# AN ONTOLOGY-BASED APPROACH FOR SEMANTIC LEVEL INFORMATION EXCHANGE AND INTEGRATION IN APPLICATIONS FOR PRODUCT LIFECYCLE MANAGEMENT

By

PEI ZHAN

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 2007

To the Faculty of Washington State University:

The members of the Committee appointed to examine the dissertation of PEI ZHAN find it satisfactory and recommend that it be accepted.

________________________________________

Chair

________________________________________

________________________________________

# ACKNOWLEDGMENTS

I would like to give special thanks to my wife, Xiangling, who is continually the love and inspiration of my life.  To my parents, their love and encouragement that enabled me to achieve my goals.  Thanks to my friends and fellow members in VRCIM Laboratory I've worked with: YoungJun, Okjoon, Lijuan, Gareth, Hrishi.  Finally, special thanks to my committee, Dr. Uma Jayaram, Dr. Sankar Jayaram, and Dr. Hakan Gurocak, whose mentoring, wisdom, and friendship have been gratefully received.

# AN ONTOLOGY-BASED APPROACH FOR SEMANTIC LEVEL INFORMATION EXCHANGE AND INTEGRATION IN APPLICATIONS FOR PRODUCT LIFECYCLE MANAGEMENT

Abstract

By Pei Zhan, Ph.d
Washington State University
August 2007

Chair: Uma Jayaram

During product lifecycle management (PLM), product information from CAD/CAE applications regularly needs to be exchanged and shared between the various applications. However, these applications often have different product data semantics and corresponding representations. The interoperability problem caused by the heterogeneous semantics and data representation is critical and needs to be addressed and automated. Recent research has focused on integration frameworks for CAD/CAE applications in order to improve interoperability. There are fundamental problems that still need to be addressed.

We identified the following important roadblocks and sought to address these specifically in our work: 1) The need for an adequate product knowledge representation of engineering design/analysis, which is easily expandable, and customizable for traditional and non-traditional (e.g. virtual prototyping) design information systems that also allows the sharing of product data semantics across all these heterogeneous systems to support distributed, collaborative engineering capabilities; 2) The need for a way to generate product data semantics by using engineering design/analysis knowledge to interpret actual product data 3) The need for a way to reconcile the differences in the different product semantics by finding underlying similarities between different knowledge representations that are from different viewports and reconcile, and use these similarities to then translate product data semantics correctly.

This dissertation proposes an ontology-based approach for a semantic level exchange and integration to improve interoperability, which includes an ontology building tool, ontology mapping tools and custom tools to associate ontologies to prduct data. For the purpose of semantic level integration, a way of representing engineering design/analysis knowledge using an engineering ontology is proposed. A layered structure is used for building knowledge into engineering ontologies so as to improve the scalability and composition adaptivity. Based on the knowledge, a semantic layer is built upon product data to use concepts/relations in ontologies to describe actual product data, which can be used to represent understandings about a product design from different perspectives. To enable translating different understandings (product data semantics) using different ontologies, an ontology mapping method is proposed to find matching concepts between different ontologies, based on three basic relation types between concepts: composition, inheritance and attribute.

A scenario is explained to describe the working mechanism of the system and to demonstrate the concept of semantic level integration framework for a simple example. A sample assembly is designed and simulated in different software packages and an integrated process is made to exchange information between them. The scenario successfully demonstrates the ontology based approach.

TABLE OF CONTENTS

TABLE OF FIGURES

# Chapter One - Introduction

Computer Aided Design (CAD), Computer Aided Manufacturing (CAM) and Computer Aided Engineering (CAE) are important technologies that have now become mainstream in the product cycle. Over the last decade, tremendous progress has been made in the development of software tools in these three areas. This has resulted in greatly expediting the product development process.

However, there often arises the problem of interoperability between different solutions involved in design, manufacturing, and analysis. The information generated in one domain can sometimes not be recognized in another program. Furthermore, the communication between engineers working in different domains is limited; a person working as a design engineer may sometimes have difficulty in understanding the comments and terms used by an analysis engineer, though they may both refer to the same product. There are several issues that contribute to this problem. In addition, people working in the design and analysis domains sometimes tend to view a product from different perspectives, and they may use different terminologies to describe the same product data, which makes communication more difficult. To make the information compatible, there may be a need to convert the data to a neutral format for use in the various applications by engineers. A lot of important design information including intent and rationale usually gets lost during the data conversion process..

To solve this problem, there are several critical challenges that need to be acknowledged in the CAD/CAM/CAE integration:

1. The challenge of multiple representations of a product from different perspectives: Since CAD/CAM/CAE are relevant to many different domains in engineering, each perspective within a domain represents specific knowledge of a given product. There is no uniform format or terminology to represent certain kinds of information, since people use different informal tools to present thoughts, such as word documents, voice, video, etc. This sometimes makes the exchange of information difficult.

2. The challenge of integrating programs working in different domains: Whether a representation is useful or not depends on what activities are being performed with the product data. For example, design features that are useful to a CAD user are not necessarily useful to an analysis engineer, and vice versa, though both sets of engineers are referring to the same product data. Furthermore, there is no simple way of mapping from one representation to another.

Ontology is a form of knowledge representation that uses a set of concepts in a domain and the relationships between them. Axioms and rules can also be used to enhance the expressiveness.

In this dissertation, an architectural framework with an ontology-based knowledge system is proposed to integrate different CAD/CAE programs. In this framework, a semantic level integration can be achieved by building representations of a product from different perspectives using ontologies and translating different representations from one perspective to another using ontology mapping.  This dissertation will also demonstrate how to improve the interoperability, reusability and scalability of an integration framework.

# Chapter Two - Literature Review

There has been much research, relevant to the integration of CAD/CAE applications, which focuses on integration frameworks, product information modeling, and the methodology of integrating applications. Ontology applications have been an active research area in which researchers try to solve the interoperability problem using ontology mapping. Also ontology is used to capture rich semantics during the design process.

## 2.1 Integration Framework

An integration framework coordinates heterogeneous tools and product models in a unified and streamlined manner and provides an infrastructure for explicitly capturing information flow and engineers' rationales in the product design and analysis processes. An integration framework gives engineers more flexibility in choosing suitable tools and enables them to focus on accomplishing their goals without having to construct a complex unified model. There have been many frameworks proposed for integration of applications in Product Lifecycle Management (PLM), mostly focused on solving the problem of interoperability. Several technologies have been developed in two key categories:

1) Middleware has been developed using CORBA, DCOM, and Java RMI to serve the whole system architecture and improve the communication between applications. All applications must obey certain protocols in order to communicate with

each other. In a distributed design simulation marketplace[1], an object model representation is presented, and the object-oriented service object serves as a middleware to communicate with the individual components and to manage information, data, or relations between user-defined data and the CAD model. Application middleware has been developed using Java RMI to integrate distributed applications such as the common manufacturing application middleware[2].

2) Standard exchange formats, such as STEP, IGES, etc. have pre-built concepts and relations to promote interoperability. In this approach, all the data exchanged in one framework must use the same format. Usually, CAD models in a standard format are stored in a central or distributed database and shared with different applications[3]. Some existing projects using these standard exchange formats include a web-based integrated product development platform for concurrent design manufacturing of sheet metal[4]. This design is based upon an information integration framework using STEP files and a STEP-based method and system for concurrent integrated design and assembly planning[5].

## 2.2 Product Information Model

In a design/analysis integration framework, different applications working on the same product design need to exchange information based on the product information model. Current research activities focused on the integration of engineering applications have identified product data representation as a key issue for design research and development[6-9], especially for solving the problem of interoperability. In the era of

PLM, this problem is becoming even more important; PLM requires that a product be managed throughout its whole lifecycle. Therefore, one of the fundamental problems in an integration framework that supports PLM is that of representing all the information about a product from different phases and in different perspectives, and thus translating product data from one representation to another.

For some integration frameworks where only certain types of products are involved, customized model templates can be used to translate product models[10]. However, the scope of this kind of product model is limited; only those products that can share the same template can be used in the integration.

In addressing the issue of interoperability, feature-based product design representation[11] is used as one of the most important technologies for CAD/CAM/CAPP integration. A feature can be regarded as a meaningful geometric construct that contains both geometry and manufacturing information[12-14]. However, there still remain several problems for integration using feature-based technology. Firstly, there is the issue of semantics. What this means is that for the same product, there can be different feature representations in different domains. There are often problems related to how to extract features from different perspectives and translate them to other representations. Secondly, it is often hard to use the concept of features in certain domains beyond design and manufacturing, such as ergonomics evaluation or assembly simulation. In these domains where some parameters and structures (e.g. ergonomics rating) cannot be described using only features, this concept lacks the necessary information for those domains and is also difficult to extend. Thirdly, from some perspectives, feature is not used to describe the design/analysis and is not a relevant tool at all.

Several other important and effective product information models have been proposed to address the issue of product lifecycle management and some models have been considered using knowledge representation to address terminological and semantic issues [15-18]. Along with the product information model, an elegant design information flow model in design is also presented[19]. Requirements of a next-generation product development system is proposed as a system “that can collaborate using a heterogeneous set of software tools, and still exchange information meaningfully and pass knowledge between various phases in the process” [15]. In order to achieve the above goal, a formal knowledge representation with the following features has been identified: 1. Not tied to a single vendor software solution; 2. Open and non-proprietary; 3. Simple and generic; 4. Extensible by allowing augmentation of the core with additional concepts to create a broader engineering context; 5. Independent of any one product development process; 6. Capable of capturing that portion of the engineering context that is most commonly shared in product development activities[14].

## 2.3 Product Data Format

Even for the same product information model, the format that the data uses could be different. As an international standard, STEP-ISO 10303[20] was adopted as a core data model in many integration frameworks[21-25]. STEP is a standard for computer-interpretable representation and exchange of product data through out the whole product lifecycle. Primarily, EXPRESS is used to describe the data model. Several data formats are provided for application data to be exchanged, including STEP-XML, STEP-file, or a shared database. Some of the shortcomings of STEP include its lack of model-reusability and extensibility and the fact that, if it is not self-describable, it is difficult to use in product data semantics.

## 2.4 Capturing Product Data Semantics

Although product data semantics can be represented in some product information models, it is still difficult to actually capture this information. The primary obstacles are the technical challenges of organizing and managing knowledge and the design challenges of a human-centered approach to building a useful and usable product data semantics recording system[7].

A team of designers model a 3-D layout and semantically-grounded behavioral description of a product or device[26]. The device's key features include integration of functional modeling with sketch-based conceptual design; and group authoring of design semantics. In MUG, a Multi-Session Distributed Conceptual CAD Environment, a user can annotate a design using a pre-defined ontology that has a function-behavior-structure form. DAML is used as the ontology language. Some design primitives are used to describe structure.

## 2.5 Ontology

"In both computer science and information science, an ontology is a data model that represents a domain and is used to reason about the objects in that domain and relations between them" [27].

As an important technology in Artificial Intelligence to share and reuse knowledge, ontology was proposed as a solution to a lack of interoperability. Ontology can be regarded as "an explicit specification of conceptualization"[28], or defined as "domain theories that specify a domain-specific vocabulary of entities, classes, properties, predicates, and functions, and a set of relationships that necessarily hold among those vocabulary items. Ontologies provide a vocabulary for representing knowledge about a domain and for describing specific situations in a domain" [29]. For example, a wine ontology[30] can define taxonomy of wine and classify Wine as type of WhiteWine, Loire, WhiteLoire etc. and in the wine ontology the relations between types of wine can also be defined such as WhiteLoire is an intersection of Loire and WhiteWine which means an instance of WhiteLoire is both Loire and WhiteWine.

### 2.5.1 Design Ontology

A general ontology for design concepts is proposed to describe the interactions between design concepts[31]. An extended device ontology is proposed to describe artifacts based on intended use as a composite of devices that processes input and produces output[32].

Process Specification Language (PSL)[33] has been developed at NIST and is written in Knowledge Interchange Format (KIF) as an interlingua for different manufacturing process applications to exchange information. The focus of PSL is to represent and capture manufacturing process-specific data only. Concepts that need to be represented in product data, such as design rationale, function, behavior, and interpart relationships, cannot be represented using PSL.

A Product Semantic Representation Language (PSRL) [34] based on formal description logic (DAML+OIL) is used to encode semantics. Mathematical logic and corresponding reasoning are used to determine semantic equivalences between an application ontology and PSRL. In the project, only exact equivalences between two ontologies are considered.  This is significant but limits ontology mapping on the feature level. Also, it is not discussed that how to automatically find equivalences between two ontologies.

Ontologies have been and are currently being used in several projects to represent conceptualizations of products and capture product data semantics [35-39]. Furthermore, in the integration of engineering applications, ontologies have been used in many projects to solve interoperability problems including semantic clashes and multiple translators. Approaches were developed in those projects can be categorized into two types: 1) using a shared, common ontology; and 2) using ontology as an interlingua[39]. As an integration framework consisting of existing heterogeneous applications that use different languages to represent data, the latter approach seems to be a better choice. For example, PSL[40;41] is used to represent ontologies in the manufacturing process, which works as an interlingua to integrate a process modeling tool and scheduling tool. However, in this integration process, it is always supposed that the semantics have already been built and that the ontology merely acts as a translator. Inference support is being added and improved. In a concurrent distributed design environment for exchanging product data semantics, it is important for an integration process to be able to capture the product data semantics during the information exchange and use an inference tool to make further assertions.

A "car seat" ontology is built to provide a shared conceptualization of the product requirement, which encapsulates the required functionalities, design parameters, performance criteria, structure and geometry[42].

### 2.5.2 Supporting Technologies for Developing Ontology

Recently, several ontology definition languages have been created during the development of the semantic web[43;44] such as RDF, RDFS[45-47], and OWL[48], and are widely accepted as important ontology languages. Some other ontology languages were developed based on these for the purpose of engineering integration, such as PSRL, developed at the University of Michigan[34].

OWL is already used extensively in describing product family modeling since it can store the structure of the product family as well as the evolution of different components of the product family. There are several other development tools supported globally by different research groups such as Jena[49] and Protégé [50]. Jena, a Java framework for building Semantic Web application, provides a programmatic environment for RDF, RDFS and OWL, and also includes a rule-based inference engine. Protégé is an ontology editor that also supports ontology building using RDF, RDFS and OWL. Similarly, OilEd also provides this kind of functionality. Reasoners such as RACER and FaCT also support reasoning for RDF, RDFS and OWL.

### 2.5.3 Ontology Mapping

Even in describing the same subject, different ontologies can use different concepts/terminologies. Using ontology in integration itself doesn't solve interoperability problem, it just raises heterogeneity from data level to a semantic level. To solve the interoperability problem, ontology Mapping is one of the critical techniques in reconciling differences between ontologies thus enabling semantic information exchange. Ontology Mapping is used to find relationships between entities, given two ontologies that describe each in a set of discrete entities. It is sometimes referred to as "Ontology Alignment".

The main issue in Ontology Mapping is finding what entity in one ontology corresponds to that in another. Basically, there are two ways of discovering mapping: either by using a *shared ontology*; or by using a *heuristic-based* approach. A general top level ontology is normally built to describe some commonly accepted concepts such as time and space. Other domain-specific ontologies are built upon the general top level ontology so that all the concepts in it can be used as standards for calculating similarities between them. This method may be difficult for semantic web integration since it is impossible for all ontologies developed globally to follow one general ontology. However, it is ideal for a specific domain, which has many sub-domains, as a core ontology for the domain can be built and other sub-domain ontologies can be built upon that. Examples include SUMO[51] and DOLCE[52]. An integration framework was developed using PSL for the purpose of integrating manufacturing processes[53].

Compared to the method of shared ontology, the other method tends to use a heuristic-based approach, such as data analysis, machine learning, statistics, or knowledge representation. For example, PROMPT[54] regards ontology as a graph, and finds similar nodes by comparing nodes in similar paths with the same start and end nodes. Other methods include combining weighted similarities in different definitions in OWL, such as domain, range, and properties[55]; and a method of finding class equivalence in two ontologies that share instances using formal concept analysis[56]. Recently a framework for integration based on ontologies has been proposed and a mapping algorithm is developed based on a whole set of mapping axioms[57]. The proposed framework uses a common upper level engineering ontology to represent semantics, and the ontology mapping is limited in the local level ontology which is used to represent each specific design model.

# Chapter Three - Problem Statement, Proposed Solution, And Scope of Work

This chapter identifies some of the higher-level limitations of the current CAD/CAE integration frameworks and seeks to identify certain specific limitations that are a sub-set of these higher-level limitations and will be addressed in this dissertation.. Also included is a brief description of the approach used to address these issues that is followed by a description of the scope of work and an organization of the disseratation.

## 3.1 Problems of CAD/CAE Integration Framework

A product is considered from different viewpoints and perspectives, and different representations and terminologies may be used by each viewpoint. For example, consider an automobile interior design evaluation. One viewpoint may focus on ergonomic comfort and accommodation while another may focus on assembly simulation for the dashboard. The representation of the product, the terminology used, and the communication of the results of these analyses vary greatly despite the fact that the underlying product is the same.

Unfortunately, most ensembles of engineering software tools only understand their own knowledge representations are not adaptable to others; furthermore, there are few general frameworks and approaches for enabling this adaptability. Although each engineering software tool can solve a problem within its scope in an effective manner, when there is a task requiring collaboration between people with different expertise and tools, these tools do not talk to each other effectively. The data formats for these tools vary greatly and, more importantly, the results generated in one system based on its own knowledge representation cannot be understood in another. Product data semantics generated throughout the design process cannot be collected efficiently and are almost always discarded. This impacts the efficiency and survivability of the ensembles that are cobbled together. There are several critical barriers:

- A lack of an approach to build **knowledge representation** of engineering design/analysis, in a manner that is a) easily expandable, b) customizable for traditional and non-traditional (e.g. virtual prototyping) design information systems. c) capable of sharing of product data semantics across all these heterogeneous systems to support distributed, collaborative engineering capabilities
- A lack of a way to capture the **product data semantics** and record design intent and design history
- A lack of a way to **translate product data** based on one knowledge representation to one based on knowledge representation.

Thus, it is becoming increasingly important for people to share their product data semantics and rationale in such a way that there exists a true integration of the product model data and support for diverse compositions with engineering application ensembles and diverse viewpoints. Although engineering firms are using a number of software tools in the product development cycle, these tools do not talk to each other very effectively. The knowledge representations behind the data vary greatly and the results generated in one system cannot be understood in another system. This miscommunication has led to a significant focus in recent years on PDM and PLM. Most traditional integration efforts have focused on pure data exchange, but there is a lack of effort in attempting to translate data at a semantic level.

## 3.2 Problem Statement

Relatively new software technologies, such as tools for developing ontology, are now making feasible the exchange of semantic information among engineering applications. For example, through proper integration of software tools, a user of commercial ergonomics software should be able to exchange product data semantics with a commercial CAD software user to communicate what needs to be modified and why. In addition to providing a common thread for communication, the use of ontology will also ensure that this information becomes part of the product model and the development process. This level of integration is referred to as “integration at a semantic level”.

In order to focus our investigation, we concentrate on the representation of product data semantics information and methods for the exchange of product data semantics. The specific objective is: To model an approach based on ontology engineering for the semantic level integration of software tools for CAD/CAE applications, after identifying fundamental issues of interoperability problem in integrating these tools.

Specifically, the following research questions are investigated:

1) How to build knowledge representation, such that different engineering applications can have their terminologies/knowledge easily represented and reused.

2) How to represent an engineer’s understanding (product data semantics) about a product design by associating CAD data to knowledge in engineering design and analysis.

3) How to find the similarities between different knowledge representations that are from different viewports, and use these similarities to translate product data semantics that is based on them.

## 3.3 Proposed Solution

The proposed approach is to build an ontology-based knowledge system in an integration framework. The requirements, specifications, parameters, description, data, and model (collectively referred to here as “**concept**”) of a design vary vastly depending on the viewpoint of the engineer or design activity. From different viewpoints, different concepts can be applied to a design. By building these concepts into ontologies, different viewpoints of a design can be represented through domain-specific concepts and terminology. This provides an environment for the variety of applications involved in the design process to describe a design/simulation decision in their own languages. By mapping the ontology of the concepts of one viewpoint to concepts in other ontologies, representations of a design from different viewpoints can be interconnected and exchanged.

## 3.4 Scope of Work

The research described in this dissertation focuses on realizing the solution proposed above through design and integration of the following: 1) An integration framework prototype that would allow applications in engineering design/analysis domains to exchange design semantic information; 2) An ontology architecture to capture and describe the product data semantics during the design and analysis processes; 3) A method to translate product data semantics used in one ontology to semantics used to describe concepts in another– specifically, how to discover mapping information between different semantic information; and, 4) A method of representing the mapping information between semantics in different domains/applications so that the information generated in the translation process can be retrieved during the integration/communication process.

Among the above tasks, the most important aspect of enabling semantic integration is that of discovering the relationship between different entities in different ontologies in order to translate semantics.

## 3.5 Organization of the dissertation

The dissertation is organized as follows:

Chapter 4 describes the overall architecture of the framework, including a description about the working mechanism of the framework.

Chapter 5, 6 describe how knowledge of engineering design and analysis in different domains is built into ontologies in a layered structure. Chapter 5 discusses the knowledge in the domain and application level. Chapter 6 discusses the knowledge in a general level.

Chapter 7 describes how product data semantics is captured using the knowledge acquired in chapter 5 and 6.

Chapter 8 describes how to calculate similarities between ontologies so to translate product data semantics described by the concepts in different ontologies.

Chapter 9 shows how the mapping information between different product data semantics is represented

Chapter 10 shows an example scenario in which a sample assembly is designed and simulated in different software packages and an integrated process is made to exchange information between them.

Chapter 11 discusses conclusion and future work

# Chapter Four - Architecture of CAD/CAE Integration Framework

This chapter discusses in detail the architecture of the proposed integration framework, which supports integrating design and analysis applications at the semantic level. The essential components of the proposed system are as follows: 1) Ontology builder; 2) Ontology mapping tool; 3) Custom tools. The requirements and details of each of these components are discussed.

## 4.1 Requirements of Integration Framework

The requirement of the proposed integration framework is to support an approach for integrating engineering software systems in a manner that can *adapt* to varying tools and points of view. There is a collection of design activities, which usually focuses on different aspects of a product and is performed by people with different areas of expertise using a variety of engineering applications tools. Opportunities for adaptive design analyses have been classified into two categories:

1) *Composition adaptability***:** The software world today is one of great diversity. Thousands of software products are available to users today, providing a wide variety of information and services in many different domains. A diverse array of design and analysis tools and software systems are available to compose an ensemble that supports collaboration for any design analysis. Although a number of methods have been proposed and implemented by various researchers in an effort to integrate engineering applications,

these methods all propose to integrate the data for a fixed set of applications. However, different design analyses require different combinations of tools. In addition, new systems may be added after the initial integration in order to provide enhanced support for a particular analysis. For example, an evaluation scenario may consist of the design engineer at one location using an immersive application to communicate with an agronomist using a human factors analysis application at a different location. This collaboration must allow the two designers to work on the same product using the application of their choice, while the software tools communicate. Getting programs to work together often necessitates extensive work by users and developers.

2) *Viewpoint adaptability*: A product is considered from different viewpoints and perspectives, and different representations and terminologies may be used in each viewpoint. For example, consider an automobile interior design evaluation. One viewpoint may focus on ergonomic comfort and accommodation while another may focus on assembly simulation for the dashboard. The representation of the product, the terminologies used, and the communication of the results of these analyses vary greatly, though the underlying product is the same.

## 4.2 Integration Framework Overview

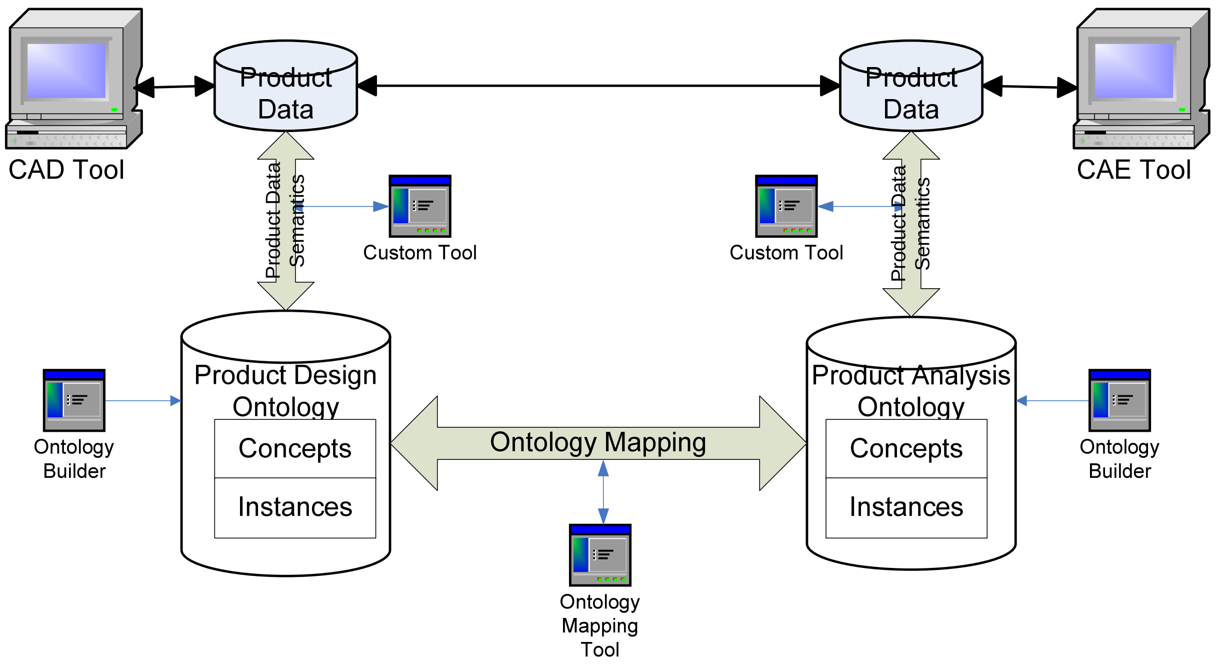

**Figure 4 - 1 Structure of Integration Framework**

In the framework, there exist the following components:

1) **Ontology Builder**, used to build ontologies for different engineering applications. Viewpoint-specific knowledge including concepts and their relationships are captured using an Ontology Builder.

2) **Ontology mapping tool**, used to translate knowledge associated in one specific domain to knowledge in another.

3) **Custom tools**, used to connect the engineering applications to the knowledge by creating an instance layer (product data semantics) to link actual data to the concepts in ontology.

The framework also contains an ontology implementation, which contains concepts about the knowledge in various areas, including design ontology and other viewpoint-specific ontologies. Product ontology defines the concepts of a product design which mainly includes the definitions of functions, behaviors and forms, and properties that necessarily describe the product. Viewpoint-specific ontology defines the concepts from specific viewpoints (e.g. ergonomics; assembly simulation). Relations between these ontologies exist so one ontology can be mapped to another. We focus on Product Design and Product Analysis in this dissertation, with the former being the source and the latter being the target.

## 4.3 Working Mechanism of Integration Framework

In the design process, product information can be described in terms of product concepts and product data. For example, an automobile dashboard can be described using concepts of Panels and Gadgets and their product data such as geometry, sizes, positions etc. Our overall solution for exchanging product data semantics is to use ontology to interpret product data, and then use ontology mapping to translate concepts into other viewpoints (different terminologies), so that the interpretation of product data can be semantically translated.

Heterogeneous applications implicitly use different knowledge bases to interpret product data. To improve the interoperability, this knowledge needs to be explicitly built into engineering ontologies as concepts and their relations. To associate the knowledge in engineering ontologies to product design, instances of concepts are instantiated with the product data. Since in different knowledge bases the concepts and relations vary significantly, through mapping between ontologies, the concepts used in a source ontology can be translated into a target ontology, and then a new instance can be created that corresponds to the translated concept in the target ontology. Thus, the product data representation in one application can be translated into another.

The procedures are as follows:

- **Capturing Product Data Semantics by defining relevant engineering ontology concepts and instantiating these concepts with specific product data**

1) Define Product data semantics by building engineering ontologies for both product design and product analysis

  - ▪ General Design Ontology (GDO) is used as a shared upper-level ontology. The commonly accepted concepts used by all the design, analysis and other domains to describe a product design are collected into this ontology. For example, to describe a design, three types of attributes, function, behavior, and form are used in the ontology. For example, in the function layer, functions of a dashboard can be described as providing dynamic feedback, control, and storage, and each function can have several sub-functions: providing visual feedback, scale plate to display text, and scale.
  - ▪ For the product design (source), a Product Design Ontology (PDO) that includes a domain-specific ontology layer for assembly design domain and an application-specific ontology layer for Pro/Engineer is created. For example, if an assembly is designed

by using Pro/Engineer, based on GDO, PDO uses concepts and terminology used assembly design domain and also the specific concepts that only used in Pro/Engineer to describe the design process, such as concepts of constraint and specific constraint types.

- On the other hand, for the design analysis (target), a design analysis ontology is built to describe the concepts; for example, an Assembly Simulation Ontology (ASO) defines the concepts of assembly simulation and their relations, such as Assembly constraint, joint, assembly hierarchy, etc.

2) Associate product data semantics to real product data by instantiating concepts in source ontology

A product is first defined using the concepts in PDO, and, according to the concepts, parametric product data is extracted from the product database. A file consists of ontology instances in OWL is created as product data semantics, which instantiates the design concepts with the actual data.

- **Translate Product Data Semantics by defining Ontology Mapping**

In both the domain of 3-D parametric feature-based design and the constraint-based assembly simulation domain, different concepts are used to describe a product design from different perspectives. However, these concepts can be translated as they eventually relate to the same information of a product design; certain concepts of a product design (for example, information in feature about geometry) in PDO can be related into concepts in ASO (geometry, material, etc.). A method is developed to calculate the similarities between these concepts, thus helping to map one concept in product design ontology to a similar concept in assembly simulation ontology.

By identifying and translating concepts that refer to the same product design information between the two ontologies, instances of concepts in PDO and their attributes are extracted from the previous product data semantics and translated into new product data semantics which consists of instances of new concepts and attributes in ASO Tthe new product data semantics described by concepts in ASO can be easily understood by assembly simulation tool and eventually will help original product data to be converted into a data representation that assembly simulation tool can read. An intermediate file is used to record the mapping information.

The process is illustrated in Figure 4-2:

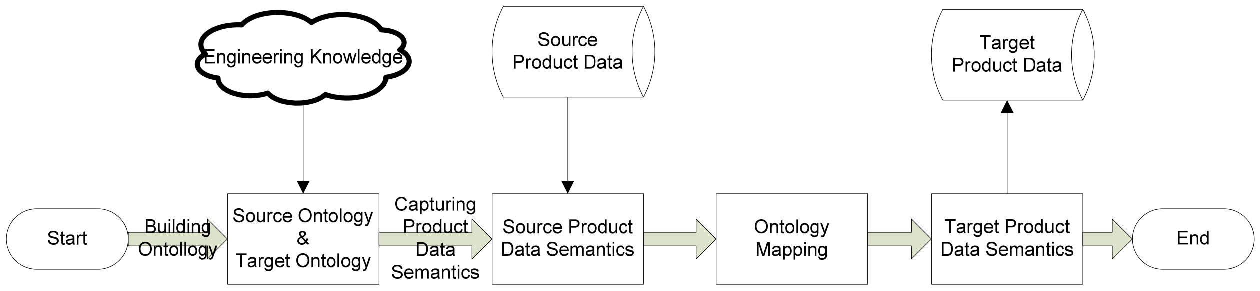
Engineering Knowledge
Source Product Data
Target Product Data
Start
Building Ontollogy
Source Ontology & Target Ontology
Capturing Product Data Semantics
Source Product Data Semantics
Ontology Mapping
Target Product Data Semantics
End

**Figure 4 - 2 Working Mechanism of Semantic Level Integration**

## 4.4 Benefits of the Framework

- 1) Interoperability: By defining ontologies from different viewpoints and building mapping between them, a product design including design intent and history can be captured and translated to different applications. A design/analysis result described using terminology of a certain viewpoint can be translated into terminology of another. Further, interoperability is enhanced by defining ontology as an intermediate data model, so that product data may be translated into or from the data model before it is exchanged between CAD/CAE applications.
- 2) Reusability: A product data model is built into concepts and product data semantics files separately. This representation, particularly the concepts, can be reused and shared.
- 3) Scalability: the layered structure of engineering ontology makes it easy to add new ontologies to the existing framework for new integration, allowing for adaptability in composition.

# Chapter Five - Defining Product Data Semantics Using Engineering Ontologies

## 5.1 Introduction

During CAD/CAE integration, heterogeneous applications use different knowledge to represent product data. Different terminologies and data types are used. As a result, the product data semantics generated in one application in a certain domain cannot be understood in other applications.

Product data semantics is defined here as an "understanding about the data of a product design based on certain engineering design/analysis knowledge". A product is considered from different viewpoints and perspectives, and different representations and terminologies may be used in these different viewpoints. Hence there are different product data semantics. For example, consider an automobile interior design; one viewpoint may focus on ergonomic comfort and accommodation while another may focus on assembly simulation for the dashboard. The representation of the product, the terminologies used, and the communication of the results of these analyses vary greatly, though the underlying product is the same.

This chapter presents an overall method of capturing product data semantics to be utilized in a CAD/CAE integration framework. In contrast to traditional integration framework where information exchange is primarily limited to a pure data level, our process deals with both concepts and instance data. It allows the discovery of mapped concepts between source and target, and then the translation of *instance data from the* *source*, which is originally in terms of the source concepts, to *instance data in the target*, now in terms of the target concepts that were discovered and specified in the mapping.

In this dissertation, a method of capturing product data semantics is created in two steps:

1) Build engineering ontologies for each specific application, which includes three layers: general design ontology layer, domain-specific layer and application-specific ontology layer; and

2) Create interpretation about product data by associating instantiating concepts with product data

## 5.2 Methodology of Building Engineering Ontologies

Methods of building engineering ontologies have been explored by many researchers. In this dissertation, in addition to the existing process of building engineering ontologies summarized [58], the following steps are used:

1) Specify the required taxonomies for general design;

2) Specify axioms for general design;

3) Specify the required taxonomies for specific engineering design/analysis domain;

4) Specify axioms in the specific design/analysis domain;

5) Specify the required taxonomies for specific application;

6) Specify axioms for specific application;

7) Identify existing taxonomies that can be used to fulfill the specifications;

8) Create new taxonomies if none exist.

## 5.3 Define the classes and their relations

As a powerful ontology editor and knowledge acquisition system, Protégé is used to define knowledge in ontologies. Figure 5-1 shows an example of classes and class hierarchies for product design in Protégé.

Based on concept definitions, relations between concepts are defined by using a basic expression in ontology that described in OWL/RDF as a Subject-Property-Object triplet form, which means a subject concept has an object concept as its property.

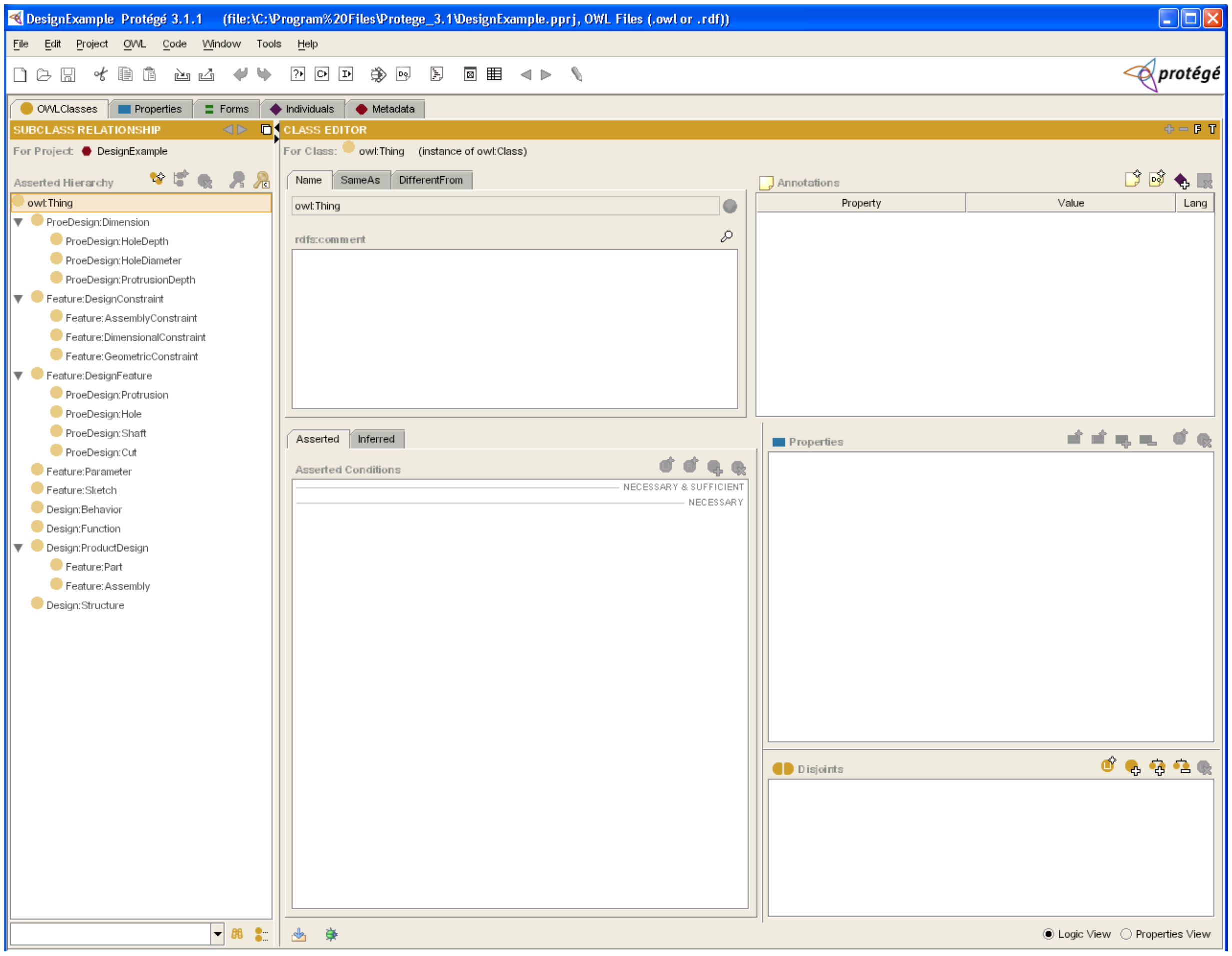

**Figure 5 - 1 Class Hierarchy in Protégé**

## 5.4 Layered Structure of the Ontology in Semantic Web

In this dissertation, methods from semantic web technologies are used to model engineering ontologies. The general structure of the semantic web layer structure developed by Tim Berners-Lee, as shown in Figure 5-2 [59], presents the fundamentals of the system. It consists of new web languages such as metadata languages (XML/RDF etc.) and logic languages, which can be used to describe rules, proofs and logic[60]. In this dissertation, we have built an infrastructure of ontology that is similar to Figure 5-2, but simplified. It has the layers shown in Figure 5-3. Each layer is incrementally built on the previous layer and incorporates more specific value into the whole architecture. A description of the individual layers is as follows:

1) The *Unique Resource Identifier* (URI) layer provides the capability to identify resources uniquely and globally.

2) The *XML layer* enables the data interoperability and the syntactic interoperability by defining the data format. It provides syntax for formatting and tagging data. XML documents are more flexible and easier to understand compared to other existing formats, such as HTML. Together with XML schema and other tools, data exchange is made much easier.

3) The *RDF/RDFS layer* uses the triplet concept of Subject-Property-Object as the foundation of the engineering ontology to describe the basic object relationships.

4) The *OWL layer* provides semantic interoperability by further enabling describing objects and their relations, and reasoning on them.

5) The *RULE layer* overcomes the expressive restriction of OWL and describes more complex relationships. In this dissertation, Semantic Web Rule Language (SWRL) is used as the RULE language.

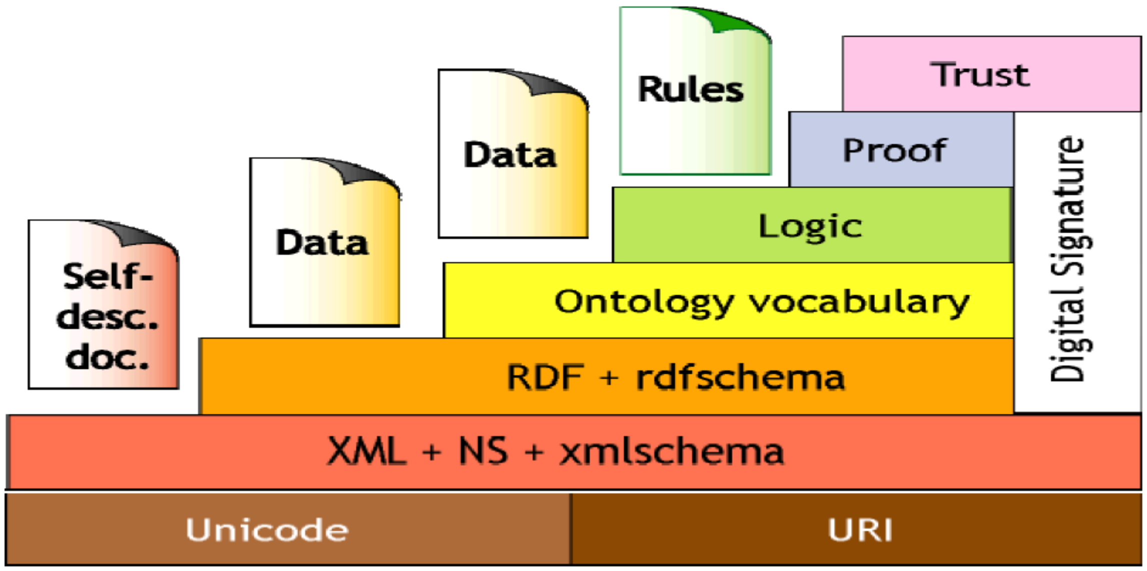

**Figure 5 - 2 W3C Semantic Web Layer Cake**

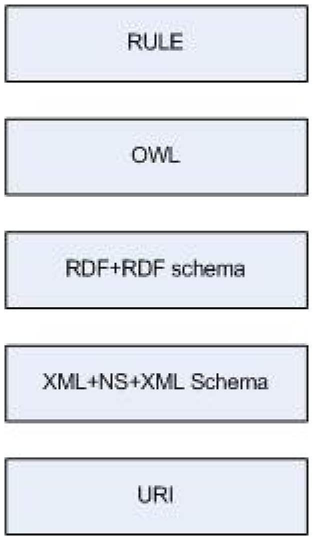

**Figure 5 - 3 Structure of Engineering Ontology**

## 5.5 Layered Structure of Engineering Ontology

In our integration method, product data semantics is built upon product data. Product data semantics is represented in two parts: knowledge base consisting of ontology concepts and their relationships; and instances of concepts that instantiated from the product data. The knowledge base describes the general and basic knowledge in different domains using concepts, relations, properties and axioms in engineering ontologies. Instance layer consists of instances which instantiate concepts in engineering ontologies, which then represent each piece of the design information in real product data. Instance layer plays as a role linking the knowledge to the actual product data. Their relationship is depicted in Figure 5-4.

An ontology is composed of the following:

1) Knowledge Representation using concepts

2) Instance data based on instantiation of concepts

Within knowledge representation are contained:

1) Axioms to state the facts in an ontology

2) Inference Rules

3) Concepts/Classes

4) Relations

5) Slots/Roles/Properties

6) Facets

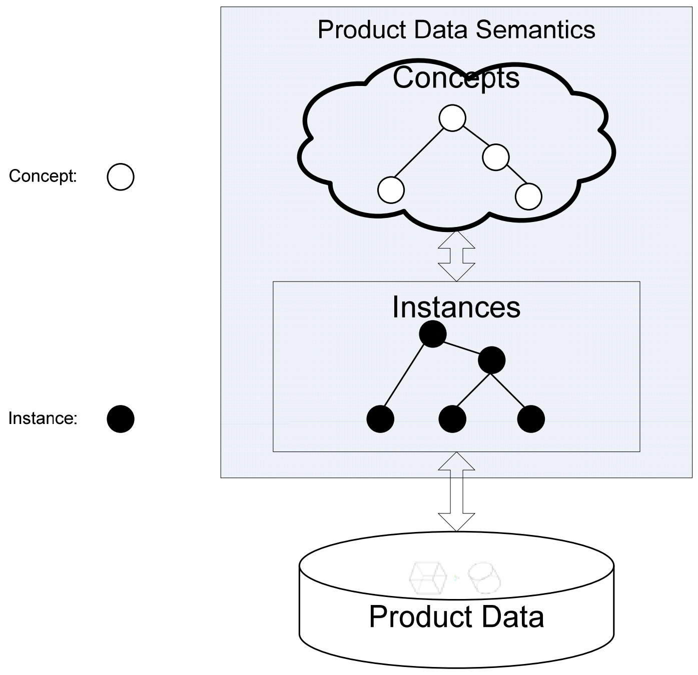

**Figure 5 - 4 Relationship between Ontology and Product data**

A layered structure of ontology as shown in Figure 5-5 is used to build engineering ontologies in the product design/analysis domain: The three-tier ontology structure includes general design ontology, domain-specific ontology, and application-specific ontology. Each layer is built upon the previous one.

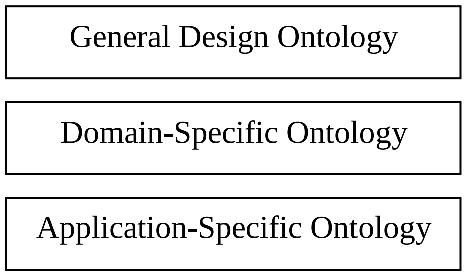

**Figure 5 - 5 Engineering Ontology Layered Structure**

1) **General design ontology** describes basic elements that represent commonalities in the product, which can be applied to any domain in design/design analysis. It includes the following commonly accepted concepts: 1) Basic attributes to describe a design, such as form-function-behavior triple concepts. 2) International standards about products, such as material names, properties, etc. 3) Common terminology, such as geometry primitives, mathematical terms, etc.

In Subject-Property-Object triplet expression, property is used to describe the relations between concepts, in general design ontology, basic properties are defined in this layer: *is-a*, *part-of; attribute-of; external-reference*, which can be used as base properties so other more specific properties can be inherited from, for example, *is_geometry_attribute_of* property is based on *attribute-of* property And *has_feature/is_feature_of* are inherited from *part-of* basic property to indicate the relation between concepts *feature* and *component*

• *Is-a* reflects the relation that one concept is a sub-concept of another, which is a common relation existing in ontology concepts definition and is a built-in type in RDF/OWL.

• *Part-of* , as a non-predefined relationship in RDF/OWL, defines ownership and is introduced as a basic property because in CAD/CAE product data, *part-of* is a very common relation. In this dissertation, we use has_Child/has_Parent to represent this basic property..

• *Attribute-of* defines the relations between an object and its attributes. *Attribute-of* property can have further sub-properties, such as hasFunctionAttribute, hasBehaviorAttribute, and hasFormAttribute, which describe the relations between a ProductDesign and its three basic attributes.

Regarding attribute-of and part-of relations, we define that the attribute-of relation can be transitive based on part-of relation, which means if A is part of B, then B will also have all the attributes that A has. A rule can be defined:

**Rule 5.5.1 for three concepts C1 and C2 and C3, if C1 has_Child C2 and C2 has_Attribute C3, then it can be deducted that C1 has_Attribute C3.**

For example, according to the rule, if a component C has a feature F, and geometry attribute A is an attribute of feature F, then attribute A is also an attribute of component C.

• *External-reference* defines the relation in which one instance is related to another other than *is-a*, *part-of* and *attribute-of* relations. For example, Part A is constrained to Part B in an assembly, the *constrained_to* relation is an *external-reference* relation type which doesn't fall under *is-a*, *part-of* and *attribute-of* relations

Figure 5-6 is part of RDF/OWL sample code that defines concepts and relations.

```
<owl:Class rdf:ID="Function"/>
  <owl:Class rdf:ID="ProductDesign"/>
  <owl:Class rdf:ID="Behavior"/>
  <owl:Class rdf:ID="Geometry"/>
  <owl:ObjectProperty rdf:ID="hasBehaviorAttribute">
    <rdfs:domain rdf:resource="#ProductDesign"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="hasFunctionAttribute">
    <rdfs:domain rdf:resource="#ProductDesign"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="hasGeometryAttribute">
    <rdfs:domain rdf:resource="#ProductDesign"/>
  </owl:ObjectProperty>
```

**Figure 5 - 6 OWL/RDF Sample Code of Concepts and Relations**

2) **Domain-specific ontology** describes elements in a particular domain. Examples of domains include 3-D parametric feature-based geometric modeling, constraint-based assembly simulation, and product design ergonomics analysis. Each domain has concepts and relations described in this layer that are widely accepted by those work in this domain, but those concepts might not be recognized in other domain-specific ontologies. Even if two domain-specific ontologies use the same terminologies, they may have different meanings. For example, in a 3-D parametric feature-based geometric modeling domain, the concept of feature is commonly understood and used by designers, regardless of what kind of specific feature-based CAD tools are used. However, in Computer Aided Manufacturing domain, the concept "feature" connotes different information than the concept "feature" in the design domain, since the latter is targeted at the manufacturing process and may imply manufacturing information.

3) **Application-specific ontology** describes knowledge in a specific application. Even though in the same domain, different applications use different terminologies and concepts and their ways of representing the structure of design are also different. For example, some feature concepts being used to describe the structure of a product in CAD applications such as Pro/Engineer are different from concepts and terminologies used in other CAD applications, like Catia, Solidworks and Unigraphics. For example, the concepts BaseExtrude and BossExtrude in Solidworks correspond to the concept Extrusion in Unigraphics[34]. Another example is the fact that every system has its own way of defining a constraint; for the same type of constraint, Pro/Engineer uses *Mate* while Catia uses *Contact.*

## 5.6 Benefits of Layered Ontology Structure

The layered structure of the ontology structure ensures the extensibility of the framework. For any new applications being included in the framework, the ontology that describes its knowledge base can be easily created and plugged into the ontology framework since there is a mechanism for considerable reuse and also openness to bring in the unique and different elemnts of the new application. For example, an ontology to represent knowledge of product design based on application Solidworks can reuse structures from the existing general design ontology layer and domain ontology layer of 3d feature based design that could have been created for an application using Pro/Engineer.

Additionally, the layered structure of ontology has the powerful advantage of limiting the ontology mapping effort to a local level and thus, improves its efficiency. For two different applications that fall under the same domain and share the same domain ontology, the ontology mapping only needs to happen on the application level. Sharing same concepts in upper-level ontologies also makes it easier to discover similar concepts in the lower-level ontologies.

# Chapter Six - Key Concepts in General Design Ontology

In this dissertation, we create a general design ontology and build the fundamentals of product design that is acknowledged in design/analysis domains. In addition to the concepts about the final product design, such as product form and materials, additional ontology concepts are developed to capture the design intent and design history in order to improve communication between design and other analysis processes.

This chapter discusses key concepts in a General Design Ontology with a focus on necessary concepts that will allow later for the integration of sample applications/domain-specific ontologies for a CAD system,Pro/Engineer, and an Assemby Simulation Environment.

## 6.1 Basic Design Entity

A design process can be defined as a process of collecting information related to product data, and the information collected can be divided into many units; each unit can have its own attribute and can be independent of the other units. In this dissertation, we define the basic element of design information that an engineer uses to describe a product design as Basic Design Entity (BDE). BDE can have following characteristics:

1) A BDE normally has three basic attributes: form attribute, behavior attribute and function attribute. In most of the cases a BDE has at least a form attribute that relate to the geometry.

2) A BDE is used as a basic information unit during design/analysis.

3) A BDE is independent of other BDEs, but there are normally interactions between them.

Examples of BDE include the Feature concept in 3-D feature-based design and feature-based manufacturing, concepts of part, assembly, and joint in assembly design/simulation. As we can see, in different design/analysis domains, design entities could differ and have different attribute definitions.

## 6.2 Attributes of a Basic Design Entity

In general design ontology, function-behavior-form triplet concepts are used to describe a product design process. All attributes of a Basic Design Entity are categorized as 1) FormAttribute; 2) BehaviorAttribute; and 3) FunctionAttribute.

It is commonly accepted to use form-behavior-function to describe a product design attribute. For a design artifact, form is used to represent physical properties such as structure, geometric shape, and material, and behavior is used to specify the response under certain input conditions[18]. However, the use of these definitions is limited to the final result of a product design. As a result, much information generated during the design process cannot be represented by the attributes, such as design intent and design history.

In our semantic integration, in order to enhance communication between applications at the semantic level, it is desirable to be able to exchange the information about how design/analysis is completed and why it is designed/analyzed in that way.

Further, if this information is captured, it can be also used for ontology mapping to discover matching concepts, as discussed in later chapters.

To represent design process information in design ontology, in addition to being used to describe attributes of a final product design result, function-behavior-form concepts are extended to describe the information generated **during** product design/analysis process, specifically by extending behavior and form attributes. For example, information generated during design process such as how a product is created, the creation/manufacturing order etc.

### 6.2.1 Form Attribute

In General Design Ontology (GDO), a geometry ontology is built as a subset of form attribute ontology to represent geometry information of product design, where geometry is divided into the following categories: 1) *Point*; 2) *Curve*; 3) *Surface*; and 4) *Solid*.

In this dissertation, we extend form attribute to describe any form attributes that are used during the design process. In other words, even those that do not become part of the final design and are only used as intermediate information in the design process are built into ontology, such as 2d profile and datum plane which are used as auxiliary information during 3d feature creation.

According to its use, geometry can be also categorized into functional geometry and auxiliary geometry. Functional geometry is used to describe geometry that supports functionality of design, for example, the shape of a shaft to support the function of a transmitting torque. Auxiliary Geometry is used to describe the geometry used during the design process that may not be part of the functional geometry and is only used as auxiliary information, such as datum plane, datum axis, feature profile, etc.

### 6.2.2 Behavior Attribute

Normally, a behavior attribute is closely associated with the final function of a design unit. Simply put, a behavior attribute is used to describe the behaviors of a product design in order to accomplish certain functions. There are different kinds of behavior, such as motions, deformation, etc.

In this dissertation, this traditional description of behavior, as described above, is defined as *functional behavior*. For example, Assembly Constraint Behavior is used when two parts (or components) are combined to form an assembly, which could be described using terms of motion (or degree of freedom): Translation in x, y, z directions, and Rotation in x,y,z directions.

However, in addition to functional behavior, we also want to capture those behaviors that occur during the design process, which can be used to present information such as design intent and history.

Under the design behaviors, we identified at least the following two sub-classes: 1) *Design Constraint*, *Behavior* such as parametric constraint, to describe the mathematical relationships between parameters; and 2) *Design Creation Behavior* to describe the way a 3-D feature is created,  such as by extruding or revolving a 2-D profile.

For example, in Pro/Engineer, a 3-D feature is usually created by a design creation behavior in a 2-D geometry (profile).  A geometric shape, such as a cylinder, can be created in one of two ways: 1) A rectangle profile revolve around an axis; or 2) A circle profile extruded in the direction of the normal of the circle profile, as depicted in Figure 6-1 and Figure 6-2.

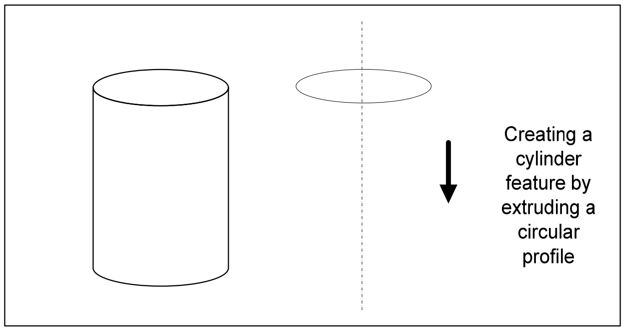

**Figure 6 - 1 Feature created using Extrude behavior**

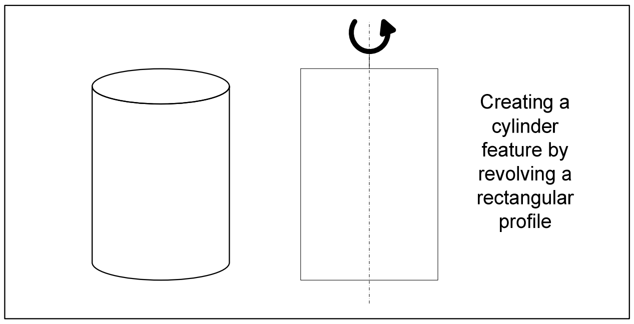
Creating a cylinder feature by revolving a rectangular profile

**Figure 6 - 2 Feature created using Revolve behavior**

To represent the design intent captured in the design process, design creation behaviors are used along with Profile concepts; code in Figure 6-3 depicts the representation of a feature created in Figure 6-1, in which  an extrude behavior is used on a circular profile.

```
        <Circle rdf:ID="circle_1">
            <has_center="#point_1">
            <has_diameter=5.0>
        </Circle>
        < Profile rdf:ID="Profile_1">
          <has_geometry_attribute rdf:resource="#circle_1">
        </Profile>
        <ExtrudeFeature rdf:ID="extrude_1">
          <has_design_creation_behavior
        rdf:resource=Extrude_z>
          <has_profile="#Profile_1"/>
          <has_extrude_depth=10.0/>
        </ExtrudeFeature>
```

**Figure 6 - 3 OWL/RDF Code of Extrude Feature**

Code in Figure 6-4 depicts the representation of the feature created in Figure 6-2, a rotate behavior on a rectangular profile.

```
        <Axis rdf:ID="axis_1">
          <has_start_point rdf:resource="#point_2"
          <has_end_point rdf:resource="#point_3"
        </Axis>
        <Rectangle rdf:ID="rectangle_1">
          <has_width=5.0>
          <has_height=10.0>
        </Rectangle>
        < Profile rdf:ID="Profile_2">
          <has_geometry_attribute
        rdf:resource="#rectangle_1">
        </Profile>
        <RevolveFeature rdf:ID="revolve_1">
          <has_design_creation_behavior
        rdf:resource=Revolve_z>
          <has_profile="#Profile_2"/>
          <has_axis=10.0/>
        </RevolveFeature>
```

**Figure 6 - 4 OWL/RDF Code of Revolve Feature**

Function attribute is another important attribute.  However, in this dissertation it is not considered and is not the focus of this work.

# Chapter Seven - Building Application-Specific Ontologies

This chapter discusses in detail the methods of building application specifc ontologies for the product design/assembly simulation based on the engineering ontology structure and the General Design Ontology concepts discussed in the previous chapters. . Specifically, it will focus on building two ontologies: one to support *Product Design* using Pro/Engineer, and the second to support *Assembly Simulation* using a virtual assembly environment called VADE. Later sections discuss in detail how product data semantics are captured using these ontologies.

## 7.1 Product Design Ontology

### 7.1.1 Foundational Domain-specific Ontologies

As a commercial product design package, Pro/Engineer covers different engineering domains. Its functionality in geometric modeling is based on the technology of feature-based parametric design, and it also facilitates assembly design using spatial constraints. To build ontology for Pro/Engineer, we first need to build *domain ontologies for parametric feature-based design and assembly design*, which are the common concepts used in these domains, so the concepts specific to Pro/Engineer can be built upon them. An investigation was performed to find some of the most common and fundamental general concepts in feature based parametric design.

In the *parametric feature-based design domain*, we identify the following concepts: 1) *Feature*; 2) *Parameter*; 3) *Design Constraint*; 4) *Feature_Tree* 5) *Feature* *element*; Figure7-1 shows the relations between the above concepts. Each link indicates *ownership* relationships.

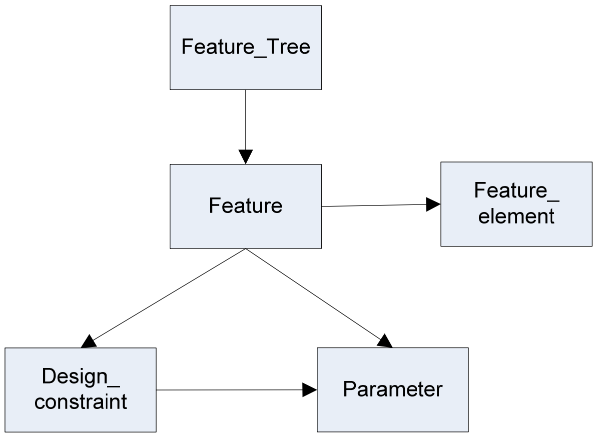

**Figure 7 - 1 Composition Relations in feature based parametric design domain ontology**

In the *assembly design* domain, we identify the following concepts: 1) *Assembly_Hierarchy*, an orderly hierarchy of the subassemblies in an assembly; 2) *Assembly*; 3) *Component*; 4) *Component Constraint*; and 5) *Degree of Freedom*. Their relations are illustrated in Figure 7-2.

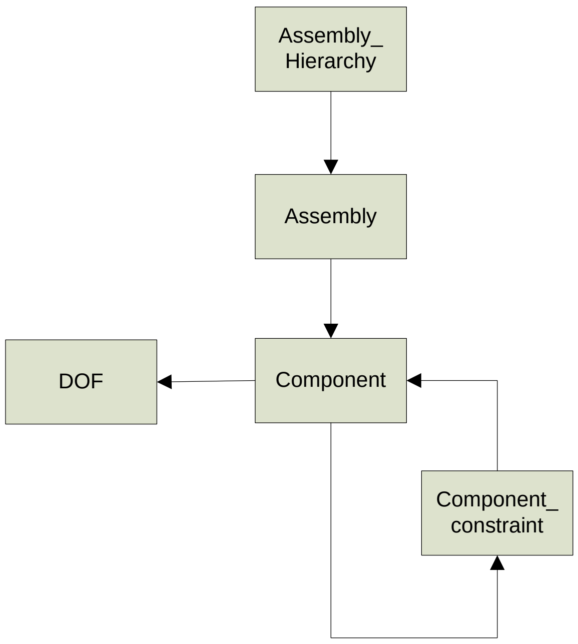

**Figure 7 - 2 Composition Relations in assembly design domain ontology**

### 7.1.2 Application-specific Ontology for Pro/Engineer

In the application-specific ontology for Pro/Engineer, we build upon concepts/terminologies that are used in *parametric feature-based design* and *assembly design*.

**Extending Concepts from Parametric Feature Based Design Domain**

Extending the feature concept in the Parametric Feature Based Design Domain, we identified two basic types in the application-specifc ontology for Pro/Engineer: Datum and Solid; Datum has the following sub-types: 1) *Datum_Coordinate_System*; 2) *Datum_Curve*; 3) *Datum_Plane*; and 4) *Datum_Point*. For a solid feature, there exist the following sub-types: 1) *Chamfer*; 2) *Extrude*; 3) *Fillet*; 4) *Hole*; 5) *Revolve*; 6) sweep and 7) *Round*.

In Pro/Engineer, a feature is defined to consist of feature elements, such as *Edge*, *Surface* , *Point*, and *Axis*, which can be regarded as its form attribute. Also behavior attribute based on concepts in GDO can be defined for each feature type.

## Extending Concepts from the Assembly Design Domain

Since, as mentioned before, the Pro/Engineer application is also based on the Assembly Design Domain, concepts in the Assembly Design domain are now similarly extended for the application specific ontology. For example, in the domain of Assembly Design, Pro/Engineer uses concept of *component constraint* to restrain the degree of freedom of each part in an assembly. Specifically, following component constraint types are used: 1) *Align*; 2) *Align_Offset* (align with offset); 3) *Coord_Sys*; 4) *Edge_On_Srf*; 5) *Insert*; 6) *Mate*; 7) *Mate_Offset* (Mate with Offset); 8) *Pnt_On_Line*; 9) *Pnt_On_Srf*; and 10) *Tangent*. Each component constraint may have one of the following parameters: *coincident* (Boolean), *offset* (float point), or *angle* (float point). Figure 7-4 shows the taxonomy of Component Constraint defined in Pro/Engineer.

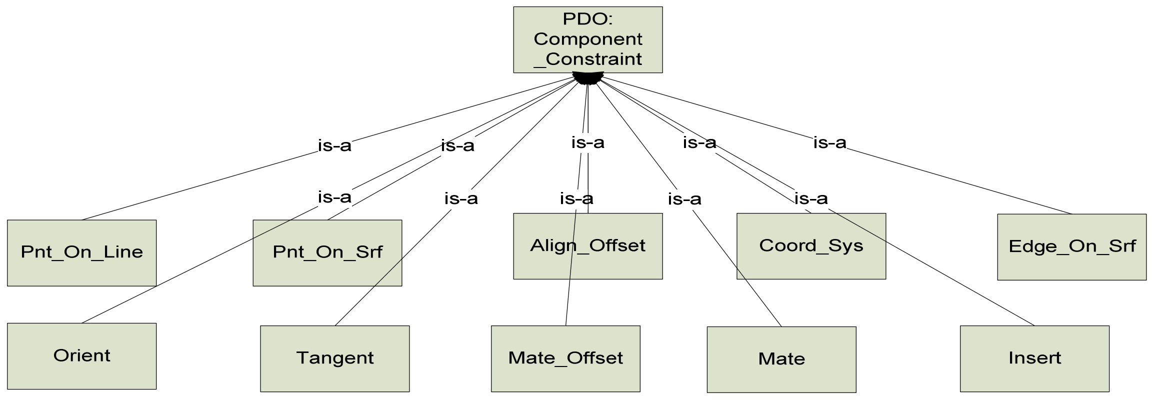

**Figure 7 - 3 PDO Component_constraint Taxonomy**

In addition to the concept definitions, there is knowledge regarding some facts about the component constraint. For example, when a constraint type is specified, it is required to specify on which geometry the component constraint is implemented. The user is asked to select a model item of a part; the valid model item types are plane, axis, point, edge, etc. According to the type of component constraint, the valid model item type could vary. The model item of the base part is defined as *assembly reference*, and the model item of the part that is to be assembled is defined as *component reference*.

. For example, in a component constraint with a type of *Pnt_On_Line,* the restriction of the selected model item of one part should be a point, and the selected model item of the other part should be a line. This information can be represented using axioms in Product Design Ontology:

**Axiom1**: An instance of *Component_Constraint* has at least one *assembly_reference* and one *component_reference,* which can be represented in the following restriction:

≥ *has_assembly_reference* min 1

≥ *has_component_reference* min 1

**Axiom2**: An instance of *Component_Constraint* with type *PNT_ON_LINE*, must have an *assembly_reference* and a *component_reference*, being one *Point* and one Line-type model item such as *Axis* and *Edge*, which can be represented in the following restrictions:

∀ *has_component_reference* only (*Axis* or *Edge* or *Point*)
∀ *has_assembly_reference* only (*Axis* or *Edge* or *Point*)

The OWL source code to describe the axiom is as shown in Figure 7-3:

```
            <owl:Restriction>
              <owl:onProperty>
                <owl:ObjectProperty rdf:ID="has_component_reference"/>
              </owl:onProperty>
              <owl:allValuesFrom>
                <owl:Class>
                  <owl:unionOf rdf:parseType="Collection">
                    <owl:Class rdf:about="#Axis"/>
                    <owl:Class rdf:about="#Edge"/>
                    <owl:Class rdf:about="#Point"/>
                  </owl:unionOf>
                </owl:Class>
              </owl:allValuesFrom>
            </owl:Restriction>
          </rdfs:subClassOf>
          <owl:disjointWith rdf:resource="#Insert"/>
          <owl:disjointWith>
            <owl:Class rdf:about="#Mate"/>
          </owl:disjointWith>
          <rdfs:subClassOf>
            <owl:Restriction>
              <owl:onProperty>
                <owl:ObjectProperty rdf:ID="has_assembly_reference"/>
              </owl:onProperty>
              <owl:allValuesFrom>
                <owl:Class>
                  <owl:unionOf rdf:parseType="Collection">
                    <owl:Class rdf:about="#Axis"/>
                    <owl:Class rdf:about="#Edge"/>
                    <owl:Class rdf:about="#Point"/>
                  </owl:unionOf>
                </owl:Class>
              </owl:allValuesFrom>
            </owl:Restriction>
```

**Figure 7 - 4 Sample OWL/RDF Code of Axioms**

Similarly, we have following the axioms for other constraints:

**Axiom3**: An instance of *Component_Constraint* with Type *INSERT* must have an *assembly_reference* and a *component_reference*, being cylindrical surface, which can be represented in the following ways:

$\forall$ *has_component_reference* only *(cylindrical_surface*)
$\forall$ *has_assembly_reference* only (*cylindrical_surface*)

**Axiom4**: An instance of *Component_Constraint* with Type of *ALIGN* must have an *assembly_reference* and a *component_reference* being *Axis*, *Edge* or *Surface*, which can be represented in the following ways:

∀ *has_component_reference* only (*Axis* or *Edge* or *Point*)

∀ *has_assembly_reference* only (*Axis* or *Edge* or *Point*)

And more similar axioms can be written for other types of constraint**.**

## 7.2 Assembly Simulation Ontology

After we choose an assembly simulation tool, an Assembly Simulation Ontology is built based on its knowledge. In this dissertation, we use Virtual Design Assembly Environment (VADE) developed in VRCIM Lab, Washington State University as our assembly simulation tool.

### 7.2.1 Domain-Specific Ontology for VADE

As VADE belongs to constraint-based assembly simulation domain, we identify the following concepts in this domain: 1) *Assembly*; 2) *subassembly* 3) *Part*; 4) *Joint*; and 5) *Assembly Constraint*.

Concepts *Assembly*, *Subassembly* and *Part* are used to represent the basic components in assembly simulation.

Joint is defined as an aggregation of all Assembly constraints between two parts which represent total interfaces between two parts that interact with each other in an assembly.

*Assembly constraint* is used to represent the concept of interface between two assembled parts in our Assembly Simulation Ontology (ASO). In an assembly, one part interacts with another through assembly constraint, and each assembly constraint is implemented based on assembly features to reduce the degree of freedom of each part. The information about each interface, including the Assembly Feature, Assembly Parameter, Behavior Attribute (Degree of Freedom), is defined as *Assembly_Constraint*.

1) **Assembly Constraint**

Assembly constraint is the spatial constraint that limits the motion of a part in an assembly.

2) **Assembly Features**

As a sub-property of Form Attribute, Assembly features refer to those geometric features involved in assembly interaction, or those features on which assembly constraint is implemented. In our Assembly Simulation Ontology (ASO), assembly features can be one of the following basic types 1) *Point*; 2) *Curve*; 3) *Surface*;

3) **Assembly Parameters**

Assembly parameters in Assembly Simulation Ontology refer to those key parameters that must be satisfied in order to meet the assembly constraint specification, for example, *Offset* and *Angle*.

4) **Behavior Attribute of Assembly Constraint**

In the previos chapters, we use function-behavior-structure triple concepts to describe Basic Design Entity. As an important BDE in assembly design/simulation ontology, assembly constraint uses Degree of Freedom as its function behavior.

### 7.2.1 Application-Specific Ontology for VADE

In the application level, specific types of assembly constraints are defined: 1) *Align*; 2) *Coincidence*; 3) *Contact*; 4) *Fix*; and 5) *Offset*. Figure 7-5 shows the taxonomy of the assembly constraint

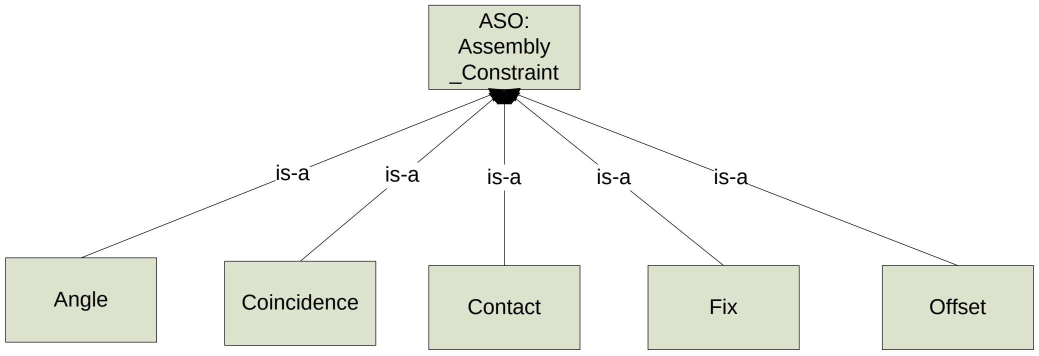

**Figure 7 - 5 ASO Assembly_Constraint Taxonomy**

Also, axioms are defined in the application-specific ontology level to specify restrictions and conditions that need to be satisfied for each of the constraint types, for example, for certain assembly constraints, following restrictions must be followed:

**Axiom5**: *Assembly_Constraint* has at least two *assembly_feature*, which can be expressed as following restriction:

$\geq$ *has_assembly_feature* min 2

**Axiom 6** *Contact* constraint must have planes as its assembly features, which can be expressed as follows:

$\forall$ *has_assembly_feature* only *(plane*)

**Axiom 7** *Align* constraint must have either lines or planes.as its assembly features, which can be expressed as follows:

$\forall$ *has_assembly_feature* only *(line* $\cup$ *plane*)

## 7.3. Capturing Product Data Semantics for product data

In relation to product data, product data semantics refers to the "meaning" of data. After concepts are built into the engineering ontologies, these concepts need to be instantiated with data from the actual product. Product data semantics is generated by associating each product data to its "meaning" and this is done by instantiating concepts with product data.

### 7.3.1 Methodology of Capturing Product Data Semantics

*The process of capturing product data semantics can be regarded as a process of instantiating concepts with instances*. To capture product data semantics, according to the concepts from a viewpoint, and 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 from the application, information about the product structure can be obtained by calling the API to build the instances. After the pre-built ontology is created, based on the existing concepts, an application is built to automatically extract the product information from the CAD database for instances.

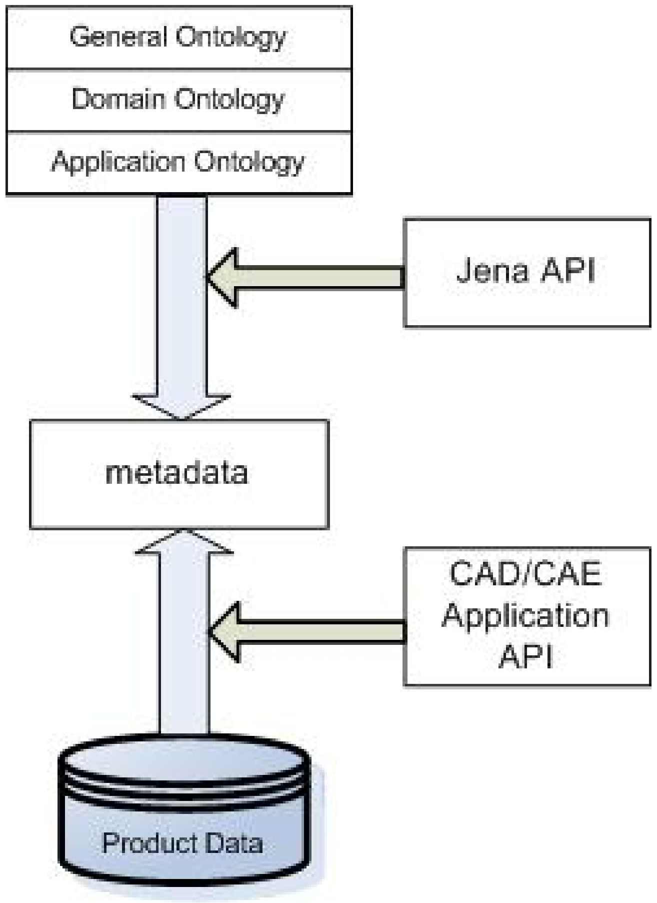

**Figure 7 - 6 Programming API to link ontology to product data**

### 7.3.2 Interpreting Product Data Using Product Data Semantics

Product data semantics describes the structure of product data using predefined concepts, instances, and their relations. Instances of concepts are used as main components in product data semantics to link product data to the concepts in the ontology layers. Figure 7-7 shows a simple part created in Pro/Engineer which consists of two features: an *extrusion* feature and a *hole* feature, including their key dimensions. Code in Figure 7-8 is generated in custom tools based on Pro/Engineer APIs and ontology toolkit Jena to represent product data semantics of the part

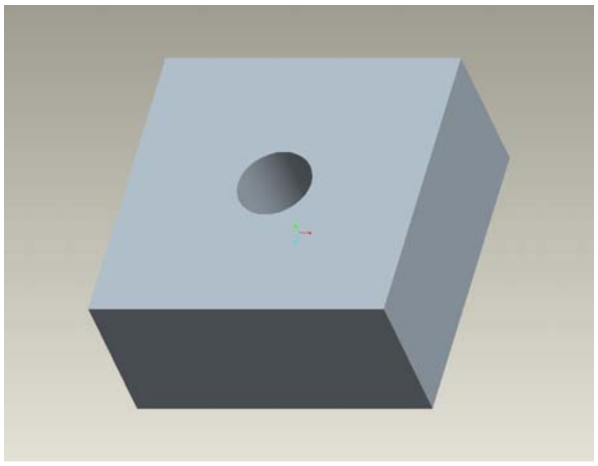

**Figure 7 - 7 Features created in Pro/Engineer**

```
        <Hole rdf:ID="Hole1">
          <has_dimension>
            <Hole_depth rdf:ID="Hole1_depth">
              <dimension_value
      rdf:datatype="http://www.w3.org/2001/XMLSchema#float"
              >50.0</dimension_value>
            </Hole_depth>
          </has_dimension>
          <has_dimension>
            <Hole_diameter rdf:ID="Hole1_diameter">
              <dimension_value
      rdf:datatype="http://www.w3.org/2001/XMLSchema#float"
              >100.0</dimension_value>
            </Hole_diameter>
          </has_dimension>
        </Hole>
        <Extrusion rdf:ID="Block1">
          <has_dimension>
            <Extrusion_depth rdf:ID="Block1_Depth">
              <dimension_value
      rdf:datatype="http://www.w3.org/2001/XMLSchema#float"
              >100.0</dimension_value>
            </Extrusion_depth>
          </has_dimension>
        </ Extrusion >
        <Component rdf:ID="Component1">
          <has_feature rdf:resource="#Hole1"/>
          <has_feature rdf:resource="#Block1"/>
        </Component>
```

**Figure 7 - 8 Instances of a Simple Part**

In this product data semantics, the following concepts in Pro/Engineer application ontology are used: *Hole*, *Extrusion*, which are two types of features; *Hole_Depth*, property of *Hole*, which is used to describe the depth of the *hole*;*Hole_Diameter*, property of *Hole*,which is used to describe the diameter of *Hole*; *Extrusion_Depth*, property of Extrusion, which is used to describe the diameter of *extrusion*. and Instances created for the concepts respectively: Hole1, Block1, Hole1_Depth, Hole1_diameter, Block1_Depth

In addition, the relations between instances are also described in the product data semantics, such as: *has_feature* property to describe the ownership between Component1 and Hole1, Block1; *has_dimentions* property to describe the composition relation between Hole1 and Hole1_diameter and Hole1_Depth.

In this CAD/CAE integration framework, the differences between the product semantics and other traditional product metadata is the data is not described using data types, but instead using ontology concepts and their relations. Thus, the product semantics can contain rich semantic information and can be easily understood. Further, the ontology makes it easier to describe the relations between data sets.

# Chapter Eight - Ontology Mapping Methods for Design/Analysis

This chapter introduces method of ontology mapping between design/analysis ontologies. Pproduct data semantics are based on different ontologies, where different concepts and relations are defined. To make product data semantics translatable between different representations, it is critical to map different concepts and relations.

Ontology mapping can be implemented in two ways: 1) Finding mapping information based on definitions; and 2) Performing similarity calculations, which calculate similarities between different concepts using heuristics method.

## 8.1 Introduction

*Ontology mapping (also called Ontology alignment) is a process to find mapping relationships between entities, including concepts, properties, and instances.* This process is one of the critical tasks necessary to achieve interoperability between engineering applications that use heterogeneous ontologies.

In a CAD/CAE integration framework, heterogeneous applications focus on the same product design and use different ontologies to describe their views of product data (product data semantics). To translate product data semantics described in heterogeneous ontologies, an important step is to align the heterogeneous ontologies. As in CAD/CAE integration, we mainly focus on product design. Hence, in this chapter, we will limit the ontology mapping to the scope of entities related to product design/analysis, defined as Basic Design Entity in this dissertation.

In a CAD/CAE integration framework, we need to translate product semantics associated with product data. So, we are looking to find mapping concepts in different engineering domains about a given product.

## 8.2 Definitions

As the focus of this work is on integration and conversion, the following definitions are made based broadly on data integration/conversion issues related to instances and their corresponding concepts:

**Definition 8.1**: **Mutual Information**: Mutual information refers to the information shared by two instances.

**Definition 8.2**: **Instance Similarity**:

$I(A \cap B)$ : Mutual information of instance A and B

$I(A \cup B)$ : Union of information of instance A and B

Instance Similarity between A and B is: $Sim(A,B) = \dfrac{I(A \cap B)}{I(A \cup B)}$

**Definition 8.3**: **Matching Instances**: In CAD/CAE integration, information about product design is exchanged between instances when common information exists. A pair of instances that have mutual information and also have the highest similarity are defined as Matching Instances, and during translation, the matching instances can be translated from one to another.

Assumption: In this dissertation, for simplicity, we assume the exclusiveness of the Matching Instances in two product data semantics; if Instances A and B are Matching

Instances which belong to different product data semantics P1 and P2, then there cannot exist a third Instance C that belongs to the same product data semantics P2 as B and is also a matching instance of A.

**Definition 8.4**: **Convertible Instance**: If Instance A and Instance B are Matching Instances, during CAD/CAE integration, and information in Instance A can be translated to information in Instance B, then we call Instance B a convertible instance of B ($Convertible(B, A)$). As in CAD/CAE integration, data flow is usually one-directional, and convertible instance is irreversible: B is a convertible instance of A but does not guarantee that A is a convertible instance of B ($Convertible(B, A) \nRightarrow Convertible(A, B)$).

**Definition 8.5**: **Matching Concepts**: For two Matching Instances, their corresponding concepts are defined as Matching Concepts.

**Definition 8.6**: **Concept Similarity**: Among all the instances $I_{1i}$ of concepts C1 and $I_{2i}$ of C2, the highest instance similarity between two instances $Max(Sim(I_{1i}, I_{2i})$ is defined as the concept similarity of C1 and C2.

**Definition 8.7**: **Convertible Concept**: For two instances, A and B, which belong to concept $C_A$ and $C_B$ respectively, if A is a convertible instance of B, then the concept $C_A$ is a convertible concept of $C_B$.

**Definition 8.8**: **Mismatching Instances**: Mismatching Instances do not share mutual information.

**Definition 8.9**: **Mismatching Concepts**: Mismatching Concepts do not have any instances that have concept similarity > 0.

During the integration process, one of the most important tasks is to find the matching instances/concepts to ensure that product data semantics are translated correctly. In the following sections, we will discuss how to find matching instances/concepts.

## 8.3 Finding Matching Information Based on Definitions

Before we look further to use heuristic method to calculate mapping information, some matching concepts can be found based on the definitions and some information that defined in ontologies. The information includes predefined explicit equivalence of classes and instances, also, since our integration is based on traditional integration framework, information generated during data conversion process of traditional integration method can also be used for ontology mapping process later.

In this dissertation the information that is used together with definitions to find matching concepts is categorized into following three types:

### 8.3.1 Ontology Mapping by Explicit Equivalent Classes

During the process of building engineering design/analysis knowledge into ontologies, some of the concepts can be readily identified to be equivalent and the information is explicitly expressed in the ontology. This is one of the simplest and most trivial cases. A rule is defined to find matching concepts according to the equivalence information that was previously built in the ontology:

**Rule 8.3.1.1 If two concepts are equivalent, then they are matching concepts.**

In OWL, equivalences can be expressed using the predefined constructs, *owl:samePropertyAs* and *owl:sameClassAs*, to indicate that two properties/classes are equivalent..

For example, in lower level ontology, common concepts are defined so that they can be reused in upper level ontologies, and it is also possible that concepts defined in upper level ontology are the actually equivalent to some of the common concepts defined in the lower level ontology. We use the predefined expression *owl:sameClassAs* to represent this relationship of equivalence. Code in Figure 8-1 can be used to indicate that the concept of "*Srf*" in PDO is actually refers to the same concept "*Surface*" as indicated in General Design Ontology:

```
<owl:Class rdf:ID="Srf">
  <owl:sameClassAs rdf:resource="&GDO;Surface"/>
</owl:Class>
```

**Figure 8 - 1 OWL/RDF Code of Explicit Equivalence**

Similarly, we can use constructs *owl:samePropertyAs* and *owl:sameIndividualAs* to represent the relationships between two properties or two instances that are the same. Also with the same constructs, more complex equivalence relations between two concepts can be represented.

For example, in Product Design Ontology for Pro/Engineer, according to the definition in Pro/Engineer, a constraint with type of Align is the same as the type "*Align_Offset*" when the offset property is set to be "*coincidence*". So in the PDO, it has the following assertion:

If a *component_constraint* with property "*has_assembly_constraint_type*" being "*Align_Offset*" and property *has_Offset* being zero, or property *is_coincident* being true, then the *component_constraint* is the same as *component_constraint* with a property "*has_assembly_constraint_type*" with value of "*Align*". Following assertion in OWL can be made as shown in Figure 8-2.

```
<owl:Class rdf:ID="Align_Offset">
  <owl:sameClassAs>
    <owl:Restriction>
    <owl:onProperty rdf:resource="is_coincident" />
    <owl:hasValue true />
    </owl:Restriction>
  </owl:sameClassAs>
</owl:Class>
```

**Figure 8 - 2 OWL/RDF code of Complex Equivalence**

### 8.3.2 Mapping Concepts based on Explicitly Equivalent Instances

At the beginning of the integration process, equivalences between existing instances can be defined to indicate the data flow information. For example, between an assembly design tool and an assembly simulation tool, data flow information is defined as from an instance of assembly in assembly design to an instance of assembly in assembly simulation. To indicate the equivalence, each instance uses an URI (Unique Resource Identifier) for the purpose of identification, when different instances share the same identifier, or the two URIs eventually redirects to the same URI, they are assumed to be equivalent.

Based on the information of instances equivalence, based on the definitions of matching instances and matching concepts, since two equivalent instances are obviously two matching instances, we can conclude that:

**Rule 8.3.2.1 If two concepts share the equivalent instance, then the two concepts are two matching concepts.**

For example, during an integration process between assembly design and assembly simulation applications, an URI of http://www.wsu.edu/~peizhan/example/#bracket_assembly is used to indicate the resource of the assembly that is being designed and simulated. In PDO, an instance of *PDO:Assembly* is instantiated to represent the assembly that is designed in the design tool, and it is assigned this URI [Figure 8-3]

```
<PDO:Assembly rdf:ID="asm001">
    <has_URI
rdf:resource="http://www.wsu.edu/~peizhan/example/#bracket_ass
embly"/>
</PDO:Assembly>
```

**Figure 8 - 3 OWL/RDF Code of Assembly Instance in PDO**

During the assembly simulation, an instance of concept *Assembly* in ASO is created to represent the assembly that is simulated,  which also has the same URI [Figure 8-4]

```
<ASO:Assembly rdf:ID="asm001">
    <has_URI
rdf:resource="http://www.wsu.edu/~peizhan/example/#bracket_ass
embly"/>
</ASO:Assembly>
```

**Figure 8 - 4 OWL/RDF Code of Assembly Instance in ASO**

Based on this information,, we can conclude that both the instances use the same URI. Hence they are equivalent, and according to the rule mentioned earlier, their concepts *PDO:assembly* and *ASO:assembly* are equivalent.

### 8.3.3 Mapping Information Generated through Data Conversion

In the traditional integration which is based on data exchanging, standard neutral formats are defined and used as a bridge format, source data is first converted to neutral format and then convert to the target data.  In our approach of semantic level integration , product data semantics is first translated and data is converted based on the mapping of product data semantics, however, the existing procedure of data conversion in traditional integration still can be used and some information generated during the data conversion can be used to help ontology mapping. for example, if it is known that one data types can be converted to another data type, the information can be used to find the matching concepts that used to interpret the two data types.

In this dissertation we regard the data conversion in CAD/CAE integration as a process of converting one type of Basic Design Entity (BDE) to another type of Basic Design Entity. During the data conversion process, if we know two sets of product data can be converted from one to the other, this information can be represented by building relationships between concepts and instances that are described in product data semantics. The following rule is defined to reflect this phenomenon:

**Rule 8.3.3.1 During data conversion, if the attribute of an instance of a BDE is converted to be the attribute of an instance of another BDE, then both Basic Design Entities share the same attribute.**

For example, when the data of a component in Pro/Engineer is converted from Pro/Engineer native format to a neutral format, like *.iv or * iges format, geometry attribute is involved in the conversion. When one instance of *ASO:Part* is converted from an instance of *PDO:Component*, according to Rule 8.3.3.1, the new instance will have the same geometry attribute as the original.

In Figure 8-5, both *PDO:Component* and *ASO:Part* are connected to a file that contains its geometry attribute, and since file Bracket.iv is converted from Bracket.prt so it can be regarded that both the files have the same geometry attribute. Hence *PDO:Component* and *ASO:Part* have the same geometry attribute.

As through data conversion, it is certain that there is common information being shared between the instances of the two concepts hence they are matching instances, so here we conclude that the two concepts are matching concepts.

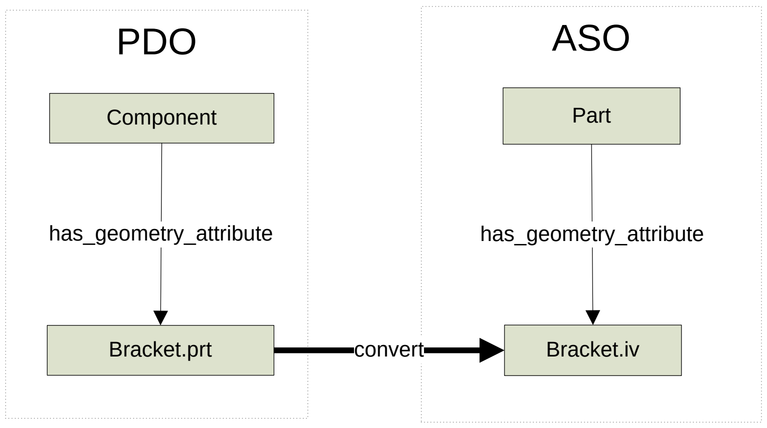

**Figure 8 - 5 Mapping Information through Data Conversion**

## 8.4 Ontology Mapping by Heuristics Method

Heuristic method is used to find matching concepts based on the calculation of similarity and concepts that have the highest similarity are used as matching concepts.

Since it is difficult to use definition 8.5 in section 8.3 to calculate similarity, and Concept Similarity can also be regarded as the relevance between two concepts, instead of using definition 8.2, we measure similarities based on different types of relationships between concepts using heuristics method.

There are several types of relationships that exist between concepts. We limit similarity calculation to the following three types: 1. Attribute (*attribute_of* relationship) 2. Composition (*part_of* relationship) 3. Inheritance (*is_a* relationship)

### 8.4.1 Attribute Similarity

Attribute relationship refers to the relation between an object and its attributes. In this dissertation, the object is limited to a Basic Design Entity (BDE)

Attribute Similarity is used to measure the similarity by calculating whether there are any common attributes that two concepts share.

For the two concepts A and B, which have attributes $T_{Ai}$ , $T_{Bi}$ from, assume similarity between two attributes as $Sim_i(T_{Ai}, T_{Bi})$ , the attribute similarity is the aggregation of all attribute similarities. If we use $Sim_1(A, B)$ to represent overall attribute similarity between A and B, we can have following equation:

$$Sim_1(A,B) = \sum_{i=1}^{n} W_i \times Sim_i(T_{Ai}, T_{Bi})$$

$W_i$ = Weight for each attribute

Note: when calculating attribute similarity, if a concept is an abstract concept which lacks a specific attribute, then an attribute similarity cannot be calculated. For example, the root concept *owl:Thing* is an abstract concept for all the other concepts, and concept is too general to have any attribute so it cannot be used to calculate attribute similarity

## 8.4.2 Composition Similarity

Composition similarity is to consider the similarity according to the *part_of* relationship, which has the following characteristics:

1) *has_parent/has_child* properties are the two basic relations defined for the composition (*part_of*) relationship

2) *has_parent/has_child* properties are inverse to each other, meaning that if A *has_parent* B, then B *has_child* A

3) *has_parent/has_child* are transitive properties, meaning that if A *has_parent/has_child* B and B *has_parent/has_child* C, then A *has_parent/has_child* C.

**Transitive relation between Composition and Attribute**

As we defined in Rule 5.5.1, the *attribute-of* relation is transitive to *part-of* relation. If concept A' in target ontology is found to have the certain attribute similarity, all the concepts that has ownership relations over A' will also have the same attribute similarity, since they also have the same attributes. This can be described as following rule:

**Rule 8.4.2.1 For a given source concept A, if a target concept A' has attribute similarity** $Sim_1(A, A')$ **, and if a Concept C' *has_Child* A', then Concept C' and A have the same attribute similarity** $Sim_1(A, C') = Sim_1(A, A')$

According to this, if two concepts A and A' have attribute similarity, all their owners will have the same attribute similarity. So we will have two composition paths with A and A' at the tail of the each of the path, all the concepts that has ownership over

A could have the same attribute similarity to a concept that has ownership over A’. In this case, composition similarity can be used to further differentiate the similarities.

**Definition 8.10**: **Composition Path:** Path that only consists of composition relations

**Definition 8.11**: **Similar Composition Path** for two composition paths: If all the nodes in one composition path have the same attribute similarity to all the nodes in the other composition path, then we define the two paths as similar composition paths. For example, in Figure 8-6, for two paths *PDO:Assembly->PDO:Component->PDO:Component_feature->PDO:Model_item*, and *ASO:Assembly->ASO:Part->ASO:Assembly_feature*, if two nodes at the tail have the same attribute similarity and based on transitivity of *attribute-of* relation over *part-of* relation, all the nodes in the first path have the same attribute similarity to the second path, then the two paths are similar composition paths.

After setting the equivalence and filtering out the class hierarchies due to inheritance(*is-a*) relationship, the following graph can be drawn for PDO and ASO considering only composition(*part-of*) relationship and the concepts limited at design entity level [Figure 8-5 and Figure 8-6]. Each of the paths in the graph is a composition path.

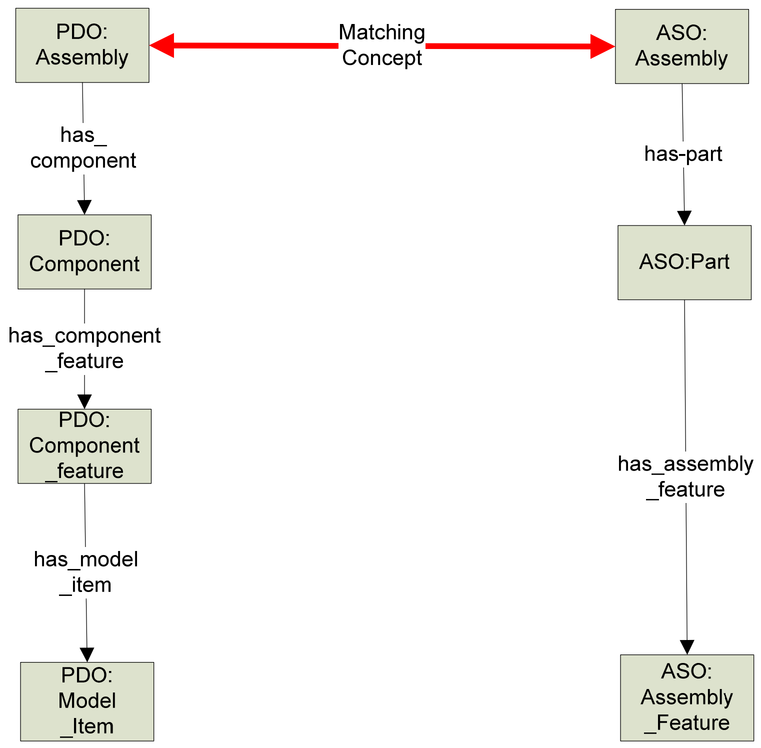

**Figure 8 - 6 Similar Composition Paths**

In a CAD/CAE integration framework, it is always easier to identify data translation from a higher composition level and to find matching concepts at the top level (e.g. both the assembly concepts in PDO and ASO). After we discover concepts that have attribute similarity in a lower composition level, plus the matching concepts that previously found at the top composition level, the concepts between them will share the attribute similarity and can be differentiated in composition similarity. So in this dissertation we always calculate the composition similarity between concepts that are in two similar composition paths.

We define the composition similarity between two concepts in similar composition paths based on the distance measure:

Distance $D(A, B) = n$

n – Number of edges in the shortest path between two nodes A and B if only considers part-of relation, for two matching concepts, the distance is 0

In two similar composition paths, if we have two nodes A and B in each of the path which has H1, T1 and H2 and T2 as head and tail nodes respectively as shown in figure 8-7, the *composition similarity* is the similarity of their relative position in each of the path, and if we use $Sim_2(A,B)$ to represent composition similarity, then an it can be calculated as $Sim_2(A,B)=1-\left|\frac{D(H1,A)}{D(H1,T1)}-\frac{D(H2,B)}{D(H2,T2)}\right|$

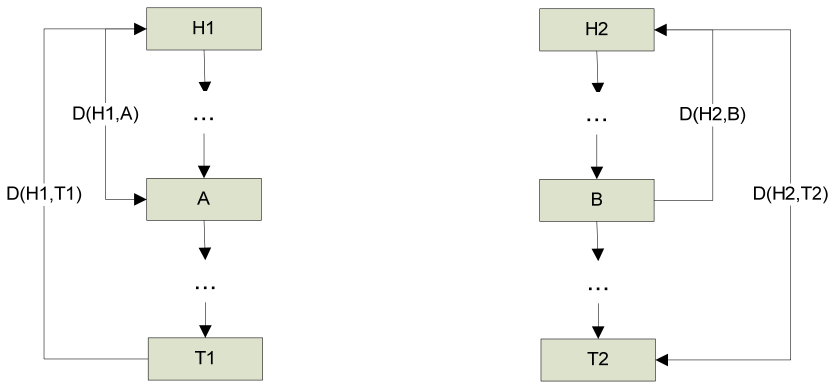

**Figure 8 - 7 Composition Similarity of Nodes in Two Paths**

For example, in Figure 8-6, if it is known that *PDO:Assembly* and *ASO:Assembly* are matching concepts, composition similarities factor between *PDO:Component* and *ASO:Part* can be calculated as:

$$Sim_2(PDO:Component, ASO:Part)=$$
$$1-\left|\frac{D(PDO:Assembly, PDO:Component)}{D(PDO:Assembly, PDO:Model\_item)}-\frac{D(ASO:Assembly, ASO:Assembly\_Feature)}{D(ASO:Assembly, ASO:Part)}\right|$$
$$=1-\left|\frac{1}{3}-\frac{1}{2}\right|=0.83$$

### 8.4.3 Similarity based on Inheritance Relation

Definition 8.12 **Inheritance Hierarchy:** Concept hierarchy that consists of *is-a* relation only.

In this dissertation, we assume the transitivity of attribute similarity and composition similarity over inheritance relation, which means all the concepts in a concept inheritance hierarchy will share the same attribute similarity and composition similarity.

A rule can be used to describe this assumption:

**Rule 8.4.3.1 For two matching concepts A and B, if each of which belongs to the two different hierarchies, then all the concepts Ai and Bi that are sub-concepts of A and B will have the same attribute similarity and composition similarity:** $Sim_1(A_i, B_i) = Sim_1(A, B)$ **and** $Sim_2(A_i, B_i) = Sim_2(A, B)$

For example, Figure 8-8 shows a concept inheritance hierarchy for assembly_constraint in ASO, compared with inheritance hierarchy in figure 7-4, all the sub-concepts of *PDO:Component_constraint* and *ASO:Assembly_constraint* share the same attribute similarity and composition similarity as *PDO:Component_constraint* and *ASO:Assembly_constraint* has.

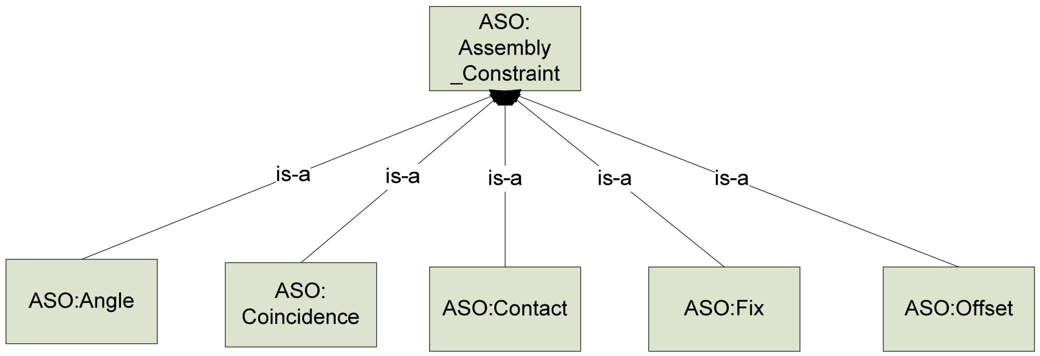

**Figure 8 - 8 Assembly_Constraint Inheritance Hierarchy**

### 8.4.4 A General Method of Finding Matching Concepts

The process of finding matching concepts can be regarded as a repetitive process: after a pair of matching concepts is found, the new matching information will be used for the next round of similarity calculations.

As it is always easier to identify the matching concepts at a higher composition level, the whole sequence flows in a top-down direction.

To find all the matching concepts between two ontologies, we use following steps:

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. In a composition graph, only the root class in an inheritance hierarchy is used, all the sub-

classes are excluded from the composition graph. And as it is always easier to identify matching concepts at the top level, it is expected that two root concepts in the composition graph are matching concepts. For example, in Figure 8-9, a two composition graphs that belong to different ontologies are created, with root node A0 and B0 as matching concepts. Figure 8-10 and Figure 8-11 show two inheritance hierarchies which use A21 and B22 as base class.

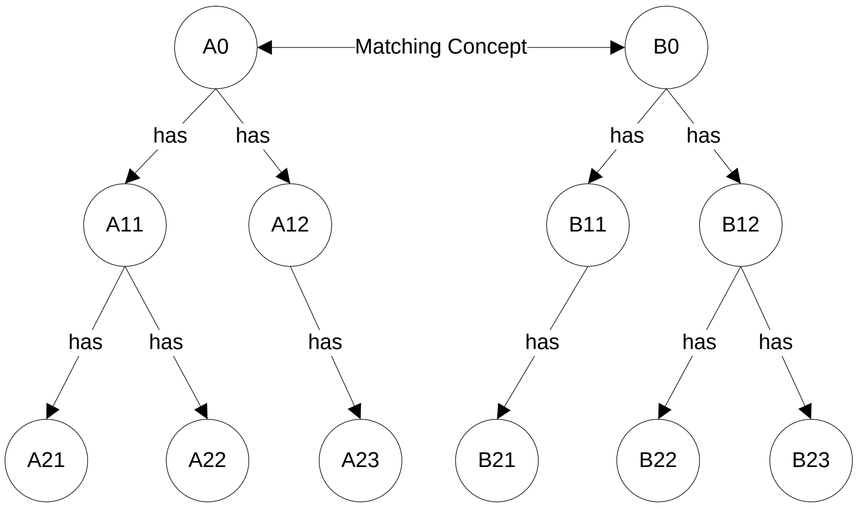

**Figure 8 - 9 Two Composition Graphs in Source and Target Ontology**

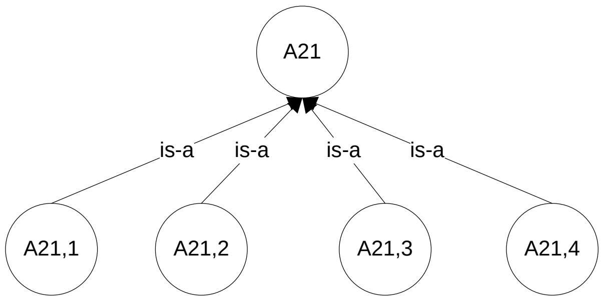

**Figure 8 - 10 Inheritance Hierarchy of Node A21 in Source Ontology**

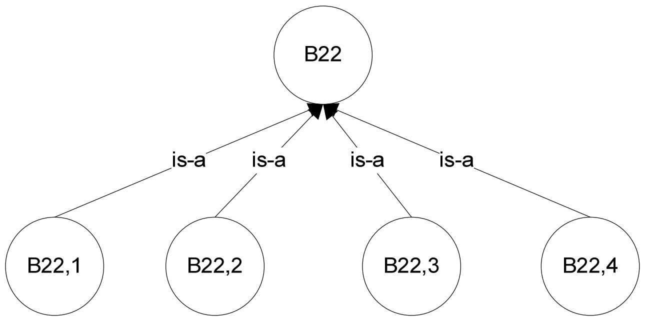

**Figure 8 - 11 Inheritance Hierarchy of Node B22 in Target Ontology**

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. For example, for concept A21, iterate all the concepts in the right composition graph of Figure 8-9 and calculate the attribute similarity between them and concept A21 to find the one has highest overall attribute similarity.

4). **Find Similar Composition Paths.** After a target concept is found to have highest attribute similarity with the given source concept, for example, A21 and B22, since the attribute is transitive over composition relation, all the classes that has owner relations over the target concept will have at least the same attribute similarity, and two similar composition paths can be created between two root nodes A0, B0 and the source concept A21 and target concept B22, as shown in Figure 8-12

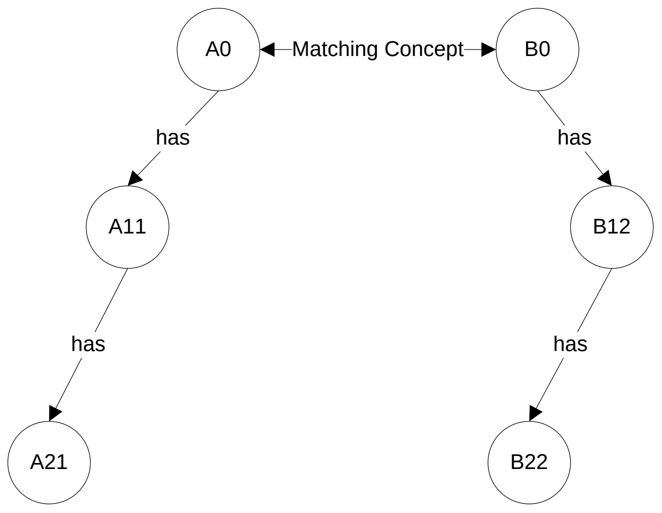

**Figure 8 - 12 Two Similar Composition Path in Source and Target Ontology**

5) **Calculate Composition Similarity.** Calculate composition similarity and choose the one with the highest composition similarity and attribute similarity as matching concept. In Figure 8-12, we can see A11 and B12, A21 and B22 have the highest composition similarity, so we can conclude that A11 and B12, A21 and B22 are matching concepts

6) **Calculate attribute similarity and composition similarity based on inheritance relation**. Composition similarity and attribute similarity can be transitive over inheritance relation, after two base concepts are found to be matching concepts, all their sub-concepts are set to have the same attribute similarity and composition similarity. For example, we can conclude that all the sub-classes of A21 and B22 as shown in Figure 8-10 and Figure 8-11 have same attribute similarity and composition similarity as A21 and B22 has.

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

8) **Human intervention**, if no similarity information can be used to find matching concept, human interference has to be used.

## 8.5 Tailoring the Method for Mapping using Basic Design Entity

In the previous section we discussed the method of calculating mapping similarity for concepts. In this dissertation the focus is to translate product information between different representations that have design entities as the basic elements. Hence it is of the highest interest to find the similarities between design entities in two different ontologies.

As discussed in 6.1 and 6.2 about specific attribute of BDE, to calculate similarity between Basic Design Entities, similarity for each of the specific attribute is calculated, they are form attribute similarity, behavior attribute similarity respectively, function attribute is not considered in this dissertation.

### 8.5.1 Form Attribute

Form attribute includes information about design such as material, geometry etc. In this dissertation we will focus on the geometry of form attribute since it contains the most important information in a design that most design/analysis activities are focused on. Geometry attribute is always the common currency during CAD/CAE integration.

To represent geometry attribute, a geometry inheritance hierarchy is defined in General Design Ontology. To simplify the calculation of geometry similarity, geometry information is categorized into four basic types: *Point*, *Curve*, *Surface*, and *Solid*, which can be further categorized such as *plane* and *cylindrical_surface* under *Surface*. We can see that the lower the level that a geometry type falls under, the more specific is the geometry information that it has.

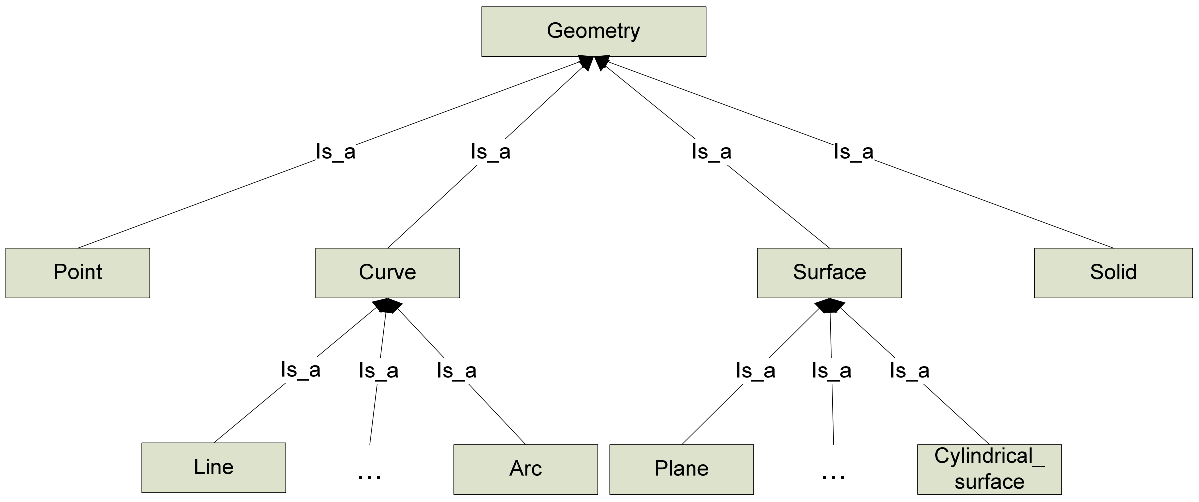

**Figure 8 - 13 Geometry Information Model**

With the simplified model, the following rules are used to determine geometry similarity:

**Rule 8.5.1.1 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**. For example, a *Line* and *Circle* have the common type of *Curve*, so their geometry similarity is proportional to the depth of the level *Curve* in the hierarchy. If we define a function F to be positive proportional to the depth, the geometry similarity between two geometry types can be expressed as $Sim(Line, Circle) = F(depth(Curve)) = F(1)$. And the geometry similarity between *Point* and *Line* is $Sim(Line, Circle) = F(depth(Geometry)) = F(0)$ since concept *Geometry* is the common category of *Point* and *Line* which has a depth of 0. We can also conclude that $Sim(Line, Circle) > Sim(Point, Line)$ since $F(1) > F(0)$

**Rule 8.5.1.2 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**. For example, concept *PDO:Feature* can have geometry types such as *Point*, *Curve*, *Surface*, *Solid* as its geometry attribute, hence their common geometry category at the lowest level *GDO:Geometry* is used to calculate geometry attribute similarity.

### 8.5.2 Behavior Attributes

Various behaviors focus on different aspects of product design which are defined and used across engineering design/analysis domains. In order to find similar behavior attribute that are shared by both the applications in ontology mapping, rules are defined to identify common behavior attribute between engineering design/analysis applications so it can be used for calculating behavior attribute similarity:

**Rule 8.5.2.3** If both applications are in the realm of 3-D feature-based product design for example, such as Pro/Engineer, Solidworks, or CATIA, then designCreationBehavior can be used as common behavior attribute.

**Rule 8.5.2.4** If both applications share a common definition in functionalBehavior, then that definition may be used as behavior attribute. For example, in the Assembly Design and Assembly Simulation domains, for the concepts PDO:component_constraint and ASO:Assembly_constraint, both of them have FunctionalBehaviorAttribute as degree of freedom, so degree of freedom can be used as behavior attribute to calculate behavior attribute similarity.

### 8.5.3 Example of Calculating Similarity for BDEs Based on Their Attributes

With the algorithm of calculating attribute similarity introduced previously, we use an example to show how to find a matching concept for a given BDE concept based on similarity calculation.

In an integration between an assembly design and assembly simulation, the knowledge of assembly design and assembly simulation are built into two ontologies: PDO (Product Design Ontology) and ASO (Assembly Simulation Ontology), both of them use their own knowledge to describe product assembly information. To help product assembly information to be translated from assembly design tool to assembly simulation tool, it is important to find corresponding matching concepts in target ontology (ASO) for a given concept in source ontology (PDO) that are used to describe product assembly data.

Following steps are used for finding matching information:

1) Matching process based on definitions.

Based on the existing data flow information, assuming data is exchanged between assembly design unit – an instance of PDO:Assembly in product design to an instance of ASO:Assembly, and we define they are equivalent instances, and based on definition in 8.3, concepts PDO:Assembly and ASO:Assembly are a pair of matching concepts.

2) Create composition graph and inheritance hierarchies

After step 1 of finding matching concepts based on definitions, first we need to create composition graph. After filtering out inheritance relations, we have two composition graphs for each of the ontology as in Figure 8-14 and Figure 8-15

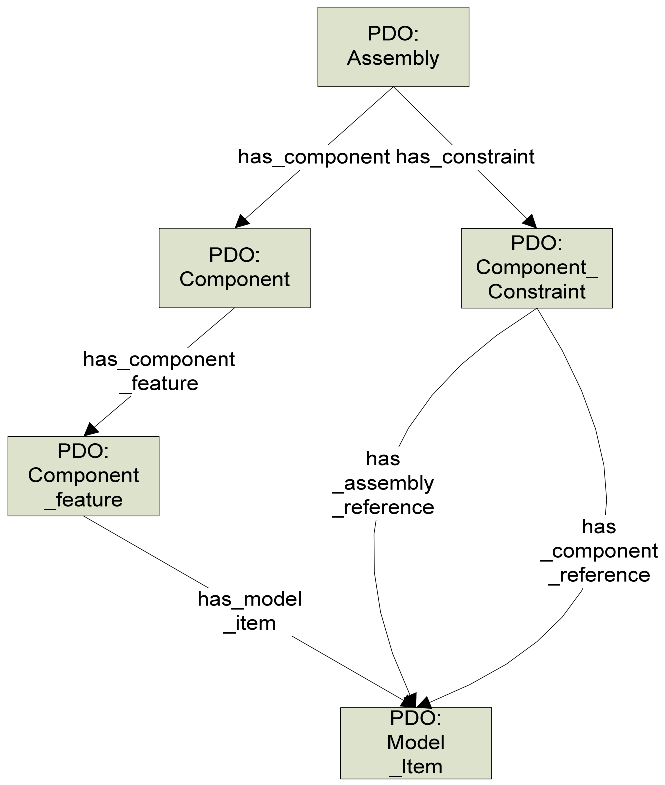

**Figure 8 - 14 PDO composition graph**

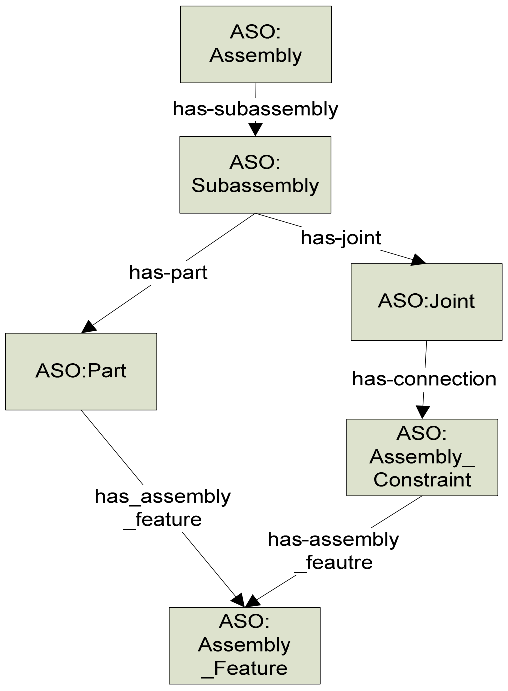

**Figure 8 - 15 ASO composition graph**

3) Calculate Attribute Similarity

In two composition graphs, for source concept *PDO:component_constraint*, we found it has following attributes:

1. Geometry Attribute: *PDO:component_constraint* has two relations *has_component_reference* and *has_assembly_reference*, which are sub-properties of composition relation (*part_of*), the object of *has_component_reference* and *has_assembly_reference*, which are both *PDO:Model_Item*, has *GDO:Geometry* as its geometry attribute. According to rule 8.4.2.1 about the transitivity of *attribute-of* relation over *part-of* relation, geometry attribute of *PDO:Model_Item* is also the geometry attribute of *PDO:component_constraint*

2. Functional Behavior Attribute, *PDO:component_constraint* has *GDO:Degree_of_Freedom* as its functional behavior attribute.

4). Find Similar Composition Paths

In target ontology ASO, after iterating all the concepts in the composition graph in figure 8-15, based on the information that *GDO:Geometry* is geometry attribute of *ASO:Assembly_feature*, and the composition relation *has_assembly_feature* between *ASO:assembly_constraint* and *ASO:Assembly_feature*, accordint to rule 8.4.2.1 about the transitivity of composition relation, we found *ASO:assembly_constraint* has *GDO:Geometry* as its geometry attribute. Also *ASO:assembly_constraint* has *GDO:Degree_of_Freedom* as its functional behavior attribute. Hence we can conclude that *PDO:component_constraint* and *ASO:assembly_constraint* has highest attribute similarity.

Since it is known that *PDO:Assembly* and *ASO:Assembly* are a pair of matching concepts, from all the BDE concepts in ASO composition graph as in Figure 8-14 and 8-15, two similar composition paths can be drawn in Figure 8 - 16,

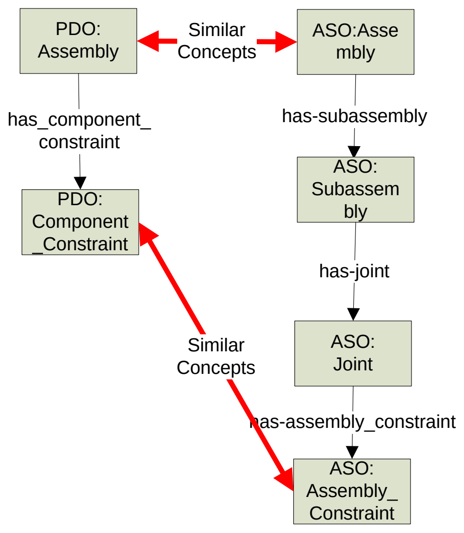

**Figure 8 - 16 Similar Composition Paths between PDO and ASO**

5) Calculate Composition Similarity.

Considering the composition similarity factor in figure 8-16, the composition similarities are:

*Sim(PDO:Component_Constraint, ASO:Joint)* = $1-\left|\frac{1}{1}-\frac{1}{2}\right|=0.5$

*Sim(PDO:Component_Constraint, ASO: Assembly_constraint)* = $1-\left|\frac{1}{1}-\frac{2}{2}\right|=1$

We can see that *ASO: Assembly_constraint* has a higher composition similarity with *PDO:Component_Constraint*, so it can be concluded that *ASO: Assembly_constraint* has the highest similarity to *PDO:Component_Constraint* and hence they are matching concepts.

6) Calculate attribute similarity and composition similarity based on inheritance relation

After calculating similarities between Basic Design Entities, if a match between two BDEs is found, e.g. *PDO:Component_Constraint* and *ASO: Assembly_constraint*, according to the RULE 8.4.3.1 about transitivity of attribute and composition attribute over inheritance relation, all subclasses of these two concepts also have the same inheritance similarity. At this level, we cannot differentiate the similarities between any two subclasses, and we will introduce how to depend on the exact properties of each instance to find matching concepts for each specific instance in the next section.

### 8.5.4 Calculating Concept Similarity Based on Instances

After finding matching concepts between two base concepts, for example, *PDO:Component_constraint* and *ASO:Assembly_constraint* in Figure 8-14 and Figure 8-

15 respectively, according to RULE 8.4.3.1 their sub-concepts share the same attribute similarity and composition similarity. In an inheritance hierarchy, compared to concepts in a higher level which usually has general information about attributes, concepts in lower level have more specific attribute information, and similarity between two concepts can be different when their attributes have different specific value(type), in order to further calculate similarity between their sub-concepts in their inheritance hierarchies as shown in Figure 7-4 and Figure 7-5, specific attribute information that depend on the actual instance of the concepts needs to be collected to calculate similarity.

For example, based on its definition, ASO:Contact can have following different assembly features and corresponding behaviors:

| **Geometry Attribute from** ***ASO:has_assembly_feature*** | **Geometry Attribute from** ***ASO:has_assembly_feature*** | **Behavior Attribute** |
|---|---|---|
| *GDO:Cylindrical_surface* | *GDO:Cylindrical_surface* | *GDO:Translate_z,* <br> *GDO:Rotate_z* |
| *GDO:Plane* | *GDO:Plane* | *GDO:Translate_x,* <br> *GDO:Translate_y,* <br> *GDO:Rotate_z* |
| *GDO:Line* | *GDO:Plane* | *GDO:Translate_x* <br> *GDO:Translate_z* <br> *GDO:Rotate_z* <br> *GDO:Rotate_y* |

**Table 8 - 1 Possible Attributes of concept ASO:Contact**

Depends on the actual instance in the product data semantics, if an instance Contact1 of class *ASO:Contact* is found to have the following attributes:

*has_assembly_feature* Plane1, which is an instance of concept *GDO:Plane* *has_assembly_feature* Plane2, which is an instance of concept *GDO:Plane*

*has_functional_behavior*: *ASO:Translate_x*

*has_functional_behavior*: *ASO:Translate_y*

*has_functional_behavior*: *ASO:Translate_z*

*has_functional_behavior*: *ASO:Rotate_z*

OWL/RDF code to represent the above attributes is shown in Figure 8-17.

```
< Contact rdf:ID=" Contact_1">
    <has_assembly_feature rdf:resource="#Plane_1"/>
    <has_assembly_feature rdf:resource="#Plane_2"/>
    <has_functional_behavior rdf:resource="#Translate_x"/>
    <has_functional_behavior rdf:resource="#Translate_y"/>
    <has_functional_behavior rdf:resource="#Translate_z"/>
    <has_functional_behavior rdf:resource="#Rotate_z"/>
  </ Contact>
```

**Figure 8 - 17 OWL/RDF Code of Contact Constraint**

In order to find the corresponding matching concept for *ASO:Contact* so that instance Contact_1 can be converted to, attribute similarity between subconcepts of *PDO:Component_Constraint* and *ASO:Contact* are calculated to find which subconcept can have highest attribute similarity with *ASO:Contact* with the specific attribute information according to Contact_1.

For example, according to the restriction and axiom built in the PDO, *PDO:Mate* can have exactly the same geometry attribute and behavior attribute as *ASO:Contact* has for instance Contact_1:

| **Geometry Attribute from** ***PDO:has_assembly_reference*** | **Geometry Attribute from** ***PDO:has_component_feature*** | **Behavior Attribute** |
|---|---|---|
| *GDO:Plane* | *GDO:Plane* | *GDO:Translate_x,* |

|  |  |  |
|---|---|---|
|  |  | *GDO:Translate_y,* <br> *GDO:Rotate_z* |

**Table 8 - 2 Attribute of Concept PDO:Mate**

From the above information, we can conclude that *PDO:Mate* .is the matching concept of *ASO:Contact* for instance Contact_1. Please note that, the matching concept that is found based on this method is dynamic and depends on the actual instance information, so it is not fixed and has to be determined every time for each of the specific instances.

# Chapter Nine - Representing Ontology Mapping Information

## 9.1 Introduction

After finding the mapping information between two ontologies, it is necessary to store the information in a certain format so that it can be reused. In order to represent the mapping information between different product data semantics, a bridge ontology is used to describe the mapping information between two different ontologies.

In this chapter, the mapping informaiotn is categorized and discussed as following:

1. Concept mapping
2. Instance mapping

## 9.2 Bridge Ontology

In ontology mapping, bridge ontology represents the mapping information between the two ontologies. If two different ontologies are regarded as two islands, bridge ontology acts as a bridge between the two ontologies – with the bridge ontology, a match of a concept or an instance in source ontology can be found in the target ontology.

Bridge ontology is an ontology which identifies concepts and instances from both the source ontology and target ontology and represents their mapping relations. Once a bridge ontology is built, it can be reused for a new integration process that involves the same source and target ontologies

## 9.3 Concept Mapping Representation

Concept mapping is to represent the mapping relations between concepts in different ontologies. In this dissertation, we only consider one to one concept mapping, which is identified to have following types:

1. **Unconditional Mapping**: Concept is unconditionally a matching concept of another concept;

2. **Conditional mapping**: Only under certain conditions can one concept be a matching concept of another concept

### 9.3.1 Unconditional Mapping

Unconditional mapping is for two concepts that are unconditionally pair of matching concepts, for example, PDO:Assembly and ASO:Assembly

To describe unconditional mapping, predefined owl expression *owl:sameClassAs* can be used to describe the explicit equivalence if two concepts are found to be matching concepts during the building ontology process. For matching concepts that are found later in ontology mapping process, in this dissertation we use concept of *ConceptMatch* to describe the matching concept relation between two concepts in bridge ontology. To describe which concepts are involved in the matching concepts, property *has_concept* is used to describe the which concepts are being mapped from one to another.

```
<ConceptMatch rdf:ID="Match_1">
    <has_concept rdf:resource="#PDO:Component"/>
    <has_concept rdf:resource="#ASO:Part"/>
</ConceptMatch>
```

**Figure 9 - 1 OWL/RDF Code of Unconditional Mapping**

### 9.3.2 Conditional Mapping

For some classes, only under certain circumstances will they be matching concepts, these circumstances are described using concept *Condition*, for example, some classes can be matching concepts only under a condition that a concept has given specific values of attribute. For a concept match, there could be more than one conditions.

For instance, concept PDO:Model_item is a general concept and only some of its subtypes can be a matching concept to ASO:Assembly_feature. The condition for PDO:Model_item to be a matching concept of ASO:Assembly_feature is when PDO:Model_item has a geometry attribute. The condition can be described using the following OWL/RDF code:

```
<Condition rdf:ID="condition1">
    <MappingAttribute rdf:resource="#has_geometry_attribute"/>
</ Condition >

 <ConditionalConceptMatch rdf:ID="Match_1">
    <has_concept rdf:resource="#PDO:Model_item"/>
    <has_condition rdf:resource="#condition1"/>
    <has_concept rdf:resource="#ASO:Assembly_feature"/>
  </ConceptMatch>
```

**Figure 9 - 2 OWL/RDF Code of Mapping Condition**

## 9.4 Instance Mapping Representation

In our semantic level integration, instances in product data semantics act as a link between concepts and product data, instance mapping links instances between different product data semantics so product data in one representation can be eventually linked to another representation.

In this dissertation, we use following concept of *InstanceMatch* together with their properties to describe the relation between two mapped instances:

1) *SourceInstance* indicates the instance in source product data semantics

2) *TargetInstance* indicates the instance in target product data semantics that is linked to the source instance.

Bellow is some sample code in OWL which describes the mapping relationship between two instances PDO:asm001 and ASO:asm001

```
<InstanceMatch rdf:ID="instancematch1">
  <sourceInstance rdf:resource="#PDO:asm001"/>
  <targetInstance rdf:resource="#ASO:asm001"/>
</InstanceMatch >
```

**Figure 9 - 3 OWL/RDF Code of InstanceMatch**

# Chapter Ten - Example Scenario

This chapter introduces an example scenario and shows the benefit of new semantic level integration compared to the traditional data level integration. In the example scenario, a product assembly is designed in one application that focus on the assembly design domain and the assembly process of the assembly design is simulated in another application that focus on assembly simulation domain. The interoperability problem that happens in the process of integrating different applications is addressed, specifically, problem about how information about assembly design is described and exchanged between the two heterogeneous applications.

To show how the new integration mechanism proposed in this dissertation works, based on the knowledge built in ontologies and the product data created in product assembly design, results and processes in different stages are discussed, such as result of original product data semantics about product design that captured in assembly design, result and process of translating product data semantics from original representation to a targeted different representation, and the mapping information between the original and converted product data semantics. And at last it is discussed that how these processes and results could come together and benefit us in converting heterogeneous product data from one presentation to another presentation.

## 10.1 Motivating Scenario

The motivation of the example is to show how CAD/CAE applications can be integrated at a new “semantic level”. Two parts, Screw and Plate, are designed in a 3d feature based geometric modeling tool as shown in Figure 10-1 and Figure 10-2. An assembly asm001 which consists of the two parts is then designed in the assembly design tool in Pro/Engineer as shown in Figure 10-3, by using knowledge of constraints and features that are defined specifically for Pro/Engineer users.

To simulate the assembly process, assembly data needs to be imported into a virtual environment that can be used to simulate the real assembly environment so user can verify that the two parts can be assembled into the final assembly. In the virtual assembly environment, the assembly process is simulated based on its own knowledge model, and it also has its own data format and ways of representing geometry and assembly information.

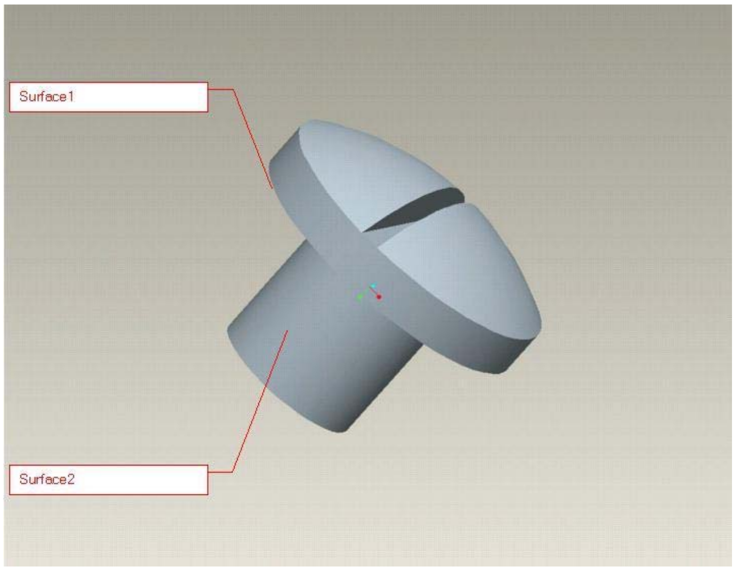

**Figure 10 - 1 Part Screw in a Test Assembly**

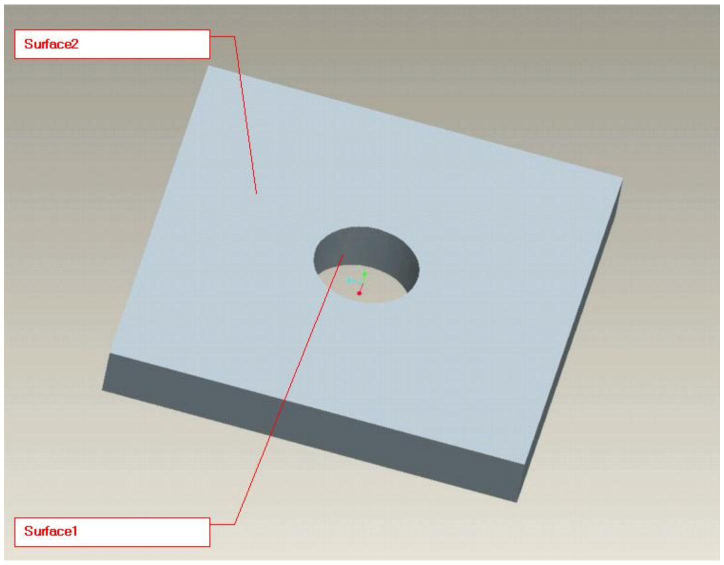

**Figure 10 - 2 Part Plate in a Test Assembly**

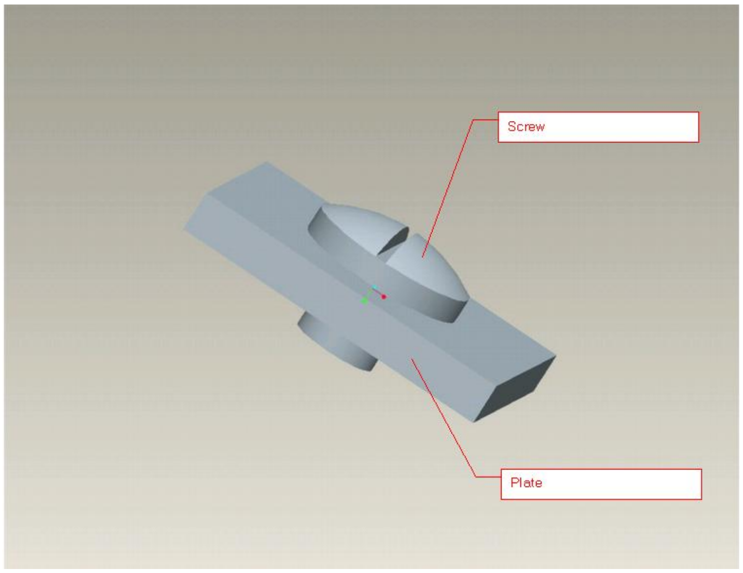

**Figure 10 - 3 Test Assembly asm001**

## 10.2 Integration in a Traditional Integration Framework

Previously in a traditional integration framework, the information exchange is limited to a pure data level. The information flow is as follows:

In order to allow the assembly simulation tool to recognize the assembly data that was originally designed in Pro/Engineer, the assembly design data in Pro/Engineer needs to be converted to another representation that can be understood by assembly simulation tool. Since Pro/Engineer and the Assembly simulation tool use different knowledge to describe constraints between parts in the assembly, for example *Align* concept is used in Pro/Engineer to describe the relationship of two planes which have the same position and are parallel to each other, whereas in the assembly simulation tool the concept *Contact* is used to describe the same relationship, during the conversion process, in a traditional integration framework, the data that is described by the original knowledge model of Pro/Eingineer cannot be understood by assembly simulation tool which is based on the other knowledge model. To reconcile the differences, normally in a traditional integration framework the original data has to be converted to a neutral knowledge representation and then converted to the target representation again. However, in an integration framework that focuses on product lifecycle management, there could be large number of different applications that co-exist and need to communicate with each other. Data is interpreted from different viewports and its representation varies so significantly that there is no common knowledge representation can cover all the different knowledge bases. This leads to information getting lost during the conversion simply because it is not able to be recognized by other applications that use a different knowledge representation.

In the next a few sections, we will discuss how to overcome the interoperability problem by using our new semantic integration approach that has been designed and implemented in this dissertation.

## 10.3 Building Ontologies for Assembly Design/Assembly Simulation

As discussed in chapter 5, 6 and 7, knowledge in assembly design using Pro/Engineer and assembly simulation is investigated and built into ontologies as Product Design Ontology (PDO) and Assembly Simulation Ontology (ASO). Using Protégé as ontology editor and OWL/RDF as ontology language, concepts and relations are represented in ontology using subject-property-object triplet expression. Figure 10-4 and Figure 10-5 show a small sample of defined classes and relations of PDO in protégé. Similarly ASO is defined.

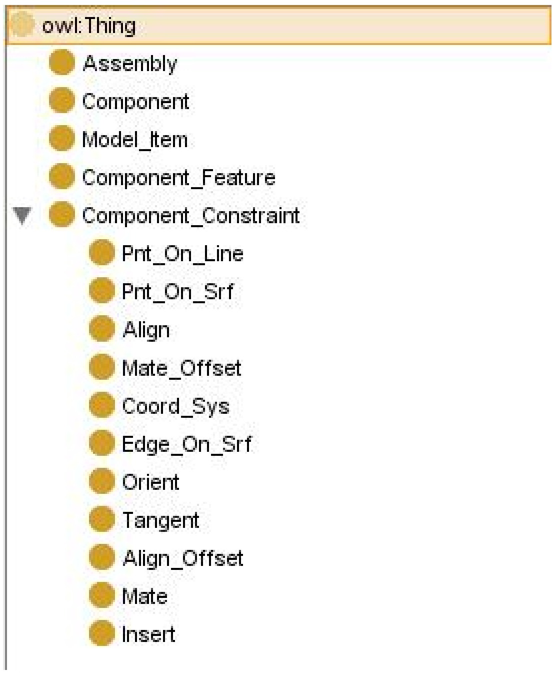

**Figure 10 - 4 Define PDO Classes in Protégé**

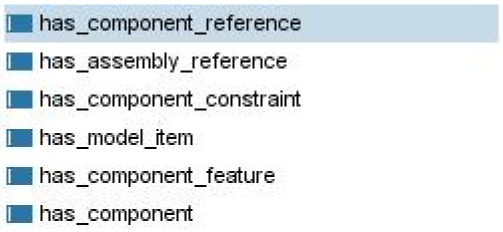

**Figure 10 - 5 Define PDO Relations in Protégé**

## 10.4 Capturing Product Data Semantics from Original Design

Based on the knowledge of assembly design in Pro/Engineer that is built into Product Design Ontology (PDO), product data semantics need to be created based on steps introduced in chapter 7. Since same knowledge is used to describe product data semantics and also to create original product assembly data, we can easily identify the concepts and relations from PDO and use them to interpret the product assembly data by creating instances.

For example, the following concepts in PDO are instantiated according to the assembly data: instance Asm001 is instantiated from concept *PDO:Assembly*, instance Plate is instantiated from concept *PDO:Component*, and instance Screw is instantiated from concept *PDO:Component,* Figure 10-6 shows the actual OWL/RDF source code.

```
        < Assembly rdf:ID="asm001"/>
        < Component rdf:ID="Plate"/>
        < Component rdf:ID="Screw"/>
```

**Figure 10 - 6 OWL/RDF Code of Specific Assembly and Component in PDO**

Also, constraints information in the assembly is described using predefined concepts in PDO, and the following instances are created:

Mate_1 is created as an instance of concept *PDO:Mate* and Insert_1 is created as an instance of concept *PDO:Insert*. Figure 10-7 shows the actual OWL/RDF source code

```
        < Mate rdf:ID="Mate_1"/>
        < Insert rdf:ID="Insert_1"/>
```

**Figure 10 - 7 OWL/RDF Code of Specific Constraints in PDO**

According to the product design in Pro/Engineer, the relations between instances can be represented as shown in Table 10-1 in form of Subject-Property-Object,. For example, as indicated in the first row, assembly Asm001 has Plate as its *PDO:has_component* property, meaning Plate is a component of Asm001. RDF/OWL source code in Figure 10-8 uses rdf:Description and predefined properties to describe these relations in supplement of the definition of instance *asm001* that described in Figure 10-4.

| Subject | Property | Object |
|---|---|---|

| | | |
|---|---|---|
| Asm001 | *PDO:has_component* | Plate |
| Asm001 | *PDO:has_component* | Screw |
| Asm001 | *PDO:has_component_constraint* | Mate_1 |
| Asm001 | *PDO:has_component_constraint* | Insert_1 |

**Table 10 - 1 Relations between Instances**

```
        < rdf:Description rdf:about="asm001">
          <has_component rdf:resource="#plate"/>
          <has_component rdf:resource="#screw"/>
          <has_assembly_constraint rdf:resource="#Align_1"/>
          <has_assembly_constraint rdf:resource="#Insert_1"/>
        </rdf:Description>
```

**Figure 10 - 8 OWL/RDF code of Assembly Relations**

In original assembly design, detailed information about each component constraint between two parts is captured and described using PDO concepts. As defined in PDO, each instance of concept *component_constraint* should have at least one instance of *model_item* as attribute of *has_assembly_reference*, and at least one instance of *model_item* as attribute of *has_component_reference*. By inquiring the product data model, we found following information that can be used to reflect this information:

Plate has Surface1 and Surface2 as two model items. Screw also has two model items named Surface1 and Surface2. The Mate_1 constraint between Plate and Screw uses Surface1 of Plate and Surface2 of Screw as component reference and assembly reference respectively, and the insert_1 constraint between the same parts uses Surface2 of Plate and Surface1 of Screw as component reference and assembly reference. To describe above information, we have following instances and properties in PDO:

Previously two instances of concept PDO:Component_Constraints, Mate_1 and Insert_1, are already created for the component constraint information. Another four instances Plate:Surface1, Plate:Surface2, Screw:Surface1, Screw:Surface2 are created for each of the model items. (Note: namespace *Plate* and *Screw* are added to the instance name to indicate the component that these model items belong to)

To reflect the relations between model items and component constraint, properties are used to link instances and we have the following subject-property-object relations as shown in Table 10-2. Figure 10 – 9 and Figure 10 – 10 shows the corresponding OWL/RDF source code.

| Subject | Property | Object |
|---|---|---|
| Mate_1 | *PDO:has_component_reference* | Plate:Surface2 |
| Mate_1 | *PDO:has_assembly_reference* | Screw:Surface1 |
| Insert_1 | *PDO:has_component_reference* | Plate:Surface1 |
| Insert_1 | *PDO:has_assembly_reference* | Screw:Surface2 |

**Table 10 - 2 Relations between Component Constraints and Model Items**

```
    < rdf:Description rdf:about="Align_1">
      <has_component_reference
  rdf:resource="#Plate:Surface2"/>
      <has_assembly_reference
  rdf:resource="#Screw:Surface1"/>
    </rdf:Description>
```

**Figure 10 - 9 OWL/RDF Code of Mate_1**

```
  < rdf:Description rdf:about="Insert_1">
    <has_component_reference
rdf:resource="#Plate:Surface1"/>
    <has_assembly_reference
rdf:resource="#Screw:Surface2"/>
  </rdf:Description>
```

**Figure 10 - 10 OWL/RDF Code of Insert_1**

## 10.5 Ontology Mapping for PDO and ASO

Now we have product data semantics which describes product data using knowledge in PDO. Before translating it to another knowledge representation, we need to find the matching concepts by using the ontology mapping algorithm described in Chapter 8. The process can be analogized to translating an article from one human language to another human language. Before we translate product data semantics, it is important to find the matching concepts in two knowledge bases, similar to finding the corresponding word in the target language that has the same meaning as the one in source language in the given context.

In this example we will reuse some result from chapter 8. By using methods introduced in chapter 8, we have identified *PDO:Assembly* and *ASO:Assembly* as matching concepts based on definitions of matching concepts and equivalent instances and Rule 8.3.2.1 as discussed in Chapter 8.3.2. Also *PDO:Component* and ASO:Part are identified as matching concepts according to Rule 8.3.3.1 as discussed in Chapter 8.3.3.

PDO:Component_Constraint and ASO:Assembly_Constraint are identified as matching concepts according to similarity calculation using heuristics method as discussed in chapter 8.4

| Concept in Source Ontology (PDO) | Concept in Target Ontology (ASO) |
|---|---|
| *PDO:Assembly* | *ASO:Assembly* |
| *PDO:Component* | *ASO:Part* |
| *PDO:Component_constraint* | *ASO:Assembly_constraint* |

**Table 10 - 3 Matching Concepts between PDO and ASO**

Based on information of existing instance Insert_1 and its concept *PDO:Insert*, we need to find the matching concept that Insert_1 can be converted to.

As we already identified *ASO:Assembly_constraint* and *PDO:Component_constraint* in 8.3.2 as matching concepts, all the sub-concepts of *ASO:Assembly_constraint* will have inheritance similarity to *PDO:Insert*. To find which concept is the most appropriate concept that instance Insert_1 can be converted to, all the sub-concepts of *ASO:Assembly_constraint* are enumerated to find the one can have the highest attribute similarity with Insert_1.

Based on existing information about instances Insert_1, its geometry attribute and behavior attribute are listed in Table 10-4.

Note: according to axiom1 in chapter 7, an instance of *PDO:component_constraint* has at least one instance of *PDO:Model_item* as *PDO:has_component_reference* property and one instance of *PDO:Model_item* as *PDO:has_assembly_reference* property, both the properties are derived from "*part-of*" relation, according to the transitivity of "*attribute-of*" relation over "*part-of*" relation, geometry attributes of *PDO:Model_item* instances are also geometry attribute of Insert_1.

| **Instance** | **Geometry Attribute From** <br> ***PDO:has_component_reference*** | **Geometry Attribute From** <br> ***PDO:has_assembly_reference*** | **Behavior Attribute** |
|---|---|---|---|

| | | | |
|---|---|---|---|
| Insert_1 | *GDO:Cylindrical_Surface* | *GDO:Cylindrical_Surface* | GDO:Translate_z<br>GDO:Rotate_z |

**Table 10 - 4 Instance of Insert component constraint**

According to definitions and restrictions in ASO, subtypes of ASO:Assembly_constraint can have attributes as shown in Table 10-5

| **Concept** | **Geometry Attribute from ASO*:has_assembly_feature*** | **Geometry Attribute From ASO*:has_assembly_feature*** | **Behavior Attribute** |
|---|---|---|---|
| *ASO:Coincidence* | *GDO:Point* | *GDO:Point* | *GDO:Rotate_x,*<br>*GDO:Rotate_y,*<br>*GDO:Rotate_z* |
| | *GDO:Line* | *GDO:Line* | *GDO:Translate_z,*<br>*GDO:Rotate_z* |
| | *GDO:Plane* | *GDO:Plane* | *GDO:Translate_x,*<br>*GDO:Translate_y,*<br>*GDO:Rotate_z* |
| *ASO:Offset* | *GDO:Plane* | *GDO:Plane* | *GDO:Translate_x,*<br>*GDO:Translate_y,*<br>*GDO:Rotate_z* |
| *ASO:Contact* | *GDO:Cylindrical_surface* | *GDO:Cylindrical_surface* | *GDO:Translate_z,*<br>*GDO:Rotate_z* |
| | *GDO:Plane* | *GDO:Plane* | *GDO:Translate_x,*<br>*GDO:Translate_y,*<br>*GDO:Rotate_z* |

**Table 10 - 5 ASO:Assembly_constraint Attributes**

From Table 10-4 and Table 10-5, we can see that *ASO:Contact* can have similar attributes in both geometry and behavior attributes, so it can be concluded that *ASO:Contact* is the matching concept of *PDO:Insert* for instance Insert_1.

Similarly, it can be found that *ASO:Coincidence* can have the similar geometry attribute and behavior attributes as *PDO:Mate* so it is the matching concept of *PDO:Mate* for instance Mate_1.

## 10.6 Translated Product Data Semantics and Instance Mapping

During the translation process, after finding the matching concept in ASO for each of the source concept in PDO, instances in PDO are translated into their counterpart instances of ASO concepts. Table 10-6 and 10-7 list the new instances and the mapping between new instances and old instances. Table 10-8 lists the new relations between the new instances. Figure 10 –9 through Figure 10-16 show the sample OWL/RDF source code.

| **Concept** | **Instance** |
|---|---|
| ASO:Assembly | ASO:Asm001 |
| ASO:Part | ASO:Plate, ASO:Screw |
| ASO:Assembly_feature | ASO:Plate:Surface1, ASO:Plate:Surface2, ASO:Screw:Surface1, ASO:Screw:Surface2 |
| ASO:Assembly_constraint | ASO:Contact_1, ASO:Coincidence_1 |

**Table 10 - 6 New Instances in ASO**

| **Source Instance** | **Target Instance** |
|---|---|
| PDO:Asm001 | ASO:Asm001 |

| | |
|---|---|
| PDO:Plate | ASO:Plate |
| PDO:Screw | ASO:Screw |
| PDO:Plate:Surface1 | ASO:Plate:Surface1 |
| PDO:Plate:Surface2 | ASO:Plate:Surface2 |
| PDO:Screw:Surface1 | ASO:Screw:Surface1 |
| PDO:Screw:Surface2 | ASO:Screw:Surface2 |
| PDO:Insert_1 | ASO:Contact_1 |
| PDO:Mate_1 | ASO:Coincidence_1 |

**Table 10 - 7 Instance Mapping between PDO and ASO**

| Subject | Property | Object |
|---|---|---|
| ASO:Asm001 | ASO:has_part | ASO:Plate |
| ASO:Asm001 | ASO:has_part | ASO:Screw |
| ASO:Coincidence_1 | *ASO:has_assembly_feature* | ASO:Plate:Surface1 |
| ASO:Contact_1 | *PDO:has_assembly_feature* | ASO:Screw:Surface2 |
| ASO:Coincidence_1 | *ASO:has_assembly_feature* | ASO:Screw:Surface1 |
| ASO:Contact_1 | *ASO:has_assembly_feature* | ASO:Plate:Surface2 |

**Table 10 - 8 New Instance Relations**

```
        <ASO:Assembly rdf:ID="Asm001">
        </Assembly_feature >
        <ASO:Part rdf:ID="Plate">
        </Part>
        <ASO:Part rdf:ID="Screw">
        </Part>
```

**Figure 10 - 11 OWL/RDF Code of New Instances**

```
        <InstanceMatch rdf:ID="im1">
          <sourceInstance rdf:resource="#PDO:asm001"/>
          <targetInstance rdf:resource="#ASO:asm001"/>
        </InstanceMatch>
        <InstanceMatch rdf:ID="im1">
          <sourceInstance rdf:resource="#PDO:Plate"/>
          <targetInstance rdf:resource="#ASO:Plate"/>
        </InstanceMatch>
        <InstanceMatch rdf:ID="im1">
          <sourceInstance rdf:resource="#PDO:Screw"/>
          <targetInstance rdf:resource="#ASO:Screw"/>
        </InstanceMatch>
```

**Figure 10 - 12 OWL/RDF Code Of Instance Mapping**

```
        <ASO:Assembly_feature rdf:ID="ASO:Plate:Surface1">
        </Assembly_feature >
        <ASO: Assembly_feature rdf:ID="ASO:Plate:Surface2">
        </Assembly_feature >

        <ASO:Part rdf:ID="plate">
          <has_assembly_feature rdf:resource="#ASO:Plate:Surface1"/>
          <has_assembly_feature rdf:resource="#ASO:Plate:Surface2"/>
        </Part>
```

**Figure 10 - 13 OWL/RDF Code of New Part Instance**

```
        < rdf:Description rdf:about="Coincidence_1">
          <has_assembly_feature rdf:resource="#Plate_Surface1"/>
          <has_assembly_feature rdf:resource="#Screw_Surface2"/>
        </rdf:Description>
```

**Figure 10 - 14 OWL/RDF Code Instance Coincidence_1**

```
        < rdf:Description rdf:about="Contact_1">
          <has_assembly_feature rdf:resource="#ASO:Plate:Surface2"/>
          <has_assembly_feature rdf:resource="#ASO:Screw:Surface1"/>
        </rdf:Description>
```

**Figure 10 - 15 OWL/RDF Code of Instance Contact_1**

```
        < rdf:Description rdf:about="asm001">
          <has_part rdf:resource="#plate"/>
          <has_part rdf:resource="#screw"/>
          <has_assembly_constraint rdf:resource="#Coincidence_1"/>
          <has_assembly_constraint rdf:resource="#Contact_1"/>
        </ rdf:Description >
```

**Figure 10 - 16 OWL/RDF Code of Instance asm001**

## 10.7 Summary

The example illustrates a new semantic integration process. In the new semantic integration, product data semantics is captured to represent an engineer's understanding about a product design, which is based on existing CAD data model and pre-built knowledge base in engineering ontologies.

Using method of ontology mapping developed in this dissertation, product data semantics can be translated to another knowledge representation.. In this example, one product data semantics based on the knowledge representation of assembly design is translated into another product data semantics based on knowledge representation of assembly simulation.

One advantage of the new integration framework is the improved interoperability. Because different applications usually describe a product design using different knowledge representations, which lead to big difference in data type and data structure, in a traditional integration framework where information is exchanged purely in a data level, this results in the problem of heterogeneous data. Using the method proposed in this dissertation, based on the knowledge of the application that the data is generated from, data is first interpreted into product data semantic, and then it is translated into another product data semantics based on knowledge of the application that converted data will be used, by using ontology mapping. In the example presented in this chapter, data about assembly design is interpreted in product data semantics using assembly design knowledge, then the original product data semantics is translated into another product data semantics which interpret data based on assembly simulation knowledge, according to the mapping information between these two product data semantics, we know what data types can be converted from one to the other. During the process, no common knowledge is required to cover all the knowledge representations, and tasks that used to require human intervention now can be accomplished automatically, which will significantly improve the efficiency of integration.

# Chapter Eleven - Summary and Future Work

This dissertation seeks to improve the interoperability of CAD/CAE applications during PLM. In a CAD/CAE integration, we think the interoperability problem caused by heterogeneous data types in various applications, is rooted from the fact that different knowledge representations are used to interpret product data,. In order to solve the problem, and also improve composition adaptivity and viewport adaptivity of an integration framework, our proposed solution is to build knowledge to interpret product data as product data semantics and translate between different product data semantics. The following research questions are investigated:

1. How to represent knowledge in engineering design and analysis in a consistent, scalable manner.
2. How to generate product data semantics by associating the engineering design/analysis knowledge to the actual product data.
3. How to reconcile the differences in different product semantics and make the semantics translations, and eventually lead to data being translated correctly.

In order to answer the above questions, Chapter 5 shows how an engineer's understanding about a product design (product data semantics) is represented using the ontologies. Chapters 6 and 7 define the structure and syntax of engineering ontologies to represent the knowledge in engineering design and analysis. Chapter 8 and 9 discuss the method of translating different product data semantics based on different knowledge. A use case is introduced in Chapter 10 to show how different product data semantics are translated between assembly design and simulation tools.

## 11.1 Summary

To address the interoperability problem in CAD/CAE integration, one solution is to explicitly build engineering knowledge that is used during different design/analysis process into ontologies, and use the knowledge to interpret the product data. Using this method, some of the interoperability problem can be improved by automatically translating the interpretation from one representation to other representations so that applications in the integration framework can understand.

With the automatic translation process and layer structured ontology based on consistent, scalable ontology language, new knowledge representation can be easily built based on existing ontologies so to improve scalability and reusability. By mapping to other knowledge, interpretation of data based on one knowledge representation can be translated into interpretation based on other knowledge representation, and eventually help the data to be converted to the correct type.

In our solution, knowledge representation, product data semantics creation and translation are the key issues.

### 11.1.1 Knowledge Representation

Ontology is a consistent and scalable way to represent knowledge in engineering design and analysis. We use a three layered structure to model engineering ontology: general ontology, domain ontology and application ontology. In an ontology, concepts are related to other concepts by different types of relations. In this dissertation, three basic relations are considered: 1) *part-of*, which represents the composition relationship; 2) *is-a*, which represents the inheritance relationships; and 3) *attribute-of*, which represents the object-attribute relationship. In addition, axioms supplement the knowledge by defining the restrictions on the concepts and their relations.

As one of the most commonly accepted ontology languages and also an actual industrial standard, RDF/OWL is a powerful ontology language to model engineering design and analysis ontologies. The capability of serializing OWL/RDF in XML format makes it easier to be accepted by different applications. The wide availability of toolkits such as Protégé and Jena makes it possible to develop new functionalities.

### 11.1.2 Product Data Semantics Creation in Source

Product Data Semantics is used to describe product data using the concepts and relations defined in engineering ontologies.

To capture product data semantics, firstly concepts and relations that are used during the design/analysis are built into engineering ontologies. In this dissertation we use two ontologies PDO and ASO as examples to show how knowledge of different domain (assembly design and assembly simulation) are built into ontologies.

Secondly, to associate the product data to concepts, instances are created by using CAD/CAE APIs and ontology toolkits to instantiate concepts in an ontology with the actual product data.

### 11.1.3 Product Data Semantics Translation To Target

One of the most challenging problems in solving heterogeneous data is finding the correct target data representation for source data. To improve interoperability, our approach is to use built-in knowledge to interpret product data into product data semantics, and then by translating different product data semantics based on ontology mapping, so that source data that is interpreted in original product data semantics can be found its matching data interpretation, and eventually to help the source data to be converted to the correct target data representation.

To translate product data semantics, one of the most critical steps is ontology mapping, which is to map different concepts and relations in two ontologies. In this dissertation, mostly focused on integration of CAD/CAE applications, we propose a method of finding matching concepts for engineering ontologies in product design and analysis domains, two steps are used: explicit matching by definitions and matching based on heuristics method of similarity calculation. For the first step, definitions are given based on information exchange in a CAD/CAE integration process. In the latter step, compared to other heuristics method of ontology mapping which don't differentiate relation types, based on the characteristics of product data, three basic types of relations are used to calculate similarities so to find matching concepts: similarity based on "*part-of*" relationship, similarity based on "*is-a*" relationship, and similarity based on "*attribute-of*" relationship, and similarity based on specific attributes of product design are also discussed. An algorithm is developed for this purpose.

### 11.1.4 Key Contributions

In summary, compared to traditional data leve integration, our proposed semantic level integration has following  features:

1) Ontologies with layered structure to represent engineering knowledge using semantic web technologies.

2) Product Data Semantics as interpretation of product data based on knowledge represented in ontologies, which can be captured by instantiating concepts in ontologies using product data.

3) An algorithm of systematically finding mapping information in two steps: 1. Finding matching information based on definitions and 2. Finding matching information based on heuristics method. In step 2, a new algorithm is developed to calculate similarity based on different types of relations between concepts.

In the example scenario we used integration between an assembly design tool and an assembly simulation tool to show the new semantic integration method. Theoretically, semantic level integration can be used for any applications in CAD/CAE that is focused on product data, including design/analysis/simulation, to integrate different applications, different attributes especially behavior attributes may need to be defined for similarity calculation.

## 11.2 Future Work

The proposed solution in this dissertation tries to address the interoperability problem in a CAD/CAE framework and improves the communications process between different applications. However, there are still many problems need to be addressed and further investigated:

1) Deployment adaptivity. As the engineering application in a CAD/CAE framework can be very different. For example, one application may require real-time

collaboration, whereas another may require a strict, well-defined, procedure-based communication. Also in some cases the deployment is based on sharing just the product data (PDM model) whereas others require the complete integration of the engineering, manufacturing, and business processes (ERP model).  Any resulting system based on current integration methods is very rigid in terms of deployment: deployment strategies often cannot be modified or replaced without subsequent rounds of negotiation and programming. There is an increasing demand for flexible deployment strategy that can support multiple deployment possibilities.

2) Representation of difference in ontology mapping for complex concepts. In this dissertation we mainly focus on finding the mapping between basic design entities, and it is also very important to find the difference between the two similar concepts so to help the translation. For example, the difference in part/sub-assembly order information between an assembly design hierarchy (as designed) and an assembly simulation hierarchy (as manufactured).

Overall, semantic level integration improves interoperability and adaptivity of integration in several aspects. There remains more ongoing research to make further improvement.

# Appendix – Acronyms

**ASO: Assembly Simulation Ontology**

**BDE: Basic Design Entity**

**CAD: Computer Aided Design**

**CAE: Computer Aided Engineering**

**DOF: Degree of Freedom**

**GDO: General Design Ontology**

**PDM: Product Data Management**

**PDO: Product Design Ontology**

**PLM: Product Lifecycle Management**

# Appendix – RULES for Ontology Mapping

**Rule 5.5.1 for three concepts C1 and C2 and C3, if C1 has_Child C2 and C2 has_Attribute C3, then it can be deducted that C1 has_Attribute C3.**

**Rule 8.3.1.1 If two concepts are equivalent, then they are matching concepts.**

**Rule 8.3.2.1 if two concepts share the equivalent instance, then the two concepts are two matching concepts.**

**Rule 8.3.3.1 During data conversion, if the attribute of an instance of a BDE is converted to be the attribute of an instance of another BDE, then both Basic Design Entities share the same attribute.**

**Rule 8.4.2.1 For a given source concept A, if a target concept A' has attribute similarity** $Sim_1(A, A')$ **, and if a Concept C'** ***has_Child*** **A', then Concept C' and A have the same attribute similarity** $Sim_1(A, C') = Sim_1(A, A')$

**Rule 8.4.3.1 if there are two matching concepts A and B which belong to each one of the hierarchies, then all the concepts Ai and Bi that are sub-concepts of A and B will have inheritance similarity** $Sim_1(A_i, B_i) = Sim_1(A, B)$ **and** $Sim_2(A_i, B_i) = Sim_2(A, B)$

**Rule 8.5.1.1 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 8.5.1.2 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 8.5.2.3 If both applications are regarding 3-D feature-based product design, then designCreationBehavior can be used as common behavior attribute.**

**Rule 8.5.2.4 If both applications share a common definition in functionalBehavior, then that definition may be used as behavior attribute.**

# Appendix – Definitions

**Definition 8.1**: **Mutual Information**: Mutual information refers to the information shared by two instances.

**Definition 8.2**: **Instance Similarity**:

$I(A \cap B)$ : Mutual information of instance A and B

$I(A \cup B)$ : Union of information of instance A and B

Instance Similarity between A and B is: $Sim(A,B) = \dfrac{I(A \cap B)}{I(A \cup B)}$

**Definition 8.3**: **Matching Instances**: A pair of instances that have mutual information and also have the highest similarity are defined as Matching Instances, and during translation, the matching instances can be translated from one to another.

**Definition 8.4**: **Convertible Instance**: If Instance A and Instance B are Matching Instances, during CAD/CAE integration, and information in Instance A can be translated to information in Instance B, then we call Instance B a convertible instance of B ($Convertible(B,A)$).

**Definition 8.5**: **Matching Concepts**: For two Matching Instances, their corresponding concepts are defined as Matching Concepts.

**Definition 8.6**: **Concept Similarity**: among all the instances $I_{1i}$ of concepts C1 and $I_{2i}$ of C2, the highest instance similarity between two instances $Max(Sim(I_{1i}, I_{2i})$ is defined as concept similarity of C1 and C2.

**Definition 8.7**: **Convertible Concept**: For two instances, A and B, which belong to concept $C_A$ and $C_B$ respectively, if A is a convertible instance of B, then the concept $C_A$ is a convertible concept of $C_B$.

**Definition 8.8**: **Mismatching Instances**: Mismatching Instances do not share mutual information.

**Definition 8.9**: **Mismatching Concepts**: Mismatching Concepts do not have any instances that have concept similarity > 0.

**Definition 8.10**: **Composition Path** is a path that only consists of composition relations

**Definition 8.11**: **Similar Composition Path** for two composition paths, if all the nodes in one composition path have the same attribute similarity to all the nodes in the other composition path, then we define the two paths as similar composition paths.

Bibliography

[1] N.Senin, D.R.Wallace, and N.Borland, "Distributed Object-Based Modeling in Design Simulation Marketplace," *Journal of Mechanical Design*, vol. 125, pp. 2-13, Mar.2003.

[2] F.Mervyn, A.Senthil Kumar, S.H.Bok, and A.Y.C Nee, "Developing distributed applications for integrated product and process design," *Computer Aided Design*, vol. 36, pp. 679-689, 2004.

[3] X.W.Xu and Q.He, "Striving for a total integration of CAD,CAPP,CAM and CNC," *Robotics and Computer-Integrated Manufacturing*, vol. 20, pp. 101-109, 2004.

[4] S.Q.Xie, P.L.Tu, D.Aitchison, D.Aitchison, R.Dunlop, and Z.D.Zhou, "A WWW-based integrated product development platform for sheet metal parts intelligent concurrent design and manufacturing," *International Journal of Production Research*, vol. 39, pp. 3829-3852, 2001.

[5] X.F.Zha and H.Du, "A PDES/STEP-based model and system for concurrent integrated design and assembly planning," *Computer Aided Deisgn*, vol. 34, pp. 1087-1110, 2002.

[6] J. Shah and P. Wilson, "Analysis of Design Abstraction, Representation, and Inferencing Requirements for Computer Aided Design," *Journal of Design Studies*, vol. 10, no. 3, pp. 169-178, 1989.

[7] W. Regli, X. Hu, M. Atwood, and W. Sun, "A Survey of Design Rationale Systems : Approaches, Representation, Capture and Retrieval," *Engineering With Computers*, vol. 16, pp. 209-235, 2000.

[8] O. Eris, P. Hansen, A. Mabogunje, and L. Leifer, "Toward a Progmatic Ontology for Product Development Projects in Small Teams," *Procedding of the International Conference on Engineering Design (ICED 99)*, 1999.

[9] C. Dym, *Engineering Design: A synthesis of View*. New Youk, NY: Cambridge University Press, 1995.

[10] Y.M.Deng, Y.C.Lam, S.B.Tor, and G.A.Britton, "A CAD-CAE integrated injection modeling design system," *Engineering with Computer*, vol. 18, no. 1, pp. 82-90, 2002.

[11] J.J.Shah, *Parametric and Feature-Based CAD/CAM: Concepts, Techniques, and Applications* Wiley-Interscience, 1995.

[12] H.Shin, G.J.Olling, Y.C.Chung, B.H.Kim, and S.K.Cho, "An integrated CAPP/CAM system for stamping die patttern machining," *Computer Aided Design*, vol. 35, no. 2, pp. 203-213, 2003.

[13] O. W. Salomons, F.J.A.M.van Houten, and H.J.J.Kals, "Review of research in feature-based design," *Journal of Manufacturing Systems*, vol. 12, no. 2, pp. 113-132, 2003.

[14] H. F. Wang and Y.-L.Z, "*CAD/CAM integrated system in collaborative development environment*," *Robotics and Computer Integrated Manufacturing*, vol. 18, pp. 135-145, 2002.

[15] S. Szykman, S. J. Fenves, S. B. Shooter, and W. Keirouz, "A Foundation for Interoperability in Next generation Product Development Systems," *Computer Aided Design*, vol. 33, no. 7, pp. 545-559, 2001.

[16] S. J. Fenves, S. Sriram, and R. Wang, "A Product Information Modeling Framework for Product Lifecycle Management," *International Symposium on Product Lifecycle Management*, 2003.

[17] D. Xue and H.Yang, "A concurrent engineering-oriented design database representation model," *Computer Aided Design*, vol. 36, no. 10, pp. 947-965, 2004.

[18] S.R.Gorti, A.Gupta, G.J.Kim, R.D.Sriram, and A.Wong, "An object-oriented representation for product and design process," *Computer Aided Design*, vol. 30, no. 7, pp. 489-501, 1997.

[19] S. B. Shooer, W. Keirouz, S. Szykman, and S. J. Fenves, "A Model of the Flow of Design Information in Product Development," *Engineering With Computers*, vol. 16, pp. 178-194, 2001.

[20] International Organization for Standardization (ISO), "Industrial Automation Systems, Standard for the Exchange of Product Model Data (STEP)," Geneva, Switzerland: ISO 10303, 1994.

[21] H. Kahn, Nick Filer, Alan Williams, and Nigel Whitaker, "A generic framework for transforming EXPRESS information models," *Computer Aided Deisgn*, vol. 33, pp. 501-510, 2001.

[22] M.J.Pratt and B.D.Anderson, "A shape modelling applications programming interface for the STEP standard," *Computer Aided Design*, vol. 33, pp. 531-543, 2001.

[23] D. T. De Martino, B. Falcidieno, and S. Habinger, "Design and engineering process integration through a multiple view intermediate modeller in a distributed object-oriented system environment," *Computer Aided Design*, vol. 30, no. 6, pp. 437-452, 1998.

[24] T. Mannisto, Hannu Peltonen, Asko Martio, and Reijo Sulonen, "Modelling generic product structures in STEP," *Computer Aided Design*, vol. 30, no. 14, pp. 1111-1118, 1998.

[25] X. F. Zha and H.Du, "A PEDS/STEP-based model and system for concurrent integrated design and assembly planning," *Computer Aided Design*, vol. 34, pp. 1087-1110, 2002.

[26] C.D.Cera, W.C.Regli, I.Braude, Y.Shapirstein, and C.V.Foster, "A collaborative 3D environment for authoring design semantics," *Compuer Graphics and Applications, IEEE*, vol. 22, no. 3, pp. 43-55, 2002.

[27] "http://en.wikipedia.org/wiki/Ontology_%28computer_science%29," 2007.

[28] T. R. Gruber, "Toward Principles of the Design of Ontologies Used for Knowledge Sharing," *International Workshop on Formal Ontology in Conceptual Analysis and Knowledge Representation*, 1993.

[29] R. Fikes and A. Farquhar, "Distributed Repositories of Highly Expressive Reusable Ontologies," *IEEE Intelligent System*, vol. 14, no. 2, pp. 73-79, 1999.

[30] "Wine Ontology," 6 A.D..

[31] I.Horvath, J.S.M.Vergeest, and G.Kuczogi, "Development and Application of Design Concept Ontologies for Contextual Conceptualization," 1998.

[32] Y.Kitamura and R.Mizoguchi, "Ontology-based Description of Functional Design Knowledge and its Use in a Functional Way Server," *Expert System Application*, vol. 24, no. 2, pp. 153-166, 2002.

[33] J.J.Michel and A.F.Cutting-Decelle, "The Process Specification Language," 2004.

[34] Lalit Patil, Debasish Dutta, and Ram Sriram, "Ontology-Based Exchange of Product Data Semantics," *IEEE Transactions On Automation Science And Engineering*, vol. 2, no. 3, pp. 213-225, 2005.

[35] M. Ciocoiu and D. S. Nau, "Ontology Based Semantics," *Seventh International Conference on Principles of Knowledge Representation and Reasoning*, 2000.

[36] Y. Nomaguchi, A. Ohnuma, and K. Fujita, "Design Rationale Acquisition in Conceptual Design by Hierachical Integration of Action, Model and Argumentation," *Proceeding of the 2004 ASME Design Engineering Technical Conference*, 2004.

[37] G. Mocko, Richard Malak, Christiaan Paredis, and Russel Peak, "A knowledge repository for behavioral models in engineering design," *2004 ASME Design Engineering Technical Conferences*, 2004.

[38] C. Kerr, Rajkumar Roy, and Peter J.Sackett, "A product ontology for automotive seat specification," *2004 ASME Design Engineering Technical Conferences*, 2004.

[39] M. Ciocoiu, D. S. Nau, and M. Gruninger, "Ontologies for Integrating Engineering Applications," *Journal of Computing and Information Science in Engineering*, vol. 1, pp. 12-22, Mar.2001.

[40] M. Gruninger, C. Schlenoff, and A. Knutilla, "Using Process Requirement as the Basis for the Creation and Evaluation of Process Ontologies fro Enterprise Modeling," *ACM SIGGROUP Bulletin Special Issue on Enterprise Modelling*, vol. 18, no. 3 1997.

[41] C. Schlenoff, A. Knutilla, and S. Ray, "Unified Process Specification Language: Requirements for Modeling Process," Technical Report NISTIR 5910, Gaithersburg, MD,1996.

[42] C.I.Kerr, R.Rajkumar, and P.J.Sackett, "A Product Ontology for Automotive Seat Specification," 2004.

[43] W3C, "W3C Semantic Web," *Available online via http://www. w3. org/2001/sw/> [accessed June 2004]*, 2001.

[44] Grigoris Antoniou and F.V.H., *A Semantic Web Primer* MIT Press, 2004.

[45] W3C, " Resource Description Framework (RDF)," *Available online via <http://www. w3. org/RDF/> [accessed June 2004]*, 2004.

[46] S. Powers, *Practical RDF*, O'Reilly, 2003.

[47] S. Ray, "Interoperability Standards in the Semantic Web," *Journal of Computing and Information Science in Engineering*, vol. 2, pp. 65-69, Mar.2002.

[48] W3C, " OWL Web Ontology Language Overview," *Available online via <http://www. w3. org/TR/owl-features/> [accessed June 2004]*, 2004.

[49] "Jena - A Semantic Web Framework for Java," *Available online via <http://jena. sourceforge. net/> [accessed June 2004]*, 2004.

[50] "The Protege Ontology Editor and Knowledge Acquisition System," *Available online via <http://protege. stanford. edu/> [accessed June 2004]*, 2005.

[51] I.Niles and A.Pease, "Towards a standard upper ontology," 2001.

[52] A. Gangemi, N. Guarino, C. Masolo, and A. Oltramari, "Sweetening wordnet with DOLCE," *AI Magazine*, vol. 24, no. 3, pp. 13-24, 2003.

[53] M.Gruninger and J.Kopena, "Semantic integration through invariants," 2003.

[54] N.F.Noy and M.A.Musen, "The PROMPT suite: Interactive tools for ontology merging and mapping," *International Journal of Human-Computer Studies*, vol. 59, no. 6, pp. 983-1024, 2003.

[55] J.Euzenat and P.Valtchev, "Similarity-based Ontology alignment in OWL-Lite," 2004.

[56] G.Summe and A.Madche, "FCA-Merge: Bottom-up merging of ontologies," 2001.

[57] T. Tudorache, "Employing Ontologies for an Improved Development Process in Collaborative Engineering." 2006.

[58] Saeema Ahmed, Sanghee Kim, and Ken M.Wallace, "A Methodology for Creating Ontologies for Engineering Design," 2005.

[59] T.Berners-Lee, "Semantic Web on XML,"2000.

[60] A.Swartz and J.Hendler, "The Semantic Web: A Network of Content for the Digital City," 2001.