tree T with a root labeled “Analysis” that has two children nodes labeled “thermal analysis” and “post test analysis”. Then terms(T) = {‘analysis’, ‘thermal’, ‘post’, ‘test’}. Let syn-terms(T) specify terms(T) ⊃ ∀t ∈ T synset(t), i.e., the union of the original terms used to label the nodes in T with the synsets of those terms as found using JWordnet. For example, the term thermal has synonyms ‘heat’, ‘solar’, etc. These terms are also included in syn-terms(T).

For each node in the OntoLT concept tree, a Z-value is calculated and an average Z-value is determined for the concept tree. Each concept trees produced by OntoLT is examined and only those meeting the user settable Z-Value significance level are to be considered for merging into the ontology. The attaching of the domain concept trees
begins at the leaves of the ontology. Note that merging may be dynamic or static. If
dynamic is used, then for each execution of the for-loop in the following pseudocode, the
leaves list may change based on the previous loops merging process.

Domain forest = All OntoLT Concept trees
Examine-list ← all leaves of ontology  // start with leaves, if static only do outside loop
For each T in Domain forest
    if (Z-value(T) >= Z-set-threshold) then
        If (dynamic merging)
            Examine-list ← all leaves of ontology  // reset for each time through loop
        Create syn-terms(T)
        Prev-Max-JV = 0, Max-JV=0       // Save the Maximimum JaccardValue
        findBestNodeMatch(Examine-list, Max-Match, Max-JV)
        While (Max-JV < Jac-threshold AND Max-JV  >= Prev-JV AND
            Max-Match not null)
            // match not big enough so ascend from best one
            // found so far looking for best match of its parents until find one or
            // until match value decreases or until run out of ancestors
            Prev-JV = Max-JV
            Examine-list = parentsOf(Max-Match)
            findBestNodeMatch(Examine-list, Max-Match, Max-JV)
        end while
        If (Max-JV >= Jac-threshold)
            Children(Max-Match) = Children(Max-Match) U {root(T)}
        else  add T as a separate tree to the pruned ontology

Procedure findBestNodeMatch( Examine-list, Max-match-one, Max-JV)
    Max-match-one = null
    For each node l in Examine-list
        Create syn-terms(l)
        Jaccard-val(l) = J(syn-terms(T), syn-terms(l))
        If (Jaccard-val(l) not = 0 & Jaccard-val(l) = = Max-JV)
            If (depth(l) > depth(Max-match-leaf) // break ties using deepest leaf
                Max-match-one = l
            If (Jaccard-val(l) > Max-JV)
Max-match-one = l
Max-JV = Jaccard-val(l)

End procedure

Initially the Examine-list contains all the leaf nodes of the pruned ontology. The syn-terms list is created for the leaf node by adding in the synsets for each term used to label the leaf. For example, the syn-terms the leaf “Avoids Deadlock” includes the terms “Avoids”, “deadlock” and all the synsets of these terms such as “prevents”, “stalemate”, “standstill” etc. These synsets are obtained using JWordNet.

Using the Examine-list, the procedure findBestNodeMatch processes each node looking for the best match for the domain concept tree T from OntoLT using the Jaccard value. If a best match leaf node is found which satisfies the minimum Jaccard threshold for matching, the tree T is added as a child of that particular leaf node in the pruned ontology. It is possible that a particular domain concept tree may have a Jaccard index value that is lower than Jaccard threshold set by the user. In such cases, the leaf node that has the highest Jaccard value is selected and the process of examining its ancestors is begun. This approach is based on the reasoning that its parents being more general concepts than the child might result in a closer match. This proceeding up the pruned ontology stops if the resulting Jaccard value between the concept tree T and an ancestor node either exceeds the threshold or decreases in value or if no more ancestors are left. If the threshold is exceeded, then the domain concept tree node is added as a child of that ancestor node; otherwise, it is added to the pruned ontology as a separate tree.

There is also a possibility of finding multiple matches for a domain concept tree where the Jaccard value happens to be the same for multiple leaf nodes. In such cases, an association of the domain concept tree T with the most specific leaf is desired. The specificity of the leaf is determined by its depth. The leaf with the greater depth is selected. If the leaves happen to have the same specificity then the first leaf found is arbitrarily kept.

A possible modification of this matching algorithm could be to rename the node in the pruned ontology. For instance if the best match for the concept tree “control team” is “provides access control” then instead of making “control team” the child of “provides
access control” we could rename the original node as “provides access control | control team” (“provides access control or control team”). Another improvement could be to investigate switching the parent child relationship, i.e., “control team” might become the parent of the node “provides access control”. These two modifications require a technique to determine which of the two concepts is more general and assigning that as the parent. This thesis did not investigate this modification.

Two approaches in generating the examine list of the pruned argument ontology can be used. In the static approach, only the leaves of the original pruned ontology are used for finding a match with a domain concept tree T. In the dynamic approach, the examine list changes dynamically. When a domain concept tree is added to the pruned ontology, new leaf nodes are added to the pruned ontology and considered for the matching process. The static approach is used first to determine the best resulting ontologies. Then the dynamic approach was run to see if it improved the results. The determination of which approach is used depends on where the statement

Examine-list ← all leaves of ontology

is executed in the above merging algorithm. If it is executed inside the outermost loop, the examine list is refreshed after processing a domain concept tree so that the merging is dynamic. For example if the first concept tree is “Control team” and its best match is “provides access control”, one of the leaves in the pruned ontology, then the concept “Control team” is assigned as a child to the concept “provides access control”. The new examine list of the pruned ontology now includes the leaf node just added, i.e., “Control team”. A static approach could be taken by moving the above statement outside of the loop but it was decided that a dynamic approach could take advantage of the Z-value significant ordering of the OntoLT concept trees.

6.4 Evaluation

The resulting ontology needs to be evaluated to ensure domain specificity. The approaches to evaluate an ontology are broadly classified as (Brank et al. 2005):

- Evaluation done by humans who are experts in the domain.
- Comparing an ontology with a Gold Standard Ontology.
- Evaluation of the results of using an ontology in an application
Comparing an ontology with a collection of documents relevant to the domain

The approach used for this research falls into the last category. The other approaches were not feasible. No gold standard ontology was available to compare the resulting ED criteria ontology to. Although a conscientious attempt was made to acquire domain experts from NASA, no one would volunteer.

Another approach to evaluating ontologies (Brank et al. 2005) assesses an ontology at its different levels instead of as a whole. They classify the levels as lexical or data level, context or application level, syntactic level, taxonomy or hierarchy, and structure, architecture, design.

The lexical level focuses on the concepts, instances, facts, etc. and vocabulary used to identify the concepts and instances in the ontology. The evaluation of these concepts is done by comparing the ontology with different sources of data. The approach used here is data driven and does not need any human evaluation or gold standard evaluation. The approach taken in this thesis also focuses on the lexical component to evaluate the ontology using the domain corpora. The lexical content of an ontology is evaluated by comparing the set of strings used as concept identifiers in the ontology with statistically significant strings retrieved from a set of documents. The evaluation can be improved by considering synonyms of a lexical entry. In the following sections, the various measures used to assess the resulting ED Criteria ontology are discussed. All the following measurements are based on those proposed in (Spyns et al. 2005).

### 6.4.1 Coverage

Coverage represents the degree to which the concepts in an ontology represent the domain. In (Spyns et al. 2005), the coverage of the ontology is based on the intersection between the vocabulary of the triples and the corpus. A triple is a subject-verb-object such as `<student studies subject>` or noun phrase-preposition-noun phrase such as `<book in library>`. The words are grouped into frequency classes, i.e. the absolute frequency of the word in the entire corpus. For example if the term “system” occur 3000 times in the corpus and another term “performance” also occurs 3000 times in the entire corpus, then the frequency class 3000 will contain the words “system” and “performance”. The
coverage is determined by finding the ratio of vocabulary intersection between each frequency class for words-triples and the corresponding frequency class for the test corpus. This is averaged over the number of classes to give a percentage value. The formula for coverage is:

\[
\text{coverage}(\text{triples}, \text{text}) = \frac{\sum_{i=1}^{n} \frac{\#(\text{words}_{\text{triples freq class}_i} \cap \text{words}_{\text{text freq class}_i})}{\#\text{words}_{\text{text freq class}_i}} \times 100}{n}
\]

The formula above finds the overlap between the words in a frequency class with the words of the same frequency in the ontology, i.e., the triples represents the learned ontology. A modified approach is used in the evaluation of the ED Criteria ontology since this ontology does not have many frequency classes. The coverage is estimated by finding the overlap between the words in a frequency class from the test corpus with all the terms labeling the nodes in the entire ontology. This calculation is done for each frequency class of the test corpus and averaged over the number of frequency classes for the test corpus.

6.4.2 Accuracy:

The accuracy measure is related to the coverage metric described above but is based on Zipf’s law (Zipf 1949) to determine coverage for only the relevant frequency classes. According to this law, the more frequently a word is used, the lesser meaning it carries. Hence the high frequency classes mainly contain words not carrying much meaning. Based on this assumption, an upper and lower bound for the frequency classes can be set. This bound determines the frequency classes containing the most relevant words. The accuracy is computed the same way as coverage only using the relevant frequency classes of the domain test corpus, not all frequency classes. The following formula determines accuracy:

\[
\text{accuracy}(\text{triples}, \text{text}) = \frac{\sum_{i=1}^{n} \frac{\#(\text{words}_{\text{triples rel freq class}_i} \cap \text{words}_{\text{text rel freq class}_i})}{\#\text{words}_{\text{text rel freq class}_i}} \times 100}{n}
\]
where words_text_rel_frequency_class refers to only the relevant frequency classes. Two approaches have been suggested for determining the relevant frequency classes. The first approach determines the median frequency class. An upper and lower bound is set by including a quarter of the classes above this median class and a quarter of the classes below this median class. These classes are all considered relevant and used in the formula above to find the accuracy.

The second approach uses the Z value statistic as described in section 6.3.1. A list of significant words is created for the domain test corpus using the Z statistic. A frequency class is considered relevant if 60% (as suggested in (Spyns et al. 2005)) of the terms in the frequency class are present in the significance list of words. All the frequency classes that satisfy the criteria are labeled relevant classes and are used in the estimation of accuracy of the ontology. The required percentage of significant words for a frequency class is a user settable parameter. The second approach was chosen for implementation.

### 6.4.3 Precision

Precision determines if all the concepts in the ontology are relevant to the domain. It is difficult to determine if the concepts in an ontology are correct without the help of human evaluation or a gold standard ontology. The approach taken in (Spyns et al. 2005) attempts to build an artificial “gold standard” by developing a list of statistically significant words for comparison to in order to determine correctness. The statistically significant list of words is found using the Z value described in section 6.3.1. Precision is then determined as the ratio of number words in the ontology overlapping with words in the significance list to the number of words in the ontology. It represents how much of the ontology is considered relevant using the artificially created “gold standard”.

\[
\text{precision}(\text{triples, text}) = \left( \frac{\#(\text{words_of_triples_mined} \cap \text{statistically_relevant_words})}{\#\text{words_of_triples_mined}} \right) \times 100
\]

where words_of_triple_mined in this thesis work is the set of all words in the ontology and statistically_relevant_words is the list of words from the domain test corpus that are determined to satisfy the Z threshold value.
6.4.4 Recall

Recall is used to determine if all the relevant triples/concepts in the domain have been retrieved. This measure is also determined using the statistically significant words list. Recall is defined as the ratio of number of words in the ontology that overlap with words in the statistically significant list and the number of statistically significant words. It represents how much of the artificially created “gold standard” is contained within the learned ontology.

\[
recall(triples, text) = \left( \frac{\#(words\_of\_triples\_mined \cap statistically\_relevant\_words)}{\#statistically\_relevant\_words} \right) \times 100
\]

7. Experiments and Results

The ontology adaptation software architecture consists of several existing software tools integrated with software developed as part of this thesis. Chapter 6 provided an overview of the architecture. In order to evaluate the performance of this system, numerous experiments were run. First a basic description of the method of experimentation is presented followed by a description of the possible parameters that may be set for an experiment. Then the specific parameters used in initial testing and refinement of the experiments based on initial results are discussed. Finally a comparison of the various experimental outcomes based on several combinations of parameters is presented along with an analysis of these outcomes.

7.1 Format of Experiments

The SEURAT Argument ontology, the domain training corpus, and the general corpus served as input to the ontology adaptation software for each experiment. These were constant throughout all the experiments. The evaluation process on the resulting adapted ontology used the general corpus and the domain testing corpus in order to determine the four performance measures coverage, accuracy, precision, and recall. These performance measures were based on the research literature for ontology evaluation and used in place of a domain expert evaluation or comparison to gold
standard ontology since these evaluation methods were not feasible. As the base case, the Argument ontology was evaluated with respect to these four measures with the results shown below.

<table>
<thead>
<tr>
<th>Coverage</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
<th># concepts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument Ontology</td>
<td>1</td>
<td>16.243</td>
<td>16.41</td>
<td>30.735</td>
</tr>
</tbody>
</table>

Table 1 Performance measures for Argument Ontology

This base case is used as comparison for the resulting four measures after performing the adaptation process, similar to a before and after comparison for analysis (see Table 4). The experiments were done in two steps following the two major processes in the adaptation methodology. First experiments were run using the various parameters defined for the pruning phase. These results provided a set of pruned ontologies for performance comparison with respect to the four measures. The approach taken for selecting the pruned ontologies for the next phase of specializing was based on selecting the pruned ontology with the highest coverage, the pruned ontology with highest accuracy, the pruned ontology with the highest precision and the pruned ontology with the highest recall. Each selected pruned ontology is then used as input to the specializing process to produce its corresponding final adapted ontology.

7.2 Experimental Parameters

The two major steps in the adaptation process are pruning and specializing the ontology. For pruning, the first selection parameter is either word pruning (WP) or phrase pruning (PP). For word pruning, the parameters are domain relevance method (TFIDF or Z), domain relevance cutoff value (2:1 or 5:1 ratio for TFIDF, 1.64 or 2.32 for Z), and percentage of terms in ontology node label which must meet relevance cutoff value (All or 60%). There are 8 possible parameter settings for word pruning. For phrase pruning, the parameters are TFIDF ratio value (2:1 or 5:1) and synonym substitution for significant term (Yes or No). There are 4 possible setting for phrase pruning. Because of time limitation the synonym substitution was not implemented. With the various parameter settings there are 12 possible variations. After initial experimenting with phrase pruning, it was determined that the phrase pruning algorithm produced extremely
low values for all the measures except for precision because it basically pruned almost the complete Argument ontology leaving only around 20 nodes in the best case. As noted previously, an improvement to use slop values and the cross product of synonyms for terms in the ontology node labels was implemented and tested but required too much computation time. Because of the poor results with phrase pruning, only word-pruned ontologies were considered as input to the specializing process.

For specializing by merging, the first parameter is the Z-value relevance cutoff for the domain concept tree (Z=1.64 or X=2.32), the failing match action (ignore or separate) and the merging environment (static or dynamic). The failing match action determines whether a domain concept tree is ignored or added as a separate tree to the ontology when it does not meet the Jaccard values (JV) threshold for merging into the ontology. The Jaccard threshold value is also settable but after experimenting and collecting data about the Jaccard values produced during initial testing, it was determine that the average Jaccard value was 0.058. The Jaccard threshold value was set at JV=0.05 and used in the final testing of the specializing process of the ontology adaptation software. Initially, the plan was to do merging with ordered Z-values but that was not implemented so that all tests were done using unordered Z-values (UO).

7.3 Comparison and Analysis of Experimental Results

As previously described, not all the word pruned ontologies were selected for the specializing process. Word pruning using the Z value relevance produced better result for all performance measures than the TFIDF relevance. The following table shows the results of word pruning with associated Z values for determining significant words list and required percentage of words in the label an ontology node that must be found in the significance words list.

<table>
<thead>
<tr>
<th>Z Value</th>
<th>Coverage</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
<th># concepts</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.64</td>
<td>14.21</td>
<td>14.35</td>
<td>33.06</td>
<td>9.93</td>
<td>159</td>
</tr>
<tr>
<td>2.32</td>
<td>13.7</td>
<td>13.84</td>
<td>35.32</td>
<td>7.59</td>
<td>113</td>
</tr>
<tr>
<td>1.64%</td>
<td>16.243</td>
<td>16.41</td>
<td>31.021</td>
<td>12.909</td>
<td>273</td>
</tr>
<tr>
<td>2.32%</td>
<td>16.243</td>
<td>16.41</td>
<td>31.32</td>
<td>12.866</td>
<td>263</td>
</tr>
</tbody>
</table>

Table 2 Word Pruning Results using Z value for Relevance
One test case WP Z=1.64, 60% has the highest or tied for highest performance values for coverage, accuracy and recall with a resulting 273 concepts. The test case WP Z=2.32, All had the best performance for precision with a resulting 113 concepts. An extra test case WP Z=1.64, All with a resulting 159 concepts was included since the number of remaining concepts fell between the other two best cases and it had better coverage, accuracy and recall values than the test case WP Z=2.32. These three pruned ontologies were selected for continuing on with the specializing by merging process. The final results for the selected parameter settings for the merging process are shown in Table 3.

<table>
<thead>
<tr>
<th>Argument Ontology</th>
<th>Coverage</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
<th># concepts</th>
</tr>
</thead>
<tbody>
<tr>
<td>WP Z=1.64, All, Merge Z=1.64 (UO), ignore, JY=0.05, stat</td>
<td>22.335</td>
<td>22.564</td>
<td>31.824</td>
<td>14.905</td>
<td>350</td>
</tr>
<tr>
<td>WP Z=1.64, All, Merge Z=2.32 (UO), ignore, JY=0.05, stat</td>
<td>22.335</td>
<td>22.564</td>
<td>32.543</td>
<td>12.605</td>
<td>290</td>
</tr>
<tr>
<td>WP Z=1.64, All, Merge Z=2.32 (UO), separate, JY=0.05, stat</td>
<td>23.857</td>
<td>24.1</td>
<td>33.83</td>
<td>14.363</td>
<td>357</td>
</tr>
<tr>
<td>WP Z=1.64, All, Merge Z=1.64 (UO), separate, JY=0.05, stat</td>
<td>25.88</td>
<td>26.15</td>
<td>32.98</td>
<td>17.66</td>
<td>447</td>
</tr>
<tr>
<td>WP Z=1.64, 60%, Merge Z=1.64 (UO), ignore, JY=0.05, stat</td>
<td>24.36</td>
<td>24.61</td>
<td>29.65</td>
<td>18</td>
<td>486</td>
</tr>
<tr>
<td>WP Z=1.64, 60%, Merge Z=2.32 (UO), ignore, JY=0.05, stat</td>
<td>24.36</td>
<td>24.61</td>
<td>30.298</td>
<td>15.81</td>
<td>414</td>
</tr>
<tr>
<td>WP Z=1.64, 60%, Merge Z=2.32 (UO), separate, JY=0.05, stat</td>
<td>25.88</td>
<td>26.15</td>
<td>31.21</td>
<td>17.227</td>
<td>474</td>
</tr>
<tr>
<td>WP Z=2.32, All, Merge Z=1.64 (UO), ignore, JY=0.05, stat</td>
<td>21.82</td>
<td>22.05</td>
<td>33.53</td>
<td>12.73</td>
<td>287</td>
</tr>
<tr>
<td>WP Z=2.32, All, Merge Z=2.32 (UO), ignore, JY=0.05, stat</td>
<td>21.82</td>
<td>22.05</td>
<td>34.5</td>
<td>10.58</td>
<td>234</td>
</tr>
<tr>
<td>WP Z=2.32, All, Merge Z=2.32 (UO), separate, JY=0.05, stat</td>
<td>23.35</td>
<td>23.589</td>
<td>35.82</td>
<td>12.86</td>
<td>314</td>
</tr>
<tr>
<td>WP Z=2.32, All, Merge Z=1.64 (UO), separate, JY=0.05, stat</td>
<td>25.88</td>
<td>26.153</td>
<td>34.96</td>
<td>18.27</td>
<td>404</td>
</tr>
</tbody>
</table>

Table 3  Performance results for selected parameters

From examining all the various columns, a reassuring result is that for coverage and accuracy, all the test cases have improved values over those taken on the Argument Ontology. For precision, most test cases have higher values than that of the Argument ontology except for points 6, 7, and 9. In each of these cases, the word pruning significance is set at the lower value 1.64 and only requires 60% of the words in an ontology node label to be found in the significant words list. These parameter values cause the denominator in the precision formula to increase because more ontology nodes are not pruned; therefore, there are more leaves for the merging process to use for
adaptation. Note that these cases include the highest and second highest value for the recall measure. Similarly, for recall, most test cases have higher values than that of the Argument ontology except for points 10, 11, and 12. These cases have the fewest number of concepts added to the ontology. All of these have severe pruning with a Z value of 2.32 and require all of the words in the ontology node label to be found in the significant words list. Note that these test cases include the highest precision value and other high precision values.

The following scatter plots reflect the data contained in the above table above. The point labels on the plots correspond to the point label in the second column of the above table.

Graph 1 Coverage vs. Accuracy
Graph 2 Precision vs Recall

Graph 3 Precision vs # concepts
Graph 4 Recall vs # concepts

Graph 5 Accuracy vs # concepts
Graph 1 reveals the close correspondence between the coverage and accuracy values with the accuracy performance measure always higher, though not by much. For accuracy, the average should be over fewer frequency classes and the calculated ratios should be higher since over a majority of the words in the frequency class are considered domain significant. As can be seen from examining the Graph 2, precision and recall are closely related. For pairs of test cases where all parameter values are the same except the Merging Z value, an improvement in precision results in a degradation in recall. This is more clearly seen by looking at Table 3. For all cases, when the Merge Z value increases from 1.64 to 2.32 but the rest of the parameter values are identical, the precision improves but the recall worsens. This result occurs because the domain concept trees require a higher average Z value before an attempt can be made at merging them into the pruned ontology.

Understanding Graph 3 is somewhat challenging. A visual examination appears to uncover two separate lines, each showing as precision increases the number of concepts decreases. The groups of points are group 1 consisting of 6, 7, 2, 3/10, and 11, and group 2 consisting of 9, 8, 5, 13, 4 and 12. Looking at the Table 3, group 1 points are all cases where merging ignores domain concept trees that do not meet the Z significance cut off value. Group 2 points are all cases where merging adds domain concept trees if they do not meet the Z significance cutoff value. There appears to be some interaction between the other parameters and the ignore/separate parameter that causes lower precision to occur for the group 1 (Ignore) points when they are in the same range of number of concepts when compared to group 2 (separate) points.

An examination of Graph 4 shows a tendency of increasing recall with increasing number of concepts. There are pairs that represent very close recall values and number of concepts where there is a small reversal: 2 and 4, 5 and 8, and 7 and 13. For the pair 2 and 4, more merging occurs with point 2 since it has a lower Z value of 1.64 but point 4 allows separate domain concept trees to be added as roots. These two parameters appear to be balancing each other. Likewise for both the other pairs, there is a balancing. For test points 5 and 13, there is more pruning with a stricter ALL words meeting significance test for point 5 and a higher Z value requirement for point 13. This however
is balanced with more merging occurring for both since a lower Z value of 1.64 is required.

Graph 5 also reveals a pattern of increasing accuracy with increasing number of concepts if the initial Argument ontology point 1 is ignored. If a vertical line is drawn at the 25 marker for accuracy, the points are separated into two groups, those to the left with less than 350 concepts and those to the right with more than 350 concepts. A linear pattern exists except for a reversal of points 6 and 13. Point 13 test case results in 404 concepts and point 6 test case has 486 concepts. The point 13 test case has stricter pruning parameters than the point 6 test case but permits adding separate domain concept trees if merging is not successful.

Due to the computational complexity of doing dynamic merging, only the best result from the above static cases were dynamically merged. The one case selected was the WP Z=1.64, 60%, Merge Z=1.64(UO), separate, JV=0.05 case since it had the highest coverage (27.411%), accuracy (27.69%) and recall (20.22%). The results from dynamic merging on this test case resulted in no significant differences for the performance measures.

The following table shows the best improvement for each performance measure based on the two best test cases, WP Z=1.64, 60%, Merge Z=1.64(UO), separate, JV=0.05 case with the highest coverage, accuracy and recall and the WP Z=2.32, All, Merge Z=2.32(UO), separate, JV=0.05 with the highest precision. These results show a substantial percentage increase in coverage, accuracy and recall and over a doubling of the number of concepts when compared to the Argument ontology.

<table>
<thead>
<tr>
<th></th>
<th>Coverage</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
<th># concepts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best Measures from test cases</td>
<td>27.411</td>
<td>27.69</td>
<td>35.82</td>
<td>20.22</td>
<td>570</td>
</tr>
<tr>
<td>Argument Ontology</td>
<td>16.243</td>
<td>16.41</td>
<td>30.735</td>
<td>12.9</td>
<td>280</td>
</tr>
<tr>
<td>Difference</td>
<td>11.168</td>
<td>11.28</td>
<td>5.085</td>
<td>7.32</td>
<td>290</td>
</tr>
<tr>
<td>% increase</td>
<td>68.76</td>
<td>68.74</td>
<td>16.54</td>
<td>56.74</td>
<td>103.57</td>
</tr>
</tbody>
</table>

Table 4 Before/After Adaptation - Change in Measures

Although an improvement is made over the initial Argument ontology, the resulting best performance measures are still low. These results, however, if compared to those reported in (Spyns et al. 2005) as 39.68% for coverage, 52.1% for accuracy, 58.78% for
precision and 9.84% for recall appear to be at a reasonable starting place for work to proceed in making improvements to the ontology adaptation software. Some ideas for future improvements are presented in the following chapter.

All the comparisons reported have been made based on these four performance measures which deal only with the content of ontologies. Intuitively, one would like to know what concepts were deleted and what ones were added by the ontology adaptation process. Appendix A shows output from performing a Prompt difference on the best adapted ontologies with the Argument Ontology. Prompt is a Protégé plug-in which reports the differences between two ontologies. In addition, one would like to visually examine the best adapted Argument ontologies. The following illustrate examples of the adapted ontology with the highest coverage, accuracy and recall, the point 9 test case.

![Figure 18 Top level Concepts for Adapted Ontology (Point 9 test case)](image-url)
This figure shows the top level concepts starting with Affordability at the top and going around clockwise, Needs Satisfaction, Maintainability, End User, Dependability, Adaptability, and Performance. There is no difference at the top level from the original to the adapted version.

The following figure illustrates an example of a domain concept tree “is-component-based” that has been added as a separate root node to the adapted ontology. This new concept tree exists as a new root and has six branches, three of which have more branches.

![Domain Concept Tree Added to Adapted Ontology](image)

Figure 19  Independent Domain Concept Tree added to Adapted Ontology
(Point 9 test case)

The following figure shows an example of domain concept trees being added to the ontology. For example, “Operating Cost” is a kind of “Affordability Criteria.” “Minimizes Communication Cost” is associated with “Operating Cost.” The OntoLT domain root concept “communication” is merged under “Minimizes Communication Cost.” Another example is “increases software support available” which is a child of “reduces support cost” which is a child of “Maintenance Cost.” Five domain concept trees have been added to “increases software support available” node.
Figure 20 Merged Domain Concept Trees
8 User Interface for Adaptation Methodology

The various pruning and specialization algorithms to covert the Software Engineering Argument ontology into an Engineering Design Ontology are implemented as a plug-in for Protégé, an ontology editor (http://protege.stanford.edu/). The plug-in allows the user to perform different pruning algorithms followed by a merging phase which specializes the ontology to a particular domain based on the selected training corpus. The ontology resulting from the different phases can be viewed from the instance tab in Protégé. The coverage, accuracy, precision and recall of the adapted ontology can then be estimated using a test corpus. These measures are displayed to assist the user in determining how well the adaptation process performed relative to the test corpus.

The figure below shows the interface of the plug-in which consists of two panes. The left panel is used to set the different parameters for the pruning phase. The user can either select the option of word pruning or phrase pruning. If word pruning is selected, then the different parameters for this approach such as selecting either TFIDF or Z value to determine the significance of ontology concept labels and then setting the appropriate threshold value for the selected one. For example, the figure below shows that Z value is selected and the threshold set is 1.64. The other parameter that the user can set is either “All words” or 60% of the words in a ontology concept label must be found in the significant words list. In the case of phrase pruning, the TFIDF value of the phrase or the TF value of the phrase can be used to determine significance of the phrase. The Z value selection is not possible for phrase pruning since the Z value for all phrases in the text corpus cannot be estimated.

The right panel allows the user to select parameters for the merging process which has three parameters. The merging takes as input domain concept trees obtained from OntoLT. The first parameter allows the user to include all trees produced by OntoLT or to include only those trees determined to be significant to the domain. Domain significance of a tree is determined by finding the average Z value of the tree. Only trees that have an average Z value greater than the user selected threshold are used in the merging process. The second parameter determines whether to use a static merging environment or a dynamic merging environment. If the static case is selected, then only nodes from the initial ontology are considered for possible merging. In the dynamic
environment domain concept trees added in the merging process are also considered as candidates for merging. The third parameter determines whether domain concept trees that cannot be merged into the existing ontology should be ignored or added as separate root nodes into the ontology. The merging Jaccard threshold is hard-coded to 0.05 based on some initial experimentation showing that the average Jaccard value was 0.058.

The resulting ontology is evaluated using the coverage, accuracy, precision and recall measures against a test corpus in order to evaluate the performance of the adaptation methodology. These measures are displayed on the right hand pane.

Figure 21 Adaptation Methodology User Interface
Figure 22 Evaluation Results from Adaptation Process
9 Research Significance and Future Directions

One approach to overcoming the bottleneck of creating ontologies is to semi-automatically adapt an existing ontology to the needs of a new but related problem domain. This thesis research has developed an ontology adaptation software system to perform the process without domain expert intervention in order to experiment with how well such a system might perform.

The SEURAT Argument ontology serves as the basis for experimenting with the adaptation methodology. It is adapted from the domain of design rationale for software engineering to the domain of design rationale for engineering spacecrafts. The ontology adaptation architecture was developed based on surveying various approaches to ontology learning and reviewing the software tools currently available for use in ontology learning. The two major components developed for this adaptation process are pruning and specializing the pruned ontology by merging new concepts into it.

The pruning algorithm created as part of this research is based on incorporating techniques used for ontology evaluation since the objective was to prune concepts from the old domain that are not relevant to the new domain. Existing pruning techniques are typically used after the learning process has created an initial ontology for purposes of removing concepts that have been included in the learned ontology but that are not considered related to the domain. The use of the Z-value statistic suggested for ontology evaluation (Spyns et al. 2005) in the pruning process of the ontology adaptation software is to our knowledge a new approach for determining whether a concept is related to the ontology domain.

The algorithm for specializing the ontology by merging the domain concept trees produced by OntoLT into the pruned ontology is inspired by the approach taken byNavigli’s (2002) as described in section 6.3.3; however, it is simpler since the matching is based on measuring the overlap of the synsets for the terms labeling the domain concept trees and the synsets for the terms labeling a node in the ontology. Navigli’s algorithm instead uses several heuristic for measuring the overlap of semantic networks and uses all the roots of the concept trees to determine the context of each root. Its objective is to position a domain concept tree in WordNet itself. For the adaptation
software the objective is to position each domain concept tree in the pruned ontology. The algorithm developed for merging OntoLT domain concept trees into the pruned ontology also incorporates the Z-value statistic to order the domain concept trees by their significance and to exclude those that do not meet the Z cutoff value. This use of the Z value is a unique application of a parameter used in ontology evaluation for the purpose of ontology learning.

Although this adaptation process was completely automated, the results described in Chapter 7 indicate that a domain expert is needed to help direct the process. The ontology adaptation software needs to be enhanced to provide domain experts with the ability to monitor the intermediate results of the adaptation phases. For example, an ordered list by significance value of all the concept nodes identified for removal from the ontology should be provided to the domain experts. The domain expert could then review the list and uncheck the removal box for each concept he or she judges to be relevant to the problem domain. Likewise, the merging process could display the Z-ordered (highest to lowest) list of domain concept trees to be merged into the ontology. The domain expert could then select the ones most related to the problem domain for merging. The system could merge those and then visually display the results. The user could then decide whether to undo a merge and/or position the new node within the ontology himself.

Another aspect for future investigation is more experimentation with OntoLT user-specified rules. OntoLT permits the definition of mapping rules to extract concepts (Protégé classes) and attributes (Protégé slots) automatically from linguistically annotated (by SCHUG) text collections. Several default mapping rules are included with the OntoLT plug-in, but alternatively the user can define additional rules. This thesis used the default mapping rules. Experimentation with user-defined mapping rules might result in OntoLT producing more relevant and specialized domain concept trees for integration into an ontology. In addition, research (Sabou 2005) suggests that a domain sublanguage might be taken advantage of to better parse the text collections looking for keywords of the domain sublanguage. These keywords might be used to pinpoint domain specific concepts for merging. A domain expert could provide the sublanguage and help develop user-defined mapping rules using the sublanguage.
One limitation of the ontology adaptation software is its dependency on SCHUG to produce the XML annotated files input to OntoLT. All the documents used in the domain training corpus and the domain test corpus had to be sent away for annotation. Although OntoLT is made readily available as a Protégé plug-in, the SCHUG software is not openly available for use. This dependency limits the use of the ontology adaptation software. Future work needs to investigate either finding other open source software or developing software to provide the SCHUG functionality. The GATE software might be one good starting point since it provides sophisticated capabilities for NLP needs.

To summarize, this thesis integrates several available software tools for ontology construction to create an ontology adaptation architecture for which pruning and specializing algorithms were developed. This software is a plug-in for the Protégé ontology editor and used in experiments to adapt the SEURAT Argument Ontology to an Engineering Design Criteria ontology for the problem domain of spacecraft design. NTRS documents were manually selected to create the domain training and test corpora.
References


6. Bontas Elena Paslaru, Mochol Malgorzata, Tolksdorf Robert, Case Studies on Ontology Reuse. In 5th International Conference on Knowledge Management (I'Know'05), Graz, Austria, 2005.


http://ontoweb.aifb.uni-karlsruhe.de/Members/ruben/Deliverable


44. Morin E., Automatic acquisition of semantic relations between terms from technical corpora. *Proceedings of the Fifth Int. Congress on Terminology and Knowledge Engineering* (TKE-99), TermNetVerlag, Vienna, 1999.


http://www.esa.int/gsp/ACT/doc/ACT-RPT-5101-ARIADNA-Natural%20Language_Rome.pdf


