A SEMANTIC APPROACH BASED ON ONTOLOGIES TO SUPPORT ENGINEERING KNOWLEDGE RETENTION AND EXCHANGE IN THE PRODUCT ASSEMBLY DESIGN AND TRAINING DOMAINS

By

OKJOON KIM

A dissertation submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Washington State University
School of Mechanical and Materials Engineering

August 2011

© Copyright by OKJOON KIM, 2011
All Rights Reserved
To the Faculty of Washington State University:

The members of the Committee appointed to examine the dissertation of OKJOON KIM find that it is satisfactory and recommend that it be accepted.

__________________________________________
Uma Jayaram, Ph.D., Chair

__________________________________________
Sankar Jayaram, Ph.D.

__________________________________________
Hakan Gurocak, Ph.D.
ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Uma Jayaram, for consistent guidance and constant attention. I also want to express my appreciation to Dr. Sankar Jayaram for continuous helpful advice and to Dr. Hakan Gurocak for serving on my committee. It has been fortunate that I could study and do research related to Virtual Reality in the Virtual Reality and Computer Integrated manufacturing Lab at WSU. In addition, I appreciate the intimate support and help from all the members in VRCIM lab.

Lastly, I would like to specially thank my wife, MiYoung for endless love, firm trust, and care. Furthermore, I am able to make a successful life because my parents, brother, and sister, and my wife’s family trust me with confidence.
A SEMANTIC APPROACH BASED ON ONTOLOGIES TO SUPPORT ENGINEERING KNOWLEDGE RETENTION AND EXCHANGE IN THE PRODUCT ASSEMBLY DESIGN AND TRAINING DOMAINS

ABSTRACT

By OkJoon Kim, Ph.D.
Washington State University
August 2011

Chair: Uma Jayaram

This research presents a semantic approach using ontologies to share knowledge and exchange product assembly knowledge at the semantic level in the two categories of domains: product assembly design domain and assembly training domains. Along with ontologies and CAD design/training tools, a typical engineering knowledge lifecycle for exchange and retention can be characterized by the following five key activities: knowledge modeling, knowledge acquisition, knowledge integration, knowledge retrieval and presentation, and knowledge maintenance.

Knowledge modeling is based on logic-based models called ontologies. Knowledge acquisition is initiated by allowing exchanges and informal interactions between the participants and then instantiates concepts in the ontologies. For knowledge
integration, we present an approach based on a shared ontology, in which a higher level of ontologies are shared among lower levels of ontologies. Key mapping strategies, such as Equivalency, Attribute Similarity, Composition Similarity, and Inheritance Similarity are defined to map concepts and properties defined in the domains.

Next, the knowledge can be retrieved and presented in the engineering tools. When presented, ontologies provide an opportunity to capture and manage common data and map concepts from one application to another in a logical and measured manner. A simple computer-based training (CBT) approach allows traditional keyboard and mouse interactions while a complex VR-based immersive training (IMT) approach provides an immersive virtual environment for more realistic experiences. Even though applications for each approach are developed completely independently, there is consequently much duplication of data and a lack of synchronization between them. Therefore, we focus on an integrated approach with support from ontologies to address this problem.

Finally, knowledge maintenance is facilitated by allowing new information that is added either through the online community or to the tools to be propagated to the other. We present an approach to build a semantic online community, CREEK (Community for Retention and Exchange of Engineering Knowledge), to facilitate numerous activities of an engineering knowledge lifecycle in a coordinated manner. The online community is
specifically focused on the procedural knowledge domain related to engineering product assembly and the target audience is a spectrum of engineers ranging from experts to newer and relatively inexperienced engineers.
TABLE OF CONTENTS

ACKNOWLEDGMENTS .............................................................................................................. iii

ABSTRACT ............................................................................................................................... iv

1. Introduction ....................................................................................................................... 1

2. Background ....................................................................................................................... 6

  2.1. Traditional Exchange of Engineering Data .......................................................... 6

  2.2. Engineering Ontologies and Ontological Mapping ............................................. 7

  2.3. Training Applications and Environment ............................................................. 9

  2.4. Semantic Web and On-line Community ............................................................. 12

  2.5. Retention and Exchange of Engineering Knowledge ........................................ 13

3. Problem Definition and Requirement ............................................................................ 15

4. Overall Approach and Architectures ............................................................................ 18

  4.1. Knowledge Management through Engineering Knowledge Lifecycle .......... 18

  4.2. Details of the Architecture ................................................................................... 23

5. Engineering Knowledge Modeling ............................................................................... 28

  5.1. Three-tiered Structure of Engineering Ontologies ............................................ 28

  5.2. General Domain Ontology ................................................................................... 30

  5.3. Product Design Domain Ontologies .................................................................... 30
5.3.1. Feature Based Modeling Domain Ontology: FBM-DO .................................... 31
5.3.2. Assembly Modeling Domain: ASM-DO .................................................. 31
5.3.3. Application Specific Ontology: PROE-AO ................................................. 31
5.4. Kinematic Simulation Domain Ontologies ...................................................... 33
5.4.1. Simulation Domain Ontology: SIM-DO .................................................... 33
5.4.2. Application Specific Ontology: VADE-AO ................................................. 35
5.5. Assembly Training Domain Ontologies ......................................................... 36
5.5.1. Training Domain Ontology: TRAIN-DO .................................................. 36
5.5.2. Application Specific Ontology: iTrain-AO ................................................. 40
5.6. Procedural Knowledge Ontology .................................................................. 43
5.7. Summary ....................................................................................................... 45
6. Engineering Knowledge Acquisition ................................................................. 46
6.1. Population of Engineering Knowledge ........................................................... 46
6.2. Instantiation of Engineering Knowledge ....................................................... 47
6.3. Case Study ..................................................................................................... 51
6.3.1. Populate the product design and kinematic simulation knowledge .......... 52
6.3.2. Populate the training knowledge and procedural knowledge .................... 53
6.4. Summary ....................................................................................................... 55
7. Engineering Knowledge Integration ................................................................. 56

7.1. A Methodology for Ontology Mappings ....................................................... 56

7.2. Our Approach for Knowledge Integration ..................................................... 58

7.2.1. Definition of a shared ontology and extensions: ........................................... 58

7.2.2. Definition of Rule-based Mapping .............................................................. 59

7.2.3. Creation of Bridge Ontology ................................................................. 60

7.2.4. Creation of Graphical Application to facilitate mapping ......................... 60

7.3. Mapping Strategies ..................................................................................... 60

7.3.1. Mapping Strategy ....................................................................................... 62

7.3.2. Overall Mapping Procedure ................................................................. 63

7.4. Bridge Ontology ......................................................................................... 66

7.5. Mapping between Multiple Domains ............................................................ 67

7.6. OMA: Ontology Mapping Application ......................................................... 69

7.7. Case Study .................................................................................................. 74

7.8 Summary ...................................................................................................... 78

8. Engineering Knowledge Retrieval and Presentation ........................................... 80

8.1. Our Approach for Knowledge Representation ............................................ 81

8.2. Detail Architecture of iTtrain ........................................................................ 84
LIST OF FIGURES

Figure 1: Overall process for knowledge modeling, acquisition, integration, retrieval, presentation, and maintenance thru the engineering applications .................................. 19

Figure 2: Three-tiered Collaborative Engineering Knowledge Retention and Exchange Environment .............................................................................................................. 24

Figure 3: iTrain – An approach to increase synchronization and sharing of training models and procedures in a digital training ensemble ........................................... 25

Figure 4: Communication between engineering tools and the engineering knowledge management unit over a distributed network .......................................................... 26

Figure 5: Three-tiered structure of engineering ontologies in product design, assembly simulation and training domain .................................................................................. 29

Figure 6: UML diagram: Concepts and relationships for PROE-AO, a product design tool .......................................................................................................................... 32

Figure 7: UML diagram: Concept and relationships for SIM-DO, the simulation domain ontology .................................................................................................................. 33

Figure 8: UML diagram: Concept and relationships for VADE-AO, an assembly simulation tool .................................................................................................................. 35
Figure 9: UML diagram: Concepts and relationships for TRAIN-DO, the train domain ontology.................................................................................................................................................. 37

Figure 10: UML diagram: Concepts and relationships iTrain-AO........................................ 42

Figure 11: UML diagram for Procedural Ontology...................................................................... 44

Figure 12: The procedure to instantiate the training ontology................................................. 48

Figure 13: Population of instances for the simulation domain in the iTrain-AO ontology by using Protégé: A part component’s information and its kinematic animation..... 49

Figure 14: An example to acquire knowledge about a training course from the training ontology through CREEK.................................................................................................................. 51

Figure 15: Training Models with different train methods and the same train task by instantiating CBT and IMT in Protégé.............................................................................................................. 54

Figure 16: Three ontology-based Integration approaches [70].............................................. 57

Figure 17: Our hybrid approach for engineering product data ontology............................ 58

Figure 18: Overall mapping procedure between ontologies.................................................... 64

Figure 19: Diagram for Bridge Ontology.................................................................................. 66

Figure 20: Ontological mapping of concepts between a product design, a procedural knowledge, and a training ontology.................................................................................................. 69

Figure 21: Product data Ontology Mapping Application (OMA)......................................... 70
Figure 22: Framework for Mapping Application.............................................................. 71

Figure 23: Inheritance & composition hierarchies in the source ontology (PROE-AO) ... 76

Figure 24: Inheritance & composition hierarchies for the target ontology (VADE-AO) .. 77

Figure 25: Knowledge lifecycle in product design, assembly simulation and training domain and knowledge sharing between various engineering applications .......... 82

Figure 26: Isolation of applications and integration of data by using MVC (Model-View-Controller) design pattern................................................................. 85

Figure 27: Knowledge transfer between product design environment and training environment and key ontologies in the iTrain Architecture................................. 86

Figure 28: Manipulation between the user interfaces and the knowledge models ...... 87

Figure 29: An example of knowledge retrieval from an online community and a training environment.................................................................................................. 90

Figure 30: Software structure and libraries of VCAT ...................................................... 91

Figure 31: A snapshot from CREEK displaying instructions and relevant resources ..... 93

Figure 32: VCAT application importing and displaying training knowledge ................. 94

Figure 33: Screenshots from Online Community describing the Step 7 in the assembly processes of the wind turbine............................................................................. 95
Figure 34: A snapshot for training replacing WFC3 (Wide Field Camera 3) in the Hubble Scope: VCAT on the left and VrToolsPro on the right sharing the geometric models

.................................................................................................................................... 97

Figure 35: Knowledge evolution by updating an assembly instruction though CREEK and propagation to the training environment .................................................................. 102
LIST OF TABLES

Table 1: Comparison between CBT and IMT according to training concepts ............... 39

Table 2: CAD model and assembly instruction for a wind turbine’s hub ..................... 53

Table 3: Examples of instantiation of PKO:Process and PKO:PKExpertKnowledge...... 55

Table 4: Sample Definitions .......................................................................................... 61

Table 5: Sample Rules for mapping strategies ............................................................... 62

Table 6: Pseudo codes – Mapping procedure and creation of a Bridge Ontology......... 74

Table 7: Matching concepts between PROE-AO and VADE-AO ................................. 77

Table 8: An Output of Bridge Ontology (OWL file) ..................................................... 78

Table 9: Programming example to read, search, and update the knowledge by modifying
the next course item of the given course item.......................................................... 88

Table 10: Example of the query and inference in the procedural ontology ................. 89

Table 11: Programming example to read the training ontology and construct the data
structure in the engineering knowledge management unit.......................................... 96
CHAPTER ONE
INTRODUCTION

Ontology is defined as an explicit formal specification of a shared conceptualization [1][2]. In other words, all of the concepts, properties, and instances are explicitly defined, machine readable, capture consensus knowledge, and represent an abstract model of the phenomenon. In addition, it supports reasoning capabilities to test concept subsumption and consistency. Because most domains do not have a standard knowledge representation, ontology engineers often end up defining variations and individual flavors of domain concepts and terminologies, and as a result, these have different levels of fidelity and commonality. Therefore, ontology helps us represent and populate knowledge representation of product engineering. Also, it leads us a way to find underlying concept similarities between different knowledge representations and described how to translate different interpretations of product data automatically. Ultimately an ontology-based approach can improve interoperability by exchanging knowledge in a semantic level.

PDM (Product Data Management) is an emerging technology for a set of rules, principles, and methodologies to manage and control the product-related information during the whole of the product lifecycle [3]. PDM keeps track of massive amount of data and information throughout design, analysis, and manufacturing as well as service and disposal. Since product data can be used by many users or applications, there are few standard definitions of product data and each application has a unique data representation that is incompatible with other applications. As a result, data integration, conversion or alignment tasks are frequently needed throughout the product lifecycle. Either a manual
data translation between applications or implementation of a standard data definition will be very complicated, time-consuming, and error-prone. The advent of ontology based strategies can help users and applications efficiently exchange product data on the semantic level with minimal effort to adhering to previously agreed upon standard data definitions.

Digital training technologies are rapidly improving and can increase training effectiveness. As computing power has increased, users have moved past using simple mouse and keyboard based training applications to more advanced simulation applications using immersive environments. This spectrum ranges from simple computer-based training applications to complex immersive training environments.

Computer-Based Training (CBT) is a process in which the user has access to a series of lessons using a PC-based application with limited interactions, while Immersive Training (IMT) is a process where the user gets immersed in a virtual environment using special VR (Virtual Reality) devices. Typically these two types of training environments are developed entirely independently but they have a lot in common. For a coordinated and efficient experience, it is logical for a user to start in a simpler computer-based training (CBT) environment and after obtaining a basic understanding proceed to the more complex Immersive Training (IMT) environment. Creation and delivery of training content to span these tools and applications is a difficult task. These two types of applications are typically developed entirely independently; however, they have a lot in common. Accordingly, techniques to share common knowledge between training applications would be very useful. Hence, in parallel with advances in hardware and software for these training simulations it is necessary to advance our knowledge of how
to create a cohesive and consistent approach that allows consolidation, sharing, and mapping of methods and models to be used in multiple tools and applications for a single overall training scenario. Our work introduces an approach called iTrain (integrated Training) to integrate CBT and IMT tools by using ontologies in the product data assembly simulation domain and the training domain. Thus, the approach leads to a training a well-organized learning content can be reused in multiple applications or locations and the maintenance, modifications or evolutions of contents is carried out with ease.

An online community has an organizational structure focused on activities requiring sharing and cooperating, and hence is effective in preserving institutional knowledge by providing a higher quality interaction [4]. Traditional online communities or web sites are commonly integrated with relational databases and depend on queries against that database to retrieve information. However, there are several disadvantages. For example, a single database is unlikely to be used for multiple web sites and results in difficulties of inconsistency of data and out of sync over a distributed network of webpages [5]. A semantic web is a “web of data”. It has the potential to overcome the difficulties of the traditional approach because the semantic web can support integration and combination of data drawn from diverse sources because it is based on formal representations, such as ontology, for recording how the data relates to real world objects [6]. Hence semantic online communities based on ontologies show great potential for retaining and exchanging engineering knowledge.

This research aims to build a semantic online community for the knowledge retention and exchange of the engineering community. The community has unique
needs. For example, a particularly complex and critical activity is the integration of the data acquired by the semantic online community with the familiar everyday tools used by engineers. In isolation, this knowledge has the potential to not be very useful. Similarly, presenting this knowledge in these applications and environments in a way that makes sense to the engineers is not straightforward. The emphasis of this thesis is on the domain of product assembly design knowledge and product assembly procedural knowledge. The engineering tools that we will focus on for the integration are the CAD design tools for assembly modeling and training tools for procedural knowledge of assembly steps.

An engineering knowledge lifecycle consists of activities related to creation, usage, update, and disposal of the engineering knowledge. We define the entire lifecycle of an engineering knowledge as 5 specific steps: knowledge modeling, knowledge acquisition, knowledge integration, knowledge retrieval and presentation, and knowledge maintenance. Knowledge modeling conceptually constructs a knowledge structure by defining concepts, relations, and axioms in the specific domain and knowledge acquisition instantiates these concepts by association with actual data. Knowledge integration merges knowledge acquired from multiple domains. Knowledge retrieval includes activities to query information from the knowledge repository and knowledge presentation includes activities to display it in specific applications. Finally, knowledge maintenance seeks to evolve the knowledge by acquiring new knowledge and updating the existing knowledge.

There are three key contributions of this thesis in terms of the engineering knowledge lifecycle. First, the ontological mapping can transfer knowledge from one
domain to the other for the knowledge integration. Second, the iTrain environment for the knowledge presentation can utilize a unified training knowledge models which can be shared and presented on training applications. Last, the semantic online community will be used for the knowledge maintenance by acquiring new engineering information and transferring it to proper domains.
CHAPTER TWO

BACKGROUND

2.1. Traditional Exchange of Engineering Data

The necessity of data translations and data integration in the product life cycle occurs in different contexts, such as application to application or user to user situations. For example, a product model drawn in a CAD software may need to be analyzed in a different CAE software. Or else, consider the situation where one engineer may design a product model using Pro/Engineer and may need to deliver it to another engineer who prefers CATIA.

In order to transfer product data, engineers typically consider two choices. An indirect method may be to use a standard neutral data format, such as IGES or STEP and direct methods would be to use a third-party tool specializing in converting data. Using neutral formats has a critical drawback because some important information obtained through CAD systems, such as feature-tree information and constraints in parametric modeling systems, can be lost. For example, constraints between parts or feature entities such as PAD, RESOLVE, and SHELL, cannot be reviewed in the target system. Direct methods such as Acc-u-Trans [7] and Pro/VADE [8] tool are good to convert the product definition data of one system to another. However, developers need to learn the APIs provided by both source and target system. In addition, the data format is also specialized for each system and persons other than the experts are not able to easily understand information from the raw data file, unless the person creating the data file is meticulous about detailed documentation. For example, when a model data is converted from Pro/Engineer, to VADE, a VR-based simulation tool created at Washington State
University, users need to know how to extract information and data from Pro/Engineer via API functions and then how to create an input data file for VADE. Both direct and indirect methods have some limitations. Product data is not written by consensus terminologies and is not shared among engineers across domains. Even though the data format is standard, such as the STEP and IGES files, some information is lost during conversions. It is safe to conclude that these problems arise because data is exchanged between applications and/or users on the syntactical level.

2.2. Engineering Ontologies and Ontological Mapping

Ontology provides a vocabulary for representing knowledge about a domain [9] and supports powerful tools that enable inference, querying, and information retrieval. W3C WEB Ontology Working Group has accepted OWL as the standardized and broadly accepted ontology language of the Semantic Web [10][11]. As a result, OWL has now been used to conceptualize engineering design semantics, to share knowledge over the web [12], and to overcome a lack of interoperability by creating a collaborative environment [13]. NIST has pioneered work in using ontology and description logics to represent product models from CAD systems, some based on a pre-existing Core Product Model (CPM) and an Open Assembly Model (OAM) [14][15][16]. An ontology-based assembly model was established to express assembly features and relations between features. Assembly constraints were explicitly represented using OWL triples and SWRL rules [17]. Furthermore, ontologies have been used to represent conceptualizations of products and represent and capture the design semantics [18][19].

The primary objective of an ontology is to capture consensual knowledge of a specific domain in a generic and formal way and allow this knowledge to be reused and
shared across applications and by groups of people. However, because ontologies are widely used in domains as well as on the semantic web, many different ontologies that cover the same domain exist. In order to come up with the sharing and interoperability of knowledge regardless of heterogeneity of ontologies in many domains, a technique to reconcile differences between ontologies is required. This technique, called ontology mapping, is used to find relationships and correspondences between semantic entities, such as concepts, properties, or instances, which are defined in two different ontologies. The area of ontology mapping is a subject of active research in different disciplines such as databases, peer-to-peer architectures, information integration, service-oriented architectures, etc. Rather than simply exchanging data on the syntactic level like current data integration approaches, research in ontology mapping on the investigation of solutions for exchanging the meaning of information between different software applications [20]. The key benefits of an ontology based on OWL is the ability to share models among applications on the web, manage data in the open environment, support consistency and quality checking across models, and support an automated reasoning tool. Ontology technique was exploited for development of an automatic transmission and showed how to exchange semantic data between a geometric model and a functional model [20].

Noy and Musen contributed significantly to the effort related to reconciling disparate ontologies semi-automatically and created an application called PROMPT [21]. They defined ontology alignment as a method to establish mappings (or links) between two ontologies by aligning paths anchored by two equivalent classes at the end. They also defined ontology merging as combining ontologies by term-matching algorithms and
ontology versioning as a way to keep track of modifications within ontologies. An Ontology Mapping FRAmework called MAFRA\[22]\[23]\[24] has been built successfully to provide an approach and conceptual framework for the overall distributed mapping process. Within MAFRA, there are various “layers”. For example, a similarity layer measures similarities between entities from the source and target ontology and a semantic bridge layer defines correspondences between source and target entities.

Thus, several possible mapping procedures exist between two ontologies. Since manual mapping techniques consist of time-consuming and error-prone tasks, most existing mapping skills provide users with tools or algorithms to (semi-)automatically map concepts and properties for large-scaled ontologies. However, they attempt to apply mapping strategies broadly to all domains and are mainly dependant on lexical matching which looks for similarities between names of classes and properties.

There is a need for research into the use of ontologies to capture engineering knowledge along and make this knowledge available for integration with engineering tools and also for access in training sessions. Also, the (semi-)automatic knowledge transition between heterogeneous domain are needed for the purpose of minimizing the knowledge loss.

2.3. Training Applications and Environment

Computer-Based Training (CBT) often involves self-paced training exercises with an interchange of information and feedback built into the modules. The benefits of CBT interactive training are as follows: 1) The trainee receives immediate feedback on performance in a structured and non-threatening environment. 2) Trainees can focus on the information they need, at any point, when needed. 3) The curriculum and pace can be
tailored to the experience level of each trainee. 4) Active learning occurs as trainees interact one-on-one with their computer [25].

Similarly, there are considerable benefits to use VR-based Immersive Training (IMT) for training in areas such as military, manufacturing, and medicine [26]-[29]. The literature states that VR-based learning can, in some cases, offer profound benefits over conventional learning methods [30].

We have created and deployed over several years a virtual assembly and design environment called VADE for training of assembly and disassembly scenarios in the automobile manufacturing industry. For example, this environment has been used to simulate the assembly of a fifth wheel on a truckcab chassis for PACCAR[31] and the assembly of a large press machine for Komatsu [31][32]. This environment is the IMT that will be used in our iTrain ensemble.

It is important that employees have access to the collective knowledge and procedures in a suitable training environment. Studies have shown that a well designed technology-supported approach can significantly improve learning and performance [33]. Computer-Based Training (CBT) and Immersive Training (IMT) can maximize engagement with information and procedures and there are many documented benefits [34]. There is considerable literature describing how immersive virtual reality (VR) has opened new realms in the area of training related to the military, manufacturing, and medicine [25]. These can, in some cases, offer profound benefits over conventional learning methods [35][36]. The term “blended training” is now becoming increasingly used by companies to describe the way they combine traditional and electronic learning
to provide the best overall results [37]-[41]. The key to a properly developed blended learning program is to know what, when and how to combine these strategies [38].

Shareable Content Object Reference Model (SCORM), developed by ADL (Advanced Distributed Learning Initiative), fosters creation of reusable learning content as "instructional objects" within a common technical framework and describes that technical framework by providing a harmonized set of guidelines, specifications and standards [42]. It specifies how to organize a content model from raw data such as audio/video, pictures or webpage, how to aggregate contents and how to fulfill content packaging. A Learning Management System (LMS) is software that automates training event administration through a set of services that launches learning content, keeps track of learner progress, and sequences learning content [43]. SCORM is being incorporated into LMS and is being widely used for training courses. It is a standard in developing interoperable learning objects [44]. A High Level Architecture (HLA) was developed by the US Department of Defense (DoD) and provides the specification of a common technical architecture for communicating between simulation systems across computer platforms [45]. Boeing has integrated learning environments with flight simulations using HLA and SCORM and increased the training efficiency and student throughput [46].

There is some ongoing research to make use of ontology techniques for managing learning objects. The SORCEROR (Source for Rapid Open Content in Learning Object Repositories) project indexes or classifies online learning resources available on the Internet by using an ontology model based on metadata of learning standard as SCORM and Dublin Core metadata [47]. A Content Repository Management System (CRMS) [48] has been developed for learning objects to be stored and managed on a web-based
learning environment. Based on CRMS, the Visualized Online Authoring Tool (VOAT) and Ontology-Based Course Editor (OBCE) [49] help users author their teaching materials by outlines of domain contents on CRMS. OntoAIMS [50] facilitates the creation, editing, maintenance and reusability of teaching materials by applying ontology. Although these provide an excellent start in the overall area of integrating ontologies and learning objects, these capabilities are not appropriate for an assembly simulation and training exercise. For example, there is no support for data exchange and flow of information between creation of 3D models and creation of learning contents. Users still design their mechanical models and separately compose learning materials.

For engineering procedural knowledge training systems that will be used by new employees, a blended training environment with a computer based environment and an immersive environment is most appropriate to support intricate manipulation of parts and realistic interactions with the artifacts in the environments for procedures such as assembly/disassembly.

Research is needed into methods to bring the knowledge captured from experts into this integrated training environment consisting of a computer-based training (CBT) environment and an Immersive Training (IMT) environment.

2.4. Semantic Web and On-line Community

The semantic web is a medium in which people can collaborate by sharing information and collaborate on the common collection of knowledge [5]. Semantic Wikis has been used to form a central knowledge base for Semantic search engine which performs Computational Fluid Dynamics simulations and assist airfoil design [51]. The semantic tools provided by the EU-funded WIDE project supports innovative product
design in the automotive field [52]. The semantic web includes technologies, such as, Resource Description Framework (RDF), RDF Schema (RDFS) and the Web Ontology Language (OWL). The meaningful definition and use of ontologies is a critical aspect of the semantic web [6].

Shneiderman [53] discusses the opportunities for development of new tools to support social creativity. Sosa and Gero [54] study the effect of interactions between designers and social ties on creativity. Gero and Maher [55] discuss that knowledge-based systems can be used to support creative design in various ways such as by providing better user interfaces, bigger knowledge bases, better knowledge representations, and a computational model with flexible search mechanism. Large communities on sites such as Facebook for general networking (58 million users), Flickr for photo sharing (4 million users), LinkedIn for business networking (17 million users), and soundpedia (3.5 million users) for music sharing show that individuals are increasingly participating in collaborations over the Internet at massive scales.

The use of the semantic web and ontologies continues to show great promise for knowledge exchange in the engineering domain.

2.5. Retention and Exchange of Engineering Knowledge

A common approach to capturing and storing product data from design tools employs database technologies. Design rules as well as their complex interdependencies can be stored in a database and database queries are permitted. However, traditional database systems are not applicable for data management involving heterogeneous but interrelated data. This is because not all the data can fit conveniently into the very structured format of database systems [56]. Also, database systems can’t provide an
environment for semi-automatically creating relationships or improving and maintaining existing relationships between participants. Other approaches supporting knowledge capturing and reasoning for engineering analysis and design tasks include Object-role modeling approaches and context based modeling approaches (CML) [57]. However, CML doesn’t provide the support for interoperability since it emphasizes the development of context models for particular applications or application domains. Another approach, case-based reasoning (CBR) systems [58], embodies expertise in a library of past cases containing descriptions of the problems and their solutions. However CBR relies heavily on previous cases and requires a large number of high quality cases to be effective. The rule-based expert system is a computer program that provides expert-level solutions to ‘important problems’ [59] with knowledge represented as rules. However, most expert systems emphasize more on technologies than knowledge itself [60] and the old expert systems are monolithic, and computer-centric. Modern expert system should be built [61].

There are many strategies for knowledge preservation but there are new approaches and opportunities that need to be considered for knowledge capture. Our research seeks to use a modern approach to capture design knowledge from experts through an online community and present it to new engineers.
CHAPTER THREE

PROBLEM DEFINITION AND REQUIREMENT

Traditionally most engineering applications are designed independently and have their own database which might not be compatible with databases of other applications. When the engineering applications try to exchange or transfer information each other on the syntactical level, some special tools are usually developed for data transfer between databases but there always exist critical problems, such as, loss of information, inconsistency of data, and out of sync. Additionally, new information or data acquired from application users, such as, feedbacks, comments, pictures, and videos, is properly not retained and not shared between other applications. These are because an ontology technology is adopted to semantically exchange the meaning of information between application and the online community is considered to retain and exchange thoughts and information between users or between applications.

The purpose of this research is to find a way to capture engineering knowledge, store it, and keep it open to extensions and modifications over different domains, applications, or persons. The overall requirement is to create shared knowledge models, to establish an integrated training environment, and to design a semantic online community. The research described in this thesis will systematically study the following in the context of the engineering knowledge lifecycle.

1. Knowledge Modeling
   
   - Establish broad categories of knowledge in engineering product assembly that an engineering organization would like to retain.
• Define a computational model based on ontologies to store this information and also to allow querying and reasoning to allow basic inferences and to derive new information not explicitly expressed in the knowledge base.

2. Knowledge Acquisition

• Extract knowledge from a CAD/CAE tools, such as, part information, assembly constraints and kinematic motions as well as resources and define assembly procedures.

• Define methods to collect information and related skills from experts in the engineering organization. Establish broad categories of knowledge in engineering design and assembly that an engineering organization would like to retain.

• Provide user-friendly interfaces or formal templates to accept users’ thoughts and information which are ultimately stored into the knowledge models.

• Populate the knowledge into ontologies

3. Knowledge Integration

• Support a semi-automatic method to exchange concepts between knowledge models in different domains.

4. Knowledge Retrieval and Presentation

• Model an approach to create an online community specifically for engineers that will enable practicing engineers to present queries and requests to a community of experts for an exchange of ideas, knowledge, problems, and solutions.

• Determine methods to present the procedural engineering design and assembly information in an integrated training environment consisting of computer-based training and immersive training modules.
• Determine methods to present the procedural engineering design and assembly information in an integrated training environment consisting of computer-based training and immersive training modules.

• Allow users to be involved in training sessions with computer-based or VR-based environment.

5. Knowledge Maintenance

• Keep the domain knowledge up to date by addressing modifications of CAD models, assembly procedures or training configurations and feedbacks or comments after experiencing the training courses.

• Formulate methods to relate this engineering knowledge in a meaningful way through relationships in computer-aided design (CAD) models, computer-aided engineering (CAE) and Product Lifecycle Management (PLM) systems since these are the master models for most engineering domains.

• Establish a collaborative engineering environment where one modification by one application is properly propagated into other applications.

• Accumulate any trainees’ feedback and convey them back to ontology.

Overall, there is a necessity to establish a systematic architecture to distribute knowledge to a variety of applications, to evolve knowledge through the communications among users, and to trigger an event for notifying a status of knowledge.
CHAPTER FOUR

OVERALL APPROACH AND ARCHITECTURES

This chapter presents our overall approach for knowledge management through the engineering knowledge lifecycle and shows architecture to integrate engineering applications.

4.1. Knowledge Management through Engineering Knowledge Lifecycle

Our current work will show that the semantic online community can be a useful method for the engineering knowledge lifecycle and can be well integrated with engineering applications. Figure 1 displays the overall process of the engineering knowledge lifecycle performed by interactions between engineering applications and knowledge repository including ontologies and resources. The semantic online community is involved in activities of knowledge acquisition, presentation and maintenance by interacting with users and the knowledge repository. Even though engineering knowledge is acquired from different users or tools, the knowledge will be integrated inside the knowledge repository and be presented to engineering tools continuously. Furthermore, the knowledge keeps being updated through the semantic online community and transferred to the training environment.
The engineering knowledge lifecycle goes through the multiple phases involving the knowledge modeling, knowledge acquisition, knowledge integration, and knowledge retrieval and presentation. Also, the knowledge will be maintained by acquiring new knowledge or updating the existing knowledge. It will mainly cover two domains: Product Assembly Design Domain and Product Assembly Training Domain. The following briefly explains roles of each phase in the engineering knowledge lifecycle.

**Knowledge Modeling:** The process of the engineering knowledge lifecycle starts with knowledge modeling. The knowledge modeling conceptually constructs a knowledge structure by defining concepts, relations and axioms in a specific domain. There will be several common terms and metadata that span across the various domains and various activities. These terms and their intent will be collected and parsed to form the basis for ontologies. Concepts will be added to the ontology as needed and properties,
rules, and axioms will be also added. These knowledge models are stored into the knowledge repository and waited to be instantiated.

**Knowledge Acquisition:** This phase gathers engineering knowledge and information from domain experts and populates them into the engineering knowledge models. That is, the knowledge acquisition instantiates concepts by associating with actual data sequentially after the knowledge modeling. Although the knowledge acquisition by instantiating concepts will be partially human-assisted, we will provide methods to automatically accumulate domain knowledge into ontologies and user-friendly interfaces to acquire data and knowledge. For examples, an add-in tool for Pro/Engineer can extract information about product designs. Also, a user interface like the web-based application will be provided to help users input new information into the knowledge repository. Furthermore, an ontology mapping method will be investigated to partially transfer knowledge between different ontologies.

**Knowledge Integration:** It intends to merge knowledge acquired from multiple domains in order to effectively exchange knowledge between different knowledge representations. Even though each domain has different semantic representations with different concepts and axioms, there probably exist similar terminologies having the same meaning. An ontology mapping technique will help finding similar or matched concepts from two different ontologies. Therefore, some knowledge populated in a product design ontology can be ultimately exchanged into either a training ontology or a procedural knowledge ontology. The ontology mapping method will be also used to interconnect knowledge from existing engineers and engineering process documents to data and knowledge from design and training systems. It is interesting to note that although
significant work has been done in capturing design knowledge during the design process, there is hardly any focus on capturing process knowledge in a form that can be stored and used with CAD and other digital data. If the process and rules used by the experienced knowledge engineer can be captured and codified, then we can start capturing true engineering and process knowledge for the future.

**Knowledge Retrieval and Presentation:** It refers to activities to query information from the knowledge repository and to display it on the engineering applications. The knowledge in the training domain will be presented to new engineers through various types of training tools. Training for engaging with procedural knowledge, such as assembly simulations, will be provided using a wide range of technologies – from a simple computer-based training (CBT) approach to a complex immersive training (IMT) approach. The CBT approach allows user interactions through traditional keyboard and mouse while the IMT approach immerses the user in a virtual environment for a more realistic experience. Creation and delivery of training contents to span these applications is a difficult task. Techniques to share common knowledge such as models, assembly planning and sequences, and practices and feedbacks between training applications would be very useful. Information between CBT and IMT tools will be integrated by using ontologies. The ontologies represent semantic knowledge about the models to be used in the training exercises, paths/sequences of the assembly process, and structures/flows for the training. The online community can also allow users to manage training contents. Furthermore, the training applications can be incorporated with SCORM, a Sharable Content Object Reference Model, so that all events and actions during training processes can be recorded, tracked and evaluated by learning management systems.
**Knowledge Maintenance**: It intends to evolve the engineering knowledge and update raw resources inside the engineering knowledge repositories. Knowledge and data in the repositories are sharable to any engineering applications so that they could be updated or removed or new knowledge might be created. Activities for the knowledge maintenance will contain knowledge propagation. For an example, when the online community modifies the assembly procedure or instruction, the modification will be committed to CBT applications.

While the engineering knowledge is created, merged, presented and updated over diverse engineering tools as seen in Figure 1, the engineering knowledge models need to be well managed and maintained through the engineering life cycle and to be independent from the engineering applications. The acquired engineering knowledge from multiple domains should be merged and can be accessed individually by engineering applications including the online community. Ultimately the engineering knowledge will be shared among the engineering applications and collaboratively maintained by them.

As well as creating the engineering knowledge management unit, the semantic online community application, CREEK (Community for Retention and Exchange of Engineering Knowledge), is designed in order to manage knowledge and data in the product assembly design and product assembly training domains over the web. It will provide a user interface to allow users to create or edit the training procedures and contents. It will also access to search engines to retrieve information from the knowledge repository so that it can present information relevant to a specific training, such as, methods, equipment, and tasks, on the web. Furthermore, it allows retired engineers or experts to continue to creatively participate in the knowledge maintenance through
informal interactions with trainees or inexperienced engineers. Interactions in this community might bring about new information which triggers the process of the knowledge acquisition. Thus, the knowledge acquisition is not limited to the initial interviews with retirees but is an on-going process through this online community.

The following section will explain about the engineering knowledge management unit and the semantic online community in detail.

4.2. Details of the Architecture

Figure 2 illustrates the overall architecture of our collaborative engineering knowledge retention and exchange environment. The overall system follows three-tier architecture: Engineering Application Unit, Engineering Knowledge Management Unit, and Engineering Knowledge Repository Unit. First, the engineering application unit, as a presentation tier, includes engineering tools to present engineering knowledge or acquire new knowledge. It includes CAD design tools, training tools, and semantic web communities and can communicate with the engineering knowledge management unit to retrieve product assembly information. Next, the engineering knowledge management unit, as a logic tier, plays a role in providing business logics, accessing to the knowledge repository and returning information necessary to the engineering applications. It can also write new information or date into the repository. It is designed with JDK (Java Development Kit) and JSP (Java Server Page) as an application programming interface. It implements various ways to access to the knowledge repository and extract information, such as, Protégé or Jena APIs for accessing to ontologies, SWRL and Pellet APIs for reasoning, and mapping strategies and algorithms for supporting ontological mapping processes. Furthermore, it provides engineering applications with business logics, such as,
data insertion/update, querying, reasoning, and decision making. Last, the engineering knowledge repository unit, as a data tier, stores all the ontology models and raw resources and retrieves information. It will be used to search engineering knowledge and to manage it through the addition of new knowledge, modification or removal so that it should be prepared for the environment where the engineering information of products keeps changing by means of new requirement and feedback after training sessions.

Figure 2: Three-tiered Collaborative Engineering Knowledge Retention and Exchange Environment

In this architecture, each engineering application will individually access to the knowledge repository and read or write information, and also modifications from on the engineering application will have influence on other applications. Therefore, the engineering knowledge repository will be collaboratively managed by all the engineering applications. This research will mainly focus on an online community, CBT and IMT, but
it can support other applications, such as, SCORM. Thus, Figure 3 shows the digital training ensemble and environments at a conceptual level and depicts that some contents can be synchronized and shared between training tools, even though each training tool requires different equipment and user interactions. When all the information and resources of both common and disparate contents are populated into one repository, these training tools can share training contents and be integrated at the data level so that they could exploit proper information and resources necessary for a specific training, such as, methods, equipment, models, and tasks. The most advantageous fact is that one modification in the repository can be propagated through all the training tools. This research only focuses on integration between CBT, IMT, and Online community, but the web-based training tool can be applicable.

Figure 3: iTrain – An approach to increase synchronization and sharing of training models and procedures in a digital training ensemble

Even though the engineering knowledge management unit in Figure 2 is established and is accessible by engineering design tools and training tools, it could not
support an interface to exchange the engineering knowledge between experienced engineers and inexperienced engineers, and trainers and trainees, or retain knowledge from retired engineers. Thus, we employ a web-based community to enable members to exchange their thought and knowledge on the web and its community will make use of the semantic knowledge instead of traditional databases.

Figure 4 displays how an online community and training applications are related with the engineering knowledge management system and also depicts the detail structure of the knowledge management system. Users will participate in the online community using web browsers and access to the knowledge management system. As a web service, Apache Tomcat 6.0 [62] is used and it is an open source platform to enable the connectivity and access to the knowledge system. Because Apache Tomcat serves as a web servlet container by using Struts2 [63] and JSP, it can communicate with the knowledge management unit importing ontologies and resources, like 3D geometric models, texts, videos, and so on.

Figure 4: Communication between engineering tools and the engineering knowledge management unit over a distributed network
The engineering knowledge management unit loads ontological models and resources from the knowledge repository and creates a proper data structure to hold concepts, instances, and relations in the ontologies. The data structure will be useful for the online community or training tools to fast access to information rather than to directly access to the ontologies. The reasoners and query engines will help verify and search information as well as infer implicit knowledge. The knowledge management unit will make logical decision and process information. Currently the online community can communicate with the knowledge management system through the web service while the training tools can do through the simple TCP/IP.

This online community application can be extended to be used in other aspects. For example, users can query and search CAD/CAM models from PLM systems and convert them for manipulation and viewing at a participant location. It can also provide methods to allow multiple people to simultaneously access, annotate, and manipulate 3D models or videos. Furthermore, a search engine can be added in order to query the raw database and repository as well the knowledge base and PDM/PLM systems.

According to Figure 4, the following chapters will give a full detail of knowledge modeling with ontologies and their instantiation in the engineering knowledge repository, how the ontologies are integrated in the engineering knowledge management unit, how knowledge is presented by CREEK, CBT and IMT, and how knowledge can be maintained through CREEK and the engineering knowledge management unit.
CHAPTER FIVE

ENGINEERING KNOWLEDGE MODELING

This chapter explains key ontologies and the structure of ontologies. In designing the ontologies, our approach divides the engineering domain into more specific domains and designs an individual ontology for each specific domain: Feature-Based Modeling Domain defining design features, Assembly Modeling Domain defining assembly components and constraints, Assembly Simulation Domain relevant to the assembly simulation, Training Domain for training environment and learning contents, and Procedural Knowledge Domain for assembly procedures. These specific domains can be combined and evolved to be appropriate for the applications or tools.

5.1. Three-tiered Structure of Engineering Ontologies

Our ontologies are constructed under a three-tiered hybrid approach, as seen in the Figure 17. An upper global ontology defines common concepts and properties and the lower local ontologies extend the concepts and properties for their each domain. In this manner, it will be easier to translate information between local ontologies by discovering similarities or relatedness.

The General Domain Ontology (GDO) is first designed to provide common terminologies, such as Component or Geometry, which can be inherited by the domain ontologies such as engineering product design domain, assembly simulation domain and training domain. Next, the Domain Specific Ontologies (like FBM-DO, ASM-DO, SIM-DO, or TRAIN-DO) which form the second level of hierarchies add new domain-specific vocabularies uniquely belonging to that particular domain. Going one level further down, the third level is the Application Specific Ontology (such as PROE-AO, and iTrain-AO)
and captures semantics unique to each application. For example, Pro/E and CATIA are two different applications in the 3D parametric feature based geometric modeling domain. The knowledge transfer between different application specific ontologies can be accomplished through mapping procedures which discovers similar or matching concepts and properties.

For this work, we seek to utilize the 3-tiered ontology architecture and extend it by creating one ontology at the domain level (TRAIN-DO) and one ontology at the application level (ITRAIN-AO). An application specific ontology for the iTTrain environment will be designed and called as iTTrain-AO. There is another possible option by creating an individual ontology for each training tool, for example, CBT-AO for CBT tools and IMT-AO for IMT tools. In this circumstance, the ontological mapping will be required for the knowledge transfer between CBT-AO and IMT-AO. However, our work seeks to design an integrated knowledge repository independent from training tools by providing one iTTrain-AO ontology.

Figure 5: Three-tiered structure of engineering ontologies in product design, assembly simulation and training domain

For this work, we seek to utilize the 3-tiered ontology architecture and extend it by creating one ontology at the domain level (TRAIN-DO) and one ontology at the application level (ITRAIN-AO). An application specific ontology for the iTTrain environment will be designed and called as iTTrain-AO. There is another possible option by creating an individual ontology for each training tool, for example, CBT-AO for CBT tools and IMT-AO for IMT tools. In this circumstance, the ontological mapping will be required for the knowledge transfer between CBT-AO and IMT-AO. However, our work seeks to design an integrated knowledge repository independent from training tools by providing one iTTrain-AO ontology.
We will discuss some of these ontologies in more detail now. We group them into 5 categories: General Domain Ontology, Product Design Ontologies, Kinematic Simulation Ontologies, Training Ontologies, and Procedural Knowledge Ontology.

5.2. General Domain Ontology

GDO describes basic concepts that represent commonalities applied to any domain ontologies. It basically defines three concepts: Component, Geometry and Behavior.

- **Component**: It is used to describe the elements of a product model.
- **Geometry**: It represents knowledge representation related to geometry.
- **Behavior**: It is defined to record the design intent and history during design processes, e.g. the way a 3D feature is created, imposed actions in an assembling process, etc.

Additionally, GDO has basic properties, such as, *is-a, has_part_of, has_attribute_of* and all the properties in domain specific ontologies and application specific ontologies should be sub-properties of one of them.

- **Is-a**: It implies the inheritance relations between concepts.
- **has_part_of**: It reflects the composition relations between concepts.
- **has_attribute_of**: It defines the relations between an object and its attributes.

5.3. Product Design Domain Ontologies

The product design domain contains a feature based modeling and an assembly modeling domain. And the application specific ontologies will import the two domain ontologies and extends concepts to be appropriate to CAD applications.
5.3.1. Feature Based Modeling Domain Ontology: FBM-DO

FBM-DO defines physically meaningful features like extrusion, revolve, hole, and round as well as geometries like point, line curve, and surface. It inherits the Geometry concept in GDO and subclasses the Feature concept for the solid modeling and the Feature_Element concept for component features like Solid, Curve and so on.

5.3.2. Assembly Modeling Domain: ASM-DO

ASM-DO represents terminologies relevant to designing assemblies in CAD tools like part, subassembly, assembly, assembly constraints. Those are defined as Part, Subassembly, Assembly, and Component_Constraint respectively.

5.3.3. Application Specific Ontology: PROE-AO

PROE-AO in Figure 17 is inherited from assembly modeling domain (ASM-DO) and the feature based modeling domain (FBM-DO) which in turn are extended from GDO. In other words, Assembly, Subassembly and Part concepts representing component compositions, Component_Constraints for relationships between components are imported from ASM-DO, and Feature for features of the solid modeling and Feature_Element for component features are borrowed from FBM-DO. Figure 6 draws an UML diagram to display the concept relationships and key properties for the application specific ontology (PROE-AO) in the product design domain.
PROE-AO inherits concepts/properties from both domains and specifies them for Pro/Engineer CAD tool. For example, it specifies $ASM-DO:Component\_Constraint$ by classifying $Align$, $Mate$, and $Insert$ and $FBM-DO:Feature$ by specifying $Extrude$, $Revolve$, and so on.

These concepts and properties can be instantiated by automatically extracting information from Pro/Engineer through a plug-in program which enables to access to geometry, feature, and assembly information through Pro/Engineer API (Application Programming Interfaces). In the training environment, knowledge required for comprising the assembly hierarchy and assembly constraint will be reused to display geometric part/assembly models, demonstrate assembly/disassembly sequences, and analyze assembly planning in a virtual assembly training environment.
5.4. Kinematic Simulation Domain Ontologies

The kinematic simulation domain contains an assembly modeling and a simulation domain. As seen in Figure 5, the assembly modeling domain can be also imported to the product design domain. Application specific ontologies will import the two domain ontologies and extends concepts to be appropriate to simulation tools.

5.4.1. Simulation Domain Ontology: SIM-DO

As another domain ontology, simulation domain ontology is inherited from GDO and can define different types of simulations, such as finite element analysis, kinematic motion analysis, and etc. This thesis only focuses on the kinematic simulation and defines types of a kinematic behavior which will belong to components including parts, subassembly, or assembly. Figure 7 displays the concepts and properties defined in SIM-DO ontology.

![Figure 7: UML diagram: Concept and relationships for SIM-DO, the simulation domain ontology](image-url)
As Figure 7 shows, a member of *KinematicSimulation*, *KinematicMotion* is subclassed into *FreeMotion* and *JointMotion*. *FreeMotion* is a set of *LinearMotion*, *RotationalMotion*, *BezierCurveMotion*, and combination of previous motions. It should be composed of *MotionPaths* which define a trajectory of the motion defining a series of locations, orientations and scales of the components over time. *JointMotion* express the conditional motion types between two components: revolute action, prismatic action, and etc. These kinematic schemas are defined in ISO 10303-105. For example, *PrismaticJointMotion* can be expressed with an axis where a component slides and transitional maximum and minimum values with respect to the local coordinate system of the axis. Also, *RevoluteJointMotion* can be constrained with an axis where a component revolves and rotational maximum and minimum values of rotation. *MotionInterpolation* will be used to calculate speed between path nodes to generate the smooth motion. The following expresses quantifier and cardinality restrictions of some concepts by using the rule language of the semantic web, called SWRL. As an example, the *MotionPathNode* can be read that there exist a *MotionPathNode* x, where x has transform y which is an instance of *Rotate*, x has transform z which is an instance of *Scale*, x has transform w which is an instance of *Translate*.

\[
\begin{align*}
\text{FreeMotion}(?x) & \rightarrow \ (\text{hasMotionPath} \geq 1) \ (?x) \\
\text{MotionPath}(?x) & \rightarrow \ (\text{hasMotionPathNode} \geq 2) \ (?x) \\
\text{MotionPathNode}(?x) & \rightarrow \ \text{hasTransform}(?x, \ ?y) \land \text{Rotate}(?y) \land \text{hasTransform}(?x, \ ?z) \\
& \land \text{Scale}(?z) \land \text{hasTransform}(?x, \ ?w) \land \text{Translate}(?w) \\
\text{PrismaticJointMotion}(?x) & \rightarrow \ \text{hasAxis}(?x, \ ?y) \land \text{Axis}(?x) \land \text{hasTransform}(?x, \ ?z) \land \text{Translate}(?z) \land \text{hasTransform}(?x, \ ?w) \land \text{Translate}(?w) \\
\text{RevoluteJointMotion}(?x) & \rightarrow \ \text{hasAxis}(?x, \ ?y) \land \text{Axis}(?x) \land \text{hasTransform}(?x, \ ?z) \land \text{Rotate}(?z) \land \text{hasTransform}(?x, \ ?w) \land \text{Rotate}(?w)
\end{align*}
\]
5.4.2. Application Specific Ontology: VADE-AO

VADE allows an engineer to consider assembly issues early in the design cycle and enables the user to be immersed in an assembly simulation environment. It should be noted that the VADE-AO focuses more on the assembling motions, besides the geometry attributes of models. VADE-AO adds the concepts/properties about assembly simulation specific in VADE system. There are different assembly constraint definitions from other ASOs, e.g. ConstraintOnPlane, ConstraintOnAxis, and several special concepts like Constraint_Sequence, Impose_behavior (Rotate, Translate), Joint, etc, which result from assembly simulation process in VADE.

Figure 8: UML diagram: Concept and relationships for VADE-AO, an assembly simulation tool
5.5. Assembly Training Domain Ontologies

The assembly training domain contains a training domain and partially a simulation domain. Application specific ontologies like iTrain-AO will import the two domain ontologies and extends concepts to be appropriate to training applications.

5.5.1. Training Domain Ontology: TRAIN-DO

The training domain ontology deals with concepts and relations existing in a training environment, such as, persons, equipment, training methods, and training tasks. In the training process, knowledge \((\text{TrainTask})\) is transferred from trainers to trainees \((\text{TrainPerson})\) through different types of methods \((\text{TrainMethod})\). And the knowledge transfer might require some devices or tools \((\text{TrainEquipment})\). Based on these explanations, Figure 9 is drawn and shows the composition and inheritance of concepts and some key properties. A \text{Train} is made up of \text{TrainPersons} and \text{TrainModes} and components of a \text{TrainMode} are \text{TrainMethod}, \text{TrainEquipment} and \text{TrainTask}. Additionally, \text{TrainTask} defines course organizations and presentations and is composed of multiple training lessons which are represented by \text{Course}, \text{CourseModule} and \text{CourseItem}. 

Some key concepts in Train-DO are as follows.

- **TrainPerson**: This is a concept related to persons involved in the training environment. A person can be an author, a mentor or a trainee.

- **TrainMethod**: This is a concept related to methods for transferring knowledge and information, such as, in-person demo, lecture, simulation or VR-based training. CBT can be an instance of *Train-DO:Simulation* and IMT can be an instance of *Train-DO:VirtualReality*.

- **TrainEquipment**: It is a concept related to equipment necessary for the training, such as, keyboard, mouse, display system, tracking system, cyber globe, and etc.
- **TrainTask**: It is a concept related to definitions of tasks and their sequences and organizations.

- **Course**: It is a concept describing a task in a broad sense. For example, assembly of the body and assembly of the generator in a wind turbine.

- **CourseModule**: It is a concept describing a sub-task where trainee will participate and it is a reusable learning content. For example, assembly of the body’s main hub and assembly of the body’s blades in a wind turbine.

- **CourseItem**: It a concept describing a specific procedure to explain each step in the assembly process or other operations. For example, assemble a chuck motor with the main hub.

- **Resource**: It a concept having linkages to raw media files including pictures, movies, audios, texts, or another webpage.

- **FeedbackAndComment**: It is a concept related to the comment and feedback captured from trainees or experts.

Each course item can be designed by adding and organizing raw materials, such as instructional texts, images, and the relevant video or audio files, which can visually depict what trainees should do. Additionally, it can give references to executable programs or websites. All these raw materials as well as ontology models will be stored into the knowledge repository. Besides, **SequenceControlMode** concept is used for controlling the flow between contents. For example, **FlowSequence**, as an instance of **SequenceControlMode**, represents that the current item can only move forward and backward within one dimensional sequence. On the other hand, another instance **ChoiceSequence** allows trainee to choose one among multiple course items.
Considering concepts defined above, we can summarize what differences exist between CBT and IMT. As seen in Table 1, the difference between them lies in the training method and equipment so that the training task and resources can be commonly used.

<table>
<thead>
<tr>
<th>Concept</th>
<th>CBT (Computer-Based Training)</th>
<th>IMT (Immersive Training)</th>
<th>Common? (YES/NO)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TrainPerson</td>
<td>Author, Trainee, Mentor</td>
<td>Author, Trainee, Mentor</td>
<td>YES</td>
</tr>
<tr>
<td>TrainMode</td>
<td>ComputerBasedTraining</td>
<td>ImmersiveTraining</td>
<td>NO</td>
</tr>
<tr>
<td>TrainMethod</td>
<td>Simulation</td>
<td>VirtualReality</td>
<td>NO</td>
</tr>
<tr>
<td>TrainEquipment</td>
<td>Keyboard, Mouse, Monitor</td>
<td>TrackingSystem, HMD, CyberGlove</td>
<td>NO</td>
</tr>
<tr>
<td>TrainTask</td>
<td>AssembleWindTurbine, AssembleHubbleScope</td>
<td>AssembleWindTurbine, AssembleHubbleScope</td>
<td>YES</td>
</tr>
<tr>
<td>Resource</td>
<td>CAD Models, Instructions, Images</td>
<td>CAD Models, Instructions, Images</td>
<td>YES</td>
</tr>
</tbody>
</table>

Table 1: Comparison between CBT and IMT according to training concepts

Accordingly, we need to put some rules into the training ontology in order to differentiate CBT from IMT in terms of training methods and equipment. For example, the immersive training should employ a virtual reality as a training method and have several devices as equipment, such as at least one tracking system, optionally HMD (Head Mounted Display) as a display device, digital glove as an interactive device or haptic devices. On the other hand, the computer-based training will use a simulation program as a method and require a monitor as a display and keyboard and mouse as an interactive device. These rules are expressed by SWRL language by putting quantifier and cardinality restrictions as follows. As an example, the TrainMode can be read *that there exist a TrainMode x, where x has equipment y which is an instance of*
TrainEquipment, x has method z which is an instance of TrainMethod, x has task w which is an instance of TrainTask.

\[
\begin{align*}
\text{Train(?x)} & \rightarrow \text{hasTrainMode(?x, ?y) \land TrainMode(?y) \land} \\
& \text{hasTrainPerson(?x, ?z) \land TrainPerson(?y) } \\
\text{TrainMode(?x)} & \rightarrow \text{hasTrainEquipment(?x, ?y) \land TrainEquipment(?y) \land} \\
& \text{hasTrainMethod(?x, ?z) \land TrainMethod(?z) \land} \\
& \text{hasTrainTask(?x, ?w) \land TrainTask(?w) } \\
\text{ComputerBasedTrain(?x)} & \rightarrow \text{hasEquipment(?x, ?y) \land (Keyboard \cup Mouse \cup} \\
& \text{Joystick)(?y) \land hasEquipment(?x, ?z) \land DisplayDevice(?z) } \\
& \text{\land hasTrainMethod(?x, ?w) \land Simulation(?w) } \\
\text{ImmersiveTrain(?x)} & \rightarrow \text{hasEquipment(?x, ?y) \land TrakcingDevice(?y) \land} \\
& \text{hasEquipment(?x, ?z) \land DisplayDevice(?z) \land} \\
& \text{hasTrainMethod(?x, ?w) \land VirtualReality(?w) } \\
\text{TrainTask(?x)} & \rightarrow \text{hasTrainCourse(?x, ?y) \land Course(?y) } \\
\end{align*}
\]

5.5.2. Application Specific Ontology: iTrain-AO

The application specific ontology, iTrain-AO, imports three domain ontologies: ASM-DO, SIM-DO and TRAIN-DO. Figure 10 illustrates the UML diagram with main concepts of the iTrain-AO ontology. iTrain-AO not only imports all the concepts and properties from the domain ontologies, but also defines a few additional concepts and properties. We define the three root concepts: ProductDesign classifying knowledge acquired from the product design domain, SimulationStudy for the assembly simulation domain, and TrainStudy for the training domain. For instance, the ProductDesign concept can be instantiated into WindTurbineDesign which embodies all the assemblies and parts relevant to the wind turbine design or HubbleScopeDesign embodying various types of Hubble telescope designs.
• *ProductDesign*: It is a concept representing a collection of components relevant to the specific product design.

• *SimulationStudy*: It is a concept representing a collection of *SIM-DO:Simulation* relevant to the specific kinematic simulation knowledge.

• *TrainStudy*: It is a concept representing a congregation of *TRAIN-DO:Train* relevant to the specific training knowledge.

• *Component*: It is a concept, as equivalent to *Component* concept in GDO, specified into three subclasses. One is *FixedSizeComponent* to represent components without any kinematic motion and any child components, the second is *MovableComponent* for components having at least one kinematic behavior and without any child component, and the other is *CompositeComponent* for assemblies or subassemblies with child components.

• *hasKinematicMotion*: As a property of *Train-DO:CourseItem*, it associates course items in TRAIN-DO with kinematic motions in SIM-DO and also associate components in ASM-DO with the kinematic motions. By doing so, certain course item can search and animate pertinent kinematic simulations in the training environment.

• *hasComponent*: As a property of *Train-DO:CourseItem*, it associates course items with components in ASM-DO. By doing so, certain course item can search and review pertinent CAD designs.

As an example, there is a course item to instruct how to get a spare tire in order to replace the flat tire. The course item will refer to pertinent components within a car assembly and kinematic simulations about a tire replacement. The components will be a
spare tire, wrench and a hoist shaft and kinematic simulations will be putting a wrench on a hoist shaft, turning the wrench counterclockwise, and detaching the spare tire from the car body.

![UML diagram: Concepts and relationships iTrain-AO](image)

**Figure 10: UML diagram: Concepts and relationships iTrain-AO**

As mentioned previously, our ontologies follow the employing three-tiered structure of engineering ontologies: General Domain Ontology (GDO), Domain Specific Ontology (ASM-DO, SIM-DO, & TRAIN-DO) and Application Specific Ontology (iTrain-AO). The lower level of ontologies import concepts and properties from the upper level of ontologies, sometimes extend them, and make relationships between concepts from different domains. This fact can result in easily integrating different domain ontologies into the one application specific ontology without altering the pre-existing knowledge.

Another fact is that each domain ontology keeps being connected to its domain tools or application. For example, concepts under the *ProductDesign* concept are connected to CAD tools, concepts under the *SimulationStudy* are to CAE or simulation tools, and concepts under *TrainStudy* are to training tools. Therefore, when the design parameters or kinematic motion paths are updated from CAD/CAE tools, the relevant knowledge in the iTrain-AO can also be updated and the course item can refer to up-to-
date design and simulation information. The methods to acquire knowledge from domain
tools and instantiate it into domain ontologies will be introduced in the next section.

5.6. Procedural Knowledge Ontology

Building on this previous works, for this research we have created a Procedural
Knowledge Ontology (PKO) that is intended to deal with processes, events and actions
that can occur in the training environment and manage expert inputs in raw formats. As a
result, it is used to transfer information about assembly sequence and planning as well as
exchanging instructions or thoughts between novices and experienced engineers.

As seen in Figure 11, a process represents a sequential state or event and is
composed of concepts such as Person, Action, Event, and Component. Each process also
has a reference to the PKExpertKnowledge which will provide special advice,
recommendations, or safety rules in order to give additional information or deliver critical
warnings. PKExpertKnowledge is composed of PKExpertRawData which in turn is
composed of one or more child concepts of PKMaterial such as video, audio or text.
The expert data is currently classified into four categories as follows:

- **PKCommentary**: It is a concept related to commentary of his/her thought processes as a problem is being solved or while using an off-line reporting techniques, like a video.

- **PKEngDesignStory**: It is a concept related to anecdotal description of an important engineering design experience with some interesting lessons or consequences.

- **PKLimitedInfoAndTask**: It is a concept related to how to drill down further when an expert has limited pieces of information to perform a task.

- **PKDecisionMethod**: It is a concept related to how to make a decision when an expert examines event or incidents to articulate the thought-processes, inferences, and strategies.
5.7. Summary

Our approach for knowledge representation is to explicitly build into ontologies the engineering product knowledge which is used in different design, analysis processes, and training. With layer-structured ontologies coded in a consistent, scalable ontology language, new ontologies can be built based on existing ontologies so as to improve scalability and reusability of the framework.
CHAPTER SIX

ENGINEERING KNOWLEDGE ACQUISITION

This chapter will introduce how to instantiate the three domain ontologies. First, the product design ontology will be instantiated by automatically extracting information from CAD tools with the help of a plug-in tool which we developed. Second, the training ontology can be mostly instantiated through the online community except for product design information. Last, the procedural knowledge ontology will be partially instantiated by knowledge engineers. Some of knowledge in both the training ontology and the procedural knowledge ontology will be instantiated by the knowledge transfer from other ontologies.

6.1. Population of Engineering Knowledge

Product data semantics refers to the “meaning” of product data. After concepts are built into engineering ontologies, concepts need to be instantiated with data from the actual product so that product data semantics are generated by the association of each product data to its concept level. To capture product data semantics based on the pre-defined concepts and their relations in engineering ontology, instances are created to represent the product data. Since most of the commercial CAD/CAE applications provide open API to access information generated in the application, information about the product can be obtained by calling the API. Instances in the application specific ontologies are built based on this information.

Another key method for on-going knowledge acquisition is an online community that brings together experienced engineers and trainee/novice into a social environment for exchange of knowledge and ideas. An easy to use, specialized web-based interface is
designed to enable the community members to interact in a manner that facilitates engineering discussions. For example, this web interface can support the following: a) viewing of CAD models, b) videos/audios, c) texts and pictures, d) visualization of process simulations, e) access to data from the training information system, etc. The members can be able to add videos, parts, and rules using this system. They are able to tie rules with parameters from the CAD model or the simulation/training knowledge or repository. They can review videos of process training content created and stored. This system is created to also enable a scenario where an experienced engineer is training a student and this information is captured during the training process itself.

We also design and create a repository for storing raw data from the knowledge acquisition process. This repository will include queries, responses, text interactions, audio, videos, and links to documents and models and these raw data will be linked to the training contents.

6.2. Instantiation of Engineering Knowledge

As seen in Figure 12, the instantiation of the training ontology starts with a CAD model which is provided by a product designer. Knowledge about design parts or assemblies and kinematic simulations are extracted from the CAD/CAE tools and then instantiated into the product design ontology. This step is automatically executed by a CAD plug-in tool we developed. Next, the semantic concepts and instances in the product design ontology are transferred to the training ontology through the ontological mapping technique. The step 3 and 4 can be sometimes different because we provide another way to directly extract knowledge in CAD models into the training ontology. As the fifth step, the training contents including methods, equipment and tasks is manually instantiated.
though the knowledge editor, like Protégé by knowledge engineers. Last, the instantiated training ontology is imported to the training tools and knowledge is presented. If there exist feedbacks getting from experienced engineers or trainees, the knowledge in the training ontology will be updated by accepting appropriate feedbacks or comments.

Figure 12: The procedure to instantiate the training ontology

Figure 13 is a snapshot taken from the Protégé knowledge editor and shows the result after some of concepts imported from the product design and simulation domain ontologies are automatically instantiated from a given CAD model, a wind turbine hub. The figure depicts instances with a purple-colored diamond icon and their properties. It also shows relations between a component, a kinematic motion and transformations. All the instances in the upper box represent assemblies or parts, an instance in the lower-left box represent the kinematic motions, and an instance in the lower-right box show the transformation information. The example instantiates a motor casing component in the wind turbine hub. The A2-0019 is an instance of the MovableComponent concept, is
linked to *Hub_Chuck_Motor_Casing.osg* as a geometric data file, and has *InstallMotorCasing* as an instance of *FreeMotion* concept which is a kind of the kinematic motion. Also, the location and orientation of the component is represented by *Translation* and *Rotate* concept. This assembly process to install the motor casing will be constructed as one of course items and the course item will cross-reference to the component and kinematic motion. The examples of creating and instantiating the training contents in the training domain will be presented in the Section 6.

![Diagram of component information and kinematic animation](image)

**Figure 13:** Population of instances for the simulation domain in the iTrain-AO ontology by using Protégé: A part component’s information and its kinematic animation

In the current training environment, the comments and feedbacks are acquired from members and manually populated into the training ontologies by knowledge engineers. However, an ontology-based online community application which we are currently developing can allow members to access to the specified web site, review the product design and manipulate the training contents by organizing training lessons. The
application will be worthy of acquiring the feedbacks from trainees and new knowledge from experienced engineers.

Since the engineering knowledge management unit provides an interface which enables an end-user application to communicate with the engineering knowledge repository, end-users can create or edit training information or upload resources into the knowledge and data repository. The Figure 14 displays how to create a training course item which is one of the steps instructing how to replace a camera of the Hubble telescope. The left side of the figure is user interfaces in CREEK to allow users to enter information about the replacement step, and the right side shows the actual instance data in the training ontology which is populated from CREEK. In this ways, the training ontology will be instantiated by capturing knowledge about training contents.

The procedural knowledge ontology will be manually instantiated by using a knowledge editor, called Protégé. It is here responsible for translating expert inputs to assembly procedures and instructions. It also accumulates expert inputs in a raw format.
6.3. Case Study

As a case study, we will make use of a wind turbine model and show how engineering knowledge is acquired in the perspective of the knowledge lifecycle. We suppose that the domain knowledge models in the engineering knowledge repository were already designed by using ontologies in the previous chapter. This section will show how the engineering knowledge is acquired by engineering applications including CAD tools and CREEK through the engineering knowledge management unit.
6.3.1. Populate the product design and kinematic simulation knowledge

CAD models in Solidworks or Pro/Engineer are provided by design engineers and assembly instruction notes to install the hub of a wind turbine are given by experts beforehand. That information is shown in Table 2. The product design and kinematic information are extracted from CAD tools and instantiated with concepts in ASM-DO and SIM-DO ontologies, as an example is shown in Figure 13. With the help of SolidWorks APIs and Pro/Engineer APIs, information of design parts and assemblies, such as, transformation, texture, assembly constraints and other properties, can be interpreted and captured into ontologies. If the CAD tools support the kinematic simulation, kinematic motions will be converted to animation path along with location/orientation data via time. As well as ontologies, each design part is exported to Wavefront OBJ and OSG (OpenSceneGraph) geometric format and saved into the knowledge repository.

<table>
<thead>
<tr>
<th>CAD Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>![CAD Model Image]</td>
</tr>
<tr>
<td>Process</td>
</tr>
<tr>
<td>---------</td>
</tr>
<tr>
<td>Step 1</td>
</tr>
<tr>
<td>Step 2</td>
</tr>
<tr>
<td>Step 3</td>
</tr>
<tr>
<td>Step 4</td>
</tr>
<tr>
<td>Step 5</td>
</tr>
<tr>
<td>Step 6</td>
</tr>
</tbody>
</table>

Table 2: CAD model and assembly instruction for a wind turbine’s hub

6.3.2. Populate the training knowledge and procedural knowledge

As briefly mentioned before, the train models and learning contents are currently instantiated by using the knowledge editor, Protégé. As shown in Figure 15, two training models are constructed: one is for CBT and the other is for IMT. The CBT instance has a monitor as equipment and the simulation as a method, but the IMT instance has a HMD as equipment and the virtual reality as a method. However, both CBT and IMT instances point out the same learning content which teaches how to attach the motor casing into the hub. The course item has references to text and image resources and a kinematic simulation as well as the relevant component in order to instantiate Step 5 from the assembly instruction shown in the Table 2. The figure below shows that the course item
in the training domain makes connections to components in the product design domain and kinematic motions in the simulation domain.

Figure 15: Training Models with different train methods and the same train task by instantiating CBT and IMT in Protégé

In addition to that, the knowledge can be populated into the procedural knowledge ontology. Table 3 shows how to instantiate steps 6 and 7 in the Table 2 as an example. An annotation \textit{HubLadderNote1} will be associated with the expert knowledge in Step 7.

<table>
<thead>
<tr>
<th>Instantiation of PKO:Process for Step 6:</th>
</tr>
</thead>
<tbody>
<tr>
<td>(∃x:Process6)(∃y:OrientHubBearing)(∃z:HubLadderNote1)(Process(Process6) ∧ Action(OrientHubBearing) ∧ Component(HubBearing1) ∧ Component(NegativeZ) ∧ is_from(y, HubBearing1) ∧ is_to(y, NegativeZ) ∧ has_location(y, Facing) ∧ has_action(x, y) ∧ has_nextprocess(x, Process7))</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instantiation of PKO:Process for Step 7:</th>
</tr>
</thead>
<tbody>
<tr>
<td>(∃x:Process7)(∃y:AttachHubLadder)(Component(HubLadder1) ∧ Component(MainHubBody) ∧ is_from(y, HubLadder1) ∧ is_to(y, MainHubBody) ∧ has_location(y, ContactWithoutOffset) ∧ has_action(x, y) ∧ has_expertknowledge(x, HubLadderNote1))</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instantiation of PKO:PKExpertKnowledge for Step 7:</th>
</tr>
</thead>
<tbody>
<tr>
<td>(∃x:HubLadderNote1)(PKExpertKnowledge(HubLadderNote1) ∧ PKText(Commentary-AssembleHubLadder) ∧ has_annotation(x, “It’s best to assemble this to Main Hub Body last”) ∧ has_expertrawdata(x, Commentary-AssembleHubLadder))</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Logic-Based Explanation for PKO:Process for Step 6:</th>
</tr>
</thead>
<tbody>
<tr>
<td>There exist a Process6 x and an OrientHubBearing y, where Process6 is an instance of Process and OrientHubBearing is an instance of Action, HubBearing1 is a component, NegativeZ is a component, y from...</td>
</tr>
</tbody>
</table>
HubBearing1 to NegativeZ, y has location Facing, x has action, x has next process Process7, and has expert knowledge HubLadderNote1”.

Table 3: Examples of instantiation of PKO:Process and PKO:PKExpertKnowledge

The training ontology populates the training courses according to the instruction note provided by an expert. Each course item will also populate appropriate resources, like texts, images and videos. Additionally, it adds two instances as a type of the training methods: CBT-Turbine having equipment as a keyboard, a mouse and a special program, and IMT-Turbine with a tracking system, HMD, digital glove and a special program.

When instantiated, ontologies are saved into the knowledge base and other CAD models and expert notes are located at the repository or database.

6.4. Summary

When domain ontologies are designed, the iTrain-AO ontology is first instantiated for the purpose of populating the knowledge of a product design domain, the assembly simulation domain, and the assembly training domain. Once an assembly or product design, kinematic simulations and training models have been designed by CAD/CAE and course editor tools, data and knowledge would be captured into different domain ontologies and be finally transferred to an ontology, called iTrain-AO. In addition, the semantic online community application, called CREEK, provides an interface to allow users to add or modify the training ontology as well as to populate knowledge about assembly procedures and experts’ feedbacks.
CHAPTER SEVEN

ENGINEERING KNOWLEDGE INTEGRATION

The primary objective of the ontology is to capture consensual knowledge of a specific domain in a generic and formal way and allow this knowledge to be reused and shared across applications. However, because ontologies are widely used in domains as well as on the semantic web, some ontologies might have concepts in common. In order to come up with the sharing and interoperability of knowledge regardless of heterogeneity of ontologies in different domains, a technique to reconcile similarities and differences between ontologies is required. This technique, called ontological mapping, will find relationships and correspondences between semantic entities, such as concepts, properties, or instances, among multiple different ontologies and helps transfer knowledge from one domain to the other domain.

The ontological mapping will expedite the knowledge integration and the engineering applications will retrieve necessary information from the knowledge repository by executing reasoning and querying. This section will describes how to integrate the three different domain ontologies and how to communicate between external applications and engineering knowledge management system.

7.1. A Methodology for Ontology Mappings

Wache et al. [64] attribute interoperability problems within the distributed community to structural (schematic) and semantic (data) heterogeneity and consider the usage of ontologies as a way to make explicit the meaning of the terms in a domain to solve the semantic heterogeneity between different software systems. They also
categorize three types of approaches for the explicit description of the information source semantics [Figure 16].

![Figure 16: Three ontology-based Integration approaches [64]](image)

The global ontology approach uses a single shared ontology to provide all the relevant information sources. This approach is suitable for environments where the information sources represent the same viewpoint on a domain. This approach is unrealistic because every domain might not follow the global ontology and the changes in one information source should be propagated to the global ontology and the mappings between the information sources.

In the multiple ontology approach, each local ontology is independently built by its own ontology. This avoids influencing other information sources when adding a new information source. It also makes mapping between local ontologies easier. However, the high possibility of semantic heterogeneity and a lack of common vocabularies can make the mapping between local ontologies difficult.

The last hybrid ontology approach overcomes the disadvantages of the first two approaches. The Shared global ontology defines common vocabularies for the local ontologies. Therefore, the local ontologies not only shared common terminologies, but also extend the concepts in the domain ontology. This feature can make it easier to translate information between local ontologies and discover correspondences between them.
7.2. Our Approach for Knowledge Integration

Our proposed method focuses on product data in the engineering domain. As a concrete starting point, our methods seek to facilitate automated mapping between a Product Design Ontology (PDO), which is the source ontology, and an Analysis & Simulation Ontology (ASO), which is the target ontology.

7.2.1. Definition of a shared ontology and extensions:

We define a shared ontology representing terminologies and constraints which different engineers in the general domain agree to use, and then allow developers to create their own specialized vocabularies by inheriting the shared ontology [Figure 17] and then adding concepts to refine and extend the general ontology. This is based on the hybrid approach discussed in section 5.1 and enables the synchronization of concepts, properties and instances between ontologies dealing with product design models.

![Diagram of ontologies]

Figure 17: Our hybrid approach for engineering product data ontology

In our case, the General Domain Ontology (GDO) is defined to provide common terminologies, such as Component or Geometry, which are shared by both the engineering product design domain and the engineering assembly simulation domain.
Next, the *Domain Specific Ontologies* (such as FBM-DO, ASM-DO or SIM-DO) which form the second level of hierarchies add new domain-specific vocabularies unique to that particular domain. For example, a 3D parametric feature-based geometric modeling tool and an ergonomic design analysis tool are relevant in different domains, but both are important for the overall engineering product development activity and are likely to have both common and separate vocabularies. Therefore, it would be appropriate to use the common concepts from the GDO and then create the domain specific ontologies for the additional concepts unique to each domain. Going one level further down, the third level is the *Application Specific Ontology* (such as PROE-AO and VADE-AO) and captures semantics unique to each application within a domain. For example, Pro/E and CATIA are two different applications in the 3D parametric feature based geometric modeling domain.

### 7.2.2. Definition of Rule-based Mapping

We then define *domain-specific rules* to express the special relationships between domain concepts for the purpose of automatic and heuristic translation between dissimilar engineering domains. Some mapping rules are found based on properties of OWL syntaxes and part-of or attribute-of relationships between classes. In addition, OWL has the advantage of permitting users to write restrictions and rules to specify classes with the aid of Jena Rules. Two constraints chosen from each ontology are passed to a reasoner, which is Pellet [65] in our case, and can be checked to determine whether or not they represent the same concept.
7.2.3. Creation of Bridge Ontology

An important requirement of a mapping ontology is to create a Bridge Ontology to help communication between two differently defined ontologies without promoting a common understanding among knowledge workers. We have designed a Bridge Ontology to store information acquired from mapping processes between two different ontologies, such as mapped concepts, properties, and necessary conditions or relations. A class or property in the Bridge Ontology defines how an entity in one ontology is related to an entity in another. It can be not only extended in order to record other mapping information within the same general domain but is also reusable for future mapping.

7.2.4. Creation of Graphical Application to facilitate mapping

An additional piece of work is a graphical application that will allow multiple target applications (or humans) to have access to the heterogeneous sources of information through the Bridge Ontology. The application we have developed will allow us to import ontologies, run mapping strategies, create Bridge Ontology, and translate instances from source to target ontology. The application uses Jena2 API to manage ontology model data formatted using the Ontology Web Language (OWL) created by the ontology editor tool Protégé. It also uses Jena Rules and Pellet for the reasoning process.

7.3. Mapping Strategies

This thesis extends the mapping definitions and rules which have been presented in previous papers [66][67][68] of ours. Table 4 enumerates some definitions used in this thesis and Table 5 lists a sample of mapping rules to find concept similarities. These mapping rules are explained in detail in the previous papers and are provided here for a quick reference.
<table>
<thead>
<tr>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mutual Information</td>
<td>Mutual information refers to the information shared by two instances.</td>
</tr>
<tr>
<td>Concept (Property) Similarity</td>
<td>Concept similarity means that one concept is similar to the other concept in terms of some characteristics or functionalities.</td>
</tr>
<tr>
<td>Matching Concept (Property)</td>
<td>Matching concept means that one concept is identical to the other concept in terms of all characteristics or functionalities.</td>
</tr>
<tr>
<td>Composition Path</td>
<td>A composition path is the path that only consists of composition relations (part_of). If all the nodes in one composition path have attribute similarity to all the nodes in the other composition path, then the two paths are defined as similar composition paths.</td>
</tr>
<tr>
<td>Attribute Similarity</td>
<td>Attribute relationship (attribute_of) refers to the relation between an object and its attributes. Attribute Similarity is used to measure the similarity by calculating whether there are any common attributes that two concepts share.</td>
</tr>
<tr>
<td>Composition Similarity</td>
<td>It is used to measure the similarity by calculating how similar two composition paths are.</td>
</tr>
<tr>
<td>Instance Similarity</td>
<td>It is to consider the similarity according to the is_a relationship. Refer to Rule-VI at Table 2.</td>
</tr>
</tbody>
</table>

**Table 4: Sample Definitions**

<table>
<thead>
<tr>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule-I</td>
<td>If two concepts are equivalent, then they are matching concepts. i.e) owl:samePropertyAs / owl:sameClassAs / owl:sameIndividualAs</td>
</tr>
<tr>
<td>Rule-II</td>
<td>If two concepts share the equivalent instance, then the two concepts are two matching concepts. Different instances share the same id, or the two URIs eventually redirects to the same URI.</td>
</tr>
<tr>
<td>Rule-III</td>
<td>During data conversion, if the attribute of an instance of a BDE(Basic Design Entity), such as feature or part, is converted to be the attribute of an instance of another BDE, then both Basic Design Entities share the same attribute.</td>
</tr>
<tr>
<td>Rule-IV</td>
<td>If two concepts have the same constraints, then they are matching concepts.</td>
</tr>
<tr>
<td>Rule-V</td>
<td>For a given source concept A, if a target concept A' has attribute</td>
</tr>
</tbody>
</table>
similarity $\text{Sim}_1(A, A')$, and if a Concept C' has Child A', then Concept C' and A have the same attribute similarity $\text{Sim}_1(A, C') = \text{Sim}_1(A, A')$

| Rule-VI | For two matching concepts A and B, if each of them belongs to the two different hierarchies, then all the concepts $A_i$ and $B_i$ that are sub-concepts of A and B will have the same attribute similarity and composition similarity: $\text{Sim}_1(A_i, B_i) = \text{Sim}_1(A, B)$ and $\text{Sim}_2(A_i, B_i) = \text{Sim}_2(A, B)$ |
| Rule-VII | If two geometry attributes fall under the same immediate geometry attribute category, then the geometry similarity of two geometry attributes is proportional to the depth of the geometry category from the top level. |
| Rule-VIII | If a concept can have different geometry types as its geometry attribute, the common geometry category at the lowest level in the hierarchy is used to calculate geometry attribute similarity. |
| Rule-XI | If both applications are in the realm of 3-D feature-based product design, the designCreationBehavior can be used as common behavior attribute. |
| Rule-X | If both applications share a common definition in functionalBehavior, then that definition may be used as behavior attribute. |

| Table 5: Sample Rules for mapping strategies |

### 7.3.1. Mapping Strategy

The principal strategies used to search for matching concepts or properties between two different engineering domain ontologies are described in detail below.

- **Equivalency**: It checks some pre-defined tags in OWL such as `<owl:sameClassAs>` and determines if two concepts (Properties) are defined as equivalent: Rule-I, Rule-II, and Rule-III.

- **Constraint Similarity**: It compares constraints of concepts written in DL (Description Logic) or FOL (First Order Logic) and decides how similar two
given concepts are. Jena Rules and an external reasoner such as Pellet will allow it to find two concepts having same constraints: Rule-IV.

- **Lexical Matching**: It exploits a lexical matching algorithm and checks how similar the names of two concepts are. This logic only shows the probability where two concepts are similar. Even though two names look similar, the semantic meaning might be different.

- **Attribute Similarity**: The geometric, functional, or behavioral attributes of classes can tell how similar two classes are: Rule-VII, Rule-VIII, Rule-XI, and Rule-X.

- **Composition & Inheritance Similarity**: Two composition paths over part-of relations and transitivity of composition relations over inheritance relations help finding similar concepts: Rule-V, and Rule-VI.

- **Property Mapping**: Most object and data properties attached to a concept in the source ontology can be mapped to properties of a matching concept in the target ontology. If a property does not exist in the target ontology, a new property will be created.

### 7.3.2. Overall Mapping Procedure

This section describes an overall process to find matching concepts and properties between two different ontologies. This section also illustrates what actions are required before and after the ontological mapping [Figure 18]. First of all, two ontologies are created by domain engineers in advance and then imported. Next, concepts are investigated to find equivalency and similarities of classes and properties according to definitions and rules. This process also includes a procedure to compute attribute similarity, composition similarity, and instance similarity [Table 4]. Lastly, the Bridge
Ontology is created and used for translating instances of a source ontology to those of a target ontology.

Figure 18: Overall mapping procedure between ontologies

The detailed procedure to find matching concepts is as follows.

1. Start with creating a composition graph and inheritance hierarchies: Create composition graph for composition relations and class hierarchies for inheritance relations.

2. Start a matching process based on definitions: Find explicit equivalences based on definitions and define them as matching concepts.

3. Find Constraint Matching: Check if there are classes whose constraints are matched to those of other classes with the help of the reasoner.
4. Check Lexical Similarity: Check the similarities based on the naming of classes and properties.

5. Calculate Attribute Similarity: Among two composition graphs from two different ontologies, for each given source concept, calculate attribute similarity between the source concept and other concepts in target ontology and find all the nodes that have similar attribute.

6. Find similar Composition Paths: Since the attribute is transitive over the composition relation, all the classes that has owner relations over the target concept will have at least the same attribute similarity.

7. Calculate Composition Similarity: Calculate composition similarity and choose the one with the highest composition similarity and attribute similarity as matching concept.

8. Calculate Attribute Similarity and Composition Similarity based on inheritance relation: When composition similarity and attribute similarity can be transitive over inheritance relation, all their sub-concepts are set to have the same attribute similarity and composition similarity.

9. Go to 5) until all the candidate concepts are calculated.

10. Human intervention, in case that user need manual operations of concept or property mappings.

11. Create the Bridge Ontology: Store information of mapped concepts and properties to the Bridge Ontology, populate knowledge (instances) from the source ontology and update knowledge in the target ontology. A detailed description about the Bridge Ontology will be introduced in the Section 5.4.
7.4. Bridge Ontology

Bridge Ontology is an ontology which identifies mapped concepts and properties between the source ontology and target ontology and represents their mapping relations. Once the Bridge Ontology is built, it can be reused for a new integration process without exploring matching concepts again and can be extended to include additional mapping information for other ontologies in the CAD/CAE/CAM domains. If one concept is not directly mapped to another concept, an extensional condition will be required to make mathematical or logical rules between two concepts. These rules can be represented by SWRL language.

![Figure 19: Diagram for Bridge Ontology](image-url)
The bridge ontology is designed such that whenever the mapping strategies are executed, pieces of useful information such as the source ontology, the target ontology, instances of mapped concepts, and mapping conditions, that have been acquired are stored.

The diagram above [Figure 19] shows the key class hierarchies and these are described below:

- **BDO:Mapping**: It logs information of both the source and target ontologies, such as, URI, name, and version and keeps mapping information from overlapping.
- **BDO:ConceptMatch**: It stores the matching concepts
- **BDO:AttributeMatch**: It stores the matching properties.
- **BDO:Condition**: It stores additional information to map two concepts or properties, such as, simple unit conversion, mathematical relations, or logical relations.
- **BDO:Rule**: It stores logical relations using FOL (First Order Logic) and DL (Description Logic). This will be used in the future because Jena2 does not support them currently.

### 7.5. Mapping between Multiple Domains

Our previous section introduced mapping methods and strategies between different engineering ontologies and presented a Bridge Ontology which identifies mapped concepts and properties between two specified ontologies and represents their mapping relations. The ontological mapping plays an important role in knowledge management. This is because when mapping relations of concepts and properties between
different ontologies are established, the knowledge transfer between domains will be accomplished with less effort.

Figure 20 shows how we have built upon that work and shows a representation of these multiple ontologies and matching concepts between product design ontology, training ontology and procedural knowledge ontology. One of the notable facts is the mapping of $PK_{\text{ExpertKnowledge}}$ concept into $\text{ExpertComment}$ in CAD tools and $\text{FeedbackAndComment}$ in the training environment. Because of that fact, the expert knowledge can be populated into the procedural knowledge ontology and then new knowledge can be distributed into annotations in an assembly model and instructions in a training content.

The figure below also represents one fact that a procedural knowledge ontology will be built based on a product design ontology and a training ontology. That is, when the knowledge in the product design ontology is acquired from CAD tools and the knowledge in the training ontology is capture from CREEK or Training tools, some of the knowledge will be conveyed to the procedural knowledge ontology because some concepts are already known as the matched concepts after the ontological mapping.
7.6. OMA: Ontology Mapping Application

Based on the shared ontology approach and supported by the Bridge Ontology design, we have designed and created an Ontology Mapping Application, OMA. It is a Java-based application that allows the user to import two ontologies as input, load additional ontologies such as General Domain Ontology automatically, calculate concept and property similarities, and display the results on the panel [Figure 21]. The contents inside of the red circle display inheritance hierarchy of the source ontology including the general domain ontologies. The contents of the blue circle show the result of concepts mapping. Each mapping strategies has a different mapping score because a concept might be mapped to multiple concepts by different mapping strategies. For example, the score of Equivalency is 1.0 and the score of Lexical Matching is 0.8 at most so that a pair of mapped concept by Equivalency is automatically chosen.
Figure 21: Product data Ontology Mapping Application (OMA)

The key functionality supported by OMA is as follows:

- Obtain and display inheritance hierarchies for source and target ontologies
- Find composition paths according to part-of and attribute-of relationships and draw them using Grappa toolkit [69]
- Search matching and similar concepts based on mapping rules and display similarity scores
- Highlight mapped concepts with a different color
- Allow the user to manually modify or add mapped concepts
- Save the mapping information into an OWL file
- Create instances for the target ontology in accordance with the mapping information
The façade of OMA has three panels.

- Panel 1 displays inheritance paths for source and target ontologies.
- Panel 2 displays the composition paths for source and target ontologies.
- Panel 3 shows the results after the mapping strategies have been executed.

There are three principal components driving OMA [Figure 22].

1. **Domain Ontology Manager**: The Domain Ontology Manager plays a key role in making ontological models ready for mapping. It can load ontologies, reads them
along with internally imported ontologies, and creates inheritance and composition graphs with the help of Jena2 APIs.

2. **Mapping Ontology Manager**: The Mapping Ontology Manager is responsible for executing the mapping processes [Figure 18] described previously and for displaying the mapping results.

3. **Bridge Ontology Manager**: The Bridge Ontology Manager populates the Bridge Ontology based on the mapping results from step 2. If the Bridge Ontology already exists, the mapping information will be merged with that. The concepts, properties, and instances of the Bridge Ontology can be viewed through Protégé tool. It also creates instances for the target ontology from those for the source ontology.

Pseudo code shown in the table below has been captured from the Mapping Ontology Manager and the Bridge Ontology Manager respectively to explain how to manage the core mapping strategies and to create classes for the Bridge Ontology.

```java
public Collection<MappedConcepts> executeConceptMapping() {
    ...
    while (true) {
        Collection<MappedConcepts> arrConcepts = null;
        switch (nMethod) {
            case 1:  // Find lexical similarities
                operator = new LexicalSimilarityOperator();
                break;
            case 2:  // Find two classes having same URI
                operator = new UriMatchingOperator();
                break;
            case 3:  // Find equivalent classes
                operator = new EquivalentMatchingOperator();
                break;
            case 4:  // Find constraint matching
```
    operator = new ConstraintMatchingOperator();
    break;

    case 5:    // Find attribute similarity
        operator = new AttributeSimilarityOperator();
        break;

    case 6:    // Find composition similarity
        operator = new CompositionSimilarityOperator();
        break;

    case 7:    // Find Inheritance similarity
        operator = new InheritanceSimilarityOperator();
        break;
    default:
        return mappedConcepts;
    }

    // Find matching concepts
    arrConcepts = operator.findMappedConcepts
                      (_mgrSourceModel, _mgrTargetModel);

    // Save matching concepts
    mappedConcepts.addAll(arrConcepts);
    ...
    return mappedConcepts;
}

public void createBridgeOntology() {
    ...
    // Mapping class, om : Ontology Model, ns = Namespace
    OntClass mapping = om.createClass(ns + "Mapping");

    // Ontology class,
    OntClass ontology = om.createClass(ns + "Ontology");
    ObjectProperty hasOntology =
        om.createObjectProperty(ns + "has_ontology");
    hasOntology.addDomain(mapping);
    hasOntology.addRange(ontology);
Table 6: Pseudo codes – Mapping procedure and creation of a Bridge Ontology

7.7. Case Study

We illustrate the mapping process by using OMA with two existing application-specific ontologies that have already been created in different domains- one for a product design domain and another for an assembly simulation domain. The former is called PROE-AO representing a product model in Pro/Engineer CAD system and the latter is VADE-AO for analyzing assembly planning in a virtual environment. Concepts shared between two different domains are defined in General Domain Ontologies. We regard the
PROE-AO as the source ontology and the VADE-AO as the target ontology in the mapping process.

Composition hierarchies according to part_of relations (properties) and inheritance hierarchies based on is-a relations are drawn for PROE-AO [Figure 23] and VADE-AO [Figure 24]. Within composition hierarchies, nodes inside circles represent classes and edges on the arcs label properties. After running mapping strategies described in the mapping process [Figure 18], a mapping result is obtained as shown in Table 7. The nodes with a red color in hierarchy graphs mean that these classes are mapped by mapping strategies.

Component class is defined in the general domain ontology, Assembly and Component_Constraint classes are defined in the domain-specific ontologies and these classes are shared in the application-specific ontologies so that they would be equivalent. By calculating lexical similarity, Part and SubAssembly classes are regarded as similar concepts, even though these classes are independently defined in different application ontologies. Mate class is inherited from Component_Constraint class in PROE-AO and Contact class is also a subclass of Component_Constraint in VADE-AO. Both classes, which are not shown on the graphs, are determined to be similar because both have the same assembly constraints such that they each reference two plane surfaces as geometry attributes and have translations in x and y axes and rotations in a z direction as behavior attributes. A reasoner tool is able to find the attribute similarity by comparing constraints of both classes. The subclasses of Geo_Attribute such as Point, Curve, and Surface in PROE-AO are mapped to the subclasses of Feature_Element class in VADE_AO by the attribute similarity. Base on the fact, Geo_Attribute can be mapped to Feature_Element
according to an inheritance similarity. Last, in order to calculate the composition similarity, two composition paths are captured from the composition hierarchies and are compared: Assembly – Component – Constraint_Sequence – Component_Constrain – Geo_Attribute from PROE-AO and Assembly – Component – Component_Constraint – Feature_Element from VADE-AO. As seen in graphs below, since most nodes are already mapped, we can say that two paths have a composition similarity. Properties in the source can be directly mapped to properties of the mapped concept in the target. For example, has_assembly_feature property in PROE-AO can be mapped to has_assemblyref_item or has_componentref_item in VADE-AO.

Figure 23: Inheritance & composition hierarchies in the source ontology (PROE-AO)
Table 7: Matching concepts between PROE-AO and VADE-AO

Table 8 below is a snippet of the Bridge Ontology file and represents how to store a matching concept of Assembly class. The first and second parts create has_ontology and has_concept properties and the last piece populates Assembly class in PROE-AO and Assembly in VADE-AO as a matching concept.

```xml
<rdf:RDF>
  // ① Mapping – has_ontology – Ontology triple
  <owl:ObjectProperty rdf:about="/BDO.owl#has_ontology">
    <rdfs:range rdf:resource="/BDO.owl#Ontology"/>
  </owl:ObjectProperty>
</rdf:RDF>
```
Our work so far has been focused on the development of the methodology, algorithms, and application. We have tested the mapping strategies in a limited manner on the two ontologies we have developed so far. We plan to test the mapping application next year on ontologies developed by others and available in this domain. We are also designing an ontology for an ergonomic analysis application and plan to do some tests in-house using this application.

7.8 Summary

This work provides ontology mapping methods that enable us to translate semantic information between domain-specific ontologies. The shared ontology approach, the rule-based mapping strategies, and the application OMA help us automatically interpret knowledge representation and diminish the barrier to interoperability. The Bridge Ontology can be considered as another knowledge representation for ontological
mapping and improve the extensibility and reusability for other mappings in the same domain. Our current application is tested between a product design and an assembly simulation area and is being expanded to integrate with other CAD/CAE fields.
CHAPTER EIGHT
ENGINEERING KNOWLEDGE RETRIEVAL AND PRESENTATION

The knowledge will be retrieved and presented through the online community like CREEK, and the training tools like CBT and IMT. This chapter will show how knowledge will be shown to users through the applications including CREEK and CBT and IMT applications.

Our approach is to create a digital training ensemble, coined iT rain, by clearly unifying common elements and separating disparate elements in a cohesive way so that different kinds for training tools, such as CBT, IMT, Online Community, Web-based learning tools, and other tools, are linked together, as seen in the Figure 3. To achieve this goal, we need to identify design requirements and an overall architecture to exchange and share data and information. Our overall approach is to take advantage of ontology representations and methods in order to integrate these tools for assembly simulations and training.

The work here introduces an approach called iT rain (integrated Training) to involve CBT and IMT tools in the shared training context implemented by ontologies in the product design, assembly simulation and training domains. To build the integrated training environment, we are first going to design a training ontology by identifying terminologies with respect to the training environment. The ontologies represent semantic knowledge about the models to be used in the training exercises, paths/sequences of the assembly process, and structures/flows for the training. Next, some concepts and properties will be semi-automatically transferred from the existing product design and
simulation ontologies to the training ontology and other training knowledge will be manually populated by the knowledge engineers. After that, the ontologies are shared and manipulated by the training tools. Thus, our approach is aimed at capturing product design, simulation and training knowledge and maintaining modifications and additions to this knowledge.

In addition to CBT and IMT, training applications like a web-based training tool will be incorporated with SCORM, a Sharable Content Object Reference Model, so that all the events and actions during training processes can be recorded, tracked and evaluated by learning management systems. Similarly, a training organization tool, such as, Reload [70], will be involved in the iTrain environment to help users organize the training sequences and resources.

8.1. Our Approach for Knowledge Representation

Our ontology-based approach focuses on creating a knowledge lifecycle: knowledge modeling, acquisition, knowledge retrieval and presentation, and knowledge maintenance. Figure 25 describes the knowledge lifecycle associated with the engineering tools and training tools. The ontologies are used for representing domains by modeling concepts and relations and capturing knowledge from the engineering tools. After populating domain knowledge from engineering tools or applications with the help of knowledge engineers, ontologies are stored into the knowledge repository, support for searching engineering knowledge, and are maintained by the addition of new knowledge, modification or removal. This can keep up with the changing environment where the engineering design, simulation and training information keeps being updated by means of new requirement and users' feedback after training sessions. The engineering knowledge,
especially training knowledge, will be presented through a variety of training tools. The following describes roles in the context of the knowledge lifecycle:

![Knowledge lifecycle in product design, assembly simulation and training domain and knowledge sharing between various engineering applications](image)

Figure 25: Knowledge lifecycle in product design, assembly simulation and training domain and knowledge sharing between various engineering applications

The knowledge repository containing ontologies and resources will be locally and remotely accessible in order that knowledge as well as digital models can be added, updated or removed by engineering tools or training applications. Also, it will be independent of any specific tools and will be shared among them so that one modification from one tool could be propagated to other tools or applications. Additionally, this architecture can increase the reusability of the ontology models. When an ontology model closely meets the requirements of one application, its usability can be increased but reusability can be decreased. For instance, when ontologies for the CBT and IMT environments are developed separately, the individual ontology is certainly well conformed to each environment but it is difficult for one ontology to be reused for the different environment. Therefore, we will design one ontology for the iTTrain environment.
by achieving agreed conceptualizations among different styles of presentation methods. It can be also shared between applications, and maintained by them.

Traditionally computer-based training tools have been mostly created by using web pages, animations, and audio/movie files. Some simulation applications now provide an interface for mouse and keyboard interactions so as to let users manipulate digital models on the screen with a slightly more realistic sense. On the other hand, immersive training applications employ expensive VR devices, such as, tracking system, gloves, and HMD (Head-Mounted Displays) to provide immersed users better stereoscopic and realistic experiences. The immersive training applications have been successfully adapted to automobile assembly, aircraft maintenance, and diverse industrial areas. Even though CBT and IMT implement different types of user interfaces and give users different levels of realistic experience, the two are designed for the same basic purpose and have many things in common, such as, digital models, assembly planning and sequences, and feedbacks. These commonalities need to be identified and exploited, so that both CBT and IMT can be run with shared data and information in spite of differences in representations and interaction mechanisms. Thus, well-coordinated methods which can provide training contents and materials to both a computer-based environment and an immersive environment are desired.

Additionally, it is desirable that a learning content which is created for one course be reusable in other courses, if necessary. For example, once a certain lesson is created to practice gripping a wrench and removing a bolt for a tire replacement course, it should be possible to embed this in another course like disassembling a bike. iTrain should also
have a service to grant users' access, to keep track of their activities during the training process, to record completion of tasks, and to evaluate each trainee.

We can summarize the requirements that the iTrain system should meet as follows:

1. **Connectivity of digital model**: The assembly simulations are based on information stored in the CAD model of the assembly. This model should be consistent across the applications and also be updated automatically when the model is updated in the CAD system.

2. **Ability to share information**: The individual simulation and training applications can share data and information on the run.

3. **Reusability**: Data or contents can be reused irrespective of applications.

4. **Maintainability**: Data and contents can be updated easily and all the events and actions can be recorded.

**8.2. Detail Architecture of iTrain**

The architecture of iTrain seeks to establish a sharable knowledge repository for CBT and IMT and creates an environment where a trainee can perform individual and sequenced lessons for a superior training experience in assembly simulations. A trainee would first get instructions in a CBT, do some simple practice assembly exercises with a mouse and keyboard to manipulate the model, and then switch to IMT for a more complex and realistic training exercise.

Figure 26 shows a schematic layout of the iTrain environment. This architecture faithfully follows the MVC (Model-View-Controller) design pattern. The model represents the knowledge repository, views are responsible for providing user interfaces, and controllers help to access the knowledge repository and retrieving necessary
information. By isolating the view or user interfaces from the model or data, the user interface can provide various kinds of views with the same data. Thus, even though both CBT and IMT applications have different level of interaction and sense of presence in the environments, users can get trained with the same model or data.

Employing ontologies enables the knowledge repository to explicitly express constraints, rules and axioms on behalf of controllers in Figure 26. This leads to the result that the responsibility of the controllers is less and the responsibility of the model is greater. Therefore, this section will present how to digitize common and disparate data and knowledge into the training ontology, share them between CBT and IMT applications, and maintain them through the knowledge lifecycle.

Figure 27 shows how ontologies are used in the training environment. We populate the knowledge from CAD tools to the pre-developed product design ontologies, for example, from the Pro/E tool to PROE-AO ontology. Next, the training ontology will be designed by including concepts and properties from assembly modeling, simulation modeling and training domain. A key ontology is an assembly training ontology to capture knowledge about organizing training resources, training contents and training
sequence as well as some knowledge about product design and simulation. Third, the knowledge will be transferred from the product design ontology to the training ontology. Last, training tools, such as, CBT, IMT or other training tool will make use of the ontology and present knowledge in different ways.

![Figure 27: Knowledge transfer between product design environment and training environment and key ontologies in the iTrain Architecture](image)

8.3. Knowledge Retrieval

8.3.1. Use of iTrain-AO ontology

The training ontology can now be imported and used by any training tools. Like a controller seen in MVC model [Figure 26], we provide interfaces or functions which allow the user interfaces to query data and information from the ontology models. As seen in Figure 28, the controller in the middle is responsible for reading the knowledge model, composing the data structure, searching data requested by the user interfaces, and updating the knowledge model. The data structure holds instances and their properties of all the concepts in the training ontology in order for the user interface to fast access to data. It is refreshed whenever the ontology model is updated. Table 9 displays an example of programming interfaces to update the next course item of the current course.
item. The code has comments to explain the each step. The function uses Protégé API, such as `createJenaOWLModelFromURI()`, which enables to access not only to concepts as `OWLNamedClass`, properties as `OWLDatatypeProperty` and `OWLObjectProperty`, and instances as `OWLIndividual`, but also to restrictions and rules inside the ontology model.

Figure 28: Manipulation between the user interfaces and the knowledge models

```java
public void setNextCourseItem(String strCrsItemName, String strNextCrsItemNames) {
    // 1. Open an ontology Model
    //----------------------------------------
    JenaOWLModel owlModel =
        ProtegeOWL.createJenaOWLModelFromURI("http://.../iTrain-AO.owl");

    // 2. Read Knowledge: Read through all the instances of TRAIN-DO:CourseItem
    //----------------------------------------------------------------------------------
    OWLNamedClass clsCrsItem =
        owlModel.getOWLNamedClass("TRAIN-DO:CourseItem");
    Collection<OWLIndividual> objCrsItems =
        (Collection<OWLIndividual>) clsCrsItem.getInstances(true);
    if (objCrsItems != null) {
        Iterator<OWLIndividual> it = objCrsItems.iterator();
        while (it.hasNext()) {
            ...
        }
    }
}
// 3. Search Knowledge: Find an instance with the identical name
//----------------------------------------------------------------------------------------
OWLDatatypeProperty propCrsItemData =
    owlModel.getOWLDatatypeProperty("TRAIN-DO:Name");
String name = (String)objCrsItem.getPropertyValue(propCrsItemData);
if (strCrsItemName == name) {
    OWLIndividual objNextCrsItem =
        owlModel.getOWLIndividual(strNextCrsItemNames);

    // 4. Update Knowledge: Set the next course item as a given course item.
    //-------------------------------------------------------------------------------
    OWLObjectProperty propCourseItemObject =
        owlModel.getOWLObjectProperty("TRAIN-DO:hasNextCourseItem");
    objCrsItem.setPropertyValue(propCourseItemObject, objNextCrsItem);
}

// 5. Write Knowledge: Save the ontology into the disk
//-------------------------------------------------------------------------------
owlModel.save(new File("http://.../iTrain-AO.owl").toURI(),
          FileUtils.langXMLAbbrev, errors);

Table 9: Programming example to read, search, and update the knowledge by modifying the next course item of the given course item

8.3.2. Reasoning and Querying

As powerful capabilities of the ontology, reasoning and querying will be supported by using a rule language, SWRL/SQWRL. The rules can infer concept relationships or axioms which are not defined explicitly. As an example, Table 10 shows a rule defined in the procedural ontology to confirm the status of a process by checking
status of components of the process and it can be performed by a CBT or IMT tool. The querying and reasoning over the product design ontology were in detail presented in the paper [71].

\[
\text{SWRL(SR1): } \text{Process}(?x) \land \text{Component}(?y) \land \\
\text{has\_component}(?x,?y) \land \text{has\_status}(?y,\text{"Assembled"}) \\
\rightarrow \text{has\_status}(?x,\text{"Complete"})
\]

Explanation: If Process x has Component y and y has status ‘Assembled’, then Process x has status ‘Complete’.

Table 10: Example of the query and inference in the procedural ontology

Figure 29 depicts that the engineering knowledge repository populates product design knowledge, assembly processes, and expert knowledge into ontologies and it can respond to queries or reasoning which is requested by an online community or training tools. As an example of assembling the hub of a wind turbine, the online community can query how to assemble a hub ladder, and update the product design information, and the training tool can query what previous processes are required prior to the hub ladder. Also, since the online community and training environment share the same knowledge base, modification in one domain can influence on the other domain accordingly.
8.4. Knowledge Presentation with Training Applications

8.4.1. CBT Application

We developed a CBT application, called VCAT (Visualization and Computer-based Assembly Training), to help users manipulate 3D geometric models, view the kinematic simulation and get familiarized with the assembly procedures by watching instructions organized with texts, images, and videos.

As seen in Figure 30, this application is written in MFC/C++ and is based on OpenSceneGraph (OSG) libraries. The OpenSceneGraph, as a high performance 3D graphics toolkit, can be used in fields such as visual simulation, games, virtual reality, scientific visualization and modeling [72]. The OWL importer on the bottom is responsible to communicate with the knowledge repository to load ontology models containing assembly hierarchies, kinematic simulations and training materials through Protégé API. It also can load 3D geometric models and raw materials. When knowledge and resources are loaded, viewers on the user interface will present its relevant information. The 3D model viewer will make the assembly hierarchies, display 3D geometric models and their information, and allow the manipulation of 3D models with a
keyboard and a mouse. The kinematic simulation viewer presents animations and the training task viewer instructs the training tasks. The application also provide an interface to edit properties and relations of product design, simulation and training contents and save back into the knowledge repository.

![Software structure and libraries of VCAT](image)

**Figure 30: Software structure and libraries of VCAT**

When the ontology models are imported, the CBT application will search training modes which have a training method as an instance of *Train-DO:Simulation* and have equipment like mouse or keyboard. Once the proper training mode is found, the application will start to load geometric models, kinematic simulations, and equipment and relevant tasks. These data are displayed on the viewers.

### 8.4.2. IMT Application

Our training environment involves two pre-developed IMT applications; one of them was developed at WSU to enable assembly actions with haptic-enabled virtual tools inside a native CAD tool, CATIA V5. The other application, called VRToolsPro, was created by IES [73] and allows users to maneuver digital models with virtual hands or tools like screw drivers or wrenches. With the help of VR hardware, user can interact
with models inside a virtual environment. In the similar way to the CBT application, it will import ontologies, geometric models, and other raw resources. However, it will only load training modes with have a training method as an instance of `Train-DO:VirtualReality` and have equipment like tracking systems or cyberglove. Even though the training method and equipment are different from the CBT application, the same training tasks are used.

**8.5. Knowledge Presentation with Online Community**

The CREEK application does not allow users to manipulate the 3D geometric models, but it can show instructions and related multimedia resources including images, videos, and audio. Through the web application, trainees can brush up the assembly sequences by navigating to previous or next steps as well as editing the current course item. Of course, the user can modify instructions and relevant resources, such as images, videos and hyperlinks to referential web sites. Figure 31 is captured from the CREEK application and teaches one assembly step by showing instructions, image or other resources.
8.6. Case Study

As case studies, we will make use of two scenarios: one is to assemble a wind turbine’s hub and the other is to replace a wild field camera in Hubble Telescope. The case studies demonstrate how product design knowledge, assembly simulation knowledge, training process knowledge, and relevant resources are respectively captured in each domain and transferred into the integrated knowledge repository.

8.6.1. Test Case – I: Assembly a Wind Turbine’s hub

*Present the knowledge in iTrain-AO ontology with CBT and IMT applications*

The snapshot [Figure 32] below was taken from the VCAT application. It imports the CBT instance in Figure 15 from the iTrain-AO ontology and visualizes assembly hierarchy and part information which are drawn from the product design ontology, kinematic simulations from the assembly simulation ontology as well as training courses...
from the assembly train ontology. The leaning contents and assembly hierarchy of the VCAT application are magnified on the bottom of the figure. With this application, user can see the product information, watch kinematic simulations, and review the assembly instruction. The comments and feedbacks from trainees can be populated back to the iTrain-AO ontology. In the future, the online-community can join in the training environment and add or update the training contents.

When a user is immersed into the virtual reality environment with the IMT application, the IMT instance from the iTrain-AO ontology will be used. Of course, the training devices are different, but the geometric models and training contents will be identical to the CBT application.

Figure 32: VCAT application importing and displaying training knowledge

The knowledge presentation will be executed by using different kinds of the training tools by importing the training ontology and the procedural knowledge ontology.
In the previous section, the training ontology instantiates the training tasks, equipment and methods. The training tasks can be in common between training tools but the equipment and methods will be differently exploited between them. In detail, CBT-Turbine, as an instance of the TrainMethod concept, can be used in the computer-based environment, and IMT-Turbine method can be done in the immersive environment by connecting to adequate interactive devices.

Similarly, Figure 33 is taken from CREEK and shows how to attach a latter on the hub body which is the Step 7 in the Table 2. CREEK can also help users manipulate the training tasks, equipment, methods and persons.

![Figure 33: Screenshots from Online Community describing the Step 7 in the assembly processes of the wind turbine](image)

To understand how the engineering knowledge management unit programmatically loads ontologies, a snippet of the TrainDomainKB java file is displayed below to show the process to read all the instances of the TrainTask concept from the
training ontology and construct a data structure. The \texttt{DSTTrainTask} class is referred to as the \textit{TrainTask} concept in the ontology.

\begin{verbatim}
public class TrainDomainKB {

    // 1. Load the training ontology
    public boolean loadModel(String strRunMode) {
        ProtegeOWL.createJenaOWLModelFromURI(m_strDocURI + m_strFileName);
        ... readTrainTaskKnowledge();
        ...
    }

    // 2. Read knowledge about the training task
    private List<DSTTrainTask> readTrainTaskKnowledge() {
        List<DSTTrainTask> lstObjects = new ArrayList<DSTTrainTask>();
        DSTTrainTask dstObject = null;

        // 3. Get all the instances of the TrainTask concept
        Collection<OWLIndividual> objTrainObjects = getOWLInstances(strClassName);
        if (objTrainObjects != null) {
            Iterator<OWLIndividual> it = objTrainObjects.iterator();
            while (it.hasNext()) {
                OWLIndividual objTrainObject = it.next();
                dstObject = new DSTTrainTask(objTrainObject);
                dstObject.readKnowledge();
                lstObjects.add(dstObject);
            }
        }

        return lstObjects;
    }

    public class DSTTrainTask {

    ...
        // 4. Get properties of the instances of the TrainTask concept
        OWLDatatypeProperty propTrainTaskData = null;
        propTrainTaskData = m_owlIndividual.getOWLModel().getOWLDatatypeProperty("Name");
        this.name = (String)m_owlIndividual.getPropertyValue(propTrainTaskData);
        ...
    }
}
\end{verbatim}

\textbf{Table 11: Programming example to read the training ontology and construct the data structure in the engineering knowledge management unit}

\section*{8.6.2. Test Case – II: Replacement of WFC3 in Hubble Scope}

The Figure 34 shows another case study where a wide field camera in the Hubble Scope is replaced with a new one. In reality, the WFC3 cannot be manipulated by one
human’s hand, but the Hubble telescope is scaled down for the training purpose. The image on the left side shows a kinematic animation and instruction in the VCAT application and the figure on the right is illustrates that a person grabs and inserts the new WFC3 into the body of Hubble telescope within the virtual environment. This example apparently shows that the same geometric model is imported and used between two different training tools.

![Figure 34: A snapshot for training replacing WFC3 (Wide Field Camera 3) in the Hubble Scope: VCAT on the left and VrToolsPro on the right sharing the geometric models](image)

8.6.3. Result from Case Studies

The two case studies test cases satisfy the four requirements: the connectivity with information about the CAD model, capability to share information, reusability and maintainability. The knowledge about geometric models is automatically captured from CAD tools to the product design ontology with the help of CAD APIs. When the product design knowledge is populated into different application specific ontology, such as PROE-AO, ontological mapping technique will help the knowledge transfer from the product design ontology to the iTrain-AO ontology. Therefore, the digital CAD models are all connected through different domain ontologies. The knowledge repository can
store ontologies and resources including CAD models and ultimately integrate data and knowledge into one place. This strategy isolates the knowledge base from the application so that multiple training applications can share knowledge irrespective of presentation styles and reusability of knowledge is increased. Last, comments and feedbacks acquired from trainees or experts can be conveyed to the knowledge engineers to update the ontologies in the training environment.

8.7. Summary

This work provides a methodology to integrate CBT and IMT application in the semantic data level by employing ontologies. To accomplish this goal, the knowledge in the iTrain-AO ontology could be shared and presented by both CBT and IMT applications in the integrated training environment. When the knowledge is presented, the training tasks and geometric models are commonly used by both the application. In addition, the shared product assembly knowledge can be presented on the semantic online community, CREEK. Also, the knowledge can be updated and managed by accepting feedbacks.
CHAPTER NINE

ENGINEERING KNOWLEDGE MAINTENANCE

Once a user edit contents though the web using any web-accessible devices, this modification will be delivered to the domain ontologies, like CAD ontology, PLM/PDM, or training ontology. These results from the fact that the online community is not only integrated with the procedural knowledge ontology, but also concepts and properties of ontologies are already mapped between different domains. For example, when a user wants to post a note about WFC3 saying “The WFC3 (Wide Field Camera 3) is a fourth-generation UVIS/IR imager and was installed in May 2009 to replaces the WFPC2 (Wide Field Planetary Camera2)”. This note will go from the procedural knowledge ontology to the training ontology or CAD ontology as expert comment.

Similarly, expert knowledge and raw data might be populated from different domain experts to different domain ontologies. For example, one annotation within the CAD system is acquired from a product design expert and is instantiated into the product design ontology or one feedback within the training environment is acquired from trainees and is instantiated into the training ontology. That expert knowledge will be translated into the procedural knowledge ontology by knowledge engineers and then be informed to participants on the online community.

When an online community is designed, an important functionality is the ability to share appropriate information with applications in the product design domain and the training domain. It is important to ensure that the user interface, the interaction, and the data exchange all focus on simplicity and ease of use. For example, if there are multiple versions of an assembly with the differences being just the number of bolts (and the
corresponding holes), and these bolts are referenced in the training content, the mapping will automatically add information to the engineering knowledge management system to link the training content to the 3D CAD product. If in the future, the size of the bolts are changed, or the number of bolts are changed, the system can trigger an event for an automatic update of the bolt assembly sequence, informing the author of the changes, and then training content and other resources including a text document will be updated. This entire methodology will allow users to collaboratively be involved in the knowledge management process, map knowledge between different domains and create methods to manage the training content in a cohesive and sustainable manner.

9.1. Case Study

*Update the knowledge according to the user’s feedback*

This training knowledge keeps being modified and maintained up to date. When there is a change in the assembly simulation domain, the change will result in modifications of training manual and environments. For example, the assembly procedures of the wind turbine hub said that the Step 5 requires an installation of a Chuck Motor Casing. As a change of the operation order, the Step 5 requires to be conducted before Steps 3 and 4 because of inadequate clearances between the components. The knowledge engineer captures this feedback and just changes the order of the assembly procedure by modifying properties of course items, such as `hasNextCourseItem` and `hasPrevCourseItem`. After the ontology models are updated, the trainee within CBT and IMT will get trained with an updated assembly procedure.
**Update the engineering knowledge through CREEK**

The knowledge can be evolved by comment or feedback from experts. Knowledge captured from experts is transferred to the knowledge engineer who is responsible for updating the existing knowledge or adding new knowledge and then the evolved knowledge is propagated up to the training environment or the product design tool. Here we will only focus on only the training domain. For instance, the assembly procedures of the wind turbine hub originally said that the Step 5 requires an installation of a Chuck Motor Casing. As a change of the operation order, the Step 5 requires to be conducted before Steps 3 and 4 because of inadequate clearances between the components. This comment is at first provided by a retired engineer though the online community. Next, the knowledge engineer populates it into the training ontology by altering the order of the assembly process as well as the procedural knowledge ontology by adding a new expert knowledge. Last, the new assembly processes will be updated within both CBT and IMT. As another example, the Figure 35 shows that one modification of the assembly instruction through CREEK will be conveyed to the training environment according to an expert note for Step 5 in the Table 2.
The semantic online community, CREEK, enables users to review, edit, and update product design and training materials and post their comments and thoughts. It is well incorporated with the knowledge management unit and the ontology-based knowledge repository so that messages from the online community could be delivered to assembly design applications and assembly training applications. In the result, CAD designers can always review any comments about the product assembly and trainees can
always practice up-to-date training courses through CBT or IMT. Additionally, since any users using web browsers or other web-accessible devices on the web can access to CREEK and retrieve or update information anytime, it will be useful to exchange ideas between engineers. Therefore, it is proven that CREEK can take responsibilities of knowledge acquisition, knowledge retrieval and presentation, and knowledge maintenance in the engineering lifecycle, it can be useful for knowledge retention and exchange, and it can be well integrated and communicated with other engineering applications.

9.2. Summary

The CREEK application was well communicated with the existing iTrain or blended training environment with CBT and IMT. It resulted from a fact that different domain applications shared the raw data including notes, images, videos and audios as well as the knowledge base composed of ontologies and their integration. Therefore, when one training application or the semantic online community modified the training contents or new knowledge was acquired from an expert, its modification would be propagated to other applications.
CHAPTER TEN

CONCLUSION AND FUTURE WORKS

The goal of this thesis is to improve methods to collaboratively maintain engineering knowledge in the product assembly design and assembly training domains. In order to achieve this goal, I employ ontologies to represent knowledge in each domain, exchange knowledge between domains, share knowledge between domain applications and maintain knowledge by providing tools to retain new information. Thus, three major research questions are raised: 1) how to transfer engineering knowledge from one domain to the other domain, 2) how to make domain applications share the unified engineering knowledge models, and 3) how to keep the engineering knowledge up-to-date through modification and acceptance of new information.

10.1. Conclusions

In this section, I present the conclusions and key contributions according to the three research topics. The each subsection is structured with my major approach to each question.

10.1.1. Ontological Mapping

The use of ontological mapping has been chosen to transfer engineering knowledge between different domains. Since our engineering knowledge models are based on three-tiered structure and represented in Chapter 5, we can find concepts and properties with the same meaning from two different application specific ontologies and transfer knowledge between them. Mapping strategies, such as, equivalency, attribute similarity, composition similarity, and instance similarity, have been defined in Section 7 and demonstrated the mapping process between a source ontology in the assembly design
domain and a target ontology in the assembly simulation domain. The work has shown that the defined mapping strategies could be very helpful to semi-automatically transfer knowledge between domain ontologies. It has also shown that matching or similar concepts from different application specific ontologies under the same domain specific ontologies could be conveniently found because many concepts and properties in the application specific ontology were inherited from the domain specific ontologies.

10.1.2. Integrated Training Environment

The integrated training environment based on the unified training ontology has been designed in Chapter 8 to integrate the training applications because the each application has different presentation methods and interfaces with users but some of contents, such as, 3D geometric models and assembly procedures, are commonly used. The iTrain-AO ontology has been designed in Section 5.5 by inheriting a simulation domain ontology and a training domain ontology. It has defined concepts, properties and relations necessary for the training environment and has also contained commonalities and differences over CBT and IMT. Demonstrations with two case studies have shown that product design models, training tasks and persons could be shared between CBT and IMT but training methods and equipment should be differentiated between them. Thus, it has been shown that the iTrain-AO ontology could be shared by training applications and could ultimately integrate them over the unified knowledge model. Also, the iTrain-AO ontology can be imported by other types of training applications, like SCORM.

10.1.3. Semantic Online Community

As an engineering applications, a semantic online community application, called CREEK, was designed. It has played an important role in the engineering knowledge
lifecycle by providing interfaces to acquire knowledge, and present knowledge as well as update knowledge relevant to the training contents. Most of all, the semantic online community could provide a place where engineering problems or questions are freely discussed over the web and could also maintain engineering knowledge by interacting the engineering knowledge repository containing ontologies and resources. Furthermore, it has made use of the procedural knowledge ontology to retain experts’ comments and information and to represent assembly procedures relevant to training tasks. Therefore, the semantic online community could be a useful approach in terms of knowledge maintenance because it could provide user interfaces to acquire engineering information, transfer the information to domain ontologies, and propagate the updated information into the other domains.

Overall, the engineering knowledge lifecycle has been achieved by knowledge modeling with ontologies, knowledge acquisition from an online community, CAD tools and training applications like CBT and IMT, knowledge integration with ontological mappings, knowledge retrieval and presentation by the online community and the training applications, and knowledge maintenance through the semantic online community. The engineering knowledge repository has been established on the basis of the product design ontology, the training ontology and the procedural knowledge ontology.

10.2. Future Works

We need to continue enhancing this work and seek to specifically address the following issues:

10.2.1. Knowledge Modeling and Knowledge Acquisition

- We need to further expand and refine the domain specific ontologies for more
domains.

10.2.2. Knowledge Integration

- Our mapping strategies can handle mapping between two concepts or between two properties. However, we need to define methods to map between a concept in source ontology and a property in the target ontology or the other way.
- When calculating composition similarity according to composition paths, we need to investigate how to map one concept to multiple concepts or vice versa.
- We need to investigate the use machine learning algorithms to calculate concept/property similarity and perform lexical matching.

10.2.3. Knowledge Retrieval and Presentation

- The other types of training application, such as a web-based training application compliant to SCORM, can be participated in the iTrain environment. Since the SCORM-compatible application is associated with the learning management system, it will be very helpful to track the trainees’ achievement and actions.

10.2.4. Knowledge Maintenance

- The semantic online community and the iTrain training environment will be integrated with PDM/PLM systems. By doing that, the history of knowledge will be traced and a business workflow-submission, approval/reject, revision-can be applied as a tool for knowledge management. In addition, a security issue, such as users’ access privileges will be resolved.
REFERENCES


APPENDIX A

Core Code for OMA
package wsu.vrcim.ontology.model.implementation;

import java.util.*;
import wsu.vrcim.ontology.mapping.*;
import wsu.vrcim.ontology.model.*;

/**
 * A general method of finding matching concepts
 * 1) Start with a matching process based on definitions
 *    : Find explicit equivalences based on definitions and define them as matching
 *    concepts.
 * 2) Create composition graph and inheritance hierarchies
 *    : Create composition graph for composition relations and class hierarchies
 *    for inheritance relations.
 * 3) Calculate Attribute Similarity
 *    : Among two composition graphs from two different ontologies, for each given
 *    source concept, calculate attribute similarity between the source concept and
 *    other concepts in target ontology and find all the nodes that have similar attribute.
 * 4) Find similar Composition Paths
 *    : Since the attribute is transitive over the composition relation, all the classes that
 *    has owner relations over the target concept will have at least the same attribute
 *    similarity.
 * 5) Calculate Composition Similarity
 *    : Calculate composition similarity and choose the one with the highest
composition similarity and attribute similarity as matching concept.

6) Calculate attribute and composition similarity based on inheritance relation:
Composition similarity and attribute similarity can be transitive over inheritance relation, all their sub-concepts are set to have the same attribute similarity and composition similarity.

7) Go to 3) until all the candidate concepts are calculated

8) Human intervention

public class ProtegeMappingImplementor implements AbstractMappingImplementor {

    // Static Variables
    //------------------

    // Logger
    static Log s_Logger = LogFactory.getLog(ProtegeMappingImplementor.class);

    // The manager for a source ontology
    private ProtegeModelImplementor _mgrSourceModel;

    // The manager for a target ontology
    private ProtegeModelImplementor _mgrTargetModel;

    /**
     * Constructor
     * @param _srcManager : OntModelManager class for the source ontology
     * @param _desManager : OntModelManager class for the target ontology
     */

    public ProtegeMappingImplementor(AbstractModelImplementor mgrSrcModel,
                                         AbstractModelImplementor mgrDesModel) {


_mgrSourceModel = (ProtegeModelImplementor)mgrSrcModel;

_mgrTargetModel = (ProtegeModelImplementor)mgrDesModel;

/**
 * Run algorithms to find matching concepts.
 */

public Collection<OMAConcept2ConceptMatch> executeConceptMapping() {
    Collection<OMAConcept2ConceptMatch> mappedConcepts = new ArrayList<OMAConcept2ConceptMatch>();

    ConceptMappingOperator operator = null;
    ConceptArrayComparer comparer = null;
    s_Logger.debug("Start mapping between two ontologies.");

    int nMethod = 1;
    // Run the algorithms. Here we will use all the algorithms defined
    // in the wsu.vrcim.ontology.mapping
    //-------------------------------------------------------------------
    while (true) {
        Collection<OMAConcept2ConceptMatch> arrConcepts = null;
        switch (nMethod) {
            case 1: // Determine if two classes has a lexical similarity
                operator = new LexicalSimilarityOperator();
                break;

            case 2: // Determine if two classes has same URI
                break;
        }
    }
}
operator = new UriMatchingOperator();
break;
case 3: // Determine if two classes are equivalent
    operator = new EquivalentMatchingOperator();
    break;
case 4: // Determine if two classes has the same restrictions
    operator = new AttributeSimilarityOperator();
    break;
case 5: // Determine if two classes has an composition
        // similarity
    operator = new CompositionSimilarityOperator();
    break;
case 6: // Determine if two classes has an attribute similarity
    operator = new InheritanceSimilarityOperator();
    break;
default:
    return mappedConcepts;
}

// Find matching concepts
//------------------------
arrConcepts = operator.findMappedConcepts(_mgrSourceModel,
    _mgrTargetModel);
// Save matching concepts
//------------------------

mappedConcepts.addAll(arrConcepts);

// Run the next algorithm
//------------------------

nMethod++;

} }
}
package wsu.vrcim.ontology.model.implementation;

import java.io.*;
import java.util.*;
import com.hp.hpl.jena.util.FileUtils;
import wsu.vrcim.ontology.model.*;
import wsu.vrcim.ontology.model.implementation.*;

// Protege-defined libraries
//-------------------------------
import edu.stanford.smi.protegex.owl.ProtegeOWL;
import edu.stanford.smi.protegex.owl.jena.JenaOWLModel;
import edu.stanford.smi.protegex.owl.model.*;
import edu.stanford.smi.protegex.owl.model.util.ImportHelper;
import edu.stanford.smi.protegex.owl.repository.*;
import edu.stanford.smi.protegex.owl.repository.impl.*;
import edu.stanford.smi.protege.exception.*;

public class ProtegeBridgeImplementor implements AbstractBridgeImplementor {
    // Static Variables
    //----------------------
    // Logger
    static Log s_LOGGER = LogFactory.getLog(ProtegeBridgeImplementor.class);
// Ontology model
private JenaOWLModel _ontModel = null;

// The URI over the network
private String _strBriDocURI = null;

// The alternative URI in the local desktop
private String _strBriAltURI = null;

// The File name
private String _strBriFileName = null;

/**
 * Constructor
 */
public ProtegeBridgeImplementor(String i_strDocURI, String i_strAltURI,
                                String i_strOntFile ) {
    // Information for a bridge ontology
    //-----------------------------
    _strBriDocURI = i_strDocURI;
    _strBriAltURI = i_strAltURI;
    _strBriFileName = i_strOntFile;
}
public boolean loadModel() {
    try {
        // 1.Create an ontology model
        _ontModel = ProtegeOWL.createJenaOWLModelFromURI(_strBriDocURI + _strBriFileName);
    } catch (OntologyLoadException ex) {
        try {
            _ontModel = ProtegeOWL.createJenaOWLModelFromURI("file:///" + _strBriAltURI + _strBriFileName);
            RepositoryManager mgrRepository = _ontModel.getRepositoryManager();
            mgrRepository.addProjectRepository(new LocalFolderRepository(new File(_strBriAltURI)));
        } catch (OntologyLoadException ex2) {
            ex2.printStackTrace();
            return false;
        }
    }
}
public void unloadModel() {
    return true;
}

/**
 * Write the ontology model into a owl file
 * @return
 */
public boolean writeModel() {
    boolean bRet = false;
    try {
        @SuppressWarnings("rawtypes")
        Collection errors = new ArrayList();
        _ontModel.save(new File(_strBriAltURI + "BDO-Test.owl")
            .toURI(), FileUtils.langXMLAbbrev, errors);
        bRet = true;
    } catch (Exception ex) {
        s_Logger.fatal("Fail to save a OWL model. " + ex.getMessage());
        s_Logger.fatal(ex.fillInStackTrace().toString());
    }
    return bRet;
}
public boolean createOntologicalMapping(String i_strMappingName,
                                        String i_strSrcUri, String i_strDesUri)
{
    // 1. Check the model
    //--------------
    if (_ontModel == null) {
        s_Logger.error("Cannot find the loaded ontology model. Check if it is created or opened!");
        return false;
    }

    // 2. Create classes for the mapping ontology
    //----------------------------------------------
    OWLNamedClass clsMapping =
        _ontModel.getOWLNamedClass("OntologicalMapping");
    OWLIndividual objMapping =
        clsMapping.createOWLIndividual(i_strMappingName);

    OWLNamedClass clsModel =
        _ontModel.getOWLNamedClass("BOModel");
OWLIndividual objModel =
    clsModel.createOWLIndividual(i_strSrcUri);

OWLDatatypeProperty propUri =
    _ontModel.getOWLDatatypeProperty("uri");

objModel.addPropertyValue(propUri, i_strSrcUri);

OWLObjectProperty propMappingObj =
    _ontModel.getOWLObjectProperty("has_model");

objMapping.addPropertyValue(propMappingObj, objModel);

objModel = clsModel.createOWLIndividual(i_strDesUri);

objModel.addPropertyValue(propUri, i_strDesUri);

objMapping.addPropertyValue(propMappingObj, objModel);

return true;

}

public boolean addConcept2ConceptMatch(OMAConcept2ConceptMatch match) {

    // 1. Check the model

    //------------------

    if (_ontModel == null) {
        s_LOGGER.error("Cannot find the loaded ontology model. Check
if it is created or opened!"");

    return false;
}

OWLNamedClass clsConcept1 =
    _ontModel.getOWLNamedClass("BOConcept");
OWLIndividual objConcept1 =
    clsConcept1.createOWLIndividual(match.getSourceBaseName()+ "-1");

OWLNamedClass clsConcept2 =
    _ontModel.getOWLNamedClass("BOConcept");
OWLIndividual objConcept2 =
    clsConcept2.createOWLIndividual(match.getTargetBaseName() + "-2");

    return true;
}

public boolean addUnpairedMatch()
{
    // 1. Check the model
    //-------------------

    if (_ontModel == null) {
        s_Logger.error("Cannot find the loaded ontology model. Check
return true;
}

/**
 * It creates a bridge ontology
 */
public boolean createBridgeOntology() {
    ...
    return true;
}

public boolean convertInstances(AbstractModelImplementor mgrSrc,
                                AbstractModelImplementor mgrDes) {
    mgrDes.saveModel();
    return true;
}
}
APPENDIX B

Core Code for CBT Application
BOOL CVCATDoc::CreateSceneGraph()
{
    m_pRootNode = new osg::Group;
    m_pRootNode->addChild(m_pHUDTitle->GetHUDPtr());
    m_pRootNode->addChild(m_pHUDDesc->GetHUDPtr());

    // Load Assembly
    //------------------
    CDSProductDesign* pdsProduct =
        m_pTrainDataModel->GetTrainSimulation()->GetProductDesign();
    if (!pdsProduct) return FALSE;
    CDSAssembly* pdsAssm = pdsProduct->GetAssembly();
    if (!pdsAssm) return FALSE;
    m_pRootNode->addChild(CreateAssemblyModel(pdsAssm));

    // Load Animations
    //------------------
    m_rpAnimationMgr = new osgAnimation::BasicAnimationManager();
    m_pRootNode->setUpdateCallback(m_rpAnimationMgr);

    CDSSimulationStudy* pdsSimStudy =
        m_pTrainDataModel->GetTrainSimulation()->GetSimulationStudy();
    if (!pdsSimStudy) return FALSE;
CDSKinematicSimulation* pdsKinematicSim =
    pdsSimStudy->GetKinematicSimulation();

if (!pdsKinematicSim) return FALSE;
CreateKinematicSimulation(pdsKinematicSim);

return TRUE;
}

osg::ref_ptr<osg::Group> CVCATDoc::CreateAssemblyModel(CDSAssembly* pAssm)
{
    osg::ref_ptr<osg::Group> grpAssm = new osg::Group;

    CString strGroupName = pAssm->GetDSName() +
        "<" + pAssm->GetSWIN() + ">";
    grpAssm->setName((LPCTSTR)strGroupName);
    grpAssm->setNodeMask(NODEMASK_ALL);

    CDSCComponentList* pCompList;
    CDSCComponentIterator it;

    pCompList = pAssm->GetComponentListPtr();
    for (it = pCompList->begin(); it != pCompList->end(); ++it)
    {
        }
CDSCOMPONENT* pComponent = *it;

CString strClassName = pComponent->GetClassName();

if (!strClassName.CompareNoCase(_T("ASSEMBLY"))) {
    CDSAssembly* pSubAssm = (CDSAssembly*)pComponent;
    grpAssm->addChild(CreateAssemblyModel(pSubAssm));
}
else if (!strClassName.CompareNoCase(_T("PART"))) {
    TCHAR szPartFile[512];
    CDSPART* pPart = (CDSPART*)pComponent;
    wsprintf(szPartFile, _T("%s\\..\OSG\%s"),
              (LPCTSTR)m_strModelPath, pPart->GetPartFileName());

    osg::ref_ptr<osg::MatrixTransform> xFormPosition =
        new osg::MatrixTransform();

    
    osg::Matrix m;
    float* pfMatrix = pPart->GetTransform()->
        GetTransformMatrix();
    m.set(pfMatrix[0], pfMatrix[4],
          pfMatrix[8], pfMatrix[12],
          pfMatrix[1], pfMatrix[5],
          pfMatrix[9], pfMatrix[13],
          pfMatrix[2], pfMatrix[6],
          pfMatrix[10], pfMatrix[14],
          pfMatrix[3], pfMatrix[7],
          pfMatrix[11], pfMatrix[15]);
pfMatrix[2], pfMatrix[6],
pfMatrix[10], pfMatrix[14],
pfMatrix[3], pfMatrix[7],
pfMatrix[11], pfMatrix[15]);

xFormPosition->setDataVariance(
    osg::Object::DYNAMIC);

xFormPosition->setMatrix(m);

xFormPosition->setName( pPart->GetDSName());
}

osg::ref_ptr<osg::MatrixTransform> xFormCallback = new osg::MatrixTransform();

osg::Node* pNode = osgDB::readNodeFile(szPartFile);
if (pNode != NULL)
{
    int nNodeMask = pPart->IsVisible() ? NODEMASK_VISIBLE : 0;
    nNodeMask |= pPart->IsPickable() ? NODEMASK_PICKABLE : 0;

    pNode->setName(pPart->GetDSName() + "<" + pPart->GetSWIN() + ">");
}

int nNodeMask = pPart->IsVisible() ? NODEMASK_VISIBLE : 0;

nNodeMask |= pPart->IsPickable() ? NODEMASK_PICKABLE : 0;
pNode->setNodeMask(nNodeMask);

CString strTexFile = pPart->GetTextureFileName();
if (!strTexFile.IsEmpty())
{
    strTexFile = m_strModelPath + "\\" + strTexFile;
    osg::ref_ptr<osg::Image> image(osgDB::readImageFile ((LPCTSTR)strTexFile));
    osg::ref_ptr<osg::Texture2D> tex = new osg::Texture2D;
    tex->setImage(image.get());

    osg::ref_ptr<osg::StateSet> nodess (pNode->getOrCreateStateSet());
    nodess->setTextureAttributeAndModes(0, tex.get(), osg::StateAttribute::ON);
}

xFormPosition->addChild(pNode);
CString strCallbackName = pPart->GetSWIN();
xFormCallback->setName((LPCTSTR)strCallbackName);
xFormCallback->setUpdateCallback(new osgAnimation
:: UpdateTransform((LPCTSTR)strCallbackName));
xFormCallback->addChild(xFormPosition);
grpAssm->addChild(xFormCallback);
}
}
}
return grpAssm.get();
}

void CVCATDoc::CreateKinematicSimulation(
     CDSKinematicSimulation* pdsKinematicSim)
{

    CDSKinematicAnimationList* pAnimList;
    CDSKinematicAnimationIterator it;

    pAnimList = pdsKinematicSim->GetAnimationListPtr();
    for (it = pAnimList->begin(); it != pAnimList->end(); ++it)
    {
        // Create an animation
        //---------------------
        CDSKinematicAnimation* pAnimation = *it;
CLinearAnimationHelper* pAniHelper =
    new CLinearAnimationHelper();

CString strTargetName = pAnimation->GetRefComponentName();
pAniHelper->CreateAnimation((LPCTSTR)pAnimation->GetDSName(),
    (LPCTSTR)strTargetName);

m_rpAnimationMgr->registerAnimation(
    pAniHelper->GetAnimationPtr());

// Enter the path
//----------------

CDSMotionPathNodeList* pNodeList;
CDSMotionPathNodeIterator it2;

pAnimation->GetMotionPath()->SortMotionPathNodes();
pNodeList = pAnimation->GetMotionPath()->
    GetMotionPathNodeListPtr();
for (it2 = pNodeList->begin(); it2 != pNodeList->end(); ++it2)
{
    CDSMotionPathNode* pPathNode = *it2;
    CDSTransform* pTx = pPathNode->GetTransform();
    pTx->ConvertMatrixToEuler();
    double fTime = pPathNode->GetTime();
osg::Vec3 position(pTx->GetPosX(), pTx->GetPosY(), pTx->GetPosZ);
pAniHelper->AddKeyframeToChannel("position", fTime, position);

osg::Vec3 rotation(pTx->GetRotX(), pTx->GetRotY(), pTx->GetRotZ);
pAniHelper->AddKeyframeToChannel("euler", fTime, rotation);
}

pAniHelper->SetPlayMode(osgAnimation::Animation::PPONG);
}
APPENDIX C

Core Code for CREEK
package wsu.vrcim.ontology.dstknowledge.trainstudy;

//0. Import libraries
//---------------------

import java.util.*;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

import edu.stanford.smi.protegex.owl.jena.JenaOWLModel;
import edu.stanford.smi.protegex.owl.model.*;

import wsu.vrcim.ontology.dstknowledge.DSTObjectStream;
public class DSTTrainDomainKB {

    //==================================================================================================
    //    STATIC VARIABLES
    //==================================================================================================

    static Log _Logger = LogFactory.getLog(DSTTrainDomainKB.class);

    private String m_strProductId = "";

    //==================================================================================================
    //    PRIVATE VARIABLES
    //==================================================================================================

    private JenaOWLModel m_owlModel = null;

    private List<DSTTrainStudy> m_lstTrainStudies = null;

    private List<DSTTrainMode> m_lstTrainModes = null;

    private List<DSTTrainPerson> m_lstTrainPersons = null;

    private List<DSTTrainTask> m_lstTrainTasks = null;

    private List<DSTTrainEquipment> m_lstTrainEquipments = null;

    private List<DSTTrainMethod> m_lstTrainMethods = null;

    private List<DSTTrainCourse> m_lstTrainCourses = null;

    private List<DSTCourseModule> m_lstCourseModules = null;

    private List<DSTCourseItem> m_lstCourseItems = null;

    private List<DSTResource> m_lstResources = null;

    private List<DSTParameter> m_lstParameters = null;

    private List<DSTFeedbackAndComment> m_lstUserComments = null;
}
public DSTTrainDomainKB(JenaOWLObject owlModel) {
    m_owlModel = owlModel;
}

private List<DSTTrainMode> readTrainModeKnowledge() {
    List<DSTTrainMode> lstObjects = new ArrayList<DSTTrainMode>();
    String strClassName = "TrainMode";
    DSTTrainMode dstObject = null;

    // 1. Get instances

    Collection<OWLIndividual> objTrainObjects =
        getOWLInstances(strClassName);
    if (objTrainObjects != null) {
        Iterator<OWLIndividual> it = objTrainObjects.iterator();
        while (it.hasNext()) {
            OWLIndividual objTrainObject = it.next();
        }
    }

    return lstObjects;
}
dstObject = new DSTTrainMode(objTrainObject);
dstObject.readKnowledge();
lstObjects.add(dstObject);
}
}
return lstObjects;

//=====================================================
//    Write knowledge about the training mode
//=====================================================

public void writeTrainModeKnowledge() {
    Iterator<DSTTrainMode> it = m_lstTrainModes.iterator();
    while (it.hasNext()) {
        DSTTrainMode item = it.next();
        item.writeKnowledge();
    }
}

//=====================================================
//    Read the knowledge of the training domain
//=====================================================

public boolean readTrainDomainKB() {

m_lstTrainStudies = readTrainStudyKnowledge();
Collections.sort(m_lstTrainStudies);

m_lstTrainModes = readTrainModeKnowledge();
Collections.sort(m_lstTrainModes);

m_lstTrainPersons = readTrainPersonKnowledge();
Collections.sort(m_lstTrainPersons);

m_lstTrainTasks = readTrainTaskKnowledge();
Collections.sort(m_lstTrainTasks);

m_lstTrainEquipments = readTrainEquipmentKnowledge();
Collections.sort(m_lstTrainEquipments);

m_lstTrainMethods = readTrainMethodKnowledge();
Collections.sort(m_lstTrainMethods);

m_lstTrainCourses = readTrainCourseKnowledge();
Collections.sort(m_lstTrainCourses);

m_lstCourseModules = readCourseModuleKnowledge();
Collections.sort(m_lstCourseModules);
m_lstCourseItems = readCourseItemKnowledge();

Collections.sort(m_lstCourseItems);

m_lstResources = readImageResourceKnowledge();

m_lstResources.addAll(readTextResourceKnowledge());

m_lstResources.addAll(readVideoResourceKnowledge());

m_lstResources.addAll(readAudioResourceKnowledge());

Collections.sort(m_lstResources);

m_lstParameters = readParameterKnowledge();

Collections.sort(m_lstParameters);

m_lstUserComments = readUserCommentKnowledge();

Collections.sort(m_lstUserComments);

return true;
public boolean writeTrainDomainKB() {
    writeTrainStudyKnowledge();
    writeTrainModeKnowledge();
    writeTrainPersonKnowledge();
    writeTrainTaskKnowledge();
    writeTrainEquipmentKnowledge();
    writeTrainMethodKnowledge();
    writeTrainCourseKnowledge();
    writeCourseModuleKnowledge();
    writeCourseItemKnowledge();
writeResourceKnowledge();

writeParameterKnowledge();

writeUserCommentKnowledge();

return true;

}