Verification of Knowledge Shared across Design and Manufacture Using a Foundation Ontology

By

Najam Akber Anjum

Under the supervision of
Dr. Jenny Harding &
Dr. Bob Young

A Doctoral Thesis

Submitted in partial fulfillment of the requirements
for the award of

Doctor of Philosophy of Loughborough University

August 2011

CERTIFICATE OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this thesis, that the original work is my own except as specified in acknowledgments or in footnotes, and that neither the thesis nor the original work contained therein has been submitted to this or any other institution for a degree.

…… (Signed)

Najam Akber Anjum

…… (Date)

Abstract

Seamless computer-based knowledge sharing between departments of a manufacturing enterprise is useful in preventing unnecessary design revisions. A lack of interoperability between independently developed knowledge bases, however, is a major impediment in the development of a seamless knowledge sharing system. Interoperability, being an ability to overcome semantic and syntactic differences during computer-based knowledge sharing can be enhanced through the use of ontologies. Ontologies in computer science terms are hierarchical structures of knowledge stored in a computer-based knowledge base. Ontologies have been accepted by all as an interoperable medium to provide a non-subjective way of storing and sharing knowledge across diverse domains. Some semantic and syntactic differences, however, still crop up when these ontological knowledge bases are developed independently. A case study in an aerospace components manufacturing company suggests that shape features of a component are perceived differently by the designing and manufacturing departments. These differences cause further misunderstanding and misinterpretation when computer-based knowledge sharing systems are used across the two domains. Foundation or core ontologies can be used to overcome these differences and to ensure a seamless sharing of knowledge. This is because these ontologies provide a common grounding for domain ontologies to be used by individual domains or department. This common grounding can be used by the mediation and knowledge verification systems to authenticate the meaning of knowledge understood across different domains. For this reason, this research proposes a knowledge verification framework for developing a system capable of verifying knowledge between those domain ontologies which are developed out of a common core or foundation ontology. This framework makes use of ontology logic to standardize the way concepts from a foundation and core-concepts ontology are used in domain ontologies and then by using the same principles the knowledge being shared is verified. The Knowledge Frame Language which is based on Common Logic is used for formalizing example ontologies. The ontology editor used for browsing and querying ontologies is the Integrated Ontology Development Environment (IODE) by Highfleet Inc. An ontological product modelling technique is also developed in this research, to test the proposed framework in the scenario of manufacturability analysis. The proposed framework is then validated through a Java API specially developed for this purpose. Real industrial examples experienced during the case study are used for validation.

Keywords: Foundation ontologies, domain ontologies, knowledge verification, ontology mediation, Common Logic, Manufacturability analysis

Dedication

To ammi, pappa

And

Mariya

Acknowledgements

I thank first of all, Allah the Almighty, the One, Who blessed me with the knowledge I possess today.

I am, then, most thankful to my supervisors Dr. Jenny Harding and my co-supervisor Dr. Bob Young for providing me the most needed guidance and for their constructive criticism which made possible the completion of this research and the production of this thesis.

I am grateful to IMCRC for funding my studies at Loughborough University, to our industrial collaborators who allowed me to spend some time in their manufacturing facility during my secondment, and to people at Highfleet Inc. for their support in building an understanding of the ontological formalism used in this research.

I am most appreciative of the support my parents provided me through continuous encouragement despite bearing a tough life, both financially and emotionally, due to my studies. I am also thankful to my wife for patiently living a lonely life during the undue extensions of my PhD in the last stages.

I must also say a big thank you to Tish for helping me in building the understanding of KFL and IODE and to Rahul for helping me in learning Java.

I also owe my humble gratitude to all the other group members including Zahid, George, Claire, and Keith and all the other friends, staff and students of Loughborough University who supported me in any way during my studies.

Abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
CG	Conceptual Graphs
CGIF	Conceptual Graph Interchange Format
CIM	Computation Independent Model
CL	Common Logic
CLIF	Common Logic Interchange Format
DAML	DARPA Agent Markup Language
DARPA	Defense Advanced Research Projects Agency
DL	Description Logic
DOLCE	Descriptive Ontology for Linguistic and Cognitive Engineering
EIF	European Interoperability Framework
FOL	First Order Logic
IC	Integrity Constraint
ICT	Information and Communications Technology
IDEF	Integration DEFinition
IMKS	Interoperable Manufacturing Knowledge System
IODE	Integrated Ontology Development Environment
KFL	Knowledge Frame Language
KIF	Knowledge Interchange Format
MAFRA	Mapping FRAmework
MDA	Model Driven Architechture
MOF	Meta Object Facility
NIST	National Institue of Standards and Technology

NLP	Natural Language Processing
ODM	Ontology Definition Metamodel
OIL	Ontology Inference Layer
OMG	Object Management Group
ONION	ONtology compositION
OWL	Web Ontology Language
PIM	Platform Independent Model
PSL	Process Specification Language
PSM	Platform Specific Model
QOM	Quick Ontology Mapping
QVT	Query View Transformation
RDF	Resource Description Framework
RL	Reference Line
RP	Reference Point
SBO	Semantic Bridge Ontology
SCL	Simple Common Logic
SMIF	Semantic Manufacturing Interoperability Framework
STEP	STandard for the Exchange of Product model data
SUMO	Suggested Upper Merged Ontology
SUO	Standard Upper Ontology
UML	Unified Modelling Language
UNSPSC	United Nation Standard Products and Services Code
VMO	Verification Meta Ontology
XML	Extensible Markup Language

Glossary of Terms

Core-concepts ontology: A core-concepts ontology is more specific than the foundation ontology but very general as compared to a domain ontology. For example a manufacturing core-concepts ontology will have all the concepts related to the whole product lifecycle. Concepts from this ontology then can be used to build a domain ontology for design, production, assembly, etc.

Domain ontology: Domain ontologies provide vocabularies about concepts within a domain and their relationships, about the activities taking place in that domain, and about the theories and elementary principles governing that domain.

Foundation ontology: Foundation ontology describes very general concepts and provides general notions under which all root terms in existing ontologies should be linked.

Ontology alignment: An automated or semi-automated discovery of correspondences between two ontologies.

Ontology articulation: A way in which the fusion or merging of ontologies has to be carried out.

Ontology mapping: A process in which concepts in two or more ontologies are connected with a relation.

Ontology mediation: The process of reconciliation between two or more ontologies.

Ontology merging: The process of creating a new ontology which is the union of source ontologies for the purpose of obtaining a bigger and richer knowledge base.

Query: A query is an expression evaluated over a model.

Transformations: performed to build a target model from a source model so that the source and target models are compatible according to the relations defined.

View: A model completely derived from another model.

Table of Contents

Chapter 1: Introduction

1.1. Research introduction

Ever soaring competition and ever increasing customer demands have forced manufacturers to identify new ways of improving quality, reducing costs and shrinking the time to market their products. In order to get the product design 'right first time', instead of going through several cycles of design modification to get the final product, manufacturers need to avoid making mistakes in the first place. This prevention of errors, in turn, requires all the departments of a manufacturing enterprise to work in unison and produce a design suitable for all the product lifecycle stages. The designer of the product, therefore, needs to be aware of all the complexities of manufacturing, inspection, assembly, maintenance, use and disposal of the product to be designed. This task is usually performed by bringing key people from all the stakeholder departments to the table to finalize the product design through critical analyses and technical discussions. The present age of computers has made this task easier by allowing product designers to browse computer-based networked knowledge bases for all the technical issues before finalizing the design. On a larger scale, there can be Information and Communications Technology (ICT) based inter-enterprise cooperation for sharing knowledge and information. For this to happen, however, independently developed knowledge management systems need to be compatible with each other. In computer science research, this problem of compatibility is studied under the title of interoperability. The European Interoperability Framework (EIF) defines interoperability as "the ability of information and communication technology (ICT) systems and of the business processes they support to exchange data and to enable the sharing of information and knowledge" (European Communities, 2004). Networked businesses today encounter recurring difficulties due to limitations in interoperability between enterprise systems (Panetto and Molina, 2008). Similar interoperability problems occur when departments within a manufacturing enterprise share information and knowledge among themselves through ICT-based knowledge management systems. Such systems assist in organizing information and management, but their ability to represent and share manufacturing knowledge is very limited (Young et al, 2010).

The necessary functionality of interoperable systems for 'enabling the sharing of information and knowledge' requires the knowledge management systems to overcome several types of incompatibilities and heterogeneities between sets of knowledge residing in

independently developed computer-based knowledge management systems. These differences occur because engineers working in different parts of the organization or different groups, with time, develop their own vocabulary for particular issues, elements or activities and this results in different information models (Lin and Harding, 2007). Two types of incompatibilities may occur. Semantic incompatibility occurs when the same word is interpreted differently by the two parties involved in knowledge sharing and syntactic incompatibility occurs due to the use of a different terminology to represent the same thing (Lin et al, 2004). To overcome these differences, ontologies are used. Ontologies, provide the basic structure or armature around which knowledge bases can be built (Devedzic, 2002). The word ontology may be confused with the same word in philosophy where it means the study of the kinds of things that exist (Chandrasekaran et al, 1999). In computer science studies, ontology is a hierarchical structure of terms. Essentially, it includes a vocabulary of terms and some specification of their meaning (Uschold and Jasper, 1999). The use of ontologies does not completely alleviate the problem of semantic heterogeneity but it helps in designing systems which can do so. Knowledge bases are therefore built around ontologies and in the event of knowledge sharing these ontologies first interact to reach an agreement on the meaning of the terms used in the knowledge bases. Once that agreement is reached, the knowledge is shared seamlessly. Reaching an agreement over the meaning of a term, however, is a contentious issue. In the field of knowledge management the resolution of this issue is known as knowledge verification and is the main focus of this thesis.

Verification of knowledge requires the overcoming of semantic and syntactic mismatches between the ontologies. This is usually achieved by one of two ways. In the first case a general upper ontology (also called a foundation ontology) is agreed upon by ontology developers, who then extend this general ontology with concepts specific to their field. The extensions of this foundation can be called domain ontologies if the concepts they contain are more specific to a domain of interest. Finding correspondence between these domain ontologies is easier, if their development is performed in a way consistent with the definition in the upper or foundation ontology (Noy, 2004). The other method of finding similarities involves the analysis of various characteristics of the two ontologies. These characteristics may include the structure of ontology, definitions of concepts, and instances

of classes (Noy, 2004). Ontological concepts having similar characteristics are then declared similar. This process of finding similarities between two ontologies is also called 'ontology matching'. The existing tools for ontology matching are mostly based on the second method. These tools, however, are limited in their automation and accuracy of matching results. As a result, a significant level of human intervention is required to successfully match ontologies which is extremely cumbersome and time consuming (Anjum et al, 2010). The foundation or upper ontologies, on the other hand, provide a basic platform for the ontology builders to commit to and this makes the process of ontology matching more automatic and accurate, which is the main assumption of this research. Since foundation and domain ontologies are the two most important concepts from the point of view of this research, they are explicitly defined below:

Foundation ontology *'describes very general concepts and provides general notions under which all root terms in existing ontologies should be linked'* (Gomez-Perez et al, 2004).

Domain ontologies *'provide vocabularies about concepts within a domain and their relationships, about the activities taking place in that domain, and about the theories and elementary principles governing that domain'* (Gomez-Perez et al, 2004).

The research explained in this thesis proposes a framework for matching two domain ontologies for the purpose of knowledge verification. The proposed framework is tested on domain ontologies of production and design concepts specialized from a manufacturing foundation ontology. The following text further describes the background, scope and significance of this research.

1.2. Research background and scope

As shown in figure 1.1, this research belongs to an area which lies at the juncture of three main research areas including computer systems interoperability, concurrent engineering in manufacturing, and knowledge management. The following descriptions of each of these areas aim to further narrow down the scope of this research.

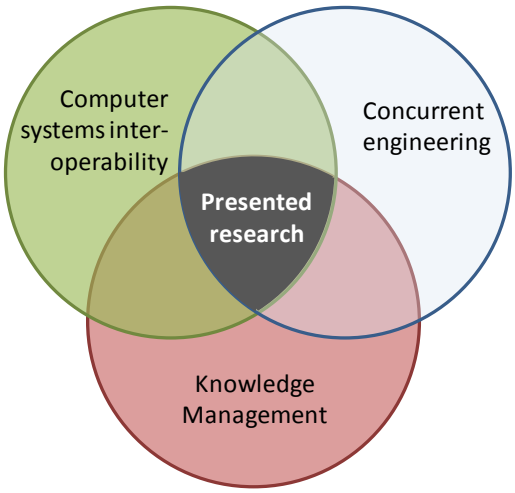


Figure 1.1. Scope of research

1.2.1. Computer Systems Interoperability

Within the research area of computer systems interoperability, the presented research lies in the region of 'mediation of ontological knowledge sources'. More specifically, it targets the reconciliation of those ontologies which are developed independently of each other but are committed to a single foundation and core concepts ontology. The terms mediation and reconciliation are used here to define a process where concepts existing in different ontologies are compared with each other to find similarities. The similarities established through this process help in a seamless and correct transfer of information and knowledge between ontological knowledge sources. The process of reconciliation and similarity finding requires the semantic and syntactic differences between ontologies to be overcome. In the presented research the ontological formalism i.e. the syntax used to build ontologies, is fixed as Common Logic. This fixation of the ontology development formalism leaves only the semantic difference to be tackled during the process of ontology matching. These semantic differences are the main focus of this research.

1.2.1. Concurrent Engineering

The methodology of concurrent engineering aims to simultaneously detect and consider manufacturability conflicts and constraints at early design stages (Li and Shen, 2009). Within the concurrent engineering research paradigm, this research is limited to the application of concurrent engineering in manufacturing. Concurrent engineering in this sense, involves a thorough scrutiny of the design of an engineering product, by the manufacturing people, during the early stages of its inception. It is assumed that previously stored and formalized manufacturing knowledge is used by the designer to optimize the design. This is where the research touches the boundaries of knowledge management.

1.2.2. Knowledge Management

Nonaka's SECI cycle of knowledge management divides the process of knowledge creation into four distinct stages. These stages are Socialization, Externalization, Combination and Internalization, hence SECI (Nonaka, 1994). The research explained in this thesis does not deal with the process of Socialization where face to face interaction of people results in knowledge transfer. Rather it remains within the boundaries of ICT-based knowledge management where the other three stages of knowledge creation are dealt with.

The process of Externalization, in the industrial scenario considered here, means that the product manufacturing knowledge is captured in the form of manufacturability rules. This externalized knowledge is then formalized in the form of ontological integrity constraints hence the process of Combination. This organized knowledge is then made a part of the manufacturability checking of the product design by making it available to the designer through an interoperable manufacturing knowledge system. Through this knowledge system, the product designers make use of this knowledge and create new knowledge of their own which is the process of Internalization.

This research involves all three dimensions, but since it aims to research an interoperable knowledge system, its main contribution is in the area of Externalization and Combination. Thus the idea is that the manufacturability knowledge is to be made available in a form understandable by all independently developed knowledge systems hence the word interoperability. More specifically, this research looks at those cases where a semantic mismatch may occur during the process of manufacturability knowledge sharing between design and manufacture. At this point it is also important to highlight the fact that this research is a part of a bigger research project entitled Interoperable Manufacturing Knowledge System (IMKS). In order to further clarify the scope of the presented research, the IMKS project is discussed in the next section.

1.3. The IMKS project

IMKS investigates the potential of foundation ontologies and accompanied verification mechanisms for improving the interoperability of knowledge management software. More specifically, the project addresses the task of developing flexible systems that can share manufacturing knowledge across the domains of product design and manufacturing planning (Young et al, 2010). Figure 1.2 depicts the setting in which this research took place. A foundation ontology with core manufacturing concepts is developed to provide a platform for domain ontologies to be based on. The design domain and manufacturing domain ontologies are then developed by using concepts from the foundation ontology. A verification system then mediates the two ontological knowledge sources in order to provide a seamless knowledge transfer across the two domains. This research is therefore

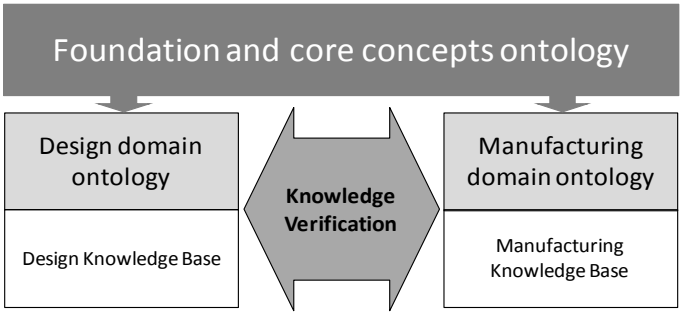


Figure 1.2. Interoperable Manufacturing Knowledge System (IMKS)

aimed at finding ways to verify knowledge when it is shared across design and manufacture in a scenario where domain ontologies are built out of a common foundation.

1.4. Significance of this research

Before the significance of this research is discussed, it is useful to clearly define it first as follows:

"This research involves the development of methods to overcome semantic differences during the process of knowledge sharing between design and manufacture ontological knowledge sources based on a common foundation ontology."

The term used in this research to define this overcoming of semantic differences is 'knowledge verification'.

The title of this research is therefore:

"Verification of knowledge shared across design and manufacture using a foundation ontology"

The significance of this research is first of highlighted by the importance of ontologies in ICT. Ontologies have been shown to be valuable in many research contexts for computer-based communication (Lin and Harding, 2007; Khilwani et al, 2009; Mascardi et al, 2008). Hence, this work is of general interest in the area of ontology research. In a more focused view, the significance of this research is proven from the research gap that exists in the area of automatic ontology mediation. A comprehensive literature review, presented in chapter 4, shows that the existing tools and methodologies for ontology mediation and matching require a fair amount of human intervention for the results to be accurate. This is extremely cumbersome and time consuming. The fundamental reasons for this high human

involvement are the mismatches in the way concepts are defined in ontologies and the way ontologies are constructed. These mismatches may lead to incorrect interpretation of knowledge associated to concepts within the ontologies. To overcome this problem, verification methods are needed which authenticate the true meaning of concepts when knowledge is shared across different domains of interest. When it comes to knowledge sharing for the purpose of concurrent engineering in manufacturing, it becomes essential that a system is present which allows a seamless transfer of manufacturability knowledge to the product designer in order to obtain an optimum design. This seamless knowledge transfer requires verification methods which are extremely accurate with minimal human assistance. This research therefore attempts to find ways to verify the authenticity and correctness of knowledge during the process of knowledge sharing across different domains. This will be shown in chapter 8 and 9 of this thesis. The two domains selected in this research to experiment with the proposed ideas are engineering design and engineering manufacture. Experimental domain ontologies in these two areas are developed and a formalized form of these ontologies can be found in the appendices.

Another dimension of the work undertaken in this research is the use of foundation ontologies. Work on several foundation ontologies can be found in the literature (Gruninger, 2004; Matuszek et al, 2006; Niles and Pease, 2001) but a comprehensive methodology for a domain ontology builder to commit to a particular foundation ontology still needs detailed description. Foundation ontologies are a step towards standardizing the domain ontology building task which is very useful in making the independently built knowledge bases interoperable. Foundation ontologies are discussed in chapter 3 and some work related to foundation and upper ontologies is reviewed in chapter 4 of this thesis.

This research is motivated by the need for achieving interoperability through knowledge authentication and verification during the process of knowledge sharing in a foundation and domain ontology based system of knowledge bases. Keeping in view the scope of this research (section 1.2) the following research questions are therefore asked:

a. How can semantic mismatches between domain ontologies based on a common foundation ontology be prevented or, if that is not possible, be addressed during the process of computer aided ontology matching?

b. How can knowledge be verified by preventing or addressing semantic mismatches when sharing knowledge between ontological knowledge sources belonging to different domains?

In this research, the second research question is answered within the field of engineering manufacturing and this will be shown in chapter 9 of this thesis. Furthermore, a general solution to the problem of semantic mismatches during the process of ontology matching is proposed in chapter 8 by answering the first question. The aims and objectives resulting out of these research questions are detailed next.

1.5. Aims and objectives

As stated earlier, being part of a bigger project, this research started from a point where some of the major objectives were already defined by the IMKS project. These objectives included:

1. To investigate the application of Model Driven Architectures and ontological formalisms in order to identify rigorous methods for the definition of libraries of 'world' objects, processes and relationships which meet the needs of interoperation in a multi-disciplinary manufacturing knowledge sharing environment.
2. To apply these methods to engineer manufacturing capability ontologies which can both capture best practice manufacturing methods and enable their sharing across the key manufacturing domains of design for manufacture, manufacturing planning and repair.
3. To explore innovations in mapping between 'world' and 'domain' models that facilitate flexible, reconfigurable and verifiable intelligent interoperation.
4. To deploy and evaluate the combination of results from objectives 1, 2 and 3 through experimental work within the context of our collaborators product development environments.

Some constituents of these aims of IMKS set the boundaries for this research. These include:

1. Use of ontological formalisms to develop a library of objects, processes and relationships.

2. Use of mapping to connect domains and world models for verified knowledge sharing.
3. Use of collaborator's product development environment which in the case of ontological formalism is IODE - the ontology editor that has been used to handle ontologies in the IMKS project.

In the light of these IMKS objectives and boundaries the aims set for this doctoral research were as follows:

1. To explore the application of ontology matching for the verification of knowledge shared between ontology-based knowledge bases belonging to different domains.
2. To develop an understanding of the application of ontologies for product modelling by using concepts from a library of design and manufacturing concepts i.e. a foundation ontology.
3. To find methods of knowledge verification when manufacturing knowledge associated with ontological product models is shared across domains.
4. To test these methods using IODE as the ontology editor and Common Logic as the ontology development formalism.

These objectives lead to the second step where the requirements of knowledge and understanding about the relevant academic and industrial issues are to be determined. A brief description of these is included in the next section.

1.6. Overview of the thesis

The thesis is comprised of ten chapters and three appendices. After outlining the methodology in chapter 2, the discussion goes on to present a comprehensive technical review of ontologies in the next chapters. Chapter 3 gives a detailed introduction to ontologies while chapter 4 reviews the contributions made in the field of ontology mediation and knowledge verificaiton. Chapter 5 explains the ontology development formalism used to test the proposed ideas. The study of shape feature based methodologies led to the development of an ontological product modelling technique which is explained in chapter 6. Chapter 7 exposes the findings of the case study conducted in an aerospace components manufacturing facility in order to study the real world industrial

interoperability problems. The design and implementation of the proposed verification framework is explained in chapter 8 and chapter 9 presents the testing and validation of this proposed framework with the help of the modelling method which was presented in chapter 6. Chapter 10 concludes the thesis with a description of further research. Also given in the appendices are the formalized ontologies used in some of the thesis chapters to explain examples.

Chapter 2: Research Methodology

2.1. Chapter overview

This chapter outlines the methodology this research followed for identifying the research problem and then proposing a solution.

A strategy of change management is used to structure this chapter. This strategy requires one to clearly outline the objectives of change or in other words to know where one wants to be after the change. These objectives are defined as the research goals in this chapter. Once that clear picture is established, the current state is then examined and requirements are identified. This step is performed in this research by reviewing the literature and through the case study in order to identigy the current interoperability requirements in the ICT based knowledge sharing systems specifically in the manufacturing industry. Finally the route to get to the intended state is decided and acted upon. This step comprises of the development of the solution for the requirements generated in the previous step. The aims and objectives of the IMKS project provided a wide but clear purpose of the development of verification methods for knowledge sharing between design and manufacture and this objective helped in outlining the more specific aims of this research.

Figure 2.1 shows a flow chart depicting the research methodology that was followed to

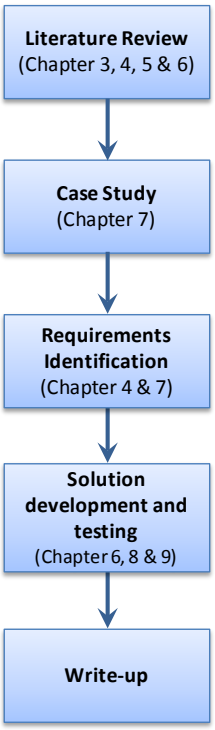


Figure 2.1. Research Methodology

achieve the research objectives. The rest of the chapter is composed of a detailed description of the methodology used.

2.2. What was the goal

The research goal can be understood with the help of the objectives of this research outlined in chapter 1. These objectives mainly comprise of first of all the development of an understanding of the application of ontologies in knowledge verification when shared across diverse domains and then the development and validation of unique methods to of knowledge verification specifically in manufacturing engineering.

2.3. What was needed to achieve this goal

The research goal defined in the previous section demands the development of an understanding in the areas of manufacturing, knowledge management, ontology development, ontology matching, knowledge sharing through ontological product models and the ontology building formalism which in this case was Common Logic. The background of the author provided the essential knowledge of manufacturing and knowledge management while the rest of the research areas had to be explored in detail. The following research questions were therefore asked:

1. What is the concept of ontology in computer science?
2. What are the existing ontologies and how are they developed?
3. What do the existing ontologies have in common with the required ontologies?
4. How are ontologies matched and mapped?
5. What are the existing matching tools and what are their limitations?
6. How helpful are the existing matching tools and techniques in verifying shared knowledge?
7. How are ontologies developed in Common Logic and edited in IODE?
8. How can the capabilities of IODE be used to develop a verification mechanism for Common Logic based ontologies?

The answers to these questions along with the existing knowledge of manufacturing processes and knowledge management would lead to the identification of the research

problem to be addressed and ultimately to its solution. A description of this process comes next.

2.4. How the goal was achieved – The core methodology

The required understanding of ontologies, their matching techniques, and existing methods of knowledge verification were primarily developed through a comprehensive literature review and partly through a case study conducted in the compressor disc machining facility of an aerospace components manufacturer.

2.4.1. Literature Review

The main focus of the literature reviewed was on ontologies and their application for knowledge verification. This led to the study of some selected ontology matching and mapping tools, techniques, methodologies and frameworks. It was identified that there is a plethora of tools available for the purpose of ontology matching and therefore a selection of the tools to be studied had to be made. This selection was made by first reviewing those research papers which presented the state of the art in this technology and a discussion on the most popular tools. A detailed study of those tools was then conducted which helped in recognizing their limitations and the identification of a research gap. This study can be found in chapter 4 of this thesis. Ideally all of these tools should have been tested on experimental ontologies in order to gain the real understanding of their capabilities. It was found, however that most of these tools were not freely available and therefore the capabilities of these tools and techniques as outlined by the research papers was considered sufficient. More time, then, was spent on studying the ontology development formalisms and how a software application for knowledge verification could be developed based on that formalism. It was identified that the available ontology matching tools are limited in their capability of automatically detecting and resolving ontological mismatches. These tools were also found to mostly detect the explication type of mismatches leaving the rest undetected. A detailed analysis of these tools can be seen in chapter 4. Chapter 4 also features a review of some of the most significant works on foundation ontology based ontology matching. This review along with the analysis of other ontology matching tools helped to identify the research gap in this field. The research gap is discussed in the conclusions to chapter 4. This research gap was used to determine the requirements for a verification tool which could address the identified weaknesses of existing tools. However,

real industrial examples were needed to totally understand the requirements of the verification methods and therefore a case study was conducted. A brief description of this case study is given here whilst a detailed account is provided in chapter 7.

2.4.2. Case Study

The case study particularly helped in identifying the requirements of knowledge and information flows in manufacturing industry especially between design and manufacture. It also helped in developing an understanding of how product shape feature based models could be converted into ontological models to associate manufacturability knowledge to them for the purpose of sharing this knowledge in order to produce a manufacturable design. This understanding could only be achieved when the initial academic knowledge of ontologies and their capabilities was gained through the literature review which preceded the case study.

2.4.3. Requirements identification

The completion of the case study led to the generation of requirements for a verification system capable of ensuring that the knowledge shared across design and manufacture is interpreted correctly and is used for the purpose of producing a design suitable for manufacturing in the available facilities. In other words it helped in identifying the real manufacturing world knowledge sharing interoperability problems. A requirements document was therefore produced, in collaboration with a co-researcher from the IMKS project, which clearly outlined these interoperability requirements and this document then proved to be the foundation for the verification system that was later developed. The requirements document can be seen in the appendices.

2.4.4. Solution development and testing

In the light of the identified requirements a knowledge verification framework was proposed. Based on this framework, an application Programming Interface (API) was developed. However, before the development of the API could be started, an adequate knowledge of the use of ontology building formalism and the functionalities of the selected ontology development environment was essential. An initial review of the tutorials showed that the query tool in the ontology editor can play a crucial role in the successful working of the intended verification mechanism. More effort was therefore spent in learning this tool and the way queries are written. The resulting knowledge, along with the knowledge of

Java, was then used to build an API which generates queries and interprets results automatically. Once functional, the API was tested on a real design scenario experienced during the case study and satisfactory results were obtained.

2.5. Conclusion

The main stages of this research were the literature review, the case study, the solution development, and the testing and validation of this solution. Each of these stages subsequently helped the next and the research led to some useful findings and results. The proposed solution was successfully tested on real industrial problems of manufacturability knowledge sharing.

Chapter 3: Ontologies – A technical review

3.1. Chapter overview

The last section of chapter 2 briefly explained the scope of the literature reviewed in this research for identifying the research gap in the area of knowledge verification through ontology mediation. This chapter, along with chapters 4 and 5, presents a comprehensive survey of the relevant literature in the field of undertaken research. This survey, first of all, helps in identifying the research gap and then the identified gap facilitates justification of the unique and innovative contribution of the research findings. The practical industrial evidence of the relevance of this research is provided in chapter 7 where the findings of a case study are presented. Together, chapters 3, 4, 5 and 7 provide a meaningful background for this research by setting the context for the findings presented later in this thesis.

3.2. Ontologies

Modelling is an essential part of the intellectual activity of human beings (Silvert, 2001). It is an approximation of reality (Studer et al, 1998). It is reasonable to assume that every single person has a subjective model of his or her own to make sense of this world. This model is constructed as people experience the events around them and the accuracy of this model determines the level of their intelligence and problem solving ability. The challenge of Artificial Intelligence (AI) is therefore the challenge of constructing an accurate model of the world in a computer interpretable form. The fundamental questions to be asked and answered in this regard should be about the existence of things. In the field of metaphysics, the systematic explanation of being as an answer to this question is called an ontology (Gomez-Perez et al, 2004). In this sense an ontology is a particular system of categories accounting for a certain vision of the world (Maedche, 2002). It is a model of discourse participants (Nirenburg and Raskin, 2004). The term 'Ontology', however, is not limited to the subject of metaphysics only. Researchers in the field of Information Science also use this term for defining a hierarchical arrangement of concepts and their relations, together with the constraints on those objects and relations between them (Alexiev et al, 2005; Antoniou and Van Harmelen, 2008). Ontologies in this sense provide a basis for shared meaning (Young et al, 2007). They provide a common terminology that helps to capture key distinctions among concepts in different domains, which aids in the translation process (Schlenoff et al, 2000). It is evident from these opinions about ontologies that, in both

philosophy/metaphysics and information science, studies of ontology have a similar aim. Both of them try to build a model of the world by defining the existence of things, their classifications and relationships. It is, however, the latter which is the subject of this research. In the next sections, ontology and issues essential for a thorough understanding of this concept are discussed.

3.2.1. Ontology defined

As defined above, an ontology attempts to define concepts, their mutual relationships and constraints on those relationships. It is a lexicon of terminology along with some specification of the meaning of terms in the lexicon (Gruninger et al, 2000). This same notion is defined by different authors using different words. The most frequently quoted definition is

'An ontology is an explicit specification of a conceptualization' (Gruber, 1993a).

This definition was later modified as:

'An ontology is an explicit and formal specification of a shared conceptualization' (Studer et al, 1998).

These definitions state some essential conditions for a specification of something to be an ontology. First of all, an ontology should be explicit. This explicit nature needs the concepts and constraints to be defined objectively leaving no space for any subjective interpretation. This is usually done by defining axioms and imposing constraints on the use of certain terminologies. Secondly, an ontology should be formal. A classification of ontologies on the basis of their formality shows that the more formal an ontology becomes, the more easily interpretable it is by computers (Gomez-Perez et al, 2004). Formality, therefore, refers to the attribute of an ontology which enables it to be read by computers. Thirdly, an ontology should be shareable. This implies that a non-shareable arrangement of explicitly defined concepts even if they are properly axiomatized and constrained will not form an ontology. The scope of this sharing, however, is arguable. A discussion on sharing leads one to the concept of 'Ontological Commitment' which will be discussed later. The word conceptualization in the above definitions refers to 'an abstract model of a phenomenon in the world by having identified the relevant concepts of that phenomenon' (Studer et al,

1998). It is an 'abstract, simplified view of the world that we wish to represent for some purpose' (Gruber, 1993a).

A mathematical definition has also been offered for an ontology by (Kalfoglou and Schorlemmer, 2003). According to this definition, 'an ontology is a pair O = (S, A), where S is the (ontological) signature – describing the vocabulary – and A is a set of (ontological) axioms – specifying the intended interpretation of the vocabulary in some domain of discourse'. Analysis of the process of ontology formation indicates that it is a process of a gradual addition of meaning to data and then to information in order to produce shareable knowledge. Recalling the fundamental concepts of knowledge management defining data, information and knowledge, a marked similarity can be observed in the way these terms are defined and the way ontologies are formed. Data are defined as discrete numbers or words like 200, 235, 222 … or 'horse', 'sky', 'Sofia'… When some meaning is added to these numbers and words they become information. For example, 200, 235 and 222 are the number of students registered in a particular course for three consecutive years. In a similar theme, when some more detail is added to this information it gets converted into knowledge. For example, the reasons for the rise and fall in the number of registered students. The concept of the hierarchy of knowledge gives a graphical representation to this notion. Figure 3.1 represents the commonalities in the concepts of the knowledge hierarchy

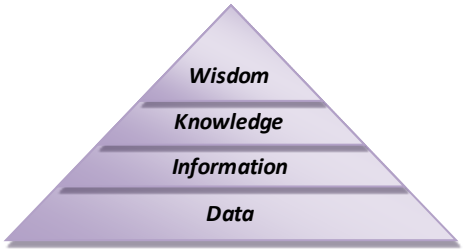


Figure.3.1. Hierarchy of Knowledge

by most authors (Zack, 1999; Gupter and Sharma, 2004; Awad and Ghaziri, 2004). The reason for explaining the knowledge hierarchy here is to appreciate that an ontology is formed in an identical manner. Data and information are arranged in the required way and then some meaning is given to them by defining axioms and imposing certain constraints. When this knowledge is made sharable it gives shape to an ontology. Thus another way of defining an ontology, as determined during this research, is as follows:

"An ontology is a sharable arrangement of objectively defined knowledge obtained by connecting, axiomatizing and constraining data."

Wisdom in this way, thus, is the combined use of these ontologies. This is because the use of knowledge in a certain way creates wisdom as shown in figure 3.1.

From the point of view of this research, however, the definition of ontologies given by (Uschold and Gruninger, 1996) is appropriate. According to them:

"An ontology is a formal description of the entities within a given domain, the properties they possess, the relationships they participate in, the constraints they are subject to, and the patterns of behaviour they exhibit".

In view of the way ontologies are used in this research, the above definition appears most fitting and therefore may be referred to whenever the word ontology is used in this thesis. Having decided upon the definition of ontology, it is now necessary to look into different types of ontologies to gain full knowledge of this field. The next sections therefore, further explicate the concept of ontology by looking into its classifications.

3.3. Classifications of ontologies

Different classifications of ontologies can be found in the literature. Ontologies are classified on the following major bases:

1. Formalization or machine readability
2. Level of rigor i.e. light weight or heavy weight
3. Logic i.e. logic-based or non-logic based and
4. Level of generality

These classifications are discussed further below.

3.3.1. Classification on the basis of formalization or machine readability

Firstly, the level of formality divides ontologies into four categories, i.e. highly informal, semi-informal, semi formal and rigorously formal (Gomez-Perez et al, 2004). Ontologies expressed in natural language are categorized as highly informal. Those which are

constructed in a restricted and structured form of natural language are referred to as semi-informal. Semi-formal ontologies are written in a formally defined language while rigorously formal ontologies are so named because they provide carefully defined terms along with machine readable semantics and axioms. Ontologies built in Ontolingua and OWL (two ontology development languages) can be designated as semi-formal ontologies (Gomez-Perez et al, 2004). The ontologies developed in this research for testing the proposed verification framework also belong to this category.

3.3.2. Classification on the basis of Level of rigor

An alternate classification is done by taking into account the rigor of restriction and constraints on terminology semantics which splits ontologies into heavy weight and light weight ontologies. Light weight ontologies entail just relationships and properties of a taxonomically arranged collection of terms while heavyweight ontologies also add axioms and constraints on the use of these terms (Gomez-Perez et al, 2004; Alexiev et al, 2005). The ontology expressiveness spectrum (McGuinness, 2003) in Figure 3.2 shows a detailed

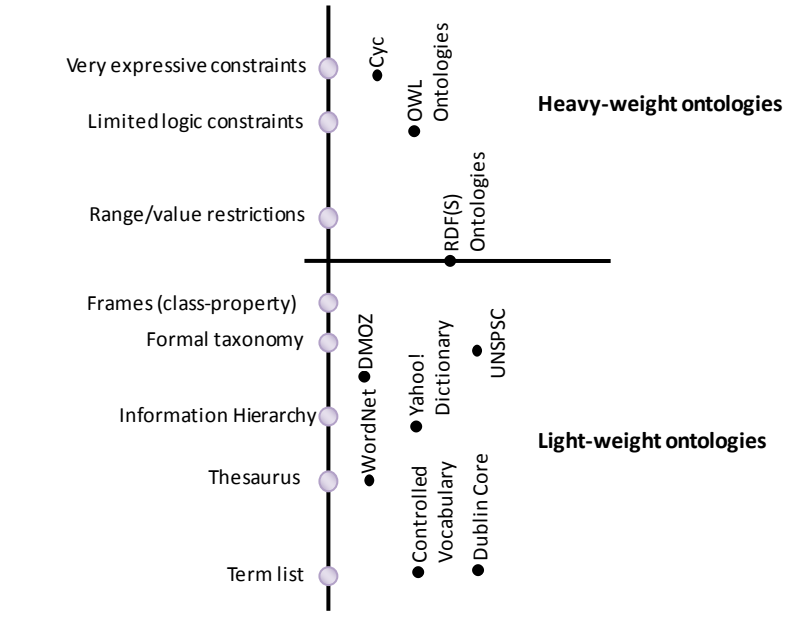


Fig 3.2: An ontology expressiveness spectrum

continuum of attributes of ontologies starting from light-weight to more heavy-weight ontologies. Different types of currently available ontologies shown in the figure like WordNet, Cyc, etc. will be considered later in this chapter when the currently available

ontologies are discussed. The example ontologies developed in this research represent heavy-weight ontologies.

3.3.3. Classification on the basis of Logic

The third classification is on the basis of the logic of the ontologies and it gives two types; Logic-based and non logic-based ontologies. A logic-based ontology explicitly specifies the semantics of terminologies through ontological definitions and axioms while a non logic-based ontology uses predefined and agreed upon meanings of terminologies and the context in which they are going to be used (Yang and Zhang, 2007). The ontologies written in this research for testing purposes belong to the category of non-logic based ontologies because the terminologies used in these ontologies are not supported by axioms. Rather they are universally agreed upon terms belonging to the domain of manufacturing engineering

3.3.4. Classification on the basis of level of generality

The fourth classification is made on the basis of the level of generality of an ontology. Different types of ontologies in this scenario are Top-level ontologies, Domain ontologies, Task ontologies and Application ontologies (Guarino, 1998). The level of generality increases as we go towards the top-level ontology from the application ontology level. Top-level or Foundational ontologies describe very general concepts. They serve very large communities from different domains. Domain ontologies contain the vocabulary from a specific domain obtained by specializing the concepts introduced in the top-level ontology. Task ontologies as their name suggests are related to a specific task within a domain. For example, the task of drilling a hole in the domain of manufacturing engineering. And finally, application ontologies are the most general type of ontologies. They represent roles performed in a certain ontology (Maedche, 2002). Ontologies written in this research contain a foundation ontology of core manufacturing concepts which is then used to develop two domain ontologies belonging to the domains of design and manufacture.

3.3.5. Other classifications

Several other forms of classifications and ontology designations can further be found in the literature. For example, content ontologies (Mizoguchi et al, 1995), communication

Table 3.1: Classifications of Ontologies

S.No	Basis of Classification	Classification		Description	Source
1	Formalization or Machine Readability	Highly Informal ontologies		The more formal that ontology is the more machine readable it is.	(Gomez-Perez et al, 2004)
		Semi-informal ontologies			
		Semi-formal ontologies			
		Rigorously formal ontologies			
2	Rigor of Restrictions and Constraints	Light weight ontologies		Less rigorous	Alexiev et al (2005), (Gomez-Perez et al, 2004)
		Heavy weight ontologies		More rigorous	
3	Logic	Logic-based ontologies		Uses axioms and constraints along with the definitions of terminologies	Yang and Zhang (2007)
		non logic-based ontologies		Uses terminologies with pre-agreed definition and use	
4	Level of Generality	Top level or Foundation ontologies		Generality decreases from top to application ontology	Guarino (1998)
		Domain ontologies			
		Task ontologies			
		Application ontologies			
5	Use	Content Ontologies	Domain ontologies (same as in 4)	Expresses conceptualization specific to a particular work area	(Mizoguchi et al, 1995)
			Task ontologies	Specific to a certain task in a domain	
		Communication Ontologies		For two way communication	
		Indexing Ontologies		For case retrieval	
		Meta or Knowledge Representation Ontologies		For knowledge sharing	
6	Conceptualization	Structure	Terminological ontologies	Such as lexicons	Van Heijst et al (1997)
			Information ontologies	Such as database schemata	
			Knowledge modelling ontologies	Includes a richer internal structure containing conceptualization of knowledge.	
		Subject	Generic ontologies (Top level as in 4)	Defined conceptualizations are generic across many fields.	
			Domain ontologies (same as in 4)	Expresses conceptualization specific to a particular work area	
			Application	Contains method and task specific extensions	

ontologies, indexing ontologies, meta or knowledge representation ontologies (Gomez-Perez et al, 2004), terminological ontologies, information ontologies, and knowledge modelling ontologies (van Heijst et al, 1997). However, these ontology types are already covered by the earlier defined classifications in one way or another and therefore are not discussed in detail here. Some aspects of these classifications can be seen in Table 3.1 which gives a consolidated view of all the classifications discussed here.

The classifications of ontologies reviewed above precisely identify the types of ontologies that are used in this research. To gain further understanding of this concept, it is also necessary to review the way ontologies are developed.

3.4. Ontology Development

Since ontologies can be considered as approximations of reality, their development is a modelling process and several issues need to be considered before starting this process. For example, a decision needs to be made whether an ontology is to be built from scratch or if it should be constructed based on an already existing ontology, what methodology and language should be used for its development, how should the compatibility issues be resolved and most importantly what should it be able to answer. Figure 3.3 (on the next page) gives a comparison of a few of the commonly quoted ontology development methodologies in the literature. There seems to be a consensus on some typical steps which include

1. Identification of scope and purpose of the ontology,
2. Definition of concepts and terms fundamental to the domain to which the ontology belongs,
3. Formalization and codification of the ontology in a suitable language,
4. Population of the ontology, and
5. Evaluation of the ontology.

One of the most important and critical issues, when starting to construct an ontology, is to determine what things exist in the domain which are to be modelled (Masolo et al, 2001). A good way of defining the scope of an ontology is to ask some competency questions (Gruninger and Fox, 1995). These competency questions are not only asked in the beginning

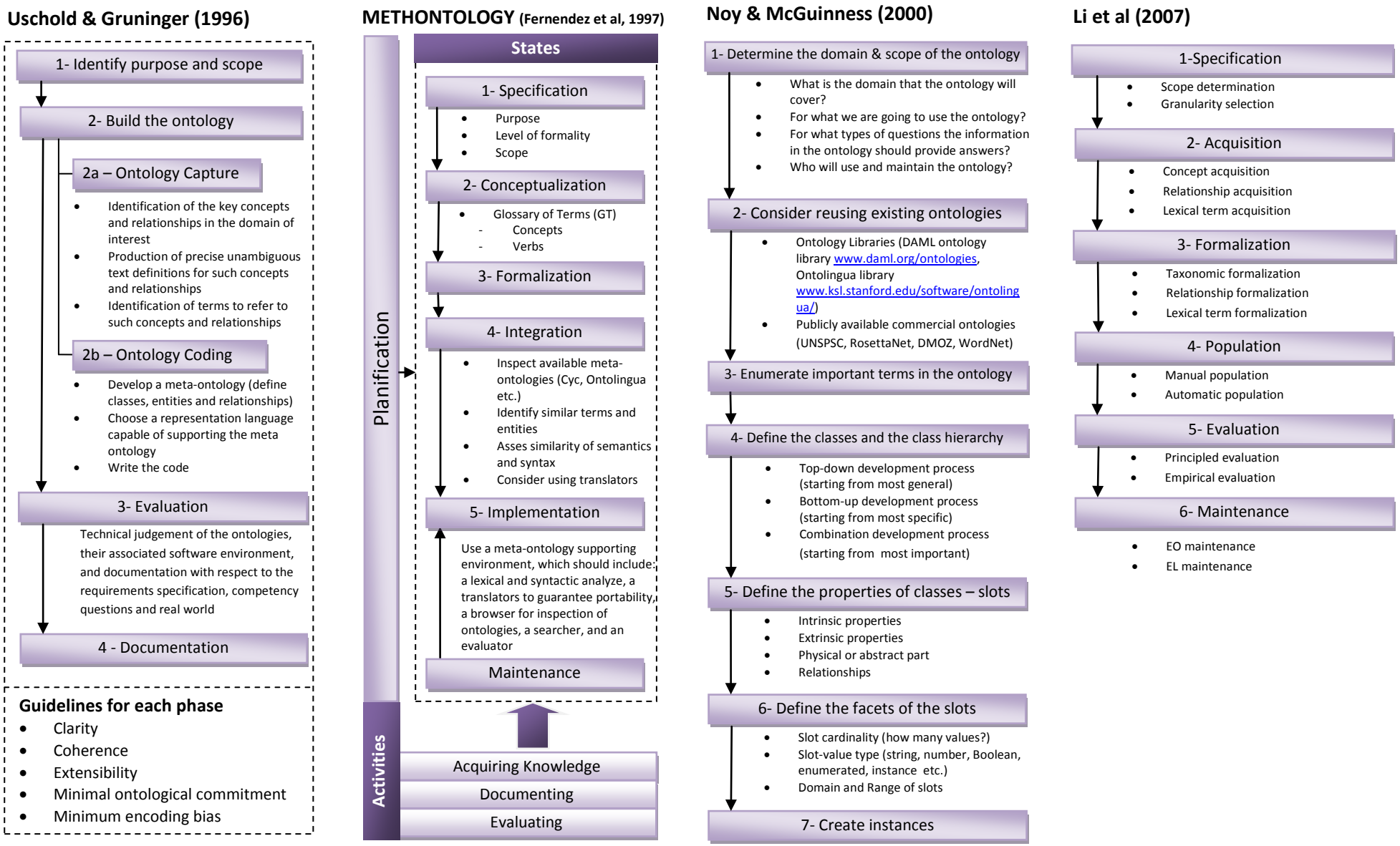


Fig 3.3: Comparison of some ontology development methodologies

to determine the scope and purpose but should also be asked at the formalization stage. They are named as formal competency questions and are asked to ensure consistency between the planned ontology in natural language and the one developed formally. Once these questions are decided, the ontology should be designed keeping in mind the answers to these questions. When defining terms and conceptualizations in class hierarchies, three different approaches can be followed (Noy and McGuinness, 2001).

1. A top-down approach in which the ontology is started from the most general classes.
2. A bottom-up approach where the most specific terms are defined first and
3. Middle-out approach where the most important terms in the middle are the starting point.

The last approach is recommended (Uschold, 1996) because it prevents the ontology becoming too specific with high level of granularity and thus reduces the chances of inconsistencies. This is, more or less, the way experimental ontologies in this research have been constructed by.

In addition to the common ontology development steps stated above there are some steps which exist in every ontological development process but are not explicitly mentioned by every author. For example, (Uschold and Grüninger, 1996) and (Fernandez et al, 1997) emphasize the need for ontology documentation. It is argued that documentation of each and every step of the development process is very important for future modifications and more importantly for the sharing of ontologies. The processes of knowledge acquisition, evaluation of ontologies and its documentation are considered to be an ongoing process (Fernandez et al, 1997). (Li et al, 2007) and (Fernandez et al, 1997) believe that after the ontology has been developed, it is also necessary to evaluate and maintain it from time to time (on an ongoing basis). Maintenance involves the modification and evolution of ontologies as the need arises. (Uschold and Grüninger, 1996) give some general guidelines for the entire ontology development process. They define the parameters of clarity, coherence, extensibility, minimum ontological commitment, and minimum encoding bias. Clarity refers to the quality of an ontology to effectively communicate the intended distinctions. Coherence is the internal logical consistency of an ontology. Extensibility reflects the provision in an ontology to extend it with new definitions and hierarchies in the future.

Minimum ontological commitment helps in increasing the extensibility of an ontology. Making as few claims as possible about the world being modelled gives freedom to the committed parties to instantiate and specialize the ontology as needed. And finally, minimum encoding bias means that the conceptualizations in the ontology should be defined at the knowledge level without depending upon a specific symbol-level encoding. (Noy and McGuinness, 2001) and (Fernandez et al, 1997) also recommend that existing ontologies are used with the new ones through integration. Ontology libraries like the Ontolingua library and DAML ontology library can be used to browse for suitable ontologies. Furthermore, some commercially available ontologies like UNSPSC, Cyc ontology, WordNet, RosettaNet, DMOZ can also be used for integration.

A more generalized view of the ontology development process can be taken from an approach called the model driven architecture. A closer look at its steps reveals that it covers all of the ontology development steps discussed above in a more structured and organized way. Following is a brief introduction to this approach and its comparison with ontology development processes.

3.5. Model Driven Architecture

MDA or model driven architecture (Miller and Mukerji, 2003) is an approach intended to make software application design and development more structured, traceable, portable, interoperable and reusable. It attempts to separate the specification of the operation of a system from the details of the way the system uses the capabilities of its platform. It is

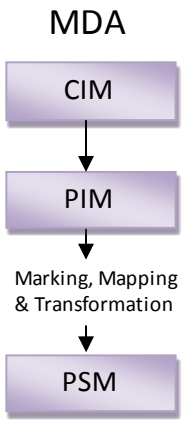


Fig. 3.4: Model Driven Architecture

model driven because models are the basic building blocks of its structure. The term model here refers to a simplification of a system built with an intended goal in mind in order to answer questions in place of the actual system (Bezivin and Gerbe, 2001). MDA divides the process of application design and development into three main models. These are:

1- Computation Independent Model (CIM),
2- Platform Independent Model (PIM) and
3- Platform Specific Model (PSM).

The first step of making a computation independent model refers to the process of specifying requirements and the environment in which the application is going to work. CIM is totally independent of any programming or modelling language and is just a representational model of system requirements and scope.

A Platform Independent Model on the other hand is more formal and requires the help of some representational software or language like UML2 to provide a more detailed view of the application, the information flow inside it and the definitions and capabilities of different modules constituting the application.

The Platform Specific Model is the computer coding or the most formal level. Here the PSM is transformed into a machine understandable form which computers can execute.

The processes of Marking, Mapping and Transformation take place in-between PIM and PSM. Marking refers to the process of indicating the roles of different PIM elements in mapping which are to be transformed into PSM elements. Mapping provides specifications for transforming a PIM into PSM for a particular platform (Miller and Mukerji, 2003). This marking and mapping not only fulfils the documentation requirements of the ontology development methodologies but also makes the whole process more traceable and easily modifiable. This can, on one hand, help in the maintenance and evaluation of the ontologies and on the other the rectification of the system in case any inconsistencies are found. The main goal of MDA is to move human involvement in an application producing process from PSM towards CIM and PIM and automate the transformation process from one model to another (Djuric et al, 2005b). In addition to the three step route of producing an application, MDA also gives a four layered meta-modelling architecture, as shown in Figure 3.5, and

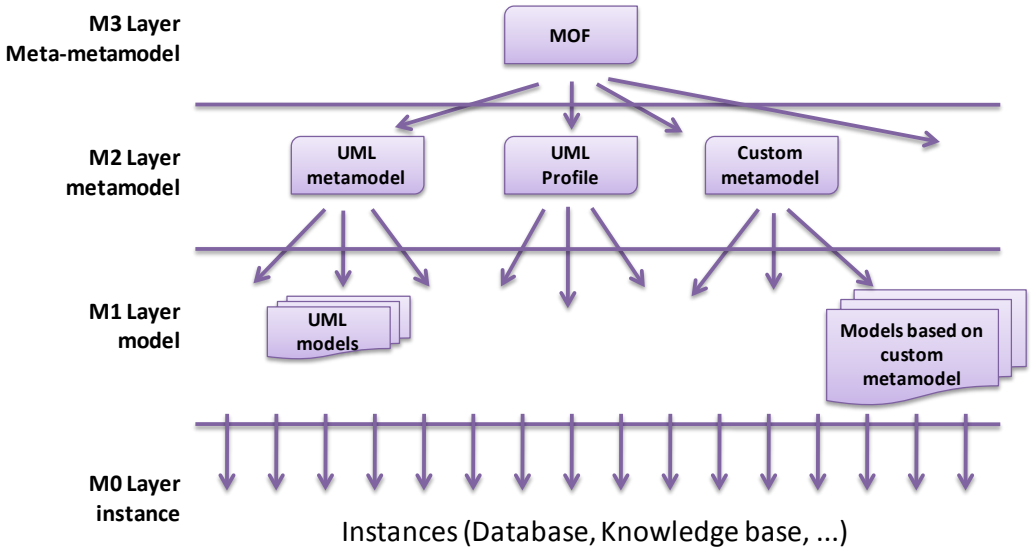


Figure 3.5: Depiction of the OMG's four layered architecture

several complementary standards provided by the Object Management Group (OMG). These standards are Meta-Object Facility (MOF), Unified Modelling Language (UML) and XML Metadata Interchange (XMI) (Djuric et al, 2005a).

3.5.1. Meta Object Facility

MOF, the meta-object facility, is a framework to specify, construct and manage technology neutral metamodels. It can therefore be used to define any modelling language such as UML. Metamodels defined in MOF can be transformed into XML documents and schemas by using XMI the XML Metadata Interchange (Djuric et al, 2005b). The MDA definition of metadata includes database schema, UML models, workflow models, business process models, business rules, API definitions, configuration and deployment descriptors, and so on (Frankel et al, 2004). It is important to note here that for any modelling language to be used with MDA tools, it must be based on MOF (Cranefield and Pan, 2007) and therefore to add a kind of metadata to the kinds of metadata that MDA tools can manage, it is necessary to define a MOF model of that kind of metadata. Such a model is known as a metamodel (Frankel et al, 2004). To cater for ontological engineering and extend the interoperability capabilities of MDA, the Object Management Group (OMG) has come up with a set of standard metamodels and mappings between them known as the Ontology Definition Metamodel (ODM) which is discussed in the following sections.

3.5.2. MDA and Ontologies

Frankel et al (2004) compare the MDA technologies and the Semantic Web. According to them, one of the important distinctions between them is their focus. The emphasis of MDA (through MOF) is on automating the physical management and interchange of metadata. Knowledge representation, on the other hand focuses more on the semantics embodied in the content of the metadata and the automated reasoning over the content. To get the best of both worlds, a standard was therefore needed to support ontology development. This need was met through the OMG's Ontology Definition Metamodel (ODM). The ontological knowledge base development process can be leveraged by developing the MOF metamodels of the existing knowledge bases and through mappings from such models to and from ODM, MOF and ODM-based knowledge engineering metadata (Frankel et al, 2004).

3.5.3. Ontology Definition Metamodel (ODM)

The Ontology Definition Metamodel is designed to enable interoperability between different ontology development languages including RDF(S), OWL, Topic Maps (TM) and Common Logic (CL). It enables Model Driven Architecture (MDA) standards to be used in ontological engineering (Djuric et al, 2005a). It caters for the forward and reverse engineering of ontologies, enabling development of ontologies in UML tools and implementation of such ontologies in OWL [and other ontology development languages] without loss of fidelity (Frankel et al, 2004). It is a collection of three main sets of standards based on MOF (OMG, 2008b). These standards need to be followed during ontology development in order to make the ontologies interoperable. It is comprised of:

1- Metamodels, for ontology development,
2- Mappings between these metamodels as well as mappings to and from UML,
3- A set of profiles that enable ontology modelling through the use of UML-based tools.

3.5.3.1. ODM Metamodels:

The version one of the ODM by OMG defines four normative metamodels (OMG, 2008a), including the metamodels for RDF, OWL, Common Logic (CL) and Topic Maps (TM). These metamodels are defined with the help of a few diagrams. The Common Logic (CL) metamodel, for example, consists of five different diagrams that represent the

infrastructure on which an ontology should be built in order to comply with ODM. These diagrams are:

1- The phrase diagram,
2- The terms diagram,
3- The atoms diagram,
4- The sentences diagram,
5- The Boolean sentences diagram and
6- The quantified sentences diagram

The phrase diagram provides mechanisms for grouping and scoping the elements that constitute an ontology authored in Common Logic or any of its syntactic variants. The Terms diagram provides additional insight into the core syntactic elements of Common Logic including names, commented terms and term sequences or functional terms. Atom diagrams can be used to handle equations. Atoms are also handled by the sentence diagram along with other sentences including Boolean, quantified, irregular and commented sentences. The Boolean sentences diagram further elaborates the handling of Boolean sentences. Similarly, the quantified sentence diagram provides further description of handling quantified sentences (OMG, 2008b). Figure 3.6 shows the main constituents and their mutual connections in a phrase diagram for common logic.

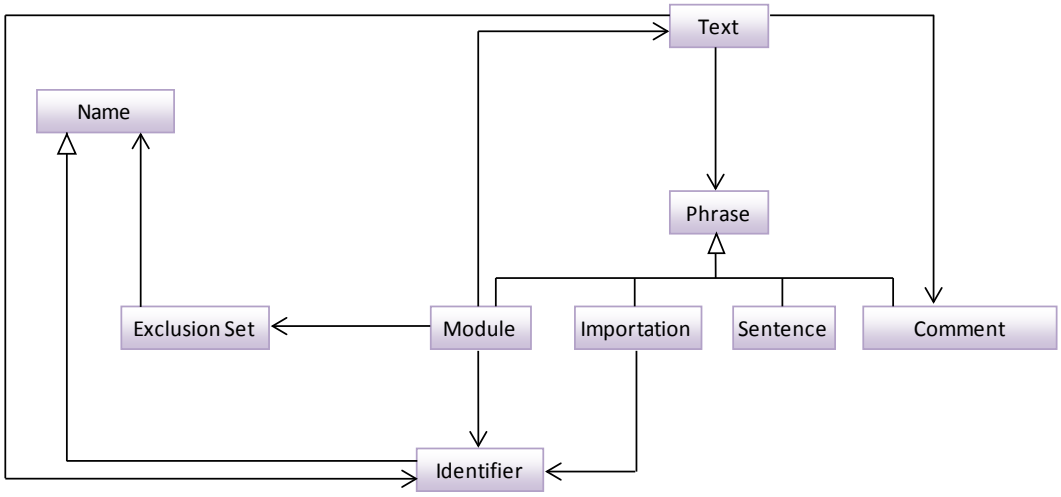


Figure 3.6: Depiction of the Phrase Diagram

3.5.3.2. Mappings

ODM provides informative mappings from each metamodel to and from OWL Full, except for Common Logic. Currently, CL only has a mapping from OWL Full but a lossy (that which loses details while transferring information) reverse mapping defined in QVT from CL to OWL is planned. ODM also gives a direct two way mapping between UML and OWL and a bi-directional mapping between UML and CL is also planned (OMG, 2008b).

3.5.3.3. UML Profiles

Profiles allow a user to generate an ontology description in an ontology development language from an ontology represented in UML. Currently, UML profiles provided by ODM enable the use of UML notation and tools for ontology modelling and facilitate generation of corresponding ontology descriptions in RDF, OWL and TM. A UML profile for CL is under consideration and may become a part of ODM in the future (OMG, 2008b).

3.5.4. Query/View/Transformation Language (QVT)

QVT is a language based on MOF 2.0 defined by the Object Management Group for automated model transformation. The QVT standard is considered essential to make MDA a success (Gardner et al, 2003). The language dimensions of QVT consist of three named language levels of Core, Relations and Operational Mappings. The core and relations layers comprise the declarative part of QVT while the operational mapping is imperative in nature. Operational mappings permit transformations to be defined using a complete imperative approach (operational transformations) or allow complementing relational transformations by providing imperative operations for implementing the relations (hybrid approach). The relations language specifies the relationships between MOF models. The core language is a small model/language which only supports pattern matching over a flat set of variables by evaluating conditions over those variables against a set of models. The operational mappings language is a standard way to be used for imperative implementation by populating the trace models of the relations language. The QVT black box represents the plug-in facility which can be used to utilize transformation resources expressed in other languages with a MOF binding (OMG, 2008a).

Before going into more details of the language some associated terminologies need to be understood.

'A query is an expression evaluated over a model'. A query may result in one or more instances from the model over which it is evaluated. For example a query can be: 'return all instances with no subclasses'.

The term 'View' refers to a model completely derived from another model. Changes in the view are dependent on the changes in the source model. If allowed, changes in the view can lead to changes in the source model. A defined reverse mapping is therefore necessary back to the base model in this case. Views are generated via transformations. A query is a restricted kind of view while a view is a restricted kind of transformation in which the target model cannot be modified independently of the source model (Gardner et al, 2003).

A Relation is the specification of a multi-dimensional transformation. Relations are not executable, they are only used to check the consistency of two models against one another and thus help in determining the validity of mapping between the models.

Mappings on the other hand are transformation implementations. They are used to refine the relations. A transformation therefore can either mean a relation or a mapping (Appukuttan et al, 2003).

Transformations are performed to build a target model from a source model in a manner that source and target models are compatible according to the relations defined (Hausmann, 2003).

In the context of MOF, the transformations between the models are defined at the

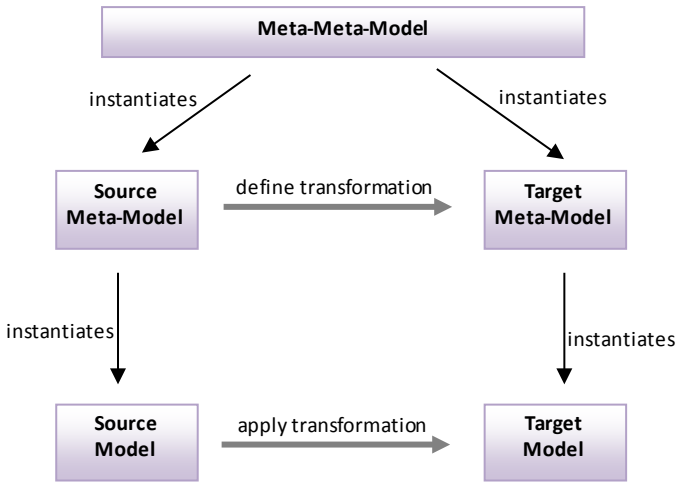


Figure 3.7: Model Transformation in MOF Context

metamodel level and are executed at the model level (Belaunde et al, 2008). The important thing to note here is that for QVT to be used as a transformation language, both the source and target models have to be instances of the same meta-metamodel i.e. MOF as shown in Figure 3.7. Mappings to any non-OMG language should be obtainable by defining a MOF metamodel for such a language (Gardner et al, 2003).

During the translation between metamodels, every single element in the source model may not have a corresponding element in the target model. This situation is called structure loss. To minimize the effects of this structure loss QVT retains the association between the source and target model elements in a mapping. This transformation itself is stored as a model as shown in Figure 3.7. Hence one can always trace back a given model instance to its original location (Colomb et al, 2004).

Some tools based on QVT are also available to perform automatic transformation between models. SmartQVT is a tool designed as a QVT parser, compiler and launcher. It implements the operational mappings in QVT and generates Java executable code realizing the transformations. SmartGen is a SmartQVT based code generator which complements the capabilities of SmartQVT. It produces text files and their directories from models (Belaunde et al, 2008).

3.5.5. Suitability of MDA for ontology development

Two main dimensions of implementation of MDA standards in the development of ontologies can be observed from the above review. Firstly, MDA can be used as a methodology to develop an ontology. The structured flow involving the development of a CIM, PIM and then PSM makes the process of ontology building more traceable and well managed. The second dimension, which is more complex but equally useful, is the use of MDA standards to structure the ontology itself. These standards specifically include MOF based ODM. A review of these standards shows that an ontology based on a MOF metamodel can be made more interoperable with other ontologies developed using the same metamodel. This gives rise to two questions. First of all “what are the languages supported by ODM?” and secondly “how acceptable are these standards in the industry?”

As far as the acceptability of these standards in the industry is concerned, the latest specifications of ODM show that big companies like IBM, TATA, Sun Microsystems and France Telecom (Orange Mobiles) are involved in the development of these standards. For the language support, it can be seen from the above review that the ontology definition metamodel (ODM) defines or aims to define standards for and transformations between RDFS, OWL and CL (among others) which are currently the three main ontology development languages (Corcho and Gomez-Perez, 2000; Mizoguchi, 2004; Pulido et al, 2006). Interoperability can therefore be enhanced, on one hand, by using the transformation support of ODM between metamodels of other major languages and on the other hand through an easier and more efficient knowledge verification process during mapping. Ontologies based on the same infrastructure will be less likely to have mismatches and therefore the knowledge verification task would be simpler. Another MOF based standard, the QVT language, complements ODM in the transformation efforts between models. If some specific deliverables of an ontology do not conflict with the ODM metamodel specifications, then developing an ontology based on it seems to create a win-win situation. If an ontology has to be developed, why not develop it using the ODM standards. The only valid objection that can be raised against the use of ODM is it not being an ISO standard. If in future it acquires the ISO status then it would be an important consideration when planning to construct an ontology.

The knowledge verification solution proposed in this research takes inspiration from the way model transformation takes place in MDA as shown in figure 3.7. The proposed solution also resembles aspects of the IMKS project where a foundation ontology acts as the meta-meta model while the domain ontologies subsume it as meta models with their instances being the assertions in the knowledge base in the form of models of engineering components and activities.

Having reviewed the ontology development methodologies, the next step is to understand the formalization of ontologies by using ontology development formalisms. The next sections, therefore, scan the research area of ontology development languages and formalisms.

3.6. Ontology Development Formalisms

An ontology, being an explicit and formal specification of a shared conceptualization (Studer et al, 1998), requires a language for its construction which can efficiently provide a means to shape up its essential characteristics. Five kinds of components are believed to specify an ontology namely: concepts, relations, functions, axioms and instances (Visser et al, 1997). To represent these components formally, ontology building languages should:

- Have a compact syntax.
- Be highly intuitive to humans.
- Have well-defined formal semantics.
- Be able to represent human knowledge.
- Include reasoning properties.
- Have the potential for building knowledge bases.
- Have a proper link with existing web standards to ensure interoperability (Pulido et al, 2006).

Some of the contemporary languages bearing these attributes and discussed here are Knowledge Interchange Format (KIF), Ontolingua, XML, RDF, RDF(S), OIL, DAML+OIL, OWL and Common Logic (CL).

3.6.1. Knowledge Interchange Format (KIF)

The Knowledge Interchange Format, as its name suggests, was developed to interchange knowledge formally between disparate computer programs (Bechhofer, 2000). KIF has declarative semantics and is based on first order predicate logic (Corcho and Gomez-Perez, 2000). KIF is the most expressive language among the languages used for representing ontologies. It allows the representation of concepts, n-ary relations (relations among more than two individuals), functions, axioms, instances and procedures. This high expressivity is however a trade-off with its reasoning power. (Corcho et al, 2003).

3.6.2. Ontolingua

KIF along with a few extensions, forms another ontology language called Ontolingua. The first of the extensions is an additional syntax to support the axiom representation in KIF and

the second is the incorporation of a Frame Ontology to define object-oriented and frame-language terms (Bechhofer, 2000). With these extensions Ontolingua is capable of building ontologies in any of the following three manners: (1) using exclusively the Frame Ontology vocabulary if axiom representation is not needed, (2) using KIF expressions and (3) using both languages simultaneously (Corcho and Gomez-Perez, 2000). The weakness of Ontolingua is that it does not have an inference functionality (Mizoguchi, 2004).

3.6.3. XML – the eXtensible Markup Language

XML, the eXtensible Markup Language, is a W3C endorsed standard for document markup. XML is a descendant of SGML (Standard Generalized Markup language) in which HTML was developed (Harold and Means, 2004). Although in contrast to HTML, XML allows user defined tags, computer agents cannot be guaranteed to determine the intended interpretation of its tags (Pulido et al, 2006). RDF (Resource Description Framework), on the other hand, provides interoperability between applications that exchange machine-understandable information on the Web (Bechhofer, 2000). A brief description of RDF follows.

3.6.4. Resource Description Framework

Through its Object-Attribute-Value triples, RDF is capable of forming a semantic network. RDF is based on the XML syntax but is different from it in that RDF is a data representation model rather than a language (Mizoguchi, 2004). An RDF data model consists of three object types: resources, properties and statements but it cannot define the relationships between properties and resources. This shortcoming is overcome through the RDF vocabulary description language which is also known as the RDF Schema (RDFS). RDF(S) is a term usually used to represent a combination of RDF and RDFS (Gomez-Perez et al, 2004). RDF(S) is not very expressive and it just allows the representation of concepts, concepts taxonomies and binary relations. To further aid the semantic web development, three more languages were developed as extensions of RDF(S) namely OIL, DAML+OIL and OWL (Corcho et al, 2003).

3.6.5. OIL – the Ontology Inference Layer

OIL, the Ontology Inference Layer is an Ontology Interchange Language and it has a well defined syntax in XML, RDF and RDF Schema and therefore can be used as a standard

language for ontology representation on the web. It is frame-based, hence, it provides a context for modelling aspects of a domain through classes, subclasses, attributes and properties. Another important element of OIL is Description Logics (DL). DL describes knowledge in terms of concepts and roles (Fensel et al, 2001). In contrast to the high expressive power of Ontolingua, OIL starts with a very simple and limited core language. This is done to encourage the development of further extensions to the language with varying degrees of expressive power (Horrocks et al, 2000).

3.6.6. DAML+OIL

In response to the limitations of RDF and RDF(S), the DAML (DARPA Agent Markup Language) group was formed by the Defence Advanced Research Projects Agency (DARPA). DAML-ONT was developed as a result with the power of expressing more sophisticated RDF class definitions than were permitted by RDFS. This language was then soon combined with the Ontology Inference Layer (OIL) to form DAML+OIL (Ouellet and Ogbuji, 2002). DAML+OIL has well defined model-theoretic semantics as well as an axiomatic specification that determine the language's intended interpretations (Pulido et al, 2006). It also adds DL-based KR primitives to RDF(S). Concepts, taxonomies, binary relations, functions and instances can all be represented in DAML+OIL (Corcho et al, 2003).

3.6.7. OWL – Web Ontology Language

Based on DAML+OIL, OWL is designed to become a common language for ontology representation on the semantic web. It is designed as a vocabulary extension of RDF (W3C, 2004). Being a language for the semantic web; extensibility, modifiability and interoperability are its main features (Mizoguchi, 2004). It is less powerful than the first order predicate logic in logical expressions for axiom writing ((Masolo et al, 2001; Mizoguchi, 2004). Although, its compatibility with SHOE and DAML+OIL gives it a good expressive power its expressiveness still has some drawbacks. It is not easy to use, its reasoning is not very efficient and some of its constructs are very complex. Its complexity is one of the causes which divide it into three different types (Pulido et al, 2006). Two of these OWL versions, i.e. OWL DL and OWL Lite are basically the subsets of OWL Full (W3C, 2004).

3.6.7.1. OWL Lite

OWL Lite is suitable for describing classification hierarchies (Delugach, 2008). It is easier to implement and is designed to provide users with a functional subset that will get them started in the use of OWL. This is done mainly to support tool builders to construct tools using a very simple form of language (Delugach, 2008).

3.6.7.2. OWL DL

DL in 'OWL DL' stands for Description Logic. The main reason for having the OWL DL sublanguage is to support the existing description logic business segment. This is because tool builders have developed powerful reasoning systems which support ontologies constrained by the restrictions required for OWL DL. Although the language constructs of both DL and Full are similar, the use of some of these and RDF features is what makes them different. OWL DL imposes some constraints on its mixture with RDF while OWL Full allows free mixing of OWL with RDF Schema.

3.6.8. Common Logic

Common Logic (CL) is a framework for a family of logic-based languages. It is designed and developed as a medium for transmitting logical content on an open communication network (ISO/IEC 24707:2007(E), 2007). Its main purpose is to provide interoperability for both syntax and semantics for a family of first order logic languages (Delugach, 2008). CL does not have a single syntax; instead, common abstract semantics for CL syntaxes are prescribed by ISO. Three different syntaxes are specified namely: Common Logic Interchange Format (CLIF), Conceptual Graph Interchange Format (CGIF) and Extended Common Logic Markup Language (XCL). The syntax of CLIF is based on KIF, that of CGIF is founded on conceptual graphs (CG), while XCL is based on XML. CL contains declarative semantics which make it possible to understand the meaning of an expression written in SCL without the help of an interpreter (Frankel et al, 2005). CL is more expressive than OWL DL and OWL Lite while OWL Full's expressiveness is comparable to CL's. However, CL being based on first order logic (FOL) has an edge over OWL Full because FOL is older and more developed than description logic (Delugach, 2008).

The ontology development languages discussed here have developed with the gradually increasing demands of interoperability and knowledge sharing. Every new language fills the

loop holes left by the preceding one. This trend had led these languages to progressively excel. The selection of a specific language for a job, however, depends upon the objectives to be achieved. Each one of these languages has its own peculiar strengths and weaknesses depending upon the task at hand. A close scrutiny is therefore needed before a decision can be made about their selection for a particular assignment. The reason why CL is used in this research is mainly due to its high expressiveness. The syntax of CL used here is KFL (Knowledge Frame Language) which allows the ontology builder to bind up to five different classes into one relation ([Anonymous], 2010). This capability cannot be found in any of the ontology development languages reviewed here.

Available literature on ontology classification, development and the languages for this purpose has been reviewed so far. It is also useful to evaluate some of the already developed ontologies which are being used for the purpose of knowledge sharing. This review is what follows.

3.7. Existing Foundation Ontologies

The ontologies discussed below, in no way represent the entire ontology landscape. These ontologies, however, can be considered as representative of their specific domain or broader working area. The PSL ontology for example is useful for representing processes while the Cyc ontology is a very general purpose ontology and can be mapped with other ontologies to provide inferential support about the commonsense knowledge. WordNet on the other hand is helpful in the natural language processing area. These ontologies are further discussed below.

3.7.1. Process Specification Language (PSL)

PSL or process specification language is an ontology designed to aid semantically sound exchange of process information between different manufacturing setups (Gruninger, 2004). It defines a neutral representation for manufacturing processes that supports automated reasoning (NIST, 2003). Included in these processes are scheduling, process modelling, process planning, production planning, simulation, project management, workflow and business process reengineering (Gruninger, 2004). It facilitates interoperability by means of the development of translators between PSL and native

formats of the applications intending to interoperate. Without a standard language like PSL, the number of translators needed for interoperability between ontologies are n(n-1). While in the presence of a standard like PSL this number reduces to n. This is because only the two-way translation between native ontologies and PSL is needed (Schlenoff et al, 2000). The primary component of PSL is an ontology containing some primitive concepts which are claimed to be adequate to define simple manufacturing, engineering and business processes. In order to make it explicit, and to make the semantics of the ontological terms sharable, the concept in a PSL ontology is defined by using three notions;

1. Language
2. Model theory
3. Proof theory

The Language part of the ontology consists of a set of symbols and the specification for the use of these symbols to make meaningful sentences. The grammar used for PSL is roughly based on the grammar of KIF (Knowledge Interchange Format) which is based on first order logic. The main purpose of Model Theory is to give a mathematical form to the semantics or meanings of the terms in the language. The Proof Theory further consists of three components (Schlenoff et al, 2000);

1. The PSL Core
2. Foundational Theories
3. PSL Extensions

The PSL core is a set of primitive axioms written in CLIF (Common Logic Interchange Format) to support the non logical part of the PSL lexicon. Its purpose is to axiomatise a set of semantic primitives for describing some fundamental concepts of manufacturing processes. The expressiveness of the PSL core is limited and therefore a set of extensions are used to enhance it (NIST, 2008). The foundational theories overcome the weaknesses of PSL core by providing precise definition of and axiomatisations for its primitive concepts. PSL extensions on the other hand add more complex processes to PSL core concepts. Since PSL core is a very basic and primitive ontology, it needs extensions for expressing concepts with higher levels of complexity (Schlenoff et al, 2000).

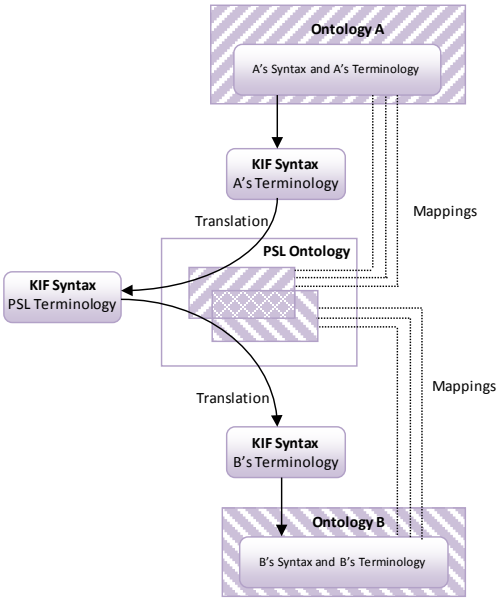


Fig. 3.8 : Mapping and translation process through the PSL ontology

Figure 3.8 shows the way in which the PSL ontology can be used to facilitate interoperation between disparate computer systems. The process starts from mapping between ontology A and ontology B with the PSL ontology. But this mapping can only be done for the overlapping areas of the two applications (the cross hatched area in the figure). The rest of the information has to be translated syntactically and semantically first into KIF syntax and then into the native terminology of an ontology. Two translators are used here and four different types of translations take place. First A's syntax and terminology is converted into a KIF syntax of A's terminology. Secondly, to acquire the required information from PSL ontology, A's terminology is translated into PSL's terminology within the KIF syntax. Thirdly, this PSL terminology is converted into B's terminology still staying in KIF syntax. Finally, KIF syntax with B's terminology is translated into B's syntax and B's terminology (NIST, 2008). Hence, the PSL ontology is used as a top or upper level ontology while ontologies of application A and application B act as domain ontologies belonging to design or manufacturing as an example.

A user interface can be developed to separate the entire processing from the user's view. This can be done by using the textual PSL encoding language which can provide a user friendly point and click interface to perform the whole process. A set of PSL tools may also

be used to obtain mapping support for the overlapping knowledge between the two ontologies (NIST, 2008).

3.7.2. SUO (Standard Upper Ontology)

The Standard Upper Ontology is a foundation or upper ontology covering a very general range of ideas. It is being developed by the SUO working group (SUO WG) under the IEEE Standards Association (IEEE SA). It is a foundation ontology because it will not include concepts specific to a certain domain, rather it will provide a base and a structure upon which other domain ontologies (like engineering, finance, medical) would be constructed (SUO WG, 2003). It will provide definitions for general-purpose terms and acts as a basis for these domain ontologies. It is estimated to contain around 2500 terms and on average ten definitions for each term. It is built in a simplified version of KIF called SUO-KIF (Niles and Pease, 2001). The main purpose of SUO is to provide automated reasoning and interoperability between several diverse areas. The key target areas are e-commerce, education and understanding of natural languages. Some other candidate ontologies are also entailed by SUO including SUMO, OpenCyc, IFF, 4D Ontology, MSO and Lattice of Theories. A few of these participating ontologies are discussed further in the following text.

3.7.3. Suggested Upper Merged Ontology (SUMO)

SUMO is an ontology obtained by mapping and merging several other publicly available ontologies into a single comprehensive form. It was developed with extensive help from the SUO working group. It is claimed to be the only ontology which has been mapped to all of the WordNet Lexicon (SUMO, 2009). It is developed to aid research in the fields of search, linguistics and reasoning. The language used for knowledge representation in SUMO is KIF-SUO. SUMO also includes the PSL Core but due to its limited variety of processes, some more verb classes are added to define a more comprehensive sub-ontology of processes. The root node of SUMO is divided into Physical and Abstract classes. The Physical class contains all those entities which have a position in time and space while the Abstract class includes everything else. Figure 3.9 shows the top level of SUMO (Niles and Pease, 2001). SUMO consists of 11 sections. These sections include representation of the structure of the ontology through relations, some fundamental ontological notions, numeric information,

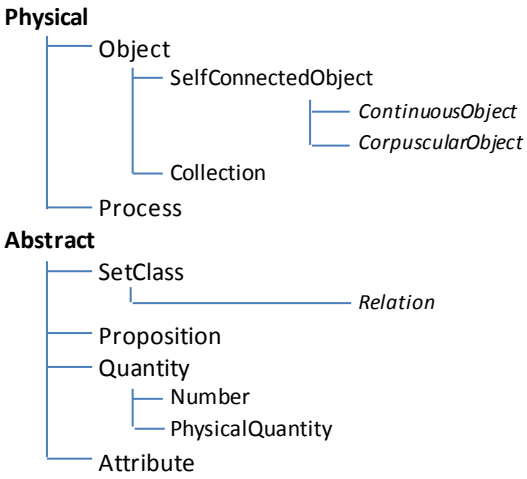


Fig. 3.9: SUMO Top Level

mereotopological content such as part-whole relations, knowledge about physical measurement units, and process, object and attribute types (Pease et al, 2002).

3.7.4. WordNet

Although not holding a unanimously agreed status of being an ontology, the inclusion of WordNet in this section is due to its widespread use in the ontology development community for obtaining linguistic support. WordNet is the largest freely available lexical database of English developed in Princeton University. It contains nouns, verbs, adjectives and adverbs grouped together into sets of cognitive synonyms called 'synsets'. These synsets express distinct concepts and are interlinked through conceptual-semantic and lexical relations (WordNet, 2006). There are around 80,000 noun synsets in WordNet (Deng et al, 2009). WordNet has become the lexical database of choice for Natural Language Processing (NLP). This is because words present in WordNet can formally be distinguished by their context and subject (Boyd-graber et al, 2008). The quality and quantity of relations between the four main parts of speech constituting WordNet are not very rigorous. For example, the number of links present between the terms are limited and are not exhaustive, there are no cross-part of speech links, and the weight or strength of relations between terms is not considered (Boyd-graber et al, 2008). Some expansion efforts are being done for WordNet. Deng and colleagues (Deng et al, 2009) for example have started a project to associate images to the terms in WordNet. The resulting lexicon will be called ImageNet and

is expected to contain 50 million cleanly labeled, full resolution images (50-100 per synset) (Boyd-graber et al, 2008).

3.7.5. Cyc Ontology

The primary aim of the Cyc project is to develop a large knowledge base appropriate for supporting reasoning in a range of domains (Matuszek et al, 2006). Developed by Cycorp, Cyc ontology is a part of Cyc technology, which is claimed to be the world's largest and most complete general knowledge base and commonsense reasoning engine. It contains 3 million assertions and describes more than 300,000 terms (Cycorp Inc., 2009). The part of the ontology available to the public is called OpenCyc and it provides a foundation of concepts plus an expressive language called CycL that supports the Cyc ontology. The openCyc is therefore useful for ontology developers. CycL provides a quoting mechanism which enables the user to differentiate between the knowledge defining a concept and the knowledge about the terms defining the concept. The Cyc foundation ontology provides knowledge representation about the individuals and their relationships with space time and human perception. The Cyc knowledge base is traditionally subdivided into three parts namely the Upper, Middle and Lower ontology depending on the level of generality of concepts it represents (Matuszek et al, 2006). A more extensive version of Cyc has recently been made available for research purposes under a research licence and is called ResearchCyc. Three recent additions to the Cyc technology are a natural language query tool, a template based fact entry tool and ontology exporter which enables the user to export selected portions of the knowledge base to OWL files (Cycorp Inc., 2009).

A brief review of these top level ontologies was necessary because the presented research also involves the use of a foundation ontology for knowledge verification through ontology mediation.

3.8. Conclusions

This chapter presented an introduction to the concept of ontologies. It is clear that ontologies can be used as a means to develop interoperable computer-based knowledge sharing systems. Some foundation ontologies exist which are already being used for this purpose.

Knowledge sharing through ontologies, however, is not without its weaknesses. The next chapter, therefore, looks into the problems that may occur during the process of knowledge sharing through ontologies.

Chapter 4: Literature review

4.1. Chapter overview

This chapter first establishes that ontology matching and mapping is directly linked to the process of verification of knowledge shared through ontologies and then a review of the literature on existing tools and techniques for ontology mediation is presented.

4.2. Cross-domain knowledge verification

When formalized knowledge is shared between different domains, prevention of its subjective interpretation becomes necessary. This process of authentication of the interpretation of knowledge, in this research, is referred to as knowledge verification. In the field of software development the term verification is usually used in conjunction with the term 'validation' and has slightly different connotations. Verification in this sense refers to the correct building of the product while validation ensures building the right product (Boehm, 1984). It is clear that in this sense these two processes correspond to a very early stage of the development of knowledge bases and they actually provide a means to measure the quality of knowledge in a knowledge base (Preece, 2001). Another description of verification is given by (Gupta, 1993). According to him knowledge verification involves

1- Checking the completeness, consistency and correctness of knowledge,
2- Determining the accuracy and consistency of the reasoning mechanism in interpreting and applying the knowledge in the knowledge base to solve problems and
3- Comparing the system with its human counterparts and grading it thereof.

The above three points indicate that verification can either be required for the knowledge contained by a knowledge base or for the software application and ontology (Lucanu et al, 2005) which organizes, facilitates accesses and presents knowledge needing verification. For the verification of knowledge itself, there can be five main approaches (Preece, 2001):

1. Inspection: is the process of human proofreading the text. This is the most commonly used and the least reliable technique of knowledge verification.
2. Static Verification: verifies the knowledge base by checking the logical anomalies (mostly redundancy and conflict). Redundancy is the uselessness of knowledge while conflict is the mutual inconsistency in the logical statements of the system.

3. Formal Proof: is a more thorough form of logical analysis than the static verification and it is used during the development process to verify the specified requirement of formal artefacts. This technique supports both verification and validation (Meseguer and Preece, 1996).
4. Cross-Reference Verification: is performed by cross checking the KBS for a particular knowledge entity at different levels. This cross checking can also be performed by using the different views of a knowledge model e.g. product view, product life cycle view etc. (Preece, 2001).
5. Empirical Testing: is done by executing different use cases. This testing can either be functional based or structure based depending upon the requirement of verification. To verify the functioning of the software application or the authenticity of the ontology working behind it the Use Case technique can be used. PRONTO is a software tool used for this purpose (Coenen et al, 2000). It operates by using the ontology to generate test cases. These test cases are then carried out against the knowledge base. In case of failure either the ontology or the knowledge base is checked for errors and is redefined. The success of the test case verifies both the knowledge and the knowledge-based system.

An important point to note here is that there is a difference in the verification of knowledge in a general sense and the verification of knowledge associated with ontologies. The ontology-based information sharing approaches usually rely on mapping between ontologies (Kaza and Chen, 2008). Mapping is the process in which correspondences are found and established between concepts in two ontologies. These correspondences are important for knowledge verification because establishment of similarities between two ontologies ensures that the understanding of knowledge shared through these ontologies is correct. Due to its importance, the ontology mapping and other relevant processes are explained further in the next subsection.

4.2.1. Ontology Mapping

Mapping is the process in which for each concept in the source ontology a corresponding concept with similar semantics in the target ontology is found (Ehrig and Sure, 2004). This is

not a standalone process, it is just a small 'fragment of a more ambitious task concerning the alignment, articulation and merging of ontologies' (Kalfoglou and Schorlemmer, 2003).

1. Ontology alignment is the automated or semi-automated discovery of correspondences between two ontologies (Lourdusamy and Ganapathy, 2008),
2. Ontology articulation defines a way in which the fusion or merging of ontologies has to be carried out (Kalfoglou and Schorlemmer, 2003) and
3. Ontology merging is the process of creating a new ontology which is the union of source ontologies for the purpose of obtaining a bigger and richer knowledge base (Lourdusamy and Ganapathy, 2008).

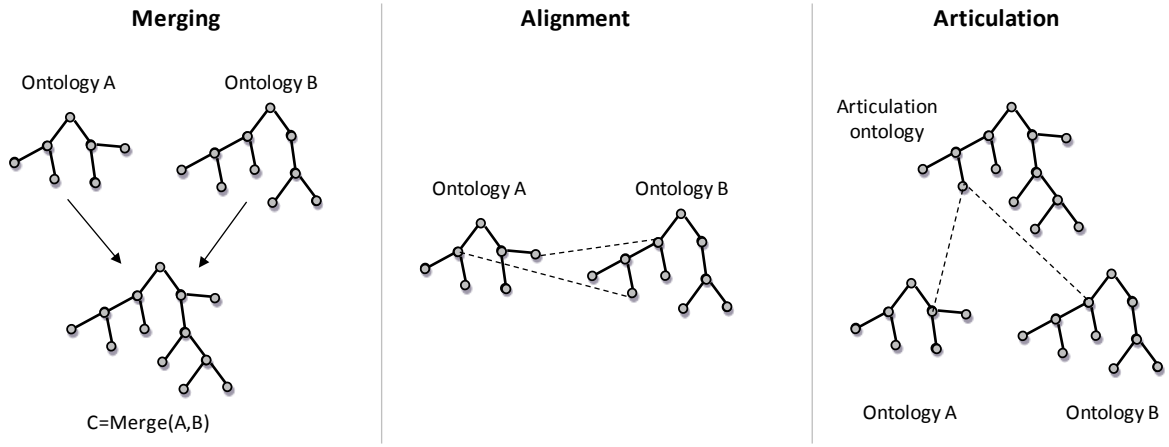


Fig. 4.1. Different Ontology Integration Methods

These three processes are represented graphically in Figure 4.1. Typically, a mapping process consists of three main stages (Bruijn et al, 2006).

1. Mapping discovery,
2. Mapping representation and
3. Mapping execution

Since there needs to be a similarity in the ontologies to be mapped, the mapping discovery stage involves a search for this similarity. Once the similarities are detected a mapping plan is generated in the mapping representation stage and finally the mapping is executed. It is important to note that there is an implicit assumption here about the syntactic structure of ontologies. This three-stage process assumes that the two ontologies to be mapped are written in the same language while in the real world scenario this might not be the case

(Lourdusamy and Ganapathy, 2008). Hence a more comprehensive and complete representation of the mapping process is given in figure 4.2, which also includes the process of ontology import.

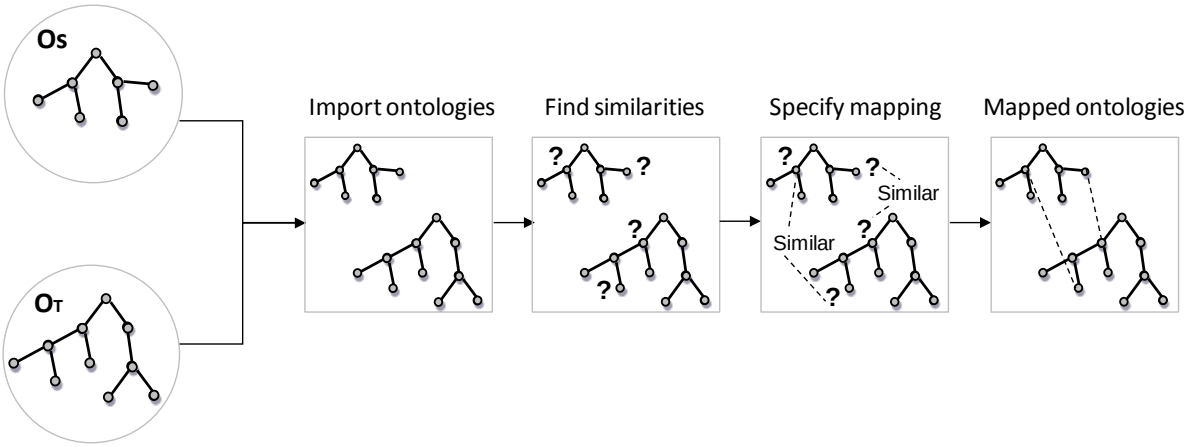


Fig. 4.2 The Mapping Process

To emphasize, it is reiterated that the knowledge verification in this research is all about the stage of mapping discovery or similarity finding across two ontologies belonging to different domains. This stage, however, is not straight forward especially when ontologies from different domains are mapped. This is because; first of all, there can be semantic differences in the use of concepts across different domains. Secondly, when ontologies are developed independently, there can be differences in their structure and content, otherwise known as ontological mismatches (Visser et al, 1997). The understanding of these mismatches is, therefore, essential for understanding the similarity-finding across two ontologies. These mismatches are discussed next.

4.3. Ontology Mismatches

Due to the possible heterogeneous nature of ontologies, the mapping process is subjected to mismatches in their components and building blocks. Being the specification of a conceptualization, an ontology consists of five components and sets of their definitions. These are: a set of Class definitions, a set of Function definitions, a set of Relation definitions, a set of Instance definitions and a set of Axiom definitions (Visser et al, 1997). Differences in the way these five components are defined give ways in which ontologies can mismatch. This section on Ontology Mismatches is mainly based upon the work of Visser et

al (1997) but in addition to this categorization, some other mismatches identified by other authors are also briefly listed afterwards.

The work of Visser et al (1997) is a pioneer in the ontology mismatch area and is quoted by several authors such as (Hameed et al, 2004; Chungoora and Young, 2008; Smart and Engelbrecht, 2008). They divide ontological mismatches into two main categories. Conceptualization mismatches and Explication mismatches. These categories are explained next with the help of an example illustrated in figure 4.3 (on next page).

4.3.1. Conceptualisation Mismatches

These mismatches occur either due to a difference in the way conceptualisations are distinguished in two ontologies or the way they are related to each other in an ontology. Hence, two different types of mismatches are Class mismatches and Relation mismatches.

4.3.1.1. Class Mismatch

This mismatch occurs due to the way classes in two ontologies are differentiated from each other. This mismatch can further be divided into two types namely a categorisation mismatch and aggregation level mismatch.

4.3.1.1.1. Categorisation mismatch

A categorisation mismatch takes place when two similar classes in two ontologies contain different subclasses. For example a class Engine can have subclasses titled Engine Block, Inlet Manifold, Exhaust Manifold or the subclasses can be Cylinders, Ignition assembly, Fuel injector etc.

4.3.1.1.2. Aggregation level mismatch

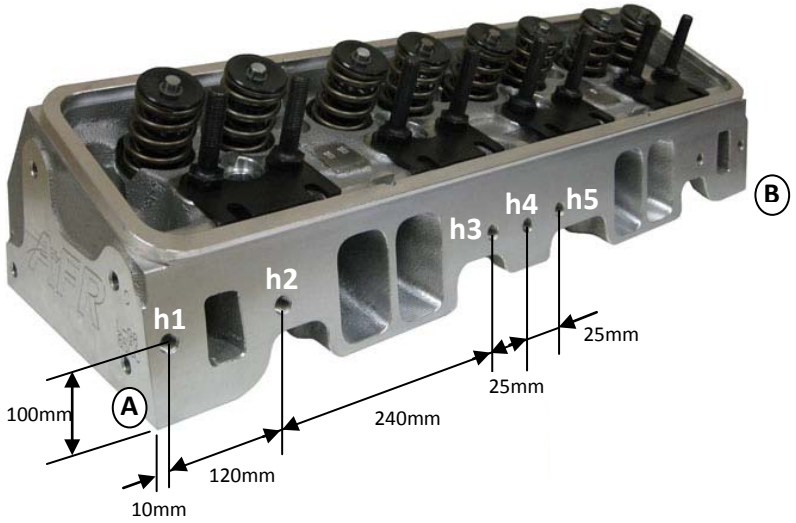
An aggregation level mismatch arises when two ontologies define a similar class at different levels of abstraction. For example, in Figure 4.3, in Ontology 1, the class FailsWhen defines the cases of cylinder head failure at a higher level of abstraction by giving a reason for the cylinder pressure loss. It shows the minimum cylinder pressure as a subclass of the machined surface flatness reduction and the hardening of the sealing gasket. While in ontology 2 the class FailsWhen directly gives the minimum pressure value without going into the details of the reasons of this pressure loss.

Ontology 1

Cylinder Head

- isMadeOf
  - Cast_Iron(SAE 1018)
- Contains
  - Machined_Surface
  - Plain_Holes
  - Tapped_Hole
- Involves
  - Surface_Machining (Parameters: Feed(a)^Speed(b))
  - Drilling
  - Tapping (h1:location(10,100)^size(M10)^h2:location(h1+120,100)^size(M10))
- IsUsedFor
  - Cylinder_Sealing (min_cylinder_pressure (x N/m^2))
  - Holding_CamShaft
  - Holding_SparkPlug
- FailsWhen
  - Machined_Surface_Flatness < yμ
    - cylinder_pressure < x N/m^2
  - Sealing_Gasket_Hardness > z bhn
    - cylinder_pressure < x N/m^2

Ontology 2



Cylinder Head

- CastedWith
  - Cast_Iron(BS 080A15)
- Contains
  - Machined_Surface —*Involves*→ Surface_machining (specifications: Feed(a)^Speed(b))
  - Plain_Holes —*Involves*→ Drilling
  - Tapped_Hole → Tapping (h1:location(10,100)^size(M10)^h2:location(130,100)^size(M10))
- IsUsedFor
  - Cylinder_Sealing
  - Maintaining_Pressure (min_cylinder_pressure (x N/m^2))
  - Contains_IgnitionAssembly
- FailsWhen
  - Cylinder_Pressure < x N/m^2

Figure 4.3. Two ontologies of a cylinder head with some mismatches

(Figure is developed especially for explaining examples)

4.3.1.2. Relation Mismatch

This mismatch happens due to the difference in relations and attributes of classes. Three further subdivisions of this type of mismatch are structure mismatch, attribute assignment mismatch and attribute type mismatch.

4.3.1.2.1. Structure mismatch

This type of mismatch comes up when a conceptualisation is specified in two ontologies using a similar set of classes or subclasses but the structuring and relation setting is different. For example, in Figure 4.3, ontology 1, the cylinder heads ontology has two different classes namely Contains and Involves for defining the components of a cylinder head and the manufacturing processes involved respectively. While in ontology 2 the manufacturing processes are a subclass of Contains class.

4.3.1.2.2. Attribute assignment mismatch

This mismatch occurs when two ontologies assign attributes to two similar classes differently. For example, in Figure 4.3, ontology 1, the minimum pressure attribute is assigned to Sealing_Cylinders, which is a sub class of UsedFor. While in ontology 2 this attribute is assigned to Maintaining_Pressure which is also a subclass of UsedFor. These two ontologies in this scenario face an attribute assignment mismatch.

4.3.1.2.3. Attribute type mismatch

This type comes into play when two ontologies in their classes contain similar instances but these instances differ in the way they are defined. For example, in Figure 4.3, ontology 1 assigns the instance of SAE 1018 to Cast_Iron which is a subclass of isMadeOf class. While in ontology 2 the instance of the same set of class and subclass is BS 080A15. These two instances represent the same type of cast iron but differ in the selection of standards. SAE is an American standard while BS is British. The two ontologies face the attribute type mismatch here.

4.3.2. Explication Mismatches

Explication mismatches are due to the difference in the way conceptualisations are defined in an ontology. The definitions of classes, relations and instances are considered to be a 3-tuple. Terms, definiens and concepts i.e. Def=<T,D,C> (Visser et al, 1997). An explication mismatch can arise when any of these three components of the 3-tuple in two ontologies

are different in some way. The relation between the terms, definiens and concepts is that *definiens* use *terms* to define a *concept*. For example, the definition of a Pen can be 'a writing device' or it can be 'a hollow cylinder with an ink refill'. Both of these definitions attempt to describe the concept of a pen but they use different definiens and different terms. In the first one the definiens target the application of the pen while in the second the structure of a pen is made the basis of its description. With these differences in terms, definiens and concepts, there can be six combinations of explication mismatches in ontologies. These are: concept mismatches (C-Mismatches), definiens mismatches (D-Mismatches), term mismatches (T-Mismatches), concept and definien mismatches (CD-Mismatches), concept and term mismatches (CT-Matches) and finally term and definien mismatches (TD-Mismatches). These mismatches are discussed below.

4.3.2.1. Concept Mismatches (C-Mismatches)

These mismatches occur when the same concept in two ontologies is defined using different terms and different definiens. For example, the definition of a pen as explained above uses different terms and different definiens but describes the same concept.

4.3.2.2. Definien Mismatches (D-Mismatches)

These mismatches arise when in two ontologies different definiens but same terms are used to define a similar concept. For example, in Figure 4.3, ontology 1, the subclass 'Tapping' contained by the main class 'Involves' is defined with these terms: [h1:location(10,100)^size(M10)^h2:location(h1+120,100)^size(M10)] where the location of hole h2 is stated by adding the x-coordinate of h1 into the distance between h1 and h2. While in ontology 2, the same hole is located by using its original coordinates i.e. (130, 100). In this case the coordinates (h1+120, 100) and (130, 100) define the same conceptualization i.e. h2 using the same terms but the approach to defining them is different. In other words they use different definiens. This type of mismatch is called the definien mismatch.

4.3.2.3. Term Mismatches (T-Mismatches)

If in two ontologies, two similar concepts are described by using identical definiens but different terms then a term mismatch occurs. An example can be the way feed and speed for the process of machining is described in two ontologies in Figure 4.3. In ontology 1 the term Specification is used to designate the feed and speed for machining while in

ontology 2 the term of Parameters is used to explain the same concept. This type of mismatch is referred to as the term mismatch.

4.3.2.4. Concept and Definien Mismatches (CD-Mismatches)

This type of mismatch occurs when definiens in two ontologies use similar terms to describe two different concepts. For example, the tool used for the tapping process (making threads in holes) is called a Tap. The same term is also used for a liquid dispensing pipe. Hence the definitions 'liquid_dispensing_pipe' and 'threading_tool' both refer to the same term i.e. a Tap but use different definiens and describe two entirely different concepts. This type of mismatch is called a concept and definiens mismatch.

4.3.2.5. Concept and Term Mismatches (CT-Mismatches)

These mismatches take place when two different concepts are defined by using different terms but the definiens used to describe the concepts are similar in two ontologies. for example in an ontology of Automobiles, a cylinder head can be defined as' cylinder head → covers_cylinders ^ maintains_pressure but similar definiens can be used to describe a CNG fuel cylinder top → covers_cylinders ^ maintains_pressure. Hence, similar definiens are used to define different terms and different concepts.

4.3.2.6. Term and Definien Mismatches (TD-Mismatches)

Finally, mismatches occurring due to ontologies referring to a similar concept but using different terms and definiens. For example the third subclass of the class Used_For in two ontologies in Figure 4.3 uses different terms and definiens to describe the function of a cylinder head to hold a spark plug. In ontology 1 the subclass is named as Holding_SparkPlug while in ontology 2 the similar subclass is named as Contains_IgnitionAssembly.

4.3.3. Other mismatches

Mismatches explained by other authors mostly overlap with those described by Visser and colleagues. For example categorization and aggregation level mismatches of Visser et al are similar to the scope differences of (Wiederhold, 1994) and scope mismatch of Klien (2001) and Qadir et al (2007). Similarly, the concept and definiens mismatches of Visser et al have a counterpart as attribute scope mismatch in Wiederhold (1994) and homonym terms mismatch of Klien (2001). On the explication mismatch side, the concept and definiens

mismatch of Visser et al (1997) has an equivalence in Wiederhold (1994) and Klien (2001) with the names of 'attribute scope' and 'homonym terms' mismatch respectively. Similarly, the 'term mismatch' and 'definien mismatches' of Visser et al (1997) are referred to as 'naming differences' and 'encoding differences' respectively in Wiederhold (1994).

Other mismatches not finding any commonality with mismatches described by Visser et al (1997) are explained below.

4.3.1.1. Concept Description Mismatches

Known as Modelling Convention mismatch in (Chalupsky, 2000), this type of mismatch comes under the class mismatch of Visser et al who define different types of class mismatches but do not explicitly mention this type. A concept description mismatch occurs when a concept is defined using different sub or super-classes. For example, Chalupsky (2000) states that to distinguish between tracked and wheeled vehicles, a choice can be to make two subclasses Vehicle as Tracked-Vehicle and as Wheeled-Vehicle. Alternatively, an attribute of Wheeled can be defined with a relation of Traction-type.

4.3.1.2. Model Coverage and Granularity Mismatch

This is another type of the class mismatch of Visser et al defined by Klien (2001) and Chalupsky (2000) as model coverage and granularity mismatch. As the name suggests, this mismatch occurs when two ontologies define the same concept with different levels of granularity. For example, a list of names can come under a class Persons or to make it more detailed, the class Person can further be divided into Male and Female. This mismatch appears to be similar to the aggregation level mismatch of Visser et al (1997) but this similarity is not recognized by Klien (2001) and Chaplusky (2000).

4.3.1.3. Single vs. Multi-Valued Property

This is the first of three mismatches Qadir and colleagues (2007) claim to be different from the mismatches pointed out by other authors before them. This mismatch occurs when a data-type or object property represents the same class but takes a different number of values in two ontologies. The authors give an example of a class named Bank_Account. In the ontology of one bank, this class might take just one value because that bank does not

allow its clients to have more than one account but in another, the class with same name might allow multiple values according to its policy.

4.3.1.4. Unique vs. Non-Unique Valued Property

This mismatch occurs when in one ontology a property can hold only one value that uniquely determines the subject, while in another ontology there can be multiple values but they cannot identify the subject uniquely (Qadir et al, 2007). An example quoted from the authors explaining this situation is where in one ontology of a university a student is identified by a unique rank number which is recognized by all departments while in another ontology the university requires multiple ranks corresponding to different departments and none of them individually determines the student uniquely.

4.3.1.5. Alignment Conflict among Disjoint Relations

A mismatch occurring when a disjoint relation in one ontology is not valid in the other. For example a class Student can be declared as disjoint with the class Employee in one ontology while in another a student is allowed to be an employee of an institution as well (Qadir et al, 2007).

4.3.1.6. Syntactic Mismatches

Unlike Visser et al (1997), Wiederhold (1994) and Klien (2001) also give details of possible syntactic mismatches. These mismatches occur due to the difference in ontology development language used by the two ontologies. In the use of different languages, not only the syntax differs but the expressivity of language can also be a barrier in successful translation and mapping of two ontologies. Since this research covers ontologies built in the same language, a review of these mismatches is not relevant here and is therefore not further discussed. Table 4.1 gives a consolidated view of the semantic mismatches reviewed above.

It has already been established that ontological knowledge verification heavily depends upon the task of similarity finding or mapping discovery between the ontologies used to share knowledge. This means, that tools need to be developed that help the user in doing so by overcoming the above discussed and other ontological mismatches. A review of the

Table 4.1. The mismatches framework (Mm stands for 'mismatch')

Mismatch Category		Visser et al (1997)	Wiederhold (1994)	Klien (2001)	Chaplusky (2000)	Qadir et al (2007)	Cummulative Mismatches
Conceptualization Mm	Class MM	Categorization Mm	Scope Differences	Scope Mm		Scope Mm	Categorization Mm
		Aggregation-level Mm					Aggregation-level Mm
				Concept Description	Modelling Conventions		Concept Description Mm
				Model Coverage and Granularity Mm	Model Coverage and Granularity Mm		Coverage Mm
						Single vs Multi valued property	Single vs Multi valued property
						Unique vs Non-unique valued property	Unique vs Non-unique valued property
	Relation MM	Structure Mm					Structure Mm
		Attribute-assignment Mm					Attribute-assignment Mm
		Attribute-type Mm					Attribute-type Mm
						Alignment conflict among disjoint relations	Alignment conflict among disjoint relations
Explication Mm		Concept & Term Mm					Concept & Term Mm
		Concept & Definiens Mm	Attribute Scopes	Homonym Terms Mm			Concept & Definiens Mm
		Concept Mm					Concept Mm
		Term & Definiens Mm		Synonym Terms Mm			Term & Definiens Mm
		Term Mm	Naming Differences				Term Mm
		Definiens Mm	Encoding Differences	Encoding Mm			Definiens Mm

literature on ontology mapping shows that there are plenty of tools available for this purpose. It is therefore necessary that these tools are reviewed, and required improvements are identified. This review is presented next. The review of these tools in the next section is, of course, not exhaustive and there are a number of other tools present to perform the task of ontology matching. The selection of these tools is made on the basis of the frequency of their appearance in the books and journal papers on ontology matching and mapping tools and techniques.

4.4. Ontology matching and mapping tools and techniques

(Noy, 2004) identifies two main approaches to mapping discovery. One set of methods, called the heuristics-based approaches, rely on the matching of several characteristics of ontologies to find similarities. The other set of approaches establishes a common grounding with an upper-level or foundation ontology in order to facilitate the process of similarity finding or mapping discovery. In the following text, the existing heuristic-based techniques are explored first and then the foundation ontology based approach is discussed.

4.4.1. Heuristics-based ontology matching approaches

This section first reviews some of the existing heuristics-based ontology matching and mapping tools and then presents an analysis of these tools for their capability to overcome the mismatches identified in the previous section. Below these tools and techniques are discussed in more detail.

4.4.1.1. MAFRA – MApping FRAmework

As shown in Figure 4.4, the conceptual framework of MAFRA is divided into five horizontal and four vertical dimensions. The horizontal dimensions represent the sequential flow of the mapping process while the vertical dimensions symbolize the ongoing support for the horizontal dimensions to operate.

The mapping process starts with the Lift and Normalization stage where the two ontologies to be mapped are brought to the same level in terms of syntax and structure. This step attempts to eliminate the syntactic differences and makes the semantic differences clearer

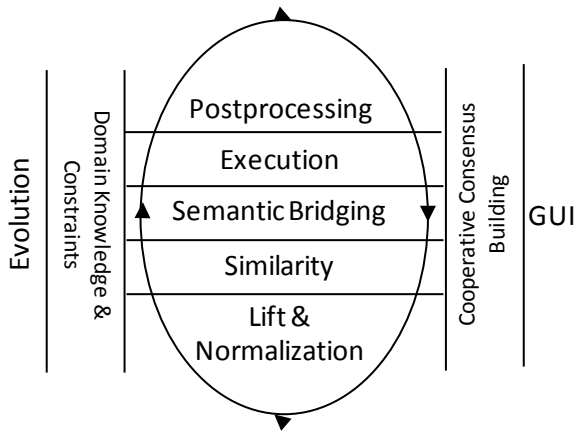


Fig. 4.4. Horizontal and vertical layers of MAFRA conceptual architecture

and more evident (Maedche et al, 2002). The output of this stage is a list of normalized lexica as the authors claim.

The next stage concentrates on finding the similarities between two ontologies in order to locate their linking points. This is the mapping discovery stage. Mainly two types of similarities are used namely the lexical similarity and the property similarity. Lexical similarity looks for identical concepts while the property similarity targets the concept attributes and relations of concepts with each other. The strategy adopted for similarity identification is to first scan the target ontology from bottom to top and then from top to bottom. The bottom to top scanning aims for the property similarity while top to bottom attempts to identify the lexical similarity. For each concept in the source ontology the entire target ontology is scrutinized and similarities are established.

In the semantic bridging phase the similarity points identified by the previous stage are used as stepping stones and a so called bridge is built between the two ontologies. The aim of this stage is to establish correspondences between the similar concepts in the two ontologies before the transformation and translation of instances from the target to the source ontology. This is done by forming a Semantic Bridge Ontology (SBO). Five distinct steps are involved in this stage.

It starts with concept bridging where by using the knowledge obtained from a thesaurus (like WordNet) terminological similarities or relations are found (synonymy, hypernymy etc).

Secondly, the properties, i.e. attributes and relations, of previously matched concepts are compared and matched. This is the point where the involvement of a domain expert might arise due to a difference in properties of the two matched concepts.

Thirdly, those instances in the source ontology are considered which cannot be matched with any of the target instances or concepts. This is the inference stage where on the basis of a similarity in the super-concepts of two different sub-concepts in two ontologies, inferences are made so as to declare and bridge the two non similar concepts. Again at this stage, human involvement might be necessary in order to examine the possibility of the two concepts being similar. Alternatively a history developed over time of the validated inferences can also be used to automatically authenticate the reliability of a particular inference.

Fourthly, the bridges established in the first three stages are verified and refined. This step is considered to be a complementary stage of the similarity module and is optional (Maedche et al, 2002).

Finally, the transformation specification step comes into play and provides directions for the execution stage on transforming and translating the instances from the target to the source ontology. Having assigned similarities and established bridges, the execution dimension finally transforms the marked concepts and instances for the source ontology.

The last horizontal module of the MAFRA conceptual structure i.e. post processing, verifies the object identity of the mapped concepts between the two ontologies. It ensures that the two concepts represent the same object in the real world.

Among the vertical dimensions of MAFRA, the evolution module helps in modifying the mapping according to the changes that take place in the source and target ontology with time. The cooperative consensus building provides coordination between the two communities, to which the two ontologies belong. This coordination is aimed at reaching a consensus on the validity of mapping relations and the interpretations of concepts. The role of ontological commitment appears useful here. The domain knowledge and constraints module provides some external help for the correct interpretation of concepts and their properties during the mapping discovery phase. This can be done by using different thesauri

or lexical taxonomies like WordNet. Finally, the graphical user interface attempts to make the handling of the whole MAFRA process easier and more manageable by providing a software interface useful in supporting the human intervention.

The uniqueness of this framework lies in the formation of a Semantic Bridge Ontology which matches the notion of articulation ontology in its description (Lourdusamy and Ganapathy, 2008). MAFRA is implemented as a plug-in of KAON which is an open-source ontology management infrastructure (Maedche et al, 2002).

4.4.1.2. PROMPT Suite

PROMPT suite (Noy and Musen, 2003) is a collection of four tools for ontology management. Among these tools, IPROMPT is designed to be an interactive ontology merging tool, ANCHORPROMPT helps in ontology mapping, PROMPTDIFF is a tool for comparing ontology versions and the fourth tool, PROMPTFACTOR provides help in factoring out a part of an ontology to form a sub-ontology. PROMPT suite is implemented as an extension or plug-in to the Protégé ontology editing environment. In the scenario of ontology mapping both ANCHORPROMPT and IPROMPT should be discussed because the latter provides the basis for the functioning of the former. The other two tools are not relevant from the ontology mapping point of view and therefore will not be discussed further.

IPROMPT is an interactive ontology-merging tool. It is interactive because it leads the user through the entire merging process. Although it is a merging tool, it does include the mapping discovery phase of the ontology mapping process. Its similarity detection function is primarily based on the lexical similarity of concepts and the relations (local context) between these concepts. But in addition to that it also suggests mapping on the basis of the

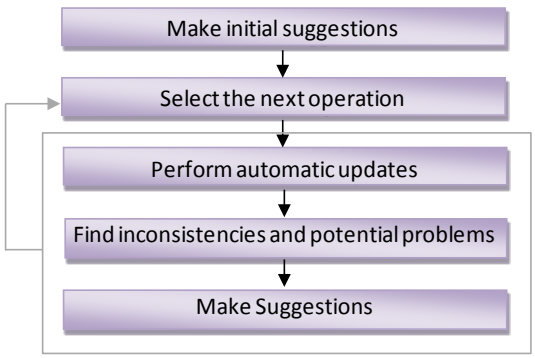


Fig. 4.5. IPROMPT Algorithm Flow

information it stores from user's actions. Furthermore, the Protégé component-based architecture allows users to plug in any term-matching algorithm. Figure 4.5 illustrates the flow of its algorithm. Users select the next operation after receiving suggestions from IPROMPT about the possible mappings. User actions are then incorporated and resulting changes are analyzed. It then highlights the problems arising due to these actions and again suggests possible actions. This process continues until a satisfactory result is obtained.

ANCHORPROMPT first finds the similar concepts in two ontologies. Alternatively these similarity suggestions can also be taken from the IPROMPT tool. The discovery phase of this tool is, however, more rigorous. It takes as input a set of pairs of similar terms (anchors) from two ontologies and automatically suggests semantic similarity between two apparently different terms. This is done by traversing along the path through which the two anchors are joined in the hierarchy. The algorithm works on the rule that if the concepts at the start and end of two hierarchical paths in two ontologies are equal then the terms connecting these

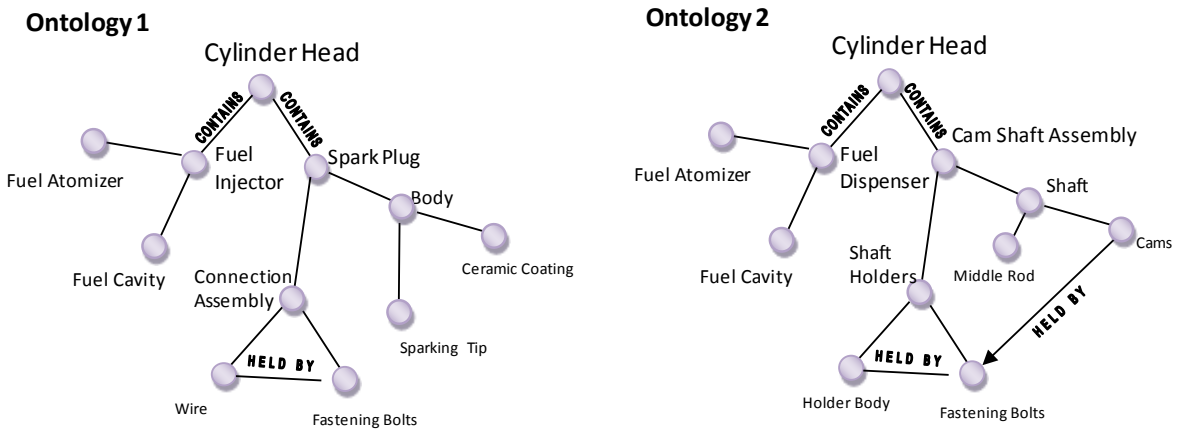


Fig. 4.6. Two Cylinder Head Ontologies
(Figure is developed especially for explaining examples)

concepts are also equal despite a difference in their names. As an example, consider Figure 4.6, which shows two ontologies modelling a cylinder head. A closer look at two ontologies reveals that there are several paths from top to bottom where the starting and ending nodes are similar but not necessarily the terms in-between. For example, Cylinder Head to Fastening Bolt or Cylinder Head to Fuel Cavity. ANCHORPROMPT calculates the number of nodes in each path and assigns a length. The length of a path is one less than the number of nodes present in it. It then calculates a similarity score based on the number of apparently

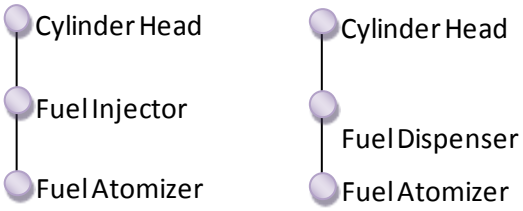


Fig. 4.7. Two paths with correctly detected similar terms

similar nodes. The paths with equal lengths are then compared and probability of apparently dissimilar terms to be similar is calculated. This probability depends upon the similarity score calculated earlier. Allowable length of detected paths is down to the user's discretion and longer paths are discouraged because of the possibility of less precise and wrong results.

As an example, consider the two paths in Figure 4.7, taken out of the ontologies in Figure 4.6. Here the starting and the ending concepts or terms are similar and therefore the two paths are anchored. Their similarity score is zero because no term apart from the starting concept matches with its counterpart node. This means that ANCHORPROMPT suggests Fuel Injector to be similar to Fuel Dispenser which is correct even though the similarity score was zero. Conversely, now consider the path "Cylinder Head – Cam Shaft Assembly – Shaft Holders – Holder Body – Fastening Bolt". This path is compared with two different paths in figure 4.8 and 4.9 but with similar anchors. In Figure 4.8, the similarity score is zero while in

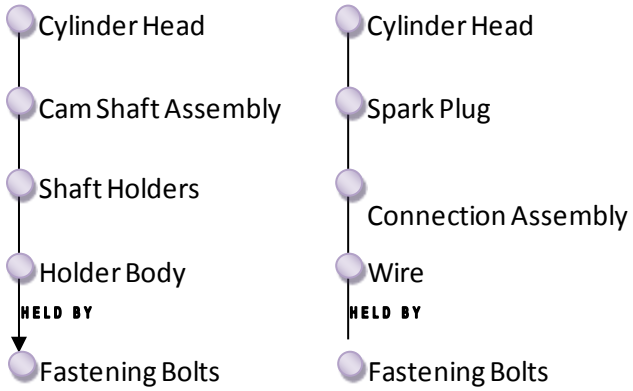


Fig. 4.8. Two similarly anchored paths with incorrectly suggested similarities and similarity score of 1

Figure 4.9 the similarity score is 1. Although the suggestions made are wrong in both comparisons a score of 1 means that the comparison in Figure 4.9 is more likely to reveal similar concepts than the comparison in Figure 4.8. To make this comparison process more reliable, ANCHORPROMPT uses a median score as the threshold below which all the

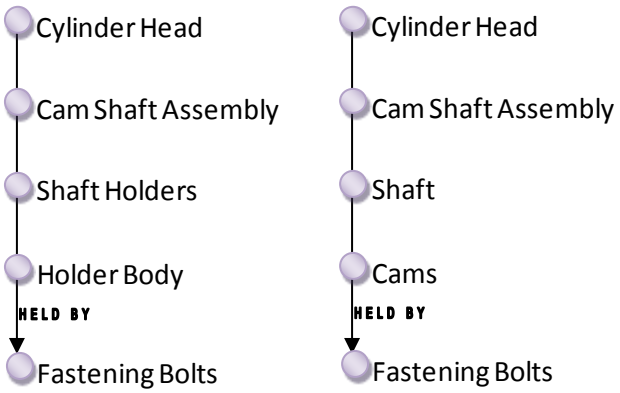


Fig. 4.9. Two similarly anchored paths with incorrectly suggested similarities with similarity score of 2

similarity measures are neglected.

Finally, ANCHORPROMPT also allows users to set the Equivalence-group size for classes, where the Equivalence-group size is the number of synonyms or is-a relations contained in an ontology. For example, the class Cylinder Head can have an is-a relation with a term, Engine Cover. The Equivalence-group size in this case will be 2. This size can be used to alter the classes scanned by ANCHORPROMPT to find similarities.

4.4.1.3. GLUE

GLUE is an ontology mapping system based on the machine learning approach. The word GLUE is not an acronym and its name actually comes from the fact that the semantic correspondences act as a 'glue' to bind the ontologies together in a semantic web (Doan et al, 2002). In GLUE, similarity measures are defined based on the joint probability distribution of the concepts involved. That is to say, for any two concepts A and B, P(A,B), P(A,B'), P(A',B) and P(A',B') are calculated. Where P(A,B') is the probability that an instance in a particular domain belongs to concept A but not to concept B. Similarly, P(A'B') is the probability that an instance in a domain does not belong to either A or B. An application can then be used to calculate a similarity measure based on these probabilities. Two applications used by the

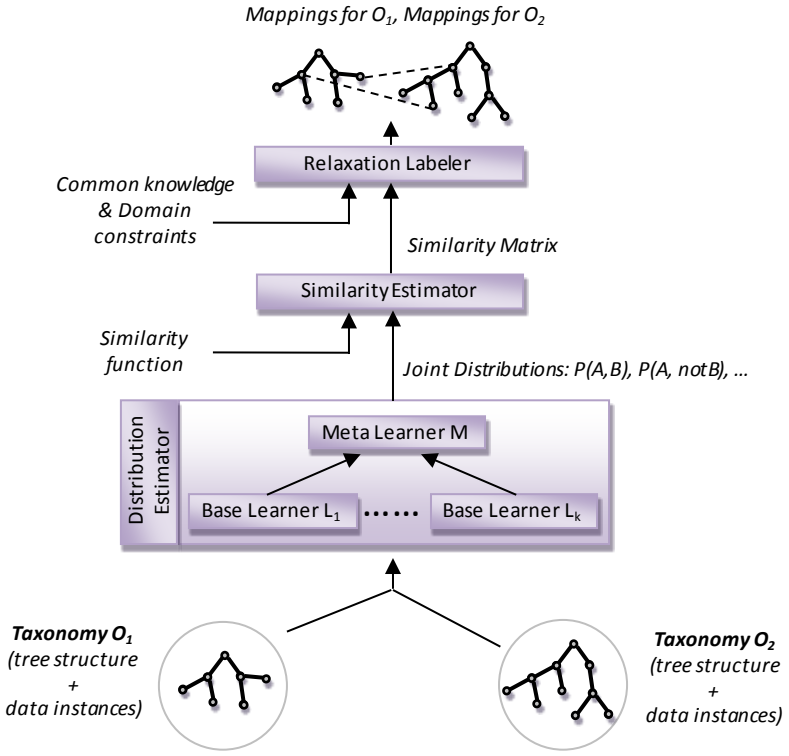


Fig. 4.10. Working of the GLUE architecture

authors to demonstrate GLUE are the Jaccard coefficient and Most-Specific-Parent similarity measures. Figure 4.10 shows the conceptual architecture of GLUE.

In the first step the ontologies to be mapped pass through the 'Distribution Estimator'. Here the joint distribution probabilities for certain instances are calculated. The Base Learner here calculates the probability of the two concepts in two ontologies being similar using different criteria. These criteria may include the checking of similarity in concept names, concept attributes, concept instances etc. Two learners are specifically described by the authors namely the Name Learner and the Content Learner. As their names suggest, the former targets the names of instances while the later aims for their textual content. The Meta Learner in the distribution estimator combines the probabilities provided by the base learners and computes an accumulated probability by giving a certain weight to each base learner as defined by the user. The joint distribution probabilities are then passed on to the Similarity Estimator at the next level. This module generates a similarity matrix based on the similarity measure defined by the user. Two similarity measures, as cited above, could be the Jaccard Coefficient and Most-Specific-Parent. Finally, the Relaxation Labeller applies some heuristics and domain constraints on this similarity matrix to find the most optimal

mapping configuration between the two ontologies. GLUE has been evaluated by using both of the above stated similarity measures and satisfactory results were obtained (Doan et al, 2002).

4.4.1.4. QOM

Quick Ontology Mapping is so named because it is a technique which is claimed to have a lower run time complexity than other mapping approaches (Ehrig and Staab, 2004) and therefore is quick. From the point of view of this literature, however, the thing which is important in this approach is its process flow. The authors give a very generic type of mapping process which is said to subsume nearly all mapping techniques as shown in figure 4.11. This process flow is then used to analyze QOM.

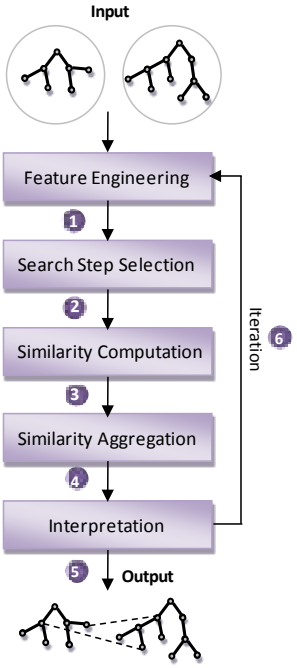


Fig. 4.11. QOM Mapping Process

The process starts with the Feature Engineering step where the ontologies to be mapped are brought to a level or a state in which they can easily be processed for similarity calculations. QOM exploits RDF triples for this purpose. The output of this process in QOM is a set of candidate-mappings, which are the possible sets of concepts to be checked for similarity.

The next step ‘Search Step Selection’ helps in choosing or restricting a certain area within the ontology in which the user is most interested for mapping in a specific iteration. QOM uses the structure of the ontology to make this choice. First of all the candidate mappings are ordered in a certain manner and some of them are totally discarded to increase mapping efficiency. Different strategies can be used to limit the number of candidate mappings. These limits can be set by either fixing a certain percentage of candidates to be processed or by considering label similarity. Additionally, a specific area in which mappings are found in previous attempts can be fixed for candidate selection or a top down approach can be used for this purpose.

The similarity computation step then uses different features of ontological entities to assess their similarity across the two ontologies. Four ontological entity features specifically mentioned are Concepts, Relations, Instances, and Property Instances. The similarity measures mentioned for this step are:

1. Object Equality, which uses the previously established iteration history or logical assertions to declare the two objects identical.
2. Explicit Equality checks if the logical assertions about the entities under consideration in two ontologies prove them similar.
3. String Equality looks for similarity in two concepts or entities in two ontologies by going through each character and comparing it with its possible counterpart in the other ontology.
4. String Similarity checks if the two concepts are nearly similar with minor differences in their names. Depending on the amount of difference a score between zero and 1 is assigned to each concept.
5. Dice Coefficient first finds the union and intersection of the sets of instances of two concepts in two ontologies. A ratio of cardinalities of the intersection to the union of these sets is then taken to calculate the similarity index.
6. SimSet measures similarity through an approach called Multidimensional Scaling. In this technique it is assumed that if two entities have equal similarity distances from some other identical entities around them in the two ontologies, then the two entities are similar.

Table 4.2. Ontological features and corresponding similarity measures used in QOM

Comparing	Feature	Similarity Measure
Concepts	Label	string similarity
	URI	string equality
	sameAs relation	SimSet
	direct properties	SimSet
	properties of direct super-concepts	SimSet
	direct super-concepts	SimSet
	direct sub-concepts	SimSet
	concept siblings	SimSet
	direct instances	SimSet
	instances of sub-concepts	SimSet
Relations	Label	string similarity
	URI	string equality
	sameAs relation	explicit equality
	domain & range	object equality
	direct super-properties	SimSet
	direct sub-properties	SimSet
	property siblings	SimSet
	property instances	SimSet
Instances	label	string similarity
	URI	string equality
	sameAs relation	explicit equality
	direct parent-concepts	SimSet
	property instances	SimSet
Property Instances	domain & range	object equality
	parent property	SimSet

Table 4.2 shows the features and similarity measures used in QOM. These similarities are then aggregated using algorithms of choice to get a single similarity value, which is then used to interpret mapping between the two ontologies either automatically or manually. Having completed the first round with a specific similarity feature the whole process is repeated with a different similarity feature. In QOM, lexical knowledge is given preference over knowledge structure when selecting the next iteration criterion. The authors of this technique believe that the restrictions imposed in step two, the search step selection, is what makes QOM more efficient than other similar techniques. However, in addition to its algorithm, QOM also serves as a framework of ontology mapping which any tool can use.

4.4.1.5. ONION

The ONtology CompositION system (ONION) attempts to semi-automatically resolve the terminological heterogeneity between ontologies and establishes articulation rules for meaningful interoperation. Through the use of an algorithm several heuristics are defined to achieve this aim. The authors of this technique claim that combining the information obtained by using multiple heuristics provides a better match between semantically related terms in the ontologies (Mitra and Wiederhold, 2002). In ONION this is done by finding linguistic similarity followed by structural matching. Based on a library of heuristic matchers, an automated articulation generator (ArtGen) suggests articulations. These suggested articulations are then rejected or accepted by a human expert. Articulation rules missed by ArtGen can also be defined manually.

The heuristic matchers used by the automated articulation generator are classified into iterative and non-iterative algorithms.

4.4.1.5.1. Non-Iterative Algorithms

As the name suggests, the non-iterative algorithms identify the matching concepts in one pass without a second attempt. The linguistic matching in ONION is performed this way. Here a similarity score is assigned to the potentially similar terms and depending upon a user defined threshold an articulation is generated or ignored. The similarity detection is done either through a thesaurus-based word relater or a corpus-based word relater. Thesaurus-based word relater uses dictionaries and thesauri like WordNet to find out if the two terms in two ontologies mean the same. The corpus-based word relater, on the other hand, uses a corpus of documents belonging to the domain of the ontologies to be matched and calculates word similarity scores based on the similarity of the contexts in which the words appear in the documents. The context is calculated by taking the cosine of context vectors. These context vectors represent the frequency of one word appearing in a 1000-character neighbourhood of another. This method is used to generate a table of word similarities to be used by the linguistic matcher. ONION does not use any instance based heuristics but the matcher can be extended to use instance information if needed.

4.4.1.5.2. Iterative Algorithms

Two types of algorithms that use multiple iterations to generate semantic matches between the concepts in two ontologies are used in ONION. The first one works on the principles of

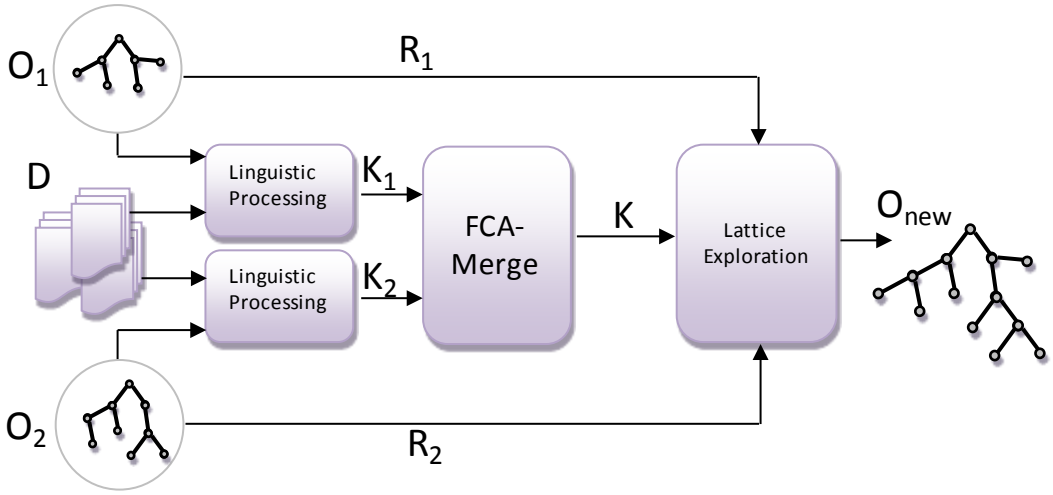


Fig. 4.12. FCA-Merge Ontology Merging Method

Structure-based heuristics and targets the subsumptions of those concepts already found similar through other heuristics. The second one uses inference-based heuristics where ontology rules are used to infer similarity or relations between concepts in two ontologies (Mitra and Wiederhold, 2002).

4.4.1.6. FCA-Merge

FCA-Merge is a semi-automatic ontology merging method, which employs natural language processing and formal concept analysis techniques to merge two ontologies. In the first step of linguistic processing, instances are extracted from a given set of domain-specific documents. A lattice of concepts is then derived from the extracted instances by using mathematical techniques taken from Formal Concept Analysis. At this stage, human intervention takes place and the lattice of concepts is explored and transformed to the merged ontology by the ontology engineer (Stumme and Maedche, 2001). Figure 4.12 shows the schematic of the FCA-Merge process. The figure shows how instances extracted from a set of documents D along with the two ontologies O1 and O2 are passed through

I_1	doc5	doc4	doc4	doc3	doc2	doc1
Concept1	✓		✓			✓
Concept2	✓	✓		✓	✓	✓
Concept3	✓	✓	✓	✓		✓
Concept4		✓	✓		✓	✓

I_2	doc5	doc4	doc4	doc3	doc2	doc1
Concept1				✓		✓
Concept2	✓	✓	✓		✓	✓
Concept3	✓		✓	✓	✓	
Concept4	✓	✓	✓		✓	✓

Fig. 4.13. The contexts K1 and K2 obtained through the linguistic processing of instances from the documents

linguistic processing to obtain two sets of formal contexts K1 and K2. The lattice K is then formed by merging the two formal contexts. The lattice is then analyzed by a human expert and a merged ontology O_{new} is created. The FCA-Merge core algorithm derives a common context from the instances taken from domain-specific documents. This method of context generation is very similar to the way contexts are generated in ONION. Figure 4.13 shows a rough depiction of how this context generation works by illustrating matrices obtained through linguistic processing of the documents. The first matrix shows that instance I_1 exists within the defined vicinity of Concept1, 2, 3 and 4 in doc1 while in doc2 it is just found near Concept2 and Concept4. The occurrences of an instance near specific concepts enables a score to be assigned to them showing the likelihood of an instance belonging to a certain context.

4.4.1.7. Chimaera

Developed by the Stanford University Knowledge System Laboratory (KSL), Chimaera is an ontology browsing, editing and merging tool. It also offers some testing and diagnostic features. The merging task from Chimaera's perspective consists of two main stages.

1. To coalesce two semantically identical terms from different ontologies so that they are referred to by the same name in the resulting ontology and
2. To identify terms that should be related by subsumption, disjointness, or instance relationships and provide support for introducing those relationships (McGuinness et al, 2000).

The most unique, and relatively the most useful, feature of Chimaera is its user interface (UI). An ontology engineer can easily browse, edit and merge ontologies with simple mouse clicks. This simple UI, however, has its disadvantages as well. The UI is only suitable for classes and slots and non-slot individuals and facets are not displayed. Furthermore, there is no support for axiom editing. Still the tool has some unique features like support for defining and editing disjoint partition information and offers an extensive set of diagnostic commands (McGuinness et al, 2000). A good display helps the user to perform many tasks not originally intended by the tool. Such as finding and resolving different structural mismatches in addition to the lexical dissimilarities in the ontologies to be merged.

For merging and evaluation, Chimaera generates name resolution and taxonomy resolution lists. A Name resolution list presents recommendations to merge two seemingly similar terms from two different ontologies while a taxonomy resolution list suggests taxonomy areas that need reorganization. The name resolution list uses term names, presentation names, term definitions, possible acronyms and expanded forms, names that appear as suffixes of other names etc. The taxonomy resolution list uses a number of heuristic strategies, e.g., one strategy involves the detection of those classes that have direct subclasses from more than one ontology (McGuinness et al, 2000).

4.4.1.8. Analysis of the heuristic-based ontology matching approaches

Table 4.3 shows a comparison of all the ontology matching and mapping tools reviewed above. These tools are analyzed for the similarity parameters and concept-matching rules that they use to match ontologies. It can be seen that most of these tools use lexical matching techniques for matching concepts, their instances and properties across two ontologies. Many tools use WordNet for finding synonyms and hypernyms of concepts and their attributes in the ontologies to be mapped. The main differences between these tools becomes apparent when their matching rules are analyzed. In this regard, an interesting and unique difference that two of the techniques ONION and FCA-Merge have, when compared to other techniques, is their use of domain specific documents to identify the context of a concept. Both of these techniques scan hundreds of textual documents relevant to the domain a concept belongs to and find the occurrence of terms around the concept. If refined to make it more accurate, this technique can be very useful in increasing the automation of mapping techniques to a considerably greater level. The effectiveness of this and other matching rules can be checked if these techniques are analyzed from the ontological mismatches viewpoint. This is done in table 4.4 (next page).

Table 4.4 uses the mismatches framework from table 4.2 and forms a matrix showing the mismatches overcome by these tools. Three symbols are used here to denote the capability of a particular method to detect and resolve a mismatch as done by Klein (Klein, 2001). ‘A’ stands for automatic and represents a capability of automatically detecting or resolving a mismatch. ‘U’ stands for user and indicates that the tool offers suggestions to the user to solve a particular mismatch, and ‘M’ denotes the mechanism provided to the user by a tool or technique, to detect or resolve a mismatch (Anjum et al, 2010). The table is

Table 4.3. Analysis of ontology matching and mapping tools

S.No.	Authors	Technique	Similarity Parameters	Matching rule used
1	Maedch et al (2002)	MAFRA (MApping FRAmework)	Lexical Similarity Property Similarity (attributes or relations)	Object Identity Establishment Statistical Analysis of Transformations
2	Noy & Musen (2003)	I PROMPT	Class names	Any term-matching algorithm can be plugged in
3	Noy & Musen (2003)	AnchorPROMPT	Anchor Points	
4	Doan et al (2003)	GLUE	Concept Instances	Similarity Metrics (Probability of similarity of Instances)
5	(Ehrig and Staab, 2004)	QOM (Quick Ontology Mapping)	Concepts, relations, instances and property instances.	Object equality, explicit equality, string equality, string similarity, dice coefficient, sim-set measures
6	Mitra & Wiederhold (2002)	ONION	Concept names	Context extracted from corpus based word relator
7	Stumme & Maedche (2001)	FCA-Merge	Concept names	Context extracted from corpus of domain specific documents
8	McGuinness et al (2000)	Chimaera	Term names, presentation names, term definitions, possible acronym and expanded forms, names that appear as suffixes of other names	Name resolution list and taxonomy resolution list

filled with the help of the review of ontology matching and mapping tools and techniques presented in this section.

A quick glimpse of this table reveals several empty fields representing a lack of available features in tools and techniques to detect and resolve conceptualization mismatches. Most of the tools and techniques provide a mechanism to the user to detect and resolve mismatches. It can be seen from table 4.4 that QOM (Quick Ontology Mapping) and Chimaera have a mechanism for the users to detect the conceptualization mismatches. This is because in QOM, the breadth of scope of similarity measure allows this technique to cover all of the possible mismatches. In Chimaera, however, it is its detailed and user

Table 4.4. Effectiveness of contemporary ontology matching tools to handle ontological mismatches

Semantic Mismatches		MAFRA		PROMPT		ANCHORPROMPT		GLUE		QOM		ONION		FCA-Merge		Chimera	
		Detection	Resolution	Detection	Resolution	Detection	Resolution	Detection	Resolution	Detection	Resolution	Detection	Resolution	Detection	Resolution	Detection	Resolution
Conceptualization Mm	Categorization Mm					M	U			M						M	
	Aggregation-level Mm					A	U			M						M	
	Concept Description Mm									M						M	
	Coverage Mm									M						M	
	Single vs Multi valued property							M		M						M	
	Unique vs Non-unique valued property							M		M						M	
	Structure Mm									M						M	
	Attribute-assignment Mm									M						M	
	Attribute-type Mm									M						M	
	Alignment conflict among disjoint relations									M						M	
Explication Mm	Concept & Term Mm									M						M	
	Concept & Definiens Mm (Homonyms)									M						M	
	Concept Mm									M						M	
	Term & Definiens Mm (Synonyms)	M		M	U	A	U	A	U	A	U	M	U	A	U	A	U
	Term Mm	M		M	U	A	U	A	U	A	U	M	U	A	U	A	U
	Definiens Mm	M		M	U	A	U	A	U	A	U	M	U	A	U	A	U

A	Automatic
U	Suggests solutions to the user
M	Provides Mechanism

friendly interface that helps the user to manually detect any kind of mismatches. Table 4.4, on one hand, shows that the available tools and techniques need to be made more automatic and on the other it indicates that these tools should be modified to target conceptualization mismatches. It is also clear from table 4.4 that the available tools and techniques mainly focus on finding the similarities rather than dissimilarities between the concepts in two ontologies and then establish correspondences. So, the main steps involved in every technique are:

1. Scanning ontologies for similar concepts,
2. Authenticating the similarity through different algorithms and tools,
3. Establishing correspondences.

The second step is the one which deals with the verification of knowledge in shared ontologies and it is here that the existing research appears to be focussed more on the explication side of terminologies and concepts than on the conceptualization side. Table 4.4 shows that only AnchorPROMPT provides an automatic detection of one of the conceptualization types of similarities. This tool also recommends correspondences that can be established between specific concepts in the ontologies to be mapped. The other two tools QOM and Chimaera just provide information about the structure of ontologies so that it becomes easier for the user to detect some conceptualization similarity (Anjum et al, 2010).

The gap identified here suggests that research is required to find ways through which different conceptualization mismatches can be detected and resolved in order to give accuracy to the process of mapping and thus verifying the knowledge being shared. An improvement in the mapping process will also aid the effort of making these tools and techniques increasingly automatic. This accuracy and automation is directly proportional to the interoperability between knowledge systems. Employment of a more accurate and automated approach is, therefore, vital from the interoperability perspective and needs to be considered seriously. The art of ontology matching is, however, not limited to the heuristic-based approaches. The other set of approaches, which require ontologies to be committed to a common ontology, also exist. This approach is discussed next.

4.4.2. Foundation ontology based ontology matching

Ontologies, being the explicit and formal specification of a conceptualization (Gruber, 1993b), provide a good platform for building shareable and interoperable knowledge repositories. They not only provide a way to preserve knowledge but also enable one to produce pre-packaged sets of information and knowledge available for individual use or for constructing large knowledge sets by using them as building blocks (Neches et al, 1991). These building blocks, if made available in the form of a shared or foundation ontology, can assist ontology builders to develop their own ontologies. Researchers agree that to make knowledge bases more shareable and expandable, instead of building them from scratch, it is more appropriate to develop them out of a single agreed upon foundation or standard (Neches et al, 1991). Foundation ontologies, as their name suggests, provide the basis for this standard. They make the expansion and integration of knowledge bases easier. This is because if two system builders build their knowledge bases on a common ontology, the system will share a common structure, and it will be easier to subsequently merge and share the knowledge bases (Swartout et al, 1997). These knowledge bases may also be accompanied by their own ontologies relevant to a specific domain. Such ontologies are called domain ontologies and as they provide a set of terms for describing some domain they can be thought of as taxonomies of relevant objects within that domain (Swartout et al, 1997). In these ontologies local vocabularies instead of standardized global vocabularies are formed in a particular context (Yang and Zhang, 2007). Example of domains may include aerospace, biology, manufacturing, arts etc.

In chapter 3 it was understood that some of the most famous foundation ontologies include Standard Upper Ontology – SUO (Niles and Pease, 2001), Suggested Upper Merged Ontology – SUMO (Niles and Pease, 2001), WordNet (Deng et al, 2009), DOLCE (Gangemi et al, 2002), and Cyc Ontology (Matuszek et al, 2006). Foundation ontologies like these may help to reduce semantic heterogeneity by restricting domain ontology builders to match their own conceptualisations against a common foundation, so that all communication is done according to the constraints derived from the ontology (Schorlemmer and Kalfoglou, 2005). These constraints, in a way, serve as a means of binding domain ontology builders to an ontological commitment. Ontological commitment is the process in which interested parties agree on the use of terminologies in an ontology. It helps in defining precisely the meaning

of a term (Gomez-Perez et al, 2004) and thus helps in sharing knowledge accurately with minimal misinterpretation. It is this agreement, in the form of constraints, which is the basis of the mediation approach discussed in this thesis.

Existing domain ontologies use foundation ontologies to communicate with other independently developed ontologies. This is done by first aligning domain ontologies with concepts in a foundation ontology and then based on these alignments between domain and foundation ontologies, the similarities between the domain ontologies are established as shown in figure 4.14. Some examples of this use of foundation or upper ontologies can be

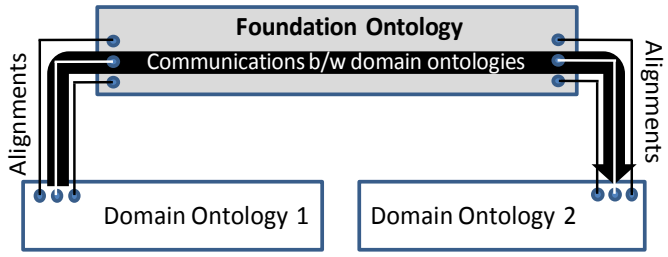


Fig. 4.14. Communication between domain ontologies via a foundation ontology

seen in the literature. In all cases, independently developed heterogeneous ontologies are provided with a pathway to communicate with each other by using an upper ontology as a semantic bridge. These examples are presented in detail in the next sections.

4.4.2.1. SMIF – Semantic Manufacturing Interoperability Framework

The semantic manufacturing interoperability framework of Changoora and Young (2010) also uses a foundation ontology and subsumed domain ontologies for semantic interoperability between knowledge sharing parties. It also features a domain ontology layer, which assumes that these domain ontologies are formed by specializing concepts from the foundation ontology as shown in figure 4.15 (next page). The foundation ontology here, thus, provides a basis to share meaning (Changoora and Young, 2010). The semantic reconciliation layer uses the logical assertions in the foundation ontology to retrieve mappings between similar concepts in two domain ontologies. This is done with the help of semantic mapping concepts. These concepts are formally defined mapping relations with added remarks for human interpretation. The mapping process involves the use of these relations to determine similarity between two concepts in independently developed domain ontologies. In the interoperability evaluation layer, these mappings are retrieved with the

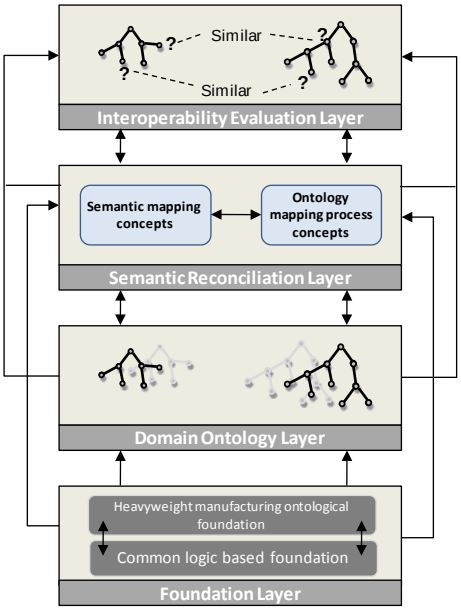


Figure 4.15. A depiction of SMIF

help of queries aimed at finding correspondences between concepts across two domain ontologies. The important aspect of this framework is the use of a foundation ontology to reconcile differences between two domain ontologies.

4.4.2.2. FOS – The Fishery Ontology Services project

Another example of the use of core ontologies for ontology mediation can be found in the work of Gangemi et al (2004). In this work, a Core Ontology of Fishery (COF) is developed and used for the purpose of reengineering, alignment, refinement, and merging of fishery knowledge organization systems. The COF is developed by specializing the DOLCE-Lite-Plus that is an extension of DOLCE foundation ontology containing ontologies of Descriptions and Situations. The part of their work that is relevant to this research includes the mapping of

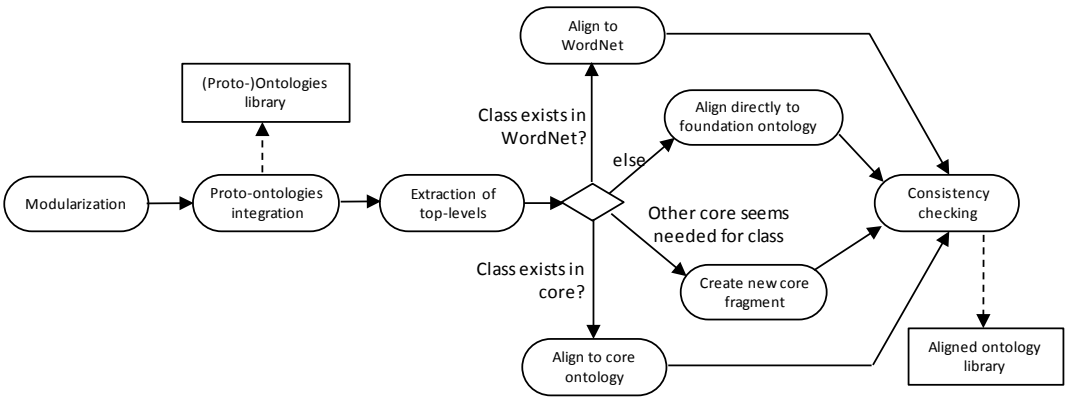


Figure 4.16. The activity diagram for modularization and alignment

domain ontology concepts with the COF. Figure 4.16 shows the step-by-step process. In the first three steps the domain ontologies, named proto-ontologies, are processed to extract the top levels. Once these levels are known, the concepts existing in the proto-ontologies are then matched against the concepts in the COF. This is done manually by making a certain class in a proto-ontology a sub class of an equivalent concept in the COF. If an equivalent concept is not found in the COF, the proto-ontology concept is then made a sub class of a similar concept in WordNet. If the concept to be aligned is very relevant to the fishery domain, a separate core ontology fragment is made for that concept and is added to the COF. If none of the above procedures work for a concept in the proto-ontology, it is aligned with a very general concept in the DOLCE foundation ontology. Once all the intended concepts are aligned, the consistency of alignments is checked. At this point all the proto-ontologies are aligned to the concepts in the upper ontologies and thus provide a bridge to share knowledge seamlessly.

4.4.2.3. Use of SENSUS for air campaign planning

SENSES is a broad coverage ontology developed by extracting and merging information from existing electronic resources. Being a 'broad coverage ontology' it also contains very general foundation terms such as 'inanimate object' along with the core concepts like 'submarine' (Swartout et al, 1997). SENSUS has more than 50,000 terms. In their work, Swartout et al (1997) first add 60 seed terms or domain concepts to the core SENSUS ontology. These concepts are added to the core concepts manually. This is illustrated in figure 4.17. The

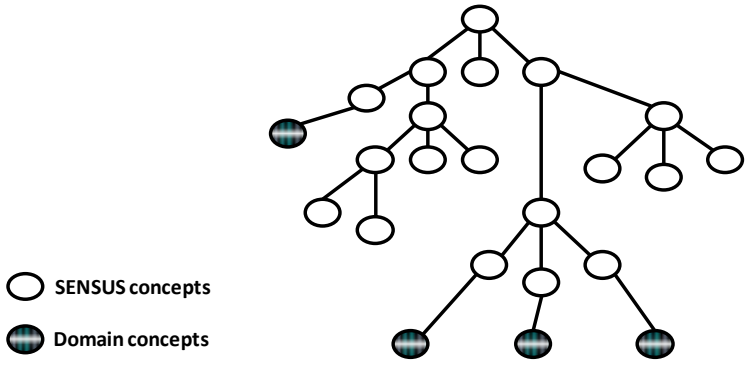


Figure 4.17. Linking domain terms to SENSUS

whole path from the newly added domain concept to the root concept in the core ontology is added to the newly formed ontology. In this way some paths are found to have many relevant terms which subsume a concept in the domain ontology. In these cases the whole

sub-tree is added to the newly formed ontology. Once the stage of concepts addition is completed, the irrelevant concepts are pruned from resulting ontology. After trimming down the irrelevant concepts and terms, the ontology contained approximately 1600 terms and concepts.

The authors argue that although this work intended on forming a domain ontology for air campaign planning, by using the structure of a core ontology, it also provides impetus to a method for accurate similarity finding during ontology mediation and thus may help in developing an interoperable system for seamless knowledge sharing.

4.4.2.4. The DOGMA framework

DOGMA (Developing Ontology-Guided Mediation for Agents) is a methodological framework for ontology engineering (Jarrar and Meersman, 2009). It deals with the cases where a centralized shared ontology is used to define more specific application ontologies to be used by certain applications. In this way the centralized ontology provides a basis of shared meaning. The framework introduces the notion of an 'Ontology Base', a 'Domain Axiomatization' and an 'Application Axiomatization'. Axiomatization here refers to a set of rules in an ontology defining the intended meaning of a concept and dictating its use. In the framework, the ontology base is intended to capture plausible domain axiomatizations. A domain axiomatization is an axiomatized theory that accounts for the intended meaning of domain vocabularies (Jarrar and Meersman, 2009). Particular applications can commit to the ontology base through an application axiomatization as shown in figure 4.18. The authors call this commitment the applications 'ontological commitment', which likewise is in

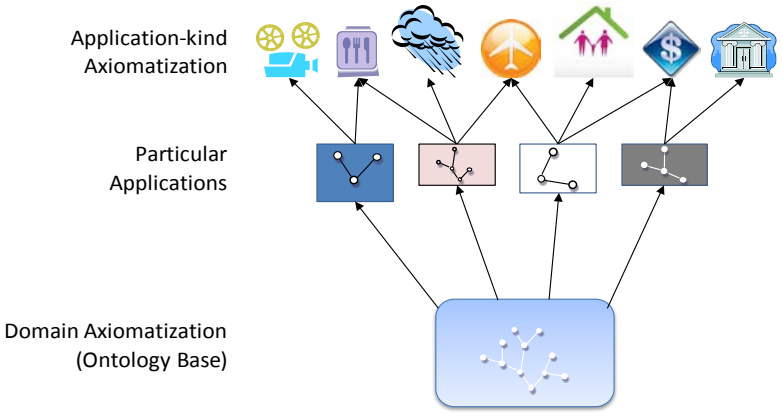


Figure 4.18. The DOGMA framework

the form of axioms and rules. The rationale behind the proposition of this framework is the fact that a complete and exact meaning of domain vocabulary is impossible to define with the help of a logical theory such as the axiomatizations mentioned here. This is because the use of a concept in different domains or at different levels of ontology may be different and a strict definition may restrict the use of that concept or vocabulary for different domain or application ontology builders, thus causing difficulties. Due to this reason, it is argued, the use of a vocabulary should not be precisely defined at the domain or shared ontology level and some leeway should be left for the application ontology builders to define at their level when building application ontologies for specific applications. The authors call this strategy a 'double articulation' which is a means of expressing knowledge in a two-fold axiomatization (Jarrar and Meersman, 2009).

The authors give an example of 'an ISBN of a book' to clarify their argument. The use of the ISBN varies across different application scenarios. A library, for example, may not give it as much importance as a bookstore does. For a bookstore, the ISBN might be an essential attribute of a book while for a library it's the authors' names that are important. This is because in some case, for example research theses and various reports, the ISBN is not available. Library applications, therefore, may not find the use of ISBN appropriate if it is fixed by the domain axiomatizations. But if its definition is relaxed at the domain level, the application axiomatization may specify its restricted meaning at the application level for its use by various applications.

The DOGMA framework provides a very useful insight into the way axioms have to be defined at different levels in an ontology-based knowledge sharing system. This is specifically relevant from the point of view of knowledge verification because a logic-based verification of shared knowledge comprises of a set of rules and axioms that need to be defined by keeping the DOGMA approach in sight.

4.4.2.5. LOM – A Lexicon-based Ontology Mapping tool

LOM (Li, 2004) is a semiautomatic ontology mapping tool which aids a human engineer to establish correspondences between two ontologies. It uses four different methods to match vocabularies from any two ontologies. The first two of these methods are purely heuristic-based or machine learning methods that involve the matching of lexical similarity through

different algorithms. The third method uses WordNet as an external source of knowledge to help identify synonyms in matching. The final method uses SUMO and a general purpose mid-level ontology called MILO (Niles and Terry, 2004) to find the ontological category of each word constituent for matching. Together SUMO and MILO contain more than six thousand concepts in the top and middle level and most popular synsets of WordNet are mapped with these concepts. It is these mappings which LOM exploits to find similarities across two ontologies. This exploitation involves first matching the concepts in the source ontology with similar terms in SUMO and MILO as suggested by WordNet and then repeating the same procedure for the terms in the target ontology. This process reveals the similarity across two ontologies to be matched because the terms that match the same concepts in SUMO and MILO are determined as similar (Li, 2004).

4.4.2.6. Unstructured vocabulary matching of Aleksovski et al

(Aleksovski et al, 2006) use an OWL DL medical terminology ontology called the DICE ontology as background knowledge to match two flat unstructured lists of concepts. The procedure is the same in which concepts in one set of vocabularies are first matched with similar concepts in the DICE ontology. This matching is called the anchoring match. In the same way, the vocabularies from the second list are anchor matched with similar concepts in the DICE ontology. Finally, the vocabularies anchored with the same terms in the DICE ontology are declared similar. Lexical matching of concept names and their synonyms is used for the process of anchor matching.

4.4.2.7. Automatic ontology matching of Mascardi et al

With an element of confidentiality in view, (Mascardi et al, 2008) propose an algorithm that uses upper ontologies to align two heterogeneous ontologies. This algorithm is based on two functions *Align* (O1, O2) and *Merge* (Al1, Al2) where O1 and O2 are the source and target ontologies respectively while Al1 and Al2 are respective alignments of concepts in O1 and O2 with terms in an upper ontology. The process of ontology matching consists of three steps. In the first two steps, the Align function tags similar concepts of source and target ontologies with concepts from an upper ontology and in the final step, these tagged alignments are merged to obtain similarities between the two ontologies. This process provides some confidentiality to virtual enterprises wanting to share information without exposing their ontologies. This is because these enterprises are only required to use the

upper ontology as reference without exposing their own ontologies. In their later work, they (Mascardi et al, 2010) experiment with OpenCyc, SUMO-OWL and DOLCE and use these foundation ontologies as semantic bridges to match ontologies. Both the concept names and structural composition are used in this work to find similarities in two ontologies.

4.4.2.8. Analysis of foundation ontology based ontology matching techniques

The work related to the use of foundation ontologies, reviewed here, shows that usually a vocabulary is developed relevant to a specific domain to provide a shared platform for building ontologies further down the level. This shared vocabulary is given the name of Core ontology. It can be seen that this approach prevents any ontological mismatches occuring because the domain or application ontology builder provides a mapping during the ontology building stage. This is done by attaching concepts in the newly constructed ontology to a similar concept in the core ontology subsumed under a foundation or upper ontology. This attaching of domain concepts, however, may cause inconsistencies as different people may interpret, and thus use, concepts and terms differently. This inconsistency can be prevented by using logic-based axioms at the core and domain levels. Care, however, needs to be taken when defining axioms, as a too tight definition of a concept at the core or foundation level may hamper its use at the domain level. This can be prevented by allowing, to some extent, the domain or application ontology builders to define the use of a concept themselves when building ontologies. The most important fact, from the point of view of this research, that can be derived from the review of foundation based techniques is that

'The selection of concepts from the core concepts ontology to build domain ontologies actually decides if the resulting knowledge is correct or not.'

For example, consider a case where a knowledge expression states a fact about the concept 'bank'. The word 'bank' may refer, either, to the bank of a river or a bank as a financial institution. It is to be made sure that the correct concept is selected from core concepts ontology when stating facts in the knowledge base. The emphasis of knowledge verification techniques using upper ontologies, therefore, has to be on defining ways to prevent an incorrect use of concepts from the core concept ontology. Jarrar amd Meersman (2008), in the above presented review, show that axiomatizations can be used for this purpose. For this reason, knowledge verification in systems with upper and core concepts ontology may

comprise of a mechanism that specifies, with the help of axiomatizations, a standard way of specializing concepts from a core concept or foundation ontology. The similarity finding technique then can be developed that uses the principles of that standard to discover correspondences between ontologies committed to the same core. In the tools, techniques and frameworks reviewed in this chapter, a description of such a mechanism cannot be found. Such a framework, therefore, is proposed in this research a detailed description of which is provided in chapter 8. This framework establishes similarities across two domain ontologies automatically by following concepts in domain or application ontologies to their origin in the core concept or foundation ontology. This framework is then tested and validated in chapter 9 by using real industrial examples.

The view of knowledge verification presented in this chapter, however, does not draw the complete picture of the rationale of knowledge verification from the perspective of this research. This view is complemented by facts presented in chapter 7, where through a case study, the industrial needs of knowledge verification are identified. To understand the rationale behind the proposed framework, the understanding of these industrial needs is also essential. The next chapter presents a brief review of the application of ontologies in manufacturing and an introduction to the techniques employed in this research for modelling ontological engineering product models.

4.5. Conclusions

It is understood that ontologies are specifications of conceptualisations and there can be differences in ontologies in the way these conceptualisations are specified which give rise to semantic ambiguities. These ambiguities get translated into incorrect knowledge understanding if knowledge is shared through ontologies. The challenge of knowledge verification is therefore the challenge of overcoming these ambiguities by finding similarities across two ontologies. It is further established that there can be a number of different ontological mismatches that may occur during ontology development and these mismatches cause problems in finding similarities between the two ontologies to be matched for seamless knowledge sharing. A correct discovery of similarities, once made after the overcoming these mismatches, rectifies the meaning of concepts across different ontologies and thus verifies the knowledge associated with those ontologies. It is also presented that

for finding similarities, two main types of techniques can be found in the literature. The first type, called the heuristic-based approaches, uses machine learning techniques to find similarities between two ontologies. The second type makes use of a shared upper ontology to reconcile differences across two ontologies. The review of these tools, techniques and frameworks draws the following conclusions:

1- Most heuristic based approaches for ontology matching target and resolve explication mismatches. Some work therefore needs to be done to develop techniques aiming at conceptualization mismatches.
2- All heuristic-based tools and techniques require human intervention at some point. A potential area of improvement, therefore, is the automation of these tools and techniques.
3- Foundation ontology based techniques resolve the issue of mismatches but bring into being the problem of inconsistencies when concepts from the core concept ontology are chosen to build domain or application ontologies and associated knowledge bases.
4- A plausible solution for these inconsistencies appears to be the use of logical theories or axiomatizations at different levels of the foundation and core concepts ontology.
5- The design and development of a mechanism capable of finding similarities between two ontologies by using their inheritance in the foundation or core ontology is needed.

These conclusions together with the conclusions of chapter 6 form a set of requirements for the proposed solution in this research.

Chapter 5: An introduction to Common Logic based ontology development formalism

5.1 Chapter overview

A brief introduction to Common Logic has already been given in chapter 3. This chapter now consolidates that introduction by describing how ontologies are formed in Knowledge Frame Language (KFL), a formalism based on one of the syntaxes of Common Logic. It also introduces the ontology editing environment used in this research, which is called Integrated Ontology Development Environment (IODE). This description is necessary in order to explain the experimentation in this research which has used this formalism and its editing environment IODE and therefore frequent examples have been given in the next chapters using this formalism and references to the editing environment. The inclusion of this chapter, therefore, makes the description of the research work clearer and more understandable. 'KFL reference' (Highfleet Inc, 2010) is the main source of information presented in this chapter.

5.2 Knowledge Frame Language (KFL)

The ISO standard 24707 defines a family of logic-based languages under the name of Common Logic (ISO/IEC 24707:2007(E), 2007). One of the syntaxes of Common Logic given in this standard is CLIF (Common Logic Interchange Format). KFL, which is the formalism used in this research, is actually a syntactic layer which sits on top of an extended syntax of CLIF called ECLIF (Extended Common Logic Interchange Format) ([Anonymous], 2010). Common Logic has been selected to build ontologies in this research, instead of OWL which is the most frequently used formalism for building and experimenting with ontologies. This selection has been made because of the high expressiveness of common logic which is important to capture the complex relationships which exist in some manufacturing contexts. As compared to other formalisms which are usually restricted to the creation of binary relations, CL provides the user with the capability to define ternary (three places), quaternary (four places) and even quinary (five places) relations ([Anonymous], 2010). In addition to that CL also provides a highly powerful syntax for logical expressions.

Ontolingua, an ontology development language, was briefly discussed in chapter 3. (Gruber, 1992) defines five essential components of an Ontolingua ontology. These are (1) Class definition, (2) Relation definitions, (3) Function definitions, (4) Instance definitions and (5) Axiom definitions. The pioneering work of (Visser et al, 1997) about ontological mismatches (see chapter 4) is also based on this definition of ontologies and therefore it is reasonable

that a KFL ontology is also analyzed on the basis of this 5-tuple. This is done by finding equivalences in a KFL ontology with the five components as shown below

Gruber's Class definitions	KFL Properties
Gruber's Relations definitions	KFL Relations
Gruber's Function definitions	KFL Functions
Gruber's Instance definitions	KFL Facts
Gruber's Axiom definitions	KFL Rules - A combination of KFL properties, relations and functions

These five components are defined next in more detail.

5.2.1 KFL properties

A KFL property is the most fundamental constituent of a KFL ontology. Its description, therefore is necessary because of its frequent use in the examples presented in the next chapters. One of the main aims achieved in this research by using ontologies is the modelling of an engineering component. As will be shown in chapter 6, an engineering component itself, its features, its dimensions, and its measuring units, are all represented through KFL properties. Furthermore, this constituent is an essential part of the rest of the four components of a KFL ontology.

The first thing to note here is that, apart from the rules, the directives in a KFL ontology start with a colon (:) as shown in the following examples. There are three essential directives to define a class in KFL as shown below.

```
: Prop      concept_1
: Inst      Type
: sup       Object
```

The 'Prop' directive stands for 'property' and is used to define a concept or a class. The 'Inst' directive stands for 'instance' and is used to associate the newly defined class with one of the categories predefined in the top level ontology contained by the ontology editing environment used in this research (a description of this environment, called IODE, is given in section 5.3 of this chapter). The 'sup' directive here defines the property 'concept_1' as a direct subsumption of the concept 'Object' predefined in the top level ontology. By using

the ‘sup’ directive, through the query tool in IODE, a specific concept can be traced back through its lineage to its origin. This is particularly useful when a concept in the domain ontology has to be traced back to its origin in the foundation ontology. This traceability attribute of an ontology is called the ‘handle’. So for the concepts in a KFL ontology the handle is its subsumption directive. For example, the feature concept ‘disc’ is defined as

```
: Prop     disc
: Inst     Type
: sup      features
```

Which says that there is a class named ‘disc’ which is an instance of ‘Type’ and its super-class is a class named ‘features’. The reason for emphasizing the existence of a handle here is that this attribute of a KFL ontology is used in this research to make concepts traceable during the process of knowledge verification. More on this can be read in chapter 8 where the process of knowledge verification using KFL ontologies is explained.

5.2.2 KFL relations

A KFL relation is that constituent of a KFL ontology which will be used to bind the shape features with the engineering component in its ontological model. It is also used, in the ontologies developed in this research, to relate a shape feature with its measuring dimensions. A clear understanding of this constituent of the KFL ontology is, therefore, crucial for the understanding of the relationships between different parts of a KFL ontology which are shape features, engineering component and their dimensions in this case.

Similar to the definition of classes, relations in a KFL ontology also require at least three directives. For example a relation named ‘relation_1’ is defined as follows:

```
: Rel      relation_1
: Inst     BinaryRel
: Sig      concept_1   concept_2
```

These three lines say that there exists a relation named ‘relation_1’ which is a binary relation i.e. existing between two concepts, and these two concepts are ‘concept_1’ and ‘concept_2’. The two related concepts are declared in the ‘Sig’ directive which stands for ‘signature’. This signature can be used as a handle to distinguish this relation from other

relations with similar names or to match it with an identical relation with a different name in other independently developed ontologies. A practical example in the field of manufacturing can be a relation which asserts that a component ‘disc’ has a ‘diameter’. This is done through a relation named ‘hasDiameter’ as shown below:

```
:Rel        hasDiameter
:Inst       BinaryRel
:Sig        disc  diameter
```

These assertions say that a binary relation ‘hasDiameter’ exists between the classes ‘disc’ and ‘diameter’. This relation is binary because it exists between two concepts. A ternary relation can also be defined with an additional concept. For example the measuring unit of diameter can also be defined within a single relation as follows:

```
:Rel        hasDiameter_mm
:Inst       TernaryRel
:Sig        disc  diameter  dia_mm
```

In these lines, a relation existing between three different concepts is defined within one set of assertions. The relation here says that the disc has a diameter attribute which is measured in millimeters. A KFL ontology can have relations between as many as five different concepts. This is an attribute of this ontology development formalism which makes it distinct from other similar logic based languages used to write ontologies. However, a more appropriate way of defining measurement units is through ‘KFL functions’ which is defined next.

5.2.3 KFL functions

KFL functions are mainly used in this research to define measurement units of numerical quantities such as geometrical dimensions of an engineering component. Its description and a clear understanding is, therefore, very important because the integrity constraints and rules defined in the domain ontologies take decisions depending upon the dimensions of shape features in an engineering component.

As is the case of the relation definition, functions in KFL are also defined using three essential directives as shown below:

```
: Fun      length_unit
: Inst     UnaryFun
: Sig      RealNumber -> length_quality
```

Like relations, functions can also be binary, ternary, quaternary and quinary. They can therefore bind up to five concepts together under one name. For example, the following function can be used to define a measuring unit along with its tolerances.

```
:Fun       angle_degrees
:Inst      TernaryFun
:Sig       RealNumber RealNumber RealNumber -> angle_quality
```

As in a relation definition, the signature in a function declaration can be used as a handle to differentiate a function from other functions of similar name or to find a similar function with a different name in other ontologies. These three lines say that there exists a function named 'angle_degrees' which is a Ternary function, the value contained by it is in the form of three individual real numbers and it belongs to the concept 'angle_quality'. A working example can be 90^{o} with tolerance of $+1^{o}$ and -0.5^{o}. Notice that the signature directive in a function declaration contains an arrow (->) unlike the relation declaration where there is simply a gap between the signatures.

5.2.4 KFL facts

KFL ontologies are populated with instances to develop knowledge bases. This is done by writing a knowledge statement and using it to form or populate the knowledge base. In the case of KFL ontologies, this knowledge statement is called a 'fact' and the processes of writing and introducing it to the knowledge bases is called 'fact assertion'. So facts are asserted in a KFL ontology in order to develop a knowledge base. Their understanding is extremely important because they are the building blocks of a knowledge base associated with an ontology. In this research, facts are used to form instances of engineering components in the knowledge base. The rules written in the manufacturing ontology, by using the properties, relations and functions, target these facts in order to verify that a certain piece of knowledge, like the dimensions of a shape feature, is correct and according to the allowances provided by the manufacturing engineer.

Facts, in a KFL ontology, are written in a syntax of common logic called simple common logic (SCL). An example of a simple fact is given below.

```
(disc   plate1)
```

This fact says that an instance of the class 'disc' exists with the name 'plate1'. Here the class 'disc' needs to be defined earlier in order for this fact to be meaningful. In a similar way, more detailed facts can be defined. For example a fact using relations and functions is declared as follows:

```
(hasDiameter   plate1   (diameter_mm   400))
```

Here the statement uses a relation 'hasDiameter' and a function 'diameter_mm' to declare a complete fact saying that the instance of the class disc named 'plate1' has a diameter of 400mm. Again, these relations and functions have to be defined earlier as explained in the previous sections. These facts can be controlled through rules existing in an ontology. The rule base of a KFL ontology can be used to decide which facts may be asserted and which may not, and this is very useful in the manufacturability analysis of an engineering component during its design. The manufacturability analysis through KFL rules is explained in detail in chapter 8. KFL rules are detailed further in the next section.

5.2.5 KFL rules

Rules can be written in a KFL ontology to assert an assumption in the form of an axiom or to control the declaration of classes or facts in the form of a constraint. They are like scrutinizing tools, used by the verification system proposed in this research, to verify the manufacturability of an engineering component as well as to ensure the consistency of the use of concepts from the foundation ontology to build the domain ontologies.

The following example can be used to understand the working of these rules, as this rule makes sure that a complete definition of an instance of a 'disc' is provided by the user.

```
(=> (disc ?x)
     (exists (?d ?h)
     (and (hasDiameter ?x ?d)
     (hasHeight ?x ?h))))
```

```
:IC hard "For a complete description of a ‘disc’ both diameter and
height are needed."
```

The use of the sign of interrogation (?), as seen in the above rule, is made whenever a variable has to be defined. In the above example therefore, ?x, ?d and ?h are all variables representing instances of a ‘disc’, its ‘diameter’, and its ‘height’ respectively. Constraints defined in KFL ontolgies can be hard or soft depending upon the requirement. This rule is a hard integrity constraint hence ‘IC hard’ in the last row and it says that if there exists a disc then its diameter and height should also both exist. A hard integrity constraint stops the user from proceeding with the faulty ontology until the fault is corrected while a soft integrity constraint just gives a warning to the user without preventing the ontology being loaded into the editing environment.

5.2.6 Other essential parts of a KFL ontology

The five components defined above form the essential skeleton of a KFL ontology. There are other parts of the complete KFL file which help in identifying the purpose and scope of the ontology built in this formalism. For example, a KFL ontology has to start with its Name, Description and Context it is using. These three attributes have to be defined before defining the classes and other building blocks. The start of a typical KFL ontology therefore looks like this:

```
: Name “Design ontology”
: Description “An ontology containing design concepts.”
: Use MLO
```

The first two lines are self explanatory but the third line needs some more explanation. The keyword ‘Use’ is used to inform the ontology editor about the context in which the ontology is defined. The context shown in the above example is MLO, the Meta Level Ontology context, which is a context predefined by the ontology editing environment that has been used in this research. Ontology builders can also define their contexts as shown below:

```
: Ctx      DSN
: Inst     UserContext
: supCtx   MLO
```

These assertions are similar to other assertions explained in the previous sections having three main directives (1) name, (2) instance type and (3) super-concept description. A user context has to be subsumed to a predefined context which is MLO in this case while DSN is the user defined context. This context can optionally be attached as a prefix of a class, relation, or function when defining rules or asserting facts. This feature of a KFL ontology is the most important of all from the perspective of this research. This is because it is the context which helps the ontology matching systems to distinguish between concepts of different ontologies. For example, the term 'mouse' in the context of computers is entirely different from a mouse in the context of biology. Likewise, a context prefix attached to an ontological term aids its identification and differentiation from other similar terms in different contexts and this is also true in KFL ontologies.

The description of a KFL ontology given in the sections above does not include some optional directives and keywords because of their irrelevance with respect to the work presented in this thesis. A consolidation of the examples given above provides an example of a simple but complete ontology as shown in table 5.1.

The ontology shown in table 5.1 (next page) provides a vocabulary to model a disc shaped component shown in figure 5.1. The instances of such a disc can be defined by asserting

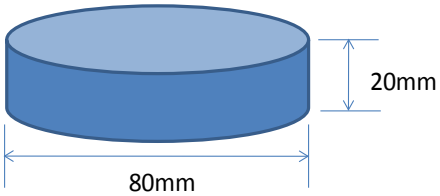


Figure 5.1 : A simple disc

facts that use the classes, relations and functions defined in the ontology. The following facts model a disc with a height of 20mm and a diameter of 80mm.

```
(disc   d1)
(hasDiameter     d1    (diameter_mm    80))
(hasHeight       d1    (height_mm    20))
```

It can clearly be seen that in order to assert these facts all the classes, relations and functions defined in the ontology earlier were essential. The rule present at the end of the ontology makes sure that both diameter and height of the disc are defined because the

Table 5.1: An example ontology

Code	Description
1 : Name “Design ontology” 2 : Description “An ontology containing design concepts.”	**Scope definition**
3 : Ctx DSN 4 : Inst UserContext 5 : supCtx MLO 6 7 : Use DSN	**Context definition**
9 10 : Prop features 11 : Inst Type 12 : sup Object 13 14 : Prop disc 15 : Inst Type 16 : sup features 17 18 : Prop measures 19 : Inst Type 20 : sup Object 21 22 : Prop diameter 23 : Inst Type 24 : sup measures 25 26 : Prop height 27 : Inst Type 28 : sup measures	**Class definitions**
29 30 : Rel hasDiameter 31 : Inst BinaryRel 32 : Sig disc diameter 33 34 : Rel hasHeight 35 : Inst BinaryRel 36 : Sig disc height	**Relation definitions**
37 38 : Fun diameter_mm 39 : Inst UnaryFun 40 : Sig RealNumber -> diameter 41 42 : Fun height_mm 43 : Inst UnaryFun 44 : Sig RealNumber -> height	**Function definitions**
45 46 (=> (disc ?x) 47 (exists (?d ?h) 48 (and (hasDiameter ?x ?d) 49 (hasHeight ?x ?h) 50 :IC hard "For a complete description of a ‘disc’ both diameter and height are needed."	**Integrity Constraint**

absence of any of these dimensions will leave the model of the disc incomplete. This integrity constraint prompts the user in case incomplete facts are asserted and thus maintains the integrity of the ontological model, hence it is called an integrity constraint. A more detailed way of modelling engineering components in the form of a KFL ontology is explained in chapter 6. The environment in which these ontologies are loaded and facts are populated is explained next.

5.3 Integrated Ontology Development Environment (IODE)

The ontology editing tool used for this work is the Integrated Ontology Development Environment (IODE™) from Highfleet systems. A snapshot of this tool can be seen in figure 5.2. IODE takes as input the ontologies written in the KFL syntax with prescribed file

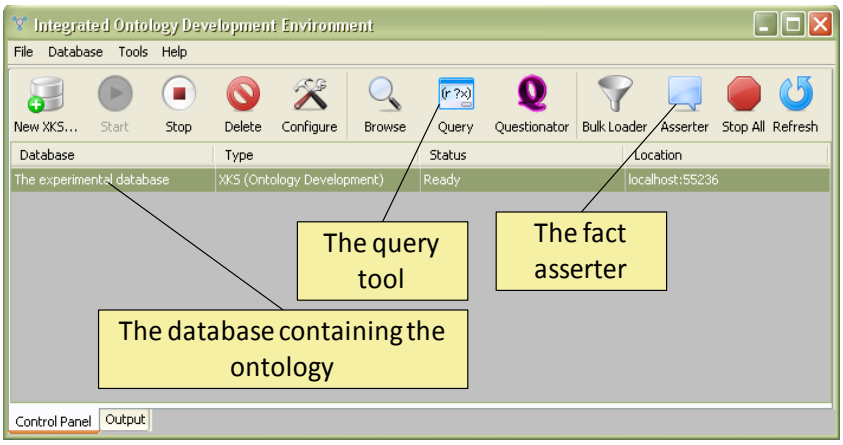


Figure 5.2 : The IODE environment

extensions. Once the ontologies are loaded the knowledge associated with them can be added, deleted and queried. From the point of view of this research, only the Fact Asserter and Query tool are important and are therefore explained further in the next sub sections. Other features of this environment which help the ontology developer in rectifying and improving the ontology before it is finally loaded into the system are not discussed further as they are not directly relevant to the research presented in this thesis. The details given here of the features of this tool are therefore based on the assumption that the ontology is already free of bugs and has been securely loaded into the system for further use.

5.3.1 The Fact Asserter

The fact asserter, as its name suggests, is used to assert facts to build the knowledge base once the ontology is loaded into the system. It takes input in the form of Simple Common Logic (SCL) assertions and optionally checks for integrity constraints. The software

application developed to demonstrate the working of the proposed framework (chapter 8) in this thesis uses the functionality of this fact asserter to build ontological models of engineering components. This software application is explained in detail in chapter 9. Figure 5.3 gives a snapshot of the fact assertion tool.

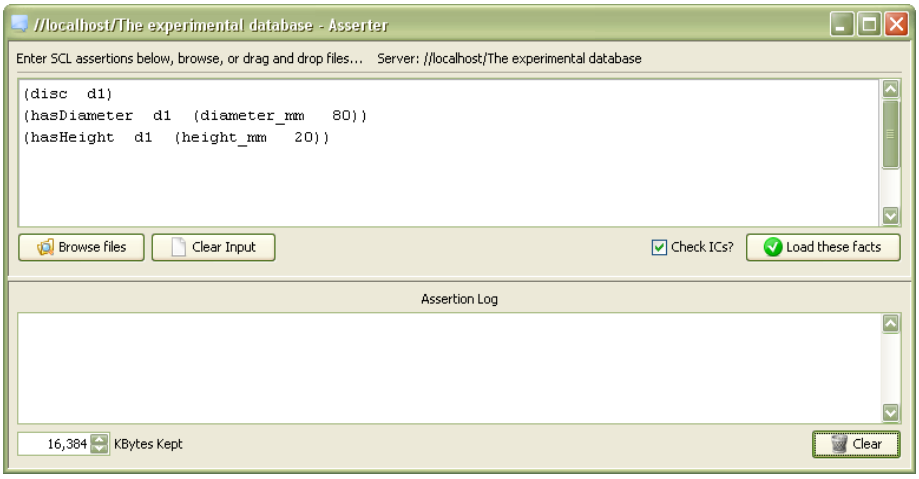


Figure 5.3 : The Fact Asserter

5.3.2 The Query tool

The query tool plays a very crucial role in the working of the proposed verification framework. It is this tool through which the system establishes similarities between concepts in the ontologies to be matched. This research features two domain ontologies one each for engineering design and engineering manufacture. The queries written by the verification system, enquire about the inheritance of a certain domain concept in the foundation and core concept ontology. Although this happens automatically because of the API developed for this task, this can be done manually as well by using the query tool as shown in figure 5.4.

The queries are to be written in the form of SCL assertions. For example, the following query is written if the existing instances of the class 'disc' are to be listed.

```
(disc ?x)
```

To find out the diameter of a disc 'd1', the following query can be written.

```
(hasDiameter    d1    (diameter_mm    ?y))
```

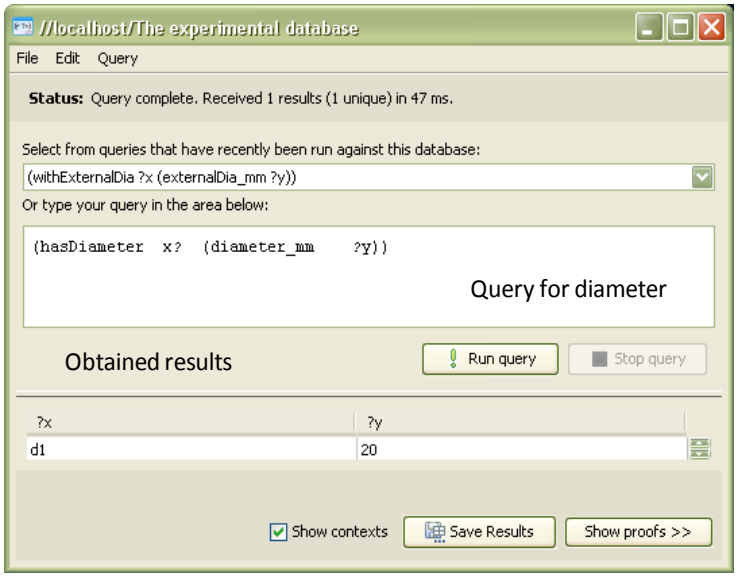


Figure 5.4 : The query tool

To find out the diameters of all the instances of class disc the following query is written.

```
(hasDiameter    ?x    (diameter_mm    ?y))
```

These queries are written on the same principles on which the facts are written. The only difference is that the entity in question is replaced with a question mark (?) prefixing an alphabet or word as can be seen in the examples above. These queries are written in the query tool and answers are obtained if any. Figure 5.4 shows a snapshot of this tool. The results obtained can be seen in the bottom part of the tool window. These results are used by the proposed knowledge verification system to make sense of the concepts in an ontology and then decide about the similarities between the ontologies to be matched.

5.4 Conclusions

This chapter gives a brief overview of the ontological formalism and ontology editing environment used in this research. The way ontologies are written in KFL and then populated and queried in IODE is also explained. This explanation is aimed at making the working of the proposed framework clearer and more understandable. A detailed use of this formalism and the introduced tools can be seen in chapter number 7 and 8. Some description of how ontologies are written in KFL can also be seen in Changoora and Young (2010).

Chapter 6: Ontology-based manufacturing knowledge sharing

6.1 Chapter overview

Chapter 5 presented an overview of the ontology development formalism used in this research. This formalism is used in this chapter to describe a methodology to develop ontological product models based on distinct shape features. The association of manufacturability knowledge to these models is also demonstrated using the same formalism. A brief literature review of the application of ontologies is presented first in section 6.2 and this is then followed by the modelling technique used in this research in section 6.3. To avoid any confusion, it is to be noted that the formation of these ontological product models is neither the development of a foundation nor a domain ontology. These models are actually formed in the knowledge base by using the concepts available in the domain ontology. The domain ontology, therefore, acts as a source of domain specific vocabulary which is used to create knowledge facts in the knowledge base. These knowledge facts, in this case, are the component models and the manufacturability knowledge associated to them.

6.2 Concurrent Engineering and Ontologies

An aim of concurrent engineering is to address the problem of insufficient manufacturability checks through detecting and considering conflicts and constraints at early design stages concurrently (Li and Shen, 2009) and this technique is accepted by all to improve productivity (Yoo and Suh, 1999). Ontologies can be used when this process of conflict detection has to be made automatic and more efficient. Defined as a formal and explicit specification of a conceptualization (Gruber, 1993b), ontologies are regarded as being useful to enhance interoperability. It has already been established in chapter 3 that ontologies not only provide a way to preserve knowledge but they can also provide pre-packaged sets of information and knowledge for individual use or for constructing large knowledge sets by using them as building blocks (Neches et al, 1991). It is these building blocks of information and knowledge which become the foundation of automating the process of concurrent engineering using ontologies.

Some examples of this approach can be found in the literature. For example, Yoo and Suh (1999) proposed a computerized concurrent engineering system consisting of three main parts; (1) an integrated product information model (IPIM) using the STEP standard (ISO/DIS

10303-224:2003(E), 2003), (2) a hierarchical database to store these models and (3) an integrity constraint validation mechanism based on EXPRESS (ISO/DIS 10303-224:2003(E), 2003), which is a formal information modelling language from STEP. They do not talk specifically about ontologies but their hierarchical database resembles such a concept. In another work Ma et al (2009) address product feature-level interoperability issues and develop a collaborative product development system. Although their emphasis is on the CAD side of feature level information sharing they do use a domain classification ontology to describe information dependencies across CAD applications. Very recently, Matsokis and Kiritsis (2010) converted a Semantic Object Model (SOM) into an ontology for better sharing and exchange of product lifecycle knowledge. SOM is a product item oriented model capable of storing data of the product's lifecycle. The reason they give for converting this model into an ontology is the reasoning capability of this technology along with its data structure layout. In another recent work, Dutra et al (2010) propose an ontology-based architecture for collaborative design. This synchronous agent-based architecture helps in conflict attenuation during the early stages of a collaborative design process. One thing which is common to all of these approaches is the use of ontologies as models which needs some more explanation and this is done in the next section.

6.2.1 Ontologies as Models

Modelling is an essential part of the intellectual activity of human beings (Silvert, 2001). A model is an approximation of reality (Studer et al, 1998). The challenge of artificial intelligence in information science is therefore the challenge of formalization of this approximation. The fundamental questions to be answered in this regard are about the existence of things. In the field of metaphysics, the systematic explanation of 'being' as an answer to this question is called an ontology (Gomez-Perez et al, 2004). In this sense ontology is a particular system of categories accounting for a certain vision of the world (Maedche, 2002) and a model of discourse participants (Nirenburg and Raskin, 2004). Most importantly, ontologies provide a common terminology that helps to capture key distinctions among concepts in different domains (Schlenoff et al, 2000). In the domain of manufacturing engineering, a common vocabulary is therefore needed to capture engineering components in a formalized state. One of the aims of IMKS is to provide that

vocabulary in the form of manufacturing core concepts. How this vocabulary can be used to build ontological models is shown in this chapter.

Some other examples of the use of ontologies for product modelling can be found in the literature. Horváth et al (1998), for example, to formalize design concepts by using an ontology paradigm. These design concepts include, along with shapes, design situations and functioning of design objects. In another work Staub-French et al (2002), formalize a vocabulary to define different types of design conditions and a feature ontology for construction and building products. In another relevant work Vegetti et al (2005) define PRoduct ONTOlogy (PRONTO) to define product concepts, their relationships and required axioms in the complex product modelling domain. In a further extension of this work, Gimenez et al (2008) introduce new concepts related to the specifications of mechanisms for aggregating and disaggregating different kinds of product-related data needed for Extended Supply Chain (ESC) logistics, as well as representing such data along a product abstraction hierarchy (Gimenez et al, 2008). Another effort in the product ontology domain is the work of Tursi et al (2007). They propose product ontologies using the IEC 62264 and STEP 10303 standards. Syntactic and semantic interoperability is then said to be achieved by finding correspondences between the terms in two ontologies. Bock et al (2010) at NIST have defined OPML an Ontological Product Modelling Language. This language claims to overcome the existing shortcomings of ontological product modelling techniques. It treats product models as ontological classifications and is capable of capturing partial and high-level product descriptions. Another effort at NIST has developed a Product Semantic Representation Language (PSRL). PSRL serves as an interlingua to enable semantic interoperability (Patil et al, 2005b). It mediates between two heterogeneous ontologies (Patil et al, 2005b; Patil et al, 2005a) to find correspondences between similar terms. Mappings are then established to connect terms in two ontologies. The important part of this work is the feature-based modelling example. In a more relevant work Catalano et al (2009) propose an ontology-based formalisation of the product design process. The relevance of this work here is again due to the fact that they also use shape of products to structure product design workflow. This use of shapes, however, is not as detailed and comprehensive as used in this research.

The research work reviewed above is in no way exhaustive and plenty of other efforts can be seen in this area. The main aim of this review was to establish that ontologies have been and are being used for product modelling. The results of these efforts, however, need some more refinement. The modelling approach proposed in this research and presented in this chapter, therefore, differs from most of these efforts primarily in its level of details in which shape features are defined and the product is modelled through their aggregation. Furthermore, the demonstrated use of ontological product models in manufacturing knowledge sharing is also what makes this work stand out from the rest. Before this modelling method is explained, however, it is necessary to understand if an approach based on shape features is plausible. This can be done by scanning the literature for shape feature based techniques. This is done in the following section.

6.2.2 Feature-based ontological modelling of engineering components

It is understood that ontologies can not only be used for the preservation of knowledge but they also help in producing pre-packaged sets of information and knowledge available to be used as building blocks (Neches et al, 1991). As found in the literature, these building blocks of information and knowledge become the underpinnings of feature based modelling of engineering components in the form of ontologies.

Product features are very useful to encapsulate engineering intent into computer systems (Ma et al, 2009). This is because once these product models are defined, related knowledge can be attached to them for use. For example, in the field of manufacturing, a model of an engineering component can have manufacturability knowledge attached to it to be used by the designer for taking design decisions. The methodologies and software tools to link this manufacturability knowledge with product models for intelligent design is currently one of the main collaborative design research problems (Li et al, 2002). Feature-based design and manufacturing offers an effective way of resolving this problem. Its benefits have been well recognized and CAD systems largely adopt it both in the mechanical and freeform domains (Catalano et al, 2009). This is endorsed by the existence of some commercial applications like FeatureCAM developed by Delcam plc. FeatureCAM is a suite of CAD/CAM software based on feature based manufacturing technology (Delcam plc, 2010). This software identifies shape features in a component and then generates the machining data. This is,

however, a manufacturing support software and not an interoperable application capable of allowing seamless sharing of knowledge, which is the main aim of the proposed methodology in this chapter.

It can be seen, from the above review, that shape feature based modelling of engineering components is considered to be a useful way of handling manufacturing knowledge and hence an ontological modelling technique based on shape features is plausible if it adds more value to the existing methods. The work done in this research, regarding the ontological modelling of engineering components, shares some commonality in one way or another with these existing ontology based collaborative design and concurrent engineering applications. The proposed system, however, is unique in its focus on overcoming the manufacturability knowledge interoperability problems caused by the differences in design and manufacturing perceptions of the features of an engineering component.

6.3 Feature-based modelling approach used in this research

Feature-based technology involves the construction of an engineering component by joining together distinct shape features. Figure 6.1 illustrates how individual features aggregate to form a complete component. It is obvious that for a feature based modelling method, individual features have to be defined first followed by their position in 3D space and the

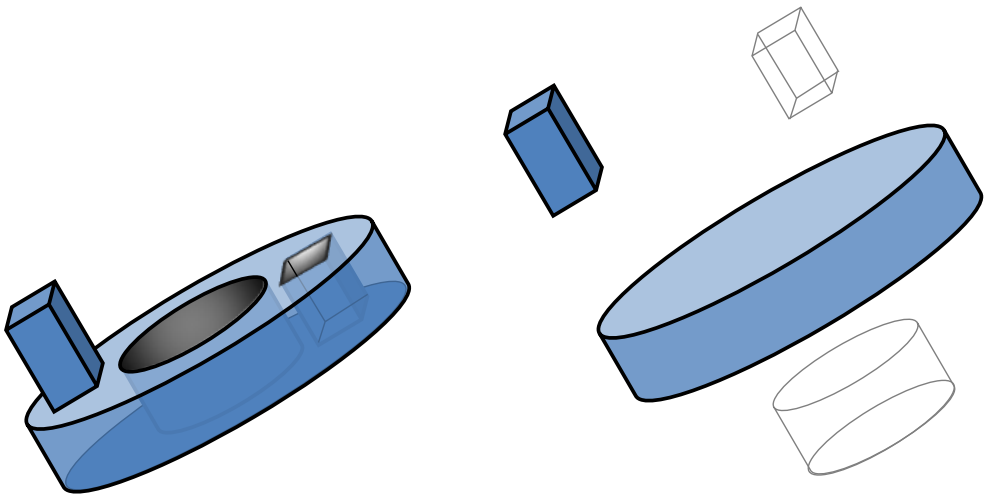


Figure 6.1. A component shown as an aggregation of shape features

definition of how they interact with each other. Figure 6.2 shows a geometrical model of a blob of material in 3D space. It can be seen that certain parameters must be specified for its definition and position. Geometrical dimensions like its height, width, and angles of the radii of curved surfaces are defined. Furthermore, its position in the 3D space is defined with the

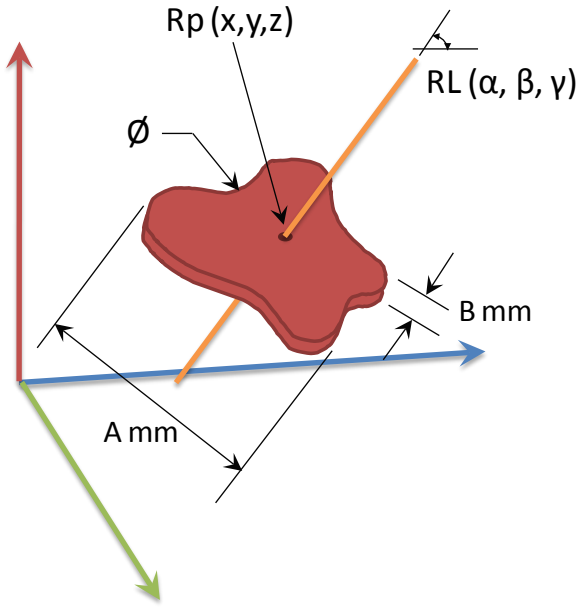


Figure 6.2. A hypothetical shape feature in 3D space

help of the coordinates of a reference point (RP) and the angles of a reference line (RL). The definitions of the reference point and reference line, however, are only required when a feature comes in relation to some other feature in the same 3D space. This happens when the feature aggregation takes place to form a complete component. Two main parts of this approach, in this regard, are therefore (1) Feature definition and (2) Feature aggregation. These two parts are discussed in detail below.

6.3.1 Feature definition

A complete definition of a shape feature primarily requires its dimensional parameters to be defined. This includes its height, width, diameter, and whatever else is deemed necessary for its complete definition. A straight hole, for example, can be defined by defining its height and diameter. Further tolerances can be added in order to aid its manufacture but height and diameter are the two essential dimensions that are needed for a complete definition of a hole.

6.3.2 Feature aggregation

Once individual features are defined, they can be put together to form a complete component. This putting together of shape features requires their positional parameters to be defined. Positional parameters refer to a shape feature's orientation in the 3D space. A straight hole, for example, can exist on a disc at any position making a certain angle with the surface of the disc. Unlike geometrical dimensions which may include a number of different parameters, positional parameters can be completely defined by defining just two key characteristics of a feature. These are its position and its orientation. The position can be defined by defining the x, y and z coordinates of the feature in the 3D space while the orientation requires the definition of the angles a shape feature makes with the three axes. Since a feature is a set of several lines and points, a certain point has to be selected within a feature to serve as a representation of the entire body. This is called here a 'reference point (RP)'. A definition of the x, y and z coordinates of this point places a feature at an exact point in the three dimensional space. The location of the reference point within the body of the feature, however, can be a debatable issue and needs to be standardized with mutual consent. On a similar basis, the definition of the angles a shape feature makes with the x, y and z axes first requires the definition of a line passing through the body of the shape feature. This line is named here as a 'reference line (RL)' as shown in figure 6.2. Again, the

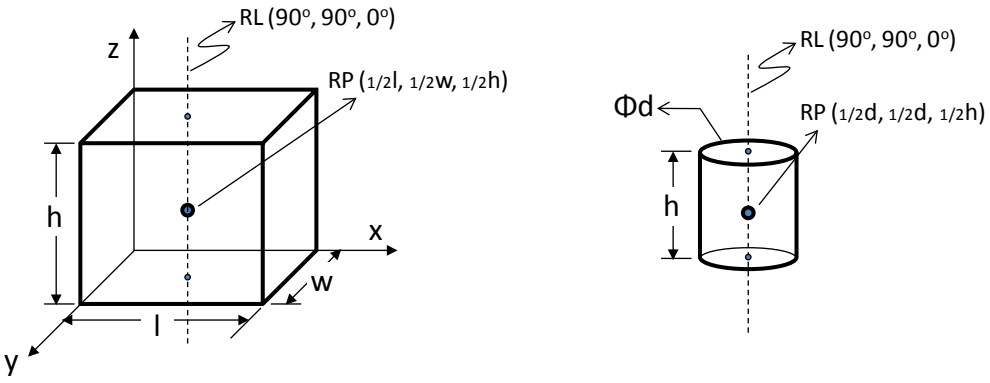


Figure 6.3. Reference Point and Reference Line standardization

angles this line makes with the body of the shape feature it belongs to needs to be standardized. In the approach presented, the position of a reference point is fixed to be at the geometrical centre of a shape feature while the reference line passes through this reference point and stays parallel with the side walls of a feature as shown in figure 6.3. In

this figure, two shape features are shown, a cube and a cylinder (or a hole if taken as a subtraction feature). It can be seen that the coordinates of the reference points are half of their dimension in each direction. Hence, for a cube the x-coordinate is half of its length (l), the y-coordinate is half of its width (w) and the z-coordinate is half of its height (h). Similarly, in the cylinder, the x and y-coordinates of the reference point are half of its diameter (d) while the z-coordinate is half of its height (h). With these coordinates, the reference points of both shapes come exactly in their geometrical centre. For the reference line, it can be seen that the angles of these lines orient them parallel to the side walls of both shapes or perpendicular to the x-y plane on which these features are placed. In an ontological model, these parameters can be fixed either with the help of axioms and constraints or simply as assumptions to be taken into consideration by the ontology and knowledge base builders. In the next section this modelling methodology is explained with the help of an example.

6.3.3 A working example

Consider a simple part comprised of a cube and a hole in the middle of one of its sides, as shown in figure 6.4. An ontological model of such a component should essentially include the dimensional and positional parameters necessary to completely define the individual

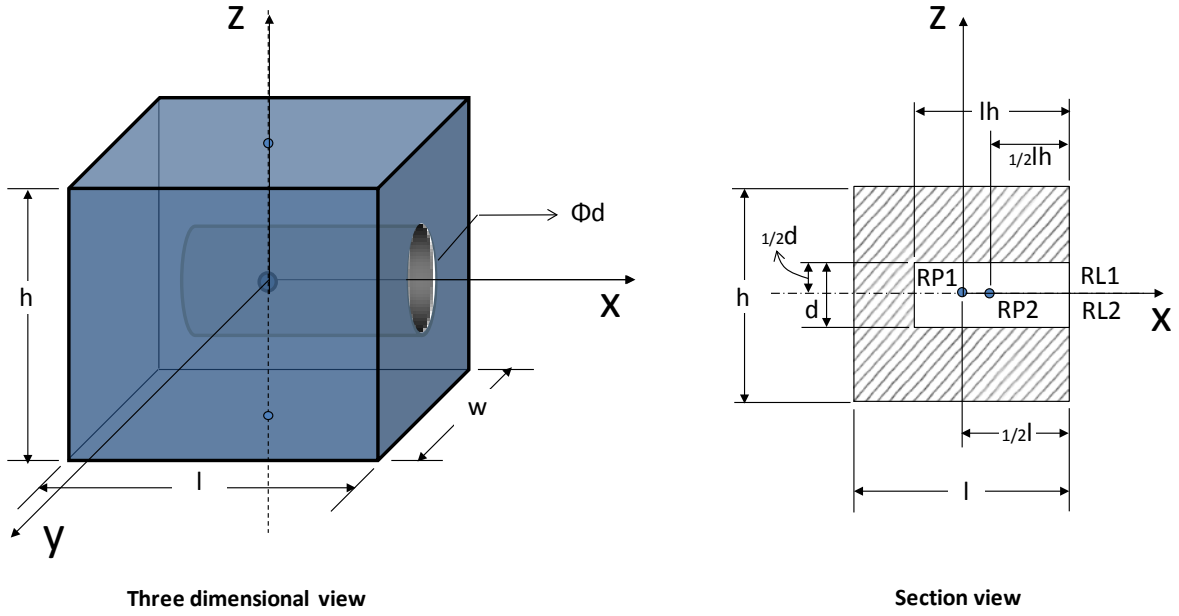


Figure 6.4. Aggregation of a cube and a hole

Table 6.1. Parameters for the modelling of the component in figure 6.4

S.No	Name	Value	Standardized Parameters
1	Block definition	cube1	A Block is a solid rectangular prismatic object with six flat square planes placed at an angle of 90^o with each other.
2	Block height	h	
3	Block length	l	
4	Block width	w	
5	Hole definition	hole1	A hole is a circular cross-section cavity in a solid object.
6	Hole height	h	
7	Hole diameter	d	
8	Block reference point	RP1	This point is in the geometrical centre of the cube.
9	RP1 x-coordinate	0	
10	RP1 y-coordinate	0	
11	RP1 z-coordinate	0	
12	Block reference line	RL1	This line passes through the reference point of the cube and is perpendicular to the top and bottom planes.
13	RL1 x-angle	90^o	
14	RL1 y-angle	90^o	
15	RL1 z-angle	0^o	
16	Datum point	RP1	All the measurements have to be taken from this point.
17	Hole reference point	RP2	This point is in the geometrical centre of the cavity.
18	RP2 x-coordinate	1/2l – 1/2lh	
19	RP2 y-coordinate	0	
20	RP2 z-coordinate	0	
21	Hole reference line	RL2	This line passes from the reference point of the cavity and is parallel to the sides surrounding it.
22	RL2 x-angle	90^o	
23	RL2 y-angle	90^o	
24	RL2 z-angle	0^o	

features as well as their assembly or aggregation. For the example in figure 6.4, table 6.1 shows the complete information to be modelled in the form of an ontology. The standardized parameters are to be fixed by the ontology builders while the parameters in the first two columns are to be defined by the user asserting knowledge about a certain part in the knowledge base. The inclusion of the non-standardized parameters, however, can be made compulsory through integrity constraints which force the knowledge asserter to define all the essential parameters of a feature for its complete definition. The key to understanding this approach correctly is the understanding of reference points and reference lines. Once this is learned properly, the placement of features to form a component is simple. The most important thing to take care of is the declaration of the datum point. In the example taken, the reference point of the solid block is taken as the datum point and thus it can be seen in the figure that the origin of all three axes is at the

solid block reference point i.e. RP1. This declaration has repercussions on all the later calculations and feature placements. In the figure above, it can be seen that although the reference points of the two features, i.e. the solid block (RP1) and the hole (RP2), are not coinciding but both of them lie on the same axis. Since the origin is at the solid block reference point, which is pre-assumed to be at the geometrical centre of the block, the coordinates of the two reference points come out to be RP1(0, 0, 0) and RP2($1/2$l – $1/2$lh, 0, 0) where 'l' is the length of the solid block and 'lh' is the length of the hole. Since both RP1 and RP2 are on the x-axis, the y and z-coordinates are zero. The declaration of the coordinates of the reference points on their own, however, is not enough because although the reference points of the two features lie on the same axis, there are still an infinite

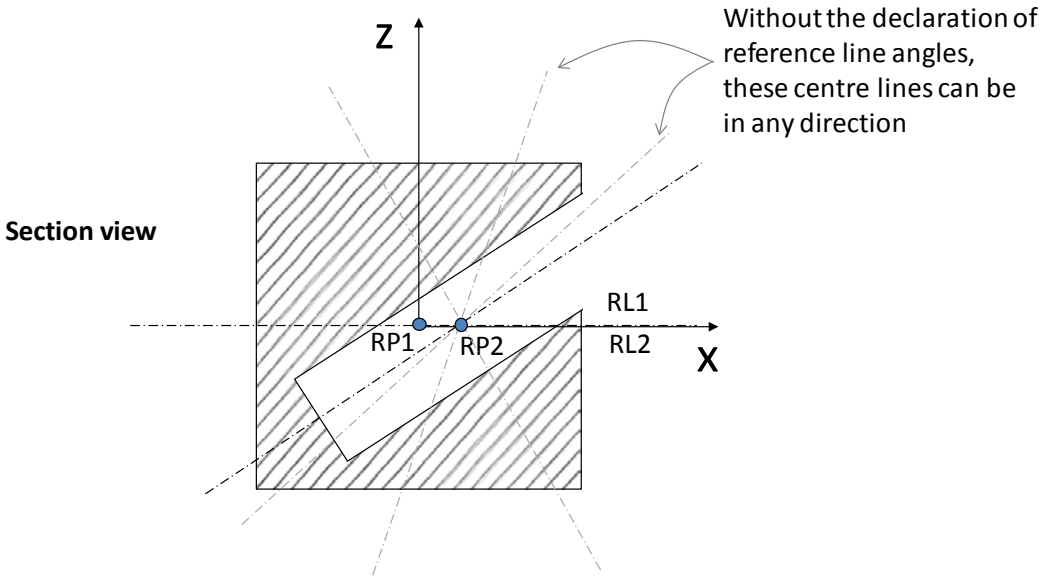


Figure 6.5. Possible directions of the hole feature without the declaration of reference line angles

number of orientations on which the two features may attach with each other as shown in figure 6.5. In this figure, the section view of the block with a hole in the middle, shown in figure 6.4, is shown with some possible directions of the hole feature without the declaration of the reference line angles. It can be seen that although RP1 and RP2 are still on the x-axis, the direction of the hole is changed. To prevent this confusion, the reference line of the solid block (RL1) and that of the hole (RL2) are declared to be parallel with each other and with the x-axis as well. Hence, the x, y and z angles being 90^o, 90^o, 0^o, respectively, are same for both RL1 and RL2. It is again to be understood that all these interpretations and resulting calculations are based on the assumptions that the reference points of the

features are at their geometrical centres with the reference line passing through that point and parallel to the sides, unless otherwise mentioned.

Once the information to be modelled is listed, the next step is to formalize it in the form of an ontology and the subsumed knowledge base. This is explained in the next section.

6.4 Ontological models

The last section presented an example to explain how an engineering component can be represented by a shape feature based model. In this section, with the help of hypothetical ontologies, the component shown in figure 6.1 is modelled in the form of an ontology. Knowledge Frame Language (KFL, as described in chapter 5) is the ontological formalism used to explain this ontological modelling.

6.4.1 The core-concepts ontology

The example ontologies developed to explain the ontological modelling approach are based on the setting proposed by the IMKS project which features a foundation and a core-

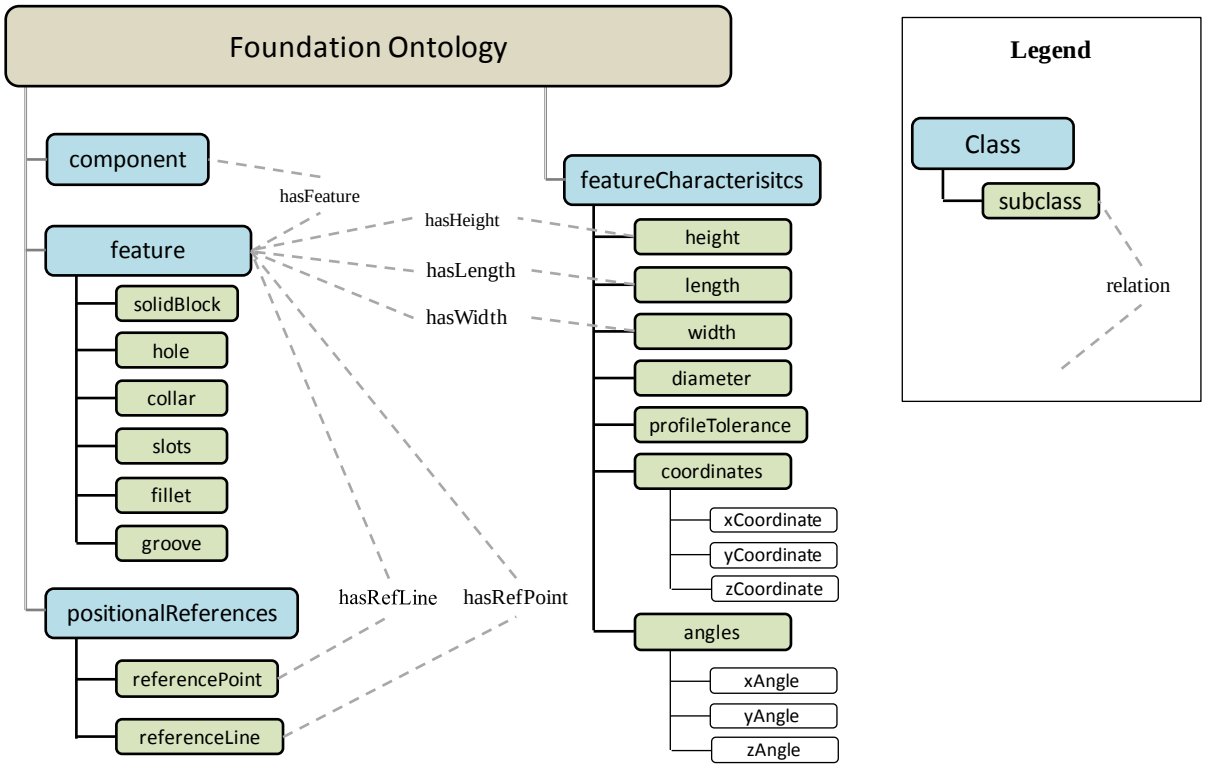


Figure 6.6. Foundation and core-concepts ontology to be used for the modelling of example component

concepts ontology subsumed by two domain ontologies one for the domain of engineering design and the other for engineering manufacture. This example, however, only illustrates the use of concepts from the core-concepts ontology to build shape feature based ontological models of engineering components. Figure 6.6 gives a view of a core-concepts ontology subsumed under a foundation ontology. It can be seen that the concepts presented in the ontology do not themselves form a model of an engineering component, rather, they provide a vocabulary to do so. The actual model of the component with all its dimensions and values is actually formed in the knowledge base by using these concepts. It is also to be noted that a mere list of concepts does not form an ontological model and some relations are needed between these concepts to bind them together and to give shape to a sensible structure. For example, a relation 'hasDiameter' may exist which connects a certain dimension e.g. diameter, to a feature e.g. hole. Some of these relations are shown in figure 6.6. Furthermore, these dimensions also need measuring units. This is done with the help of functions in KFL that attach a dimensional parameter to its measuring unit. The next section gives a formalised form of the ontologies depicted in figure 6.6.

6.4.2 Formalization of the ontology

The complete ontology from figure 6.6 is not formalized here and only those concepts are created which are needed to build the ontological model of the component shown in figure 6.4. This means, that it is to be assumed that the foundation ontology provides the very general concepts needed by the core concepts to form knowledge facts. These may include concepts like Top, AbstractEntity, ConcreteEntity, Particular etc. the formalization of an ontology in KFL includes the declaration of properties, relations, functions and axioms (as explained in chapter 5). Only the first three constituents are declared here starting with the creation of concepts i.e. classes (using KFL properties). Axioms are not written here to keep the focus on the modelling approach rather than the reasoning capability of the ontology which will be discussed in detail in chapter 8.

6.4.2.1 Definition of classes

```
component      :Prop    component
               :Inst    type
               :sup     Particular
```

feature and positionalReferences classes are also defined in this way.

featureCharacterisitcs	:Prop	featureCharacteristics
	:Inst	type
	:sup	AbstractEntity

solidBlock	:Prop	solidBlock
	:Inst	type
	:sup	feature

The hole class is also defined in a similar way subsuming it to the feature parent class.

referencePoint	:Prop	referencePoint
	:Inst	type
	:sup	positionalReferences

The referenceLine class is also defined in this way subsuming it to the parent class of positionalReferences.

height	:Prop	height
	:Inst	type
	:sup	featureCharacteristics

The length, width, diameter, coordinates, and angles classes are also defined in this way.

xCoordinate	:Prop	xCoordinate
	:Inst	type
	:sup	coordinates

The yCoordinate and zCoordinate classes are also defined in this way.

xAngle	:Prop	xAngle
	:Inst	type
	:sup	angles

6.4.2.2 Definition of relations

hasFeature	:Rel	hasFeature	
	:Inst	BinaryRel	
	:Sig	component	feature

hasHeight

```
:Rel      hasHeight
:Inst     BinaryRel
:Sig      feature     height
```

The binary relations `hasLength`, `hasWidth`, `hasDiameter`, `hasRefPoint` and `hasRefLine` have been defined in very similar ways to the relation `hasHeight` shown above.

hasXcoordinate

```
:Rel      hasXcoordinate
:Inst     BinaryRel
:Sig      referencePoint   xCoordinate
```

The binary relations `hasYcoordinate` and `hasZcoordinate` have been defined in very similar ways to the relation `hasXcoordinate` shown above.

hasXangle

```
:Rel      hasXangle
:Inst     BinaryRel
:Sig      referenceLine   xAngle
```

The binary relations `hasYangle` and `hasZangle` have been defined in very similar ways to the relation `hasXangle` shown above.

datumPoint

```
:Rel      datumPoint
:Inst     UnaryRel
:Sig      referencePoint
```

6.4.2.3 Definition of functions

height_mm

```
:Fun      height_mm
:Inst     UnaryFun
:Sig      RealNumber -> height
```

The unary functions `length_mm`, `width_mm`, `diameter_mm`, `xCoord_mm` etc have all been defined in very similar ways to `height_mm` shown above.

6.4.2.4 *Ontology formalization summary*

The formalized ontological segments shown in the last section only contain the essential entities for the formation of an ontological model of the example in figure 6.4. These formalized concepts have to be a part of the foundation ontology so that the properties, relations and functions defined here can be used by the domain ontology builders in their own way and then these domain concepts subsequently by the knowledge base builders to model engineering components and the associated manufacturability knowledge in the knowledge base. In one way, these ontological segments are actually a simplified form of the example foundation ontology developed and used in the validation of the proposed framework by using industrial case study examples.

In ontological terms, the formation of the ontological product models in the knowledge base is actually the process of ontology population. This population of the above presented ontology is done in the next section.

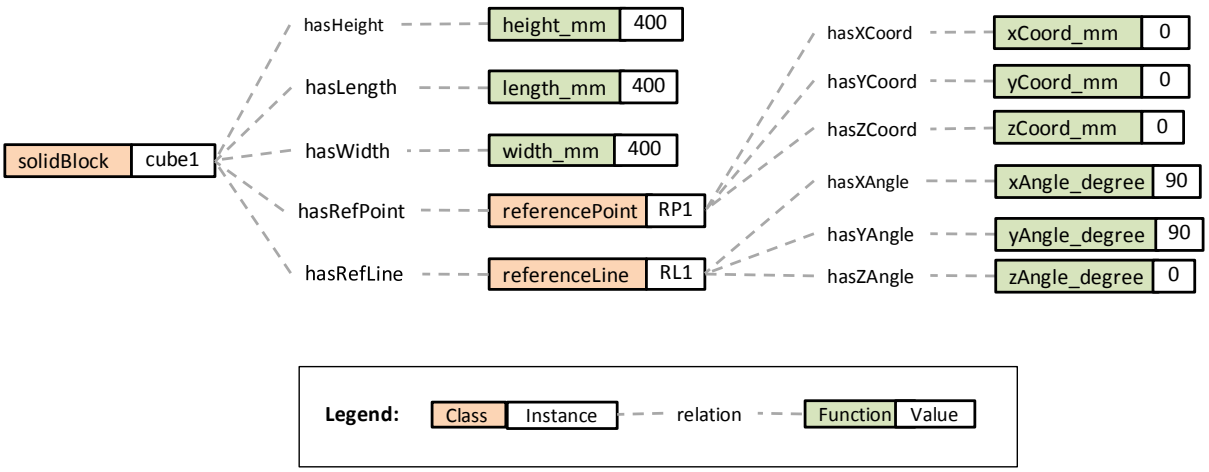


Figure 6.7. Instances of classes

6.4.3 Ontology population – knowledge base building

Figure 6.7 shows the essential instances to be created in order to completely define a feature, in this case a 'solidBlock'. It can be seen that to create an instance of the solidBlock feature the instances of its height, length, width, reference point, and reference line also need to be created by assigning them a numerical value. The functions defined in the

ontology are used here for this purpose. Similarly to create an instance of a reference point its x, y and z-coordinates are needed and for a reference line, its x, y and z angles all need to

Table 6.2. Instance of classes – the ontological model of the component in figure 6.4

Description	Line	Code
Defining an instance of solidBlock named cube1.	1	`(solidBlock cube1)`
Defining the dimensional parameters of cube1 i.e. height, length and width.	2	`(hasHeight cube 1 (height_mm 400))`
	3	`(hasLength cube 1 (length_mm 400))`
	4	`(hasWidth cube 1 (width_mm 400))`
Defining an instance of referencePoint	5	`(referencePoint RP1)`
Defining an instance of referenceLine	6	`(referenceLine RL1)`
Associating the reference point and line with cube1	7	`(hasRefPoint cube1 RP1)`
	8	`(hasRefLine cube1 RL1)`
Defining the x, y and z coordinates of the reference point. Since this point is now associated with cube1, defining its coordinates actually positions the cube1 in 3D space. In this case its the datum point as declared in line 12 so it lies on the origin i.e. 0,0,0.	9	`(hasXcoord RP1 (xCoord_mm 0))`
	10	`(hasYcoord RP1 (yCoord_mm 0))`
	11	`(hasZcoord RP1 (zCoord_mm 0))`
	12	`(datumPoint RP1)`
Defining the angles of the reference line of cube1 and hence orienting it correctly in 3D space.	13	`(hasXangle RL1 (xAngle_degree 90))`
	14	`(hasYangle RL1 (yAngle_degree 90))`
	15	`(hasZangle RL1 (zAngle_degree 0))`
Defining an instance of hole named hole1	16	`(hole hole1)`
Defining the compulsory dimensional characterisitcs of hole1. This includes the length and the diameter.	17	`(hasLength hole1 (length_mm 300))`
	18	`(hasDiameter hole1 (diameter_mm 100))`
Instantiating a reference point.	19	`(referencePoint RP2)`
Instantiating a reference line.	20	`(referenceLine RL2)`
Associating the instantiated reference point and line with hole1.	21	`(hasRefPoint hole1 RP2)`
	22	`(hasRefLine hole1 RL2)`
Positioning hole1 in the 3D space by giving coordinates to its reference point.	23	`(hasXcoord RP2 (xCoordinate_mm 0))`
	24	`(hasYcoord RP2 (yCoordinate_mm 15))`
	25	`(hasZcoord RP2 (zCoordinate_mm 0))`
These directives orient hole1 reference line in the 3D space. Being 90,90,0 and the reference point on the same axis, the hole reference line overlaps with that of cube1.	26	`(hasXangle RL2 (xAngle_degree 90))`
	27	`(hasYangle RL2 (yAngle_degree 90))`
	28	`(hasZangle RL2 (zAngle_degree 0))`

be instantiated as well, again with the help of their respective functions. These instances are then associated with the solidBlock instance with the help of respective relations. In chapter 5, it was shown how instances of classes are created. The same syntax of simple common logic is used here to create instances of classes defined in the ontology in section 6.4.2.3. Table 6.2 shows all the instances needed to create an ontological model of the component shown in figure 6.4. The table also gives a line-by-line description of the code that needs to be written to instantiate classes created in the ontology. These lines present the formalization of the information listed in table 6.1. In the same way different components and assemblies can be modelled in an ontology supported knowledge base. Once these models are built they can be used to state conditions of manufacturability, which in the case of an interoperable system, can be used by the designer to examine the design of a product from the manufacturability point of view. Details of how this can be achieved are explained in the next section.

6.5 Manufacturability verification

Manufacturability verification of a feature, or a set of features in the form of a component, can be done by writing ontological rules or integrity constraints. The process of model creation is the process of the assertion of knowledge facts about the dimensional and positional parameters of a component. Every fact asserted in the knowledge base is sieved through these rules and its validity is checked according to the conditions stated in the rule. These conditions are checked by the ontology browser IODE which has been introduced in chapter 5. When ontologies are loaded along with a set of knowledge facts, IODE checks for the integrity of ontological assertions and knowledge statements and gives an error if they do not comply with either the inbuilt or user written constraints. An example of this can be seen in figure 6.8 (next page) where a window shows two sections. In the section in the top part of the window some knowledge facts can be seen which instantiate a hole named h1 and assign the value of 1mm to its diameter. In the bottom part of the window an error message can be seen which says that there is a hard IC violation and a hole with a diameter less than 1mm cannot be drilled by using the available tooling. The window shown in figure 6.8 is the fact asserter tool as introduced in chapter 5, which aids in asserting knowledge facts, or creating ontological models of features and components in our case, in an ontology that is already loaded into the database. This process of error generation is actually a

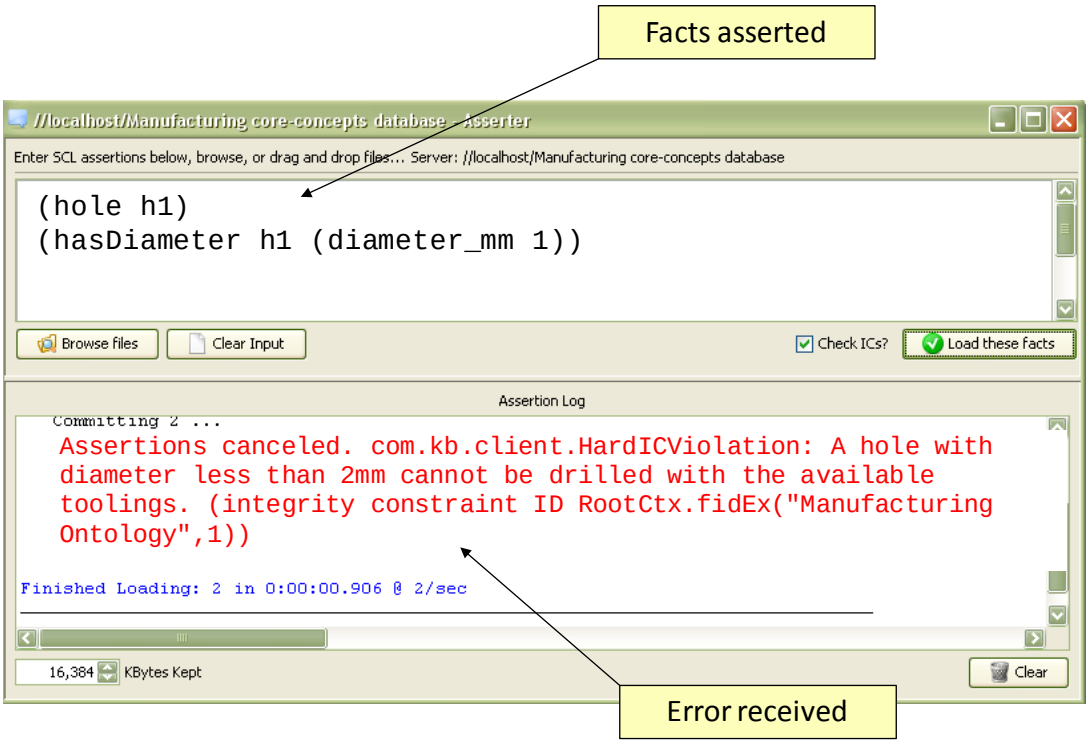


Figure 6.8. Integrity constraint violation pops up an error and knowledge facts are not asserted

process of manufacturability verification. If the facts are asserted successfully, the feature or component modelled is manufacturable. Otherwise the design needs to be changed according to the limitations of the available manufacturing facility.

Two types of manufacturability limitations may occur. In the first case an individual feature, independent of other features in the component, may become impossible to manufacture within a facility due to its dimensions. In the second case a component may become non-manufacturable due to the way features in that component interact with each other or certain dimensions of one feature hinder the creation of one or more of the other features. These two cases are explained here separately.

6.5.1 Individual feature manufacturability constraint

This is the simplest form of manufacturability constraint which may exist due to the unavailability of a manufacturing method or tool within the facility in which a component has to be manufactured. Consider a very simple example of a hole. There are certain sizes of holes that can be produced by a machine or tooling used at that machine. If there is a lower limitation of the diameter of a hole that can be produced in a manufacturing facility, a

Table 6.3. The manufacturability rule for a hole

Description	Line	Code
Starting the rule with the AND condition. The symbol => denotes IF.	1	(=> (and
The first condition is that their exists a hole.	2	(hole ?h)
The second condition is that this hole has a diameter.	3	(hasDiameter ?h (diameter_mm ?d)))
At this point the THEN part of the condition starts whcih says that the diamter has to be greater than 2mm.	4	(gteNum ?d 2))
'IC hard' means that this is a hard integrity constraint which means the user cannot proceed unless the error is removed. The statement in the quotations appears in the error in the asserter window as shown in figure 6.8	5	:IC hard "A hole with diameter less than 2mm cannot be drilled with the available toolings."

manufacturability constraint shown in table 6.3 needs to be written. The table also shows a line-by-line code description. It can be seen that this is a simple IF-Then rule where the IF part of the constraint (lines 1 to 3) states the conditions about the existence of a hole and its diameter and the THEN part (line 4) declares the conditions that need to be fulfilled in order for the diameter to be correct. In the end, a statement in quotations holds the message to be given to the user in case a violation occurs. IC hard before the quotations means this constraint stops the user from proceeding further with the modelling process unless the dimension of the feature is changed. Another possible constraint is 'IC soft' which only gives a caution to the user about the violation and does not end the process. The user in that case can continue with the model creation as the violation may not be very significant. In the example given here, the minimum size of a hole that can be drilled is 2mm and sizes smaller than that are declared to be non-manufacturable. In a similar way different constraints for the size or position of a certain feature can be written. The positional constraints, however, are needed more when two or more features are used to state a manufacturability condition. This is explained next.

6.5.2 Manufacturing constraints due to feature dependability

Since, the manufacturability constraint for an individual feature first needs the declaration of the existence of that feature along with its dimensional parameter (lines 2 and 3 in table 6.3), a rule about more than one feature requires this condition for all of the features that take part in fulfilling the asserted condition. Furthermore, in the case of one feature, only

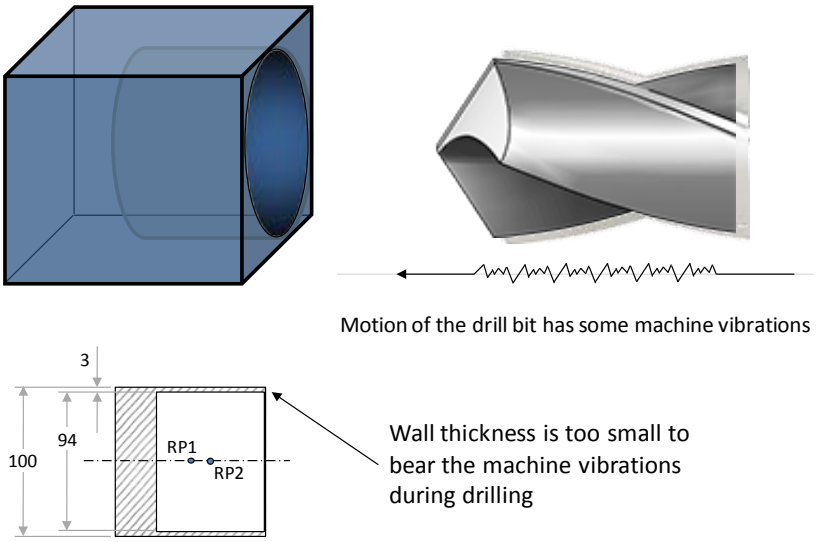


Figure 6.9. An impossible to drill hole in a cube due to machine vibrations

dimensional parameters need to be declared. In the presence of multiple features, the positional parameters also need to be included in the condition. In fact, it is the positional parameters which play the most vital role in the verification of manufacturability. Consider an example where a hole has to be drilled in a solid block of metal. In such a situation, there is a limit to the wall thickness that is left behind after the drilling of the hole as shown in figure 6.9. This is due to the inherent machine vibrations which require a certain allowance between the hole edge and the outer side of the cube for them to be absorbed without distorting the shape of the cube. In this scenario, two features, i.e. a solid block and a hole, are dependent upon each other and an integrity constraint requires the declaration of dimensional and positional parameters that model the component exactly as depicted in figure 6.9. This integrity constraint is shown in table 6.4 (next page). Lines 3 to 16 model a solid block by defining its dimensional and positional parameters in the form of variables. Lines 18 to 30 create the model of a hole through the definition of its parameters. At this point the IF part of the rule ends and the THEN part starts. Three variables for the calculations of the difference between the length, height and width of the block and the diameter of the hole are first introduced. This is done in lines 39, 40 and 41 of table 6.4 respectively. These variables represent the wall thickness of the cube left after the drilling of the hole. Lines 42 to 44, then, state the minimum possible thickness that is manufacturable

Table 6.4. Integrity constraint for the manufacturability of the component shown in figure 6.9.

Description	Line	Code
Starting the IF part of the rule	1	(=> (and
	2	
Defining a variable ?c being a solidBlock	3	(solidBlock ?c)
Defining the dimensional parameters of ?c i.e. height, length and width.	4	(hasHeight ?c (height_mm ?ch))
	5	(hasLength ?c (length_mm ?cl))
	6	(hasWidth ?c (width_mm ?cw))
Defining an instance of referencePoint	7	(referencePoint ?rp1)
Defining an instance of referenceLine	8	(referenceLine ?rl1)
Associating the reference point and line with ?c	9	(hasRefPoint ?c ?rp1)
	10	(hasRefLine ?c ?rl1)
Since rp1 is now associated with ?c, defining its coordinates actually positions ?c in 3D space.	11	(hasXcoord ?rp1 (xCoord_mm ?rp1x))
	12	(hasYcoord ?rp1 (yCoord_mm ?rp1y))
	13	(hasZcoord ?rp1 (zCoord_mm ?rp1z))
Defining variables for the angles of the reference line of ?c	14	(hasXangle ?rl1 (xAngle_degree ?rl1a))
	15	(hasYangle ?rl1 (yAngle_degree ?rl1b))
	16	(hasZangle ?rl1 (zAngle_degree ?rl1c))
	17	
Defining a variable instance of hole ?h	18	(hole ?h)
?hl and ?hd being the variables for the length and diameter of ?h	19	(hasLength ?h (length_mm ?hl))
	20	(hasDiameter ?h (diameter_mm ?hd))
Defining a reference poing variable	21	(referencePoint ?rp2)
Defining a reference line variable	22	(referenceLine ?rl2)
Associating the reference point and line variables with ?h	23	(hasRefPoint ?h ?rp2)
	24	(hasRefLine ?h ?rl2)
Positioning ?h in the 3D space by giving coordinates to its reference point variable	25	(hasXcoord ?rp2(xCoordinate_mm ?rp2x))
	26	(hasYcoord ?rp2(yCoordinate_mm ?rp2y))
	27	(hasZcoord ?rp2(zCoordinate_mm ?rp2z))
Defining variables for the angles of the reference line of ?h	28	(hasXangle ?rl2 (xAngle_degree ?rl2a))
	29	(hasYangle ?rl2 (yAngle_degree ?rl2b))
	30	(hasZangle ?rl2 (zAngle_degree ?rl2c))
	31	
For the two features to be in the position as illustrated in figure 6.9, these are the equivalences that need to exist. These assertions declare the two reference points to coincide and the lines to overlap	32	(eqNum ?rl1a ?rl2a)
	33	(eqNum ?rl1b ?rl2b)
	34	(eqNum ?rl1c ?rl2c)
	35	(eqNum ?rp1y ?rp2y)
	36	(eqNum ?rp1z ?rp2z)
Closing the IF part of the rule and entering into the THEN part	37	)
	38	
These conditions say that the differentce between the diameter of the hole and any of the sides of a solid block should be gretater than or equal to 5mm	39	(numMinus ?cl ?hl ?result1)
	40	(numMinus ?ch ?hd ?result2)
	41	(numMinus ?cw ?hd ?result3)
	42	(and (gteNum ?result1 5)
	43	(gteNum ?result2 5)
	44	(gteNum ?result3 5))
Closing the THEN part and the rule as well	45	)
Stating the type of constraint and the error message to be displayed in case of violation of the rule	46	:IC hard “The machine vibrations do not allow the leftover thickness of the walls to be less than 5mm during the process of drilling. A smaller thickness will result in distortion.”

without the part getting distorted due to machine vibrations. The last part of the rule declares this rule to be a hard integrity constraint and states the error message.

One thing to be noted here is that the rule models the hole and cube aggregations in the same way as was done in table 6.2 apart from one thing. There is no declaration that the reference point of the cube is the datum point. This is because the rule mainly deals with the difference in dimensions of the two features which does not require knowledge of the datum point.

6.6 Conclusions

This chapter described an ontological modelling technique for engineering components and demonstrated how manufacturability verification can be done through the use of integrity constraints. It is clear from the examples explained above that an engineering component can easily be translated into the form of an ontological product model and then manufacturability constraints can be associated with this model. Since the research presented in this thesis is related to knowledge interoperability, this modelling method has to be analyzed accordingly. It is therefore important to find what differences may occur when models in two independent ontologies are built. This is because these differences hinder the interoperability of these models and thus render the mechanism of manufacturability verification ineffective. A review of the literature on ontological mismatches was presented in chapter 4. This review, however, does not present the full picture of the problems that occur in practical situations in manufacturing setups. To gain an understanding of these practical industrial problems a case study in a manufacturing setup was performed. The findings of this case study are presented in the next chapter.

Chapter 7: The case study

7.1. Chapter overview

The findings of a case study are presented in this chapter to help understand those requirements of the undertaken research that are specific to the problem of knowledge interoperability in the manufacturing industry. These requirements are based on the ways in which concurrent engineering is applied in design context in a large aerospace company. This chapter, therefore, first of all highlights the knowledge sharing needs of a typical manufacturing company and secondly, it provides the practical industrial evidence of the importance of the research problem at hand.

7.2. Purpose and scope of the case study

An understanding of the real industrial interoperability problems is essential for an effective design of the proposed interoperable manufacturing knowledge system. In order to achieve that, a case study was conducted in the compressor disc manufacturing plant of an aerospace manufacturer. This study had two main dimensions. The first dimension dealt with the study of communication needs between design and production departments in order to produce a manufacturable design. As this research is all about computer-based knowledge sharing, the second dimension entailed the study of the criteria for modelling an engineering component in the form of an ontology to which knowledge can be attached for seamless sharing. The literature review in the area of shape feature based design and manufacture of engineering components (see chapter 6) has already established the importance and usefulness of ontology techniques. The manufacturing setup of the aerospace manufacturer, where the case study was conducted, was also aiming to adopt this engineering methodology and therefore, shape feature based design and manufacture of engineering products was chosen as the methodology to produce computer-based models.

This case study, therefore, studied the manufacturer's existing efforts to standardize some of the shape features in a compressor disc, and also carried out a fresh analysis of the geometry, function and manufacturing methods of these discs to examine the existing shape features and, if needed, introduce some new ones. The case study entailed studies of detailed drawings, assessment of the intended functions of different areas of a compressor disc, and the study of the production routes that the disc goes through starting from a

forged stock and ending in a polished packed form ready to be delivered. The primary sources of data were individual one-to-one interviews but the company intranet and archival information also contributed to this 12 week case study. The following points were considered during interviews and data analysis:

a. What are the real world design-associated manufacturability problems?

b. What are the design and manufacturing perceptions of product features?

c. In a typical manufacturing setup, what does a designer want a production engineer to know and what does a production engineer want the designer to know?

d. What is a typical design modification process?

Once the data and information about the above points was gathered, a tool to analyze the existing information flows between design and manufacture and to propose improvements was required. IDEF-0 diagrams (Draft Federal Information, 1993) were therefore used for this purpose. These diagrams could not be included in the thesis due to a non-disclosure agreement between the researchers and the case study company. The results of the analysis of this case study, however, are discussed to argue the nature of knowledge and information flow between design and manufacture. The understanding achieved by the analysis of the case study materials and the methods used for the analysis are the most important aspects of the case study work from the viewpoint of this thesis.

7.3. Case study findings

The findings detailed here are divided into two sections. The first section presents the analysis of the information and knowledge flows between design and manufacture while the second section deals with the component which was studied in order to propose a feature-based knowledge sharing mechanism.

7.3.1. Information flow study

It was understood that the best approach to study the information flow between design and manufacture was to study the documents that were used by the two departments to communicate with each other. These documents included part drawings, manufacturing

instructions, design change requests, quality improvement initiatives etc. Some important documents analyzed during the case study are given in table 7.1 along with their description and purpose. Since, the case study used the shape feature based design and manufacturing approach, these documents were specially examined for their suitability to carry the same information when a component is considered as an accumulation of different shape features. The document names used here are changed from the original titles for the purpose of maintaining confidentiality.

Table 7.1. Some important documents analyzed during the case study

Terms	Meaning	Description
PDD	Preliminary Design Document	Generated by the design department as an answer to the initial design requirements from an organizational unit. This is sent back to the organizational unit which requested the design initially. It contains the agreement or disagreement about the design requested.
GCR	Geometry Change Request	Sent to the engineering department usually by manufacturing for requesting a change in a certain dimension of a component.
DND	Design Negotiation Document	A document which is generated by manufacturing to negotiate with engineering design on a certain dimension of a component. If the manufacturing department finds a certain part of the component difficult to make, it suggests changes to the engineering department to make the manufacturing process easier and more accurate. The engineering department after analyzing the suggestion and judging its implications on the performance of the component allows or disallows the change.
PQII	Process Quality Improvement Initiative	A document initiated to request a change in the design of a certain part of the component in order to improve its quality. This improvement in quality is aimed at achieving better production results.
CPMI	Critical Parts Manufacturing Instructions	A document generated by engineering to describe the way critical parts need to be manufactured.
IMF	Important Monitored Features	These are the features which are currently monitored by the manufacturing department. Their process capability charts are drawn and the results are used in the design of future similar features.

Most of the documents listed in table 7.1 were used to communicate a certain design requirement or some information about the component. The most important documents from the point of view of this research were PDD, GCR, DND, PQII, CPMI, and IMF because they exchange knowledge that can easily be replaced with a vocabulary based on standard shape features. The details of these documents can be seen in table 7.1. Since the intent of this part of the case study was to establish an understanding of the knowledge sharing

requirements, these documents were analyzed from the point of view of knowledge exchange. The careful analysis of these documents and the way they were being used revealed that the information and knowledge they contain and were used to exchange can certainly be made more meaningful if standard shape features are used. This is because during the case study it was established that different perceptions of the same part of the component exist in different parts of the same company. Since these differences have occurred over time, it is reasonable to assume that they are dynamic in nature i.e. they will keep changing with time if a benchmark is not defined. Details of these differences are given in the next section where the component geometry is used for this purpose. It is, however, very important at this point to keep in mind that when a computer-based knowledge sharing system is used to exchange knowledge between independently developed knowledge bases, these differences may get worse rendering the computer systems unable to establish similarity during the process of knowledge sharing and thus affecting interoperability. These interoperability problems indicate that there is a need for a dynamic system where design and manufacture people are provided with a standard vocabulary to independently develop their own knowledge bases using their own perceptions. Since the changes in these knowledge bases are made using the same common vocabulary, a knowledge verification system can use this vocabulary to translate the meanings accurately across departments. In the case of this research, this vocabulary is provided in the form of a foundation and core-concepts ontology.

One last thing to note here is that during this part of the case study, it was found that the documents like the DND and GCR were designed to involve manufacturing people in the design of a new product or the modification of an existing one. A GCR, for example, is raised and sent to the designer when the manufacturer has some problems with the production of a component. Similarly, the DND requires the approval of the manufacturer before the design of a component is finalized. If analyzed closely, it can be seen that these documents, in one way or another, deal with the manufacturability of a component. These documents, therefore, will not be needed in the presence of an automatic manufacturability analysis mechanism. The focus of this research in presenting potential solutions to the studied industrial problems is thus on computer-based manufacturability knowledge sharing. The

next section explains the findings of the next part of the case study where a component was studied with special emphasis on manufacturability problems.

7.3.2. The component study

The study focused on a high pressure compressor disc as illustrated in Figure 7.1. Different parts of this disc are important in different lifecycle stages. These stages include manufacturing, assembly, maintenance, use, and disposal. This case study, however, only dealt with the analysis of information and activities of design and manufacturing, and more precisely, the process of machining for this particular disc. For proprietary reasons, the exact

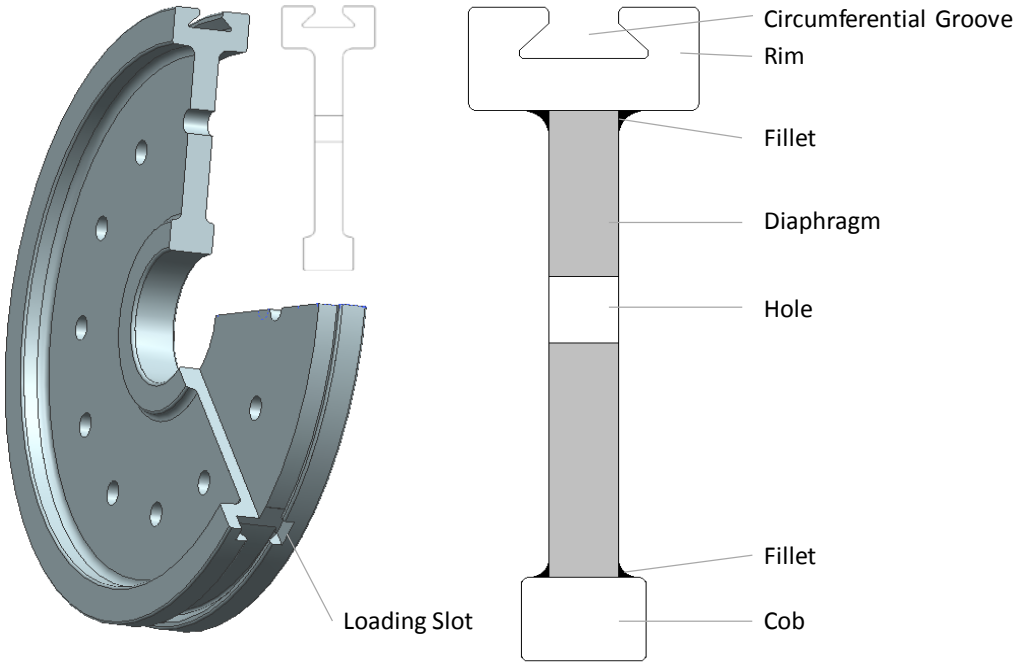


Figure 7.1. The example component

form of the studied disc is not shown here but the illustrated shape fulfils the essential requirements to present the study and its findings. One point to note here is that a feature of an engineering component can include different things, e.g. its shape, its material, or its colour, however the word feature used in this document refers only to shape features unless otherwise specified.

7.3.2.1. Design Features

The design of discs of this type usually starts with the modification of an existing component the geometry of which closely resembles the intended design. A disc is a uniform cross-

section component, obtained through a 360^{o} revolution of this cross-section. Its design, therefore, requires its features to be mainly divided into two categories. The first category includes those features which exist in the cross-section of the disc throughout its full revolution. These features are named as 2D or two dimensional features because a two dimensional image of these features is enough for their definition. The second category belongs to those features which have a specific location in the disc and a simple cross-section of the disc at that location is not enough to define their geometry completely. These features are called 3D or three dimensional features. These features are very important here because this categorization plays a crucial role in defining the manufacturing processes associated with these features. The features which are going to be the focus of this study are listed below.

2D Features	**3D Features**
Circumferential Groove	Loading Slot
Rim	Holes
Diaphragm	
Cob	
Rim-Diaphragm Fillet	
Diaphragm-Cob Fillet	

The analysis of design features is based on the functionalities attached to them. So the translation of design requirements into a final component would involve the creation and incorporation of certain features which fulfill the functional requirements of the intended design. A designed component, in this case, would therefore be considered as an accumulation of design features. At this point a design feature can be defined as:

"A distinct part of an engineering component to which a specific function or a set of functions can be associated"

When it comes to a set of functions associated to a design feature, a rating has to be developed which labels a specific functionality according to its importance. Figure 7.2, for example, illustrates a feature-functionality matrix of the features shown in Figure 7.1.

This feature functionality matrix is especially helpful when a design in modified. This is because a designer needs to know the unwanted changes that may occur to the

		Rim	Circumferential Groove	Diaphragm	Cob	Rim-Diaphragm Fillet	Diaphragm-Cob Fillet	Holes	Loading Slot
1	Transmit torque	9		5	1			5	
2	Join with other discs			1				9	
3	Position blades axially	1	9	1					
4	Position blades radially	1	9					5	
5	Prevent stress concentration					9	9		
6	Ensure concentricity when joining with other discs	1						5	
7	Balance the radial forces and stresses				9				
8	Load blades into the circumferential groove								9

Figure 7.2. A feature functionality matrix

functionality of a feature due to an alteration in the design of other features. A more important reason for such a matrix, which is also more relevant here, is the identification of distinct features in a component. If a major function can possibly be associated to an identified feature then that feature qualifies as a design feature. It can be seen in the matrix that every single listed feature has got at least one major function. This is determined through their score. A score of 9 makes a certain function the major reason for the existence of a specific feature in a component. Scores of 5 and 1, similarly, show the lower relevance of a feature for a particular function. Hence, the reason why the selected disc is divided into these eight features is the distinct functionality of each of these features and this is the criteria which can be used to identify design features in any component. For the disc studied here, three main divisions can be as follows:

Base Features	**Stress Relieving Features**	**Joining Features**
Rim	Rim-diaphragm fillet	Holes
Diaphragm	Diaphragm-cob, fillet	Circumferential groove
Cob		Loading slot

There are two different types of divisions for design features defined here. The division according to the function a particular feature performs is helpful during the initial stages of design when the concept starts to come into being, while the division into 2D and 3D features becomes handy during the analysis of the initially proposed design. Examples of

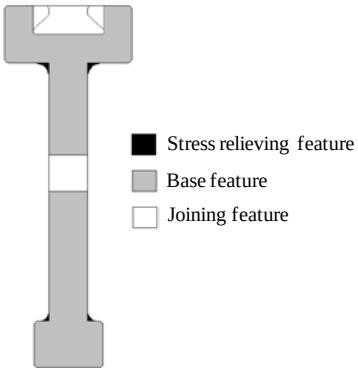


Figure 7.3. Design Features

this are thermal analysis and stress analysis where the initial analyses are performed mainly for the 2D features and then the process is further extended to 3D features if necessary. Other uses of the latter division relate to the manufacturing features which are explained in the next section. Figure 7.3 gives an image of how a designer might view the cross-section of this disc in terms of the features of which it is comprised.

7.3.2.2. Manufacturing Features

While the criteria for the identification of design features are the functions they perform, the criteria used for the recognition of manufacturing features are the manufacturing methods required for their production. Hence, it is necessary to analyze the manufacturing processes the selected disc goes through. A look at the production route, or machining route to be more specific, of the studied disc reveals that it mainly requires the processes of turning and milling for its production. It is important to note here that the demarcation made between design features to make them distinct does not work when manufacturing

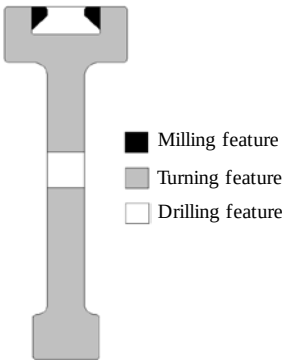


Figure 7.4. Manufacturing Features

features are to be identified. This is because more than one design features can be produced during a single manufacturing process. Therefore, a manufacturing feature may possibly be a combination of several design features. Going back to the division of design features into 2D and 3D, a connection can be seen between this division and the manufacturing processes. All two dimensional design features can be produced through the process of turning while the 3D features are either produced through drilling or milling. This connection is very helpful when manufacturing knowledge is attached to design features for the designer to understand. An obvious designation of manufacturing features is therefore turning features, milling features and drilling features as shown in figure 7.4 and categorized below.

Turning Features	**Milling Features**	**Drilling Features**
Rim	Loading slot	Holes
Diaphragm		
Cob		
Circumferential groove		
Rim-diaphragm fillet		
Diaphragm-cob fillet		

7.3.2.3. Standard Features

One of the main objectives of this case study was to analyze how a standard feature-based design and manufacturing system may work in a manufacturing setup. For this reason it was necessary to first define what is considered to be a standard feature. A standard feature is therefore defined as:

"A geometrical feature, having a specific design function associated to it and a fixed manufacturing method assigned for its production."

It was understood after an in depth analysis of the manufacturing methods of features that to standardize a feature, its manufacturing method, the tooling it uses and the way it is finally inspected have to be fixed. A feature therefore is standardized if it is always manufactured and inspected in the same way. For this to happen, a feature designated as a standard needs to exist in most of the components belonging to a certain part family. This is because the fixation of a manufacturing and inspection method has to be justified by the

high volume of production of a feature or its variants. For this reason, the feature ubiquity was the primary consideration when features in the selected disc were determined, as shown in Figure 7.1.

The features shown in Figure 7.1 clearly have distinct functions attached to them as proven by the matrix in figure 7.2, and their manufacturing methods were also found to be similar across a range of discs belonging to the same part family. Hence, with a fixed set of tooling, these features can be designated as standard. This set of eight features, therefore, are considered to be standard features and will be dealt with as so in the rest of the document. These standard features provide a common ground for the design and manufacture domains to use them as the basis for their individual knowledge bases. To further increase the interoperability, these features can be named by using an existing standard like ISO 10303-224 (ISO/DIS 10303-224:2003(E), 2003). This standard gives a detailed account of machining features that may exist in an engineering component and therefore can become the basis of the naming conventions of standard features.

7.3.2.4. Interdependence of standard features and manufacturability limitations

Since the recognition of features is different in the design and manufacturing domains, more than one standard feature may get produced in a single manufacturing or machining operation. For example, the initial turning operation produces the circumferential groove, the rim, and partly the diaphragm. In another turning operation the cob and rest of the diaphragm are produced. It is Important to also note here that the interfaces between these standard features, which are called fillets, are also produced during these operations because, clearly, in terms of manufacturing they are not a separate entity. Some of these interdependencies and manufacturing limitations experienced during the case study are now discussed.

7.3.2.4.1. Limited tool life

During the study some cases were discovered where accumulated machining (i.e. because of limited tool life) affects the dimensional requirements of a feature. For example, during the turning of the lower part of the disc, the tool machines the surface starting from cob, going through the fillet and machining some part of the diaphragm as shown in figure 7.5. Due to the limited tool edge life, the tool has to be lifted off the surface for replacement.

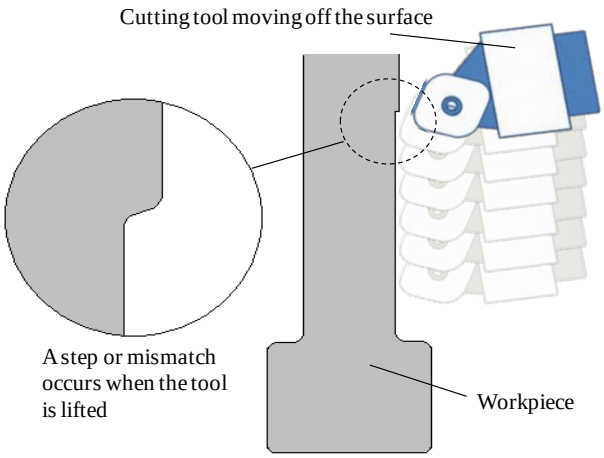


Figure 7.5. Limited tool life

This lifting off of the tool causes a step to be formed on the diaphragm surface. Obviously, a better and more expensive tool might solve this problem but when the tooling is standardized, these things have to be taken care of by the designer when prescribing the dimensional requirements of a feature.

7.3.2.4.2. Undersized fillets

Another case witnessed was when one of the fillets in the disc had a radius smaller than the smallest available tool. This is shown in figure 7.6. Again a smaller tool can do the job but a sharper tool tip makes it more delicate and vulnerable to breakage and breaking a tool during machining might mean ruining the whole component. These are the limitations

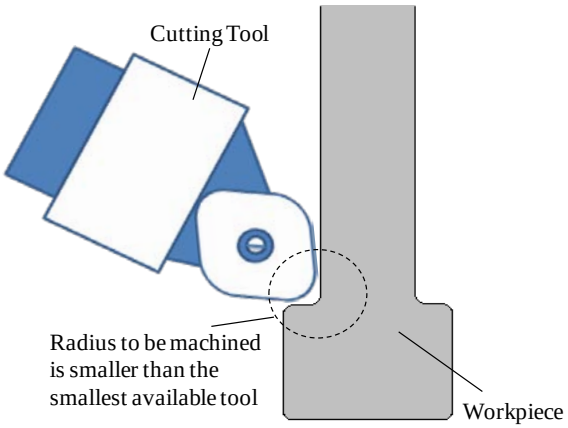


Figure 7.6. Undersized fillets

therefore which need to be considered when a component or features in that component are designed.

7.3.2.4.3. Tool accessibility limitations

Another limitation which designers need to be aware of is the accessibility of cutting tools. For the component selected for this case study, the circumferential groove feature posed this problem. A tradeoff exists between the groove angle that is needed for firm blade holding and the opening width. Too big an opening might reduce the amount of force needed by the groove edges to hold the blade and too small an opening offers the problem of tool asscessibility as shown in figure 7.7. For this reason, the available tools were limited to a certain range of angles and groove depths that could be achieved.

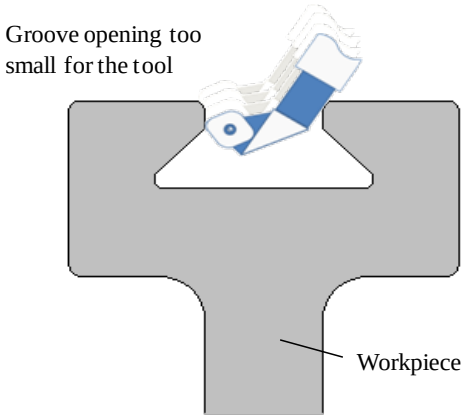


Figure 7.7. Tool accessibility limitations

7.3.2.4.4. Fixture suitability

The way a component is held may also cause problems during its machining. It was found that the holding location, holding mechanism and tool motion path all contribute to create difficulties during the machining of a disc.This issue, therefore, also needs to be accounted for during designing and relevant features need to be modified accordingly.

7.3.2.5. Perceived Interoperability Issues

It has so far been established that the interpretation of different parts of the same component may differ in design and manufacturing domains. Designers look at a feature from its functionality point of view while manufacturers try to translate the component features into the language of manufacturing methods and processes. An example of this difference is illustrated in figure 7.8. In addition to that for a manufacturer the terms joining

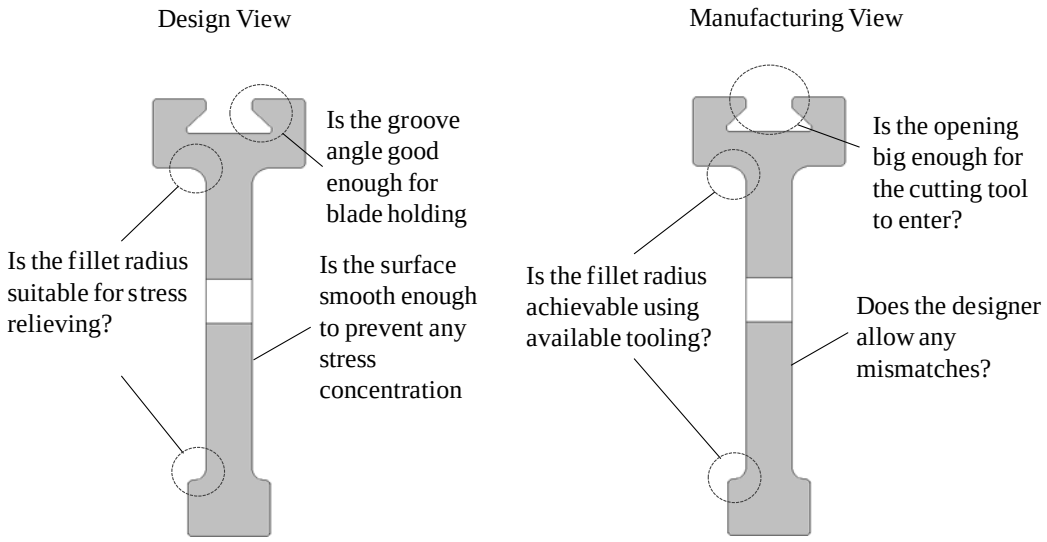


Figure 7.8. Comparison of designer's and manufacturer's views

feature or stress relieving feature do not make any sense and the same features or a combination of them are identified as drilling features and turning features. This is a typical example of conceptual differences that may lead to conceptual mismatches when knowledge is mapped between two domains. But this is not all that may go wrong and some terminological differences were also found to exist. For example, the part of the disc between the edge and the centre was known as diaphragm while the manufacturing engineers and machine operators were calling it a web. Similarly, the central portion was referred to as a 'cob' in design and a 'hub' on the shop floor. These conceptual and terminological differences are a clear indication of the potential semantic mismatches when knowledge bases are built independently.

To handle these issues, a verification system is needed which mediates between design and manufacturing knowledge bases to reconcile any conceptual and terminological differences. This case study provided practical examples of these differences and the verification system therefore needs to be capable of alleviating these and other similar problems.

7.4. Case study findings summarized

It was established during the study that design and manufacturing engineers have their own perceptions and views of a component and its features. Designers look at a part from its functionality point of view while the manufacturing engineers percieve it from the manufacturing method viewpoint. To ease the solution of this problem, a component can

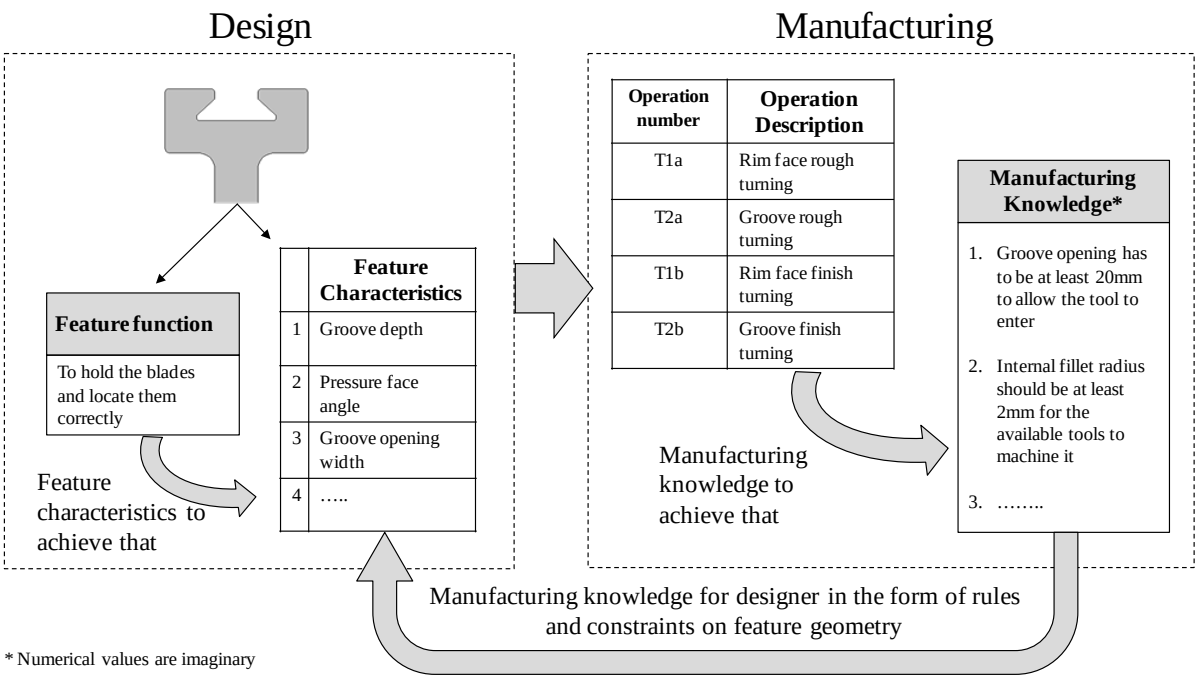


Figure 7.9. Manufacturing knowledge flow to the designers

therefore be divided into standard features. These features when dealt with on an individual basis make it easier for the two parties to associate knowledge to them and thus understand each other by sharing this knowledge. Figure 7.9 shows how this system works. A designer designs a component by translating the intended functionalities into physical features. These features are then analyzed and modified on a detailed basis according to the function they are supposed to perform. This process gives the features their geometrical characteristics. An accumulation of these features then produces the whole component. The output of this entire course of action is an engineering drawing. When this engineering drawing goes to the manufacturing engineer it is dealt with on the basis of manufacturing methods and tooling available. This gives rise to the manufacturability issues. A coordination process then takes place to collect this manufacturing knowledge and make it available to the designer so that the next time a similar feature is included in a component, the designer is aware of its manufacturing requrements. Presently this co-ordination process uses documents listed in the first section of this case study. In the initial stages of the design, documents like the DND are used. When the component is designed, the manufacturability issues are resolved by initiating documents like the GCR. This exchange of documents,

however, is very time consuming and is not capable of preventing a non-manufacturable product at its early design stages.

The process described above can be made automatic and very quick if a computerized system is in place to stop the designer whenever a manufacturability constraint is violated. This can be achieved by building interoperable knowledge bases in both the design and manufacturing domains. The interoperability can be further enhanced by providing a verification capability within the system. This verificaiton and mediation system would increase the interoperability of knowledge by reconciling conceptual and terminological differences between the design and manufacturing domains. This is what this research encompasses and this case study gave the researcher an idea of the real engineering design and associated manufacturability problems that need to be resolved through an interoperable manufacturing knowledge system. The following points more explicitly state the research requirements identified during the case study:

1- The information flow between design and manufacture is mainly regarding the manufacturability of a component. This is evident from the analysis of documents exchanged between design and manufacture.
2- Designing and manufacturing departments perceive engineering components differently depending upon the aspect that is important to them. Many examples of this perceptional difference were identified in the case study.
3- In the interpretation of different features of a component, some terminological differences also exist in addition to the perceptional differences across two departments as can be seen in the examples given in this chapter.
4- At present, exchange of knowledge is largely achieved through exchange of documents such as the ones listed in table 7.1. In the presence of a seamless automated knowledge sharing system the knowledge exchange and the process of manufacturability analysis can easily be made faster and more efficient.

In the light of these findings, this research proposes a knowledge mediation and verification framework (described in chapter 8) specifically targeting the manfuacturability analysis at early design stages of a product. The proposed framework caters for all the semantic i.e. perceptional and syntactic i.e. terminological differences found during the case study. This

verification framework is unique in the way it combines feature-based design and manufacturing with ontologies thus reaping the benefits of both technologies. Through the use of ontologies this framework alleviates the problem of semantic and syntactic differences and through the use of feature-based design and manufacture it provides a resourceful platform for the manufacturability analysis of components. An automatic manufacturability analysis software application is also developed which works on the principles of the proposed verification framework. The validaiton of the verification application (presented in chapter 9) is purely based on the manufacturability problems observed during the cases study. This successful case study based validation speaks volumes of the potential such a seamless knowledge sharing system has for an automated and quick analysis of the manufacturability of a component in similar manufacturing setups.

7.5. Conclusions

The findings of this case study endorse the understanding developed during the literature review. In the literature review it was found that there is a requirement of mechanisms capable of mediating between the independly developed knowledge sources, specifically those based on ontologies. This case study helps in experiencing the real world industrial problems where it is understood that the need for verification of knowledge is even greater when technical knowledge is shared, by using computer based systems, across departments even when these are departments within the same company. The next chapters aims to present the proposed solution which is intended to alleviate these knowledge verification problems.

Chapter 8: A novel knowledge verification framework for foundation ontology based knowledge bases

8.1. Chapter overview

After highlighting the research gap in chapter 4, understanding the capabilities of the chosen ontological formalism in chapter 5, demonstrating the use of ontologies in manufacturing knowledge sharing in chapter 6, and identifying the industrial knowledge interoperability problems in chapter 7 the proposed knowledge verification framework can now be explained in this chapter. Only the design and implementation aspects of the proposed framework are considered in this chapter. The validation comes in chapter 9.

8.2. Revisiting the findings so far

Before the verification framework is explained, it is useful to revisit the research gap and industrial interoperability requirements identified in the previous chapters. It was concluded after the review of ontology mediation techniques in chapter 4 that:

1- Most heuristic based approaches of ontology matching, target and resolve explication mismatches. Some work therefore needs to be done to develop techniques aimed at conceptualization mismatches.
2- All heuristic-based tools and techniques require human intervention at some point. A potential area of improvement, therefore, is the automation of these tools and techniques.
3- Foundation ontology based techniques resolve the issue of mismatches but introduce the problems of inconsistencies when concepts from the core concept ontology are chosen to build domain or application ontologies and associated knowledge bases.
4- A plausible solution for these inconsistencies appears to be the use of logical theories or axiomatizations at different levels of the foundation and core concepts ontology.
5- The design and development of a mechanism capable of finding similarities between two ontologies by using their inheritance in the foundation or core ontology is needed.

Since this research is a part of the IMKS project which mainly deals with domain ontologies underpinned by a foundation ontology, an ontology matching approach which makes use of these links from domain ontologies to the foundation ontology will be most effective and appropriate. As point 5 in the above list states, that a mechanism finding similarities through

the foundation inheritance identification seems to be the logical solution to the ontology matching problem in the IMKS scenario, however, some additional challenges also need to be considered because of the industrial problems identified during the case study which include the following:

1- The information flow between design and manufacture is mainly regarding the manufacturability of a component,
2- Designing and manufacturing departments perceive engineering components differently depending upon the aspect that is important to them,
3- In the interpretation of different features of a component, some terminological differences also exist in addition to the perceptional differences across two departments,
4- At present, exchange of knowledge is largely achieved through exchange of documents. In the presence of a seamless automated knowledge sharing system the knowledge exchange and the process of manufacturability analysis can easily be made faster and more efficient.

Thus, the five requirements obtained from the literature review when viewed in the light of the findings of the industrial case study highlight the need of:

"A knowledge verification system, specializing in manufacturability analysis, capable of automatically resolving the semantic and syntactic inconsistencies resulting from the perceptional differences between design and manufacture through the use of logic in heavy weight foundation ontology based knowledge bases."

The keywords in the above statement are underlined to emphasize the fact that the proposed solution has to *automatically resolve the perceptional differences through the use of logic*. How this is achieved by the proposed knowledge verification framework is explained next.

8.3. A novel knowledge verification framework

Keeping in view the requirements set by the above presented findings, a novel knowledge verification framework is proposed in this research. A schematic of this framework can be seen in figure 8.1. This verification framework proposes the use of logic not to overcome differences in domain ontologies during knowledge sharing but to prevent them from

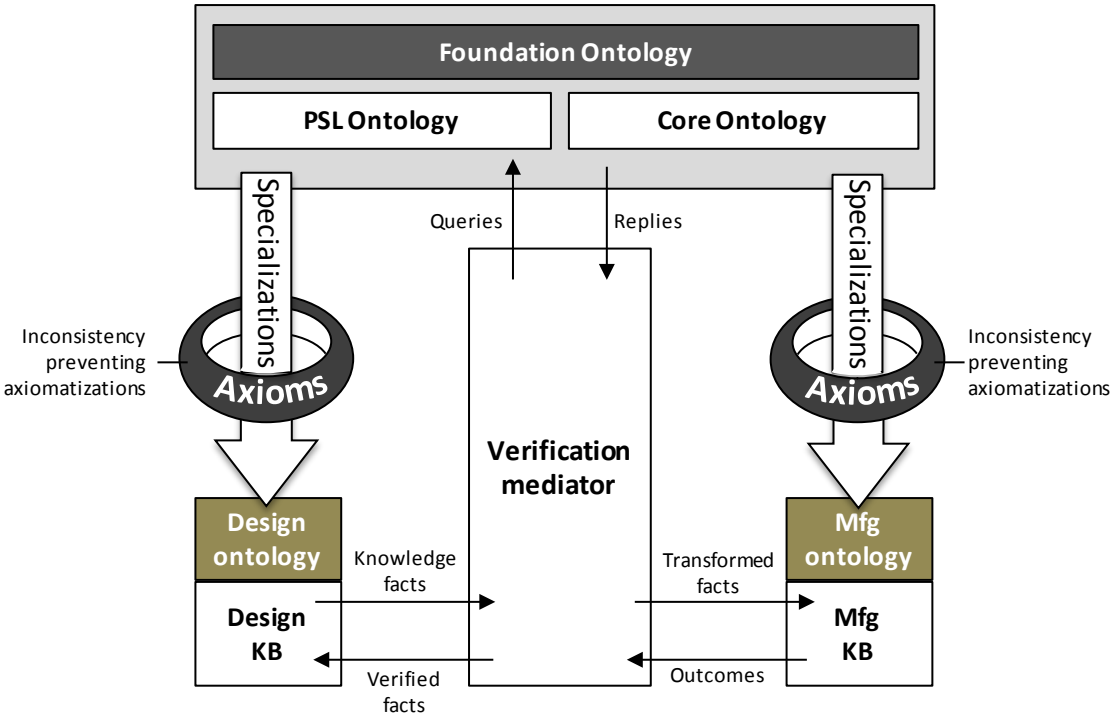


Figure 8.1. The verification framework

happening in the first place. This is done firstly at the domain ontology level i.e. during the process of domain ontology development and secondly at the knowledge base level during the process of knowledge base population. To materialize this idea into an understandable form, a verification framework is proposed to contain a set of 'inconsistency preventing axiomatizations' (shown as axioms in figure 8.1) and a 'verification mediator'. As has already been explained in chapter 5, the ontological formalism used in this research allows the user to write integrity constraints and axioms to control the creation of classes in the ontology and their use in the knowledge base. With this capability in view, the proposed axiomatizations are aimed at standardizing the way concepts from the foundation and core-concepts ontology are specialized in the domain ontology. This standard way is needed to ensure that no inconsistencies are created during the process of concept specialization due to subjective interpretations of foundation and core concepts by the domain experts. These constraints are proposed to either exist in the foundation and core-concepts ontology or as a separate attachable ontology. The 'verification mediator', which is the second part of the proposed verification framework, is designed to detect similarities on the basis of the assumption that the specializations in the domain ontologies are created by following the standard way. The design and implementation details of this framework come next.

8.4. Design of the verification framework

Figure 8.1 illustrates the setting in which this verification mechanism is designed to work. Five main components of this framework are:

1- Foundation and core-concepts ontologies,
2- Domain ontologies,
3- Knowledge bases,
4- A set of inconsistency preventing axiomatizations and
5- The verification mediator

A detailed design of each of these components can be a research project in its own right. For that reason, the presented research only focuses on the last two components which specifically deal with the verification of knowledge shared between foundation ontology based domain ontologies. In the following text, a brief description is given of the first three components while a detailed design description of the last two components is presented.

8.4.1. Foundation and core-concepts ontologies

At the top of figure 8.1, two layers of ontologies can be seen. The first layer is that of the foundation ontology which then subsumes in the second layer - the PSL ontology and a core ontology of manufacturing concepts. A description of the PSL ontology can be found in chapter 3. PSL is used here to provide core-concepts for defining processes. The manufacturing core-concepts ontology, on the other hand, provides concepts for defining products. Together, these two ontologies provide the vocabulary for building domain ontologies through a controlled specialization of their concepts and subsequently the knowledge base through controlled modelling of design and manufacturing processes.

8.4.2. Domain ontologies

The foundation and core-concepts ontologies hold very general concepts. Which means, although these concepts can directly be used to build the knowledge base, only a very general level of modelling can be done without going into the low level details of manufacturing processes. For this reason, domain ontologies need to be built, with idiosyncratic and specialized concepts, before knowledge is modeled in the knowledge bases. These idiosyncrasies depend upon the preferences of a certain domain. Two domains

studied during the case study were design and manufacture and therefore two domain ontologies belonging to these domains are included in the framework.

To explain the proposed framework, those examples from the case study are used where the domain ontologies are needed to cater for different perceptions and terminologies existing in different departments. In these cases, therefore, domain ontologies are used as a source of localized vocabulary to model knowledge in departmental knowledge bases. These domain ontologies, however, are assumed to have connections in the foundation and core-concepts ontology with concepts and terms of similar meaning. To make sure that these connections are built whilst building the domain ontology, it is proposed, that some axiomatizations and constraints should be used. These are called here the inconsistency preventing axiomatizations and will be explained later.

8.4.3. Knowledge bases

At the bottom of all the layers of foundation, core-concepts, and domain ontologies lie the knowledge bases. These knowledge bases are built by using concepts from the domain ontology of a respective department. Since there are differences in the terminologies used by the design and manufacturing domain ontologies, the knowledge facts built under these ontologies will also be different. It is the work of the verification mediator then to identify similarities and verify knowledge. The scenario that will be used to explain the implementation of the proposed framework assumes that the design knowledge base contains the facts that model a component. It is proposed in this research that these design models of components are examined by the manufacturability constraints existing in the manufacturing domain ontology before they are finalized. This is done through the verification mediator as can be seen in figure 8.1. It is shown that, through the use of queries, the verification mediator verifies facts by first sending them to the manufacturing knowledge base to be checked by the manufacturability constraints existing in the manufacturing domain ontology. If no objections are raised by the manufacturing domain ontology the verification mediator sends the verified facts back to the design knowledge base for them to be finally asserted in the form of a product model. Further explanation of the functioning of the verification mediator is given in the implementation section of this chapter.

8.4.4. Inconsistency preventing axiomatizations

Some deliberation is required to enable correct decisions to be made about the type of constraints which should be written to address the possible inconsistencies that may occur during concept specialization. The ideal way to address this challenge is to conduct a social scientific survey to discover the types of inconsistencies that may occur by building independent ontologies during the process of concept specialization. Alternatively (or additionally), existing literature on this topic can be searched to find results of any similar research conducted in the past. The literature on foundation and upper ontologies, however, does not offer any such knowledge and the scope of this research does not allow the social science aspects of the use of foundation ontologies to be thoroughly investigated. Hence, the design and testing of the framework proposed in this research has been based on possible inconsistencies identified through the industrial case study work reported in chapter 7. For that reason, some obvious possible inconsistencies are considered in this research to explain and test the proposed approach. Since the concepts from the foundation and core-concepts ontology are used first to build the domain ontology and then the knowledge base, two types of axioms need to be written. The first type is proposed to scrutinize the concept specialization in the domain ontology and the second type is aimed at examining the completeness of the knowledge facts in the knowledge base. These two types are explained below with the help of examples. The word 'axiom' is used, instead of axiomatization, from now on for brevity.

8.4.4.1. Axioms for the domain ontologies

The most vital issue for the foundation ontology based domain ontologies is the linking of concepts in the domain ontologies to similar concepts in the foundation and core-concepts ontology. Two types of cases may exist in this regard. In the first case, an independently developed domain ontology may need to be linked to a foundation by mapping similar concepts. In the second case, a domain ontology is built by using the concepts from the foundation ontology as building blocks. In both of these cases, there needs to be a standard way of linking the concepts in the two ontologies together such that the structure of the domain ontology is consistent with the foundation and core-concepts ontology. The compliance to this standard way can be ensured through axioms or integrity constraints present in the domain ontology. The existence of these axioms is extremely vital for the

functioning of the verification mediator which is the second main part of the verification framework proposed in this research. The reason being that the main assumption on which the verification mediator works is that the domain ontologies are consistent with the foundation and core-concepts ontologies in terms of their structure if not content. Hence, the method of subsuming the domain concepts under similar foundation or core concepts needs to be defined explicitly. As far as this research is concerned, the standard way of doing this is defined to be the super-concept relation. This is depicted in figure 8.2 where a

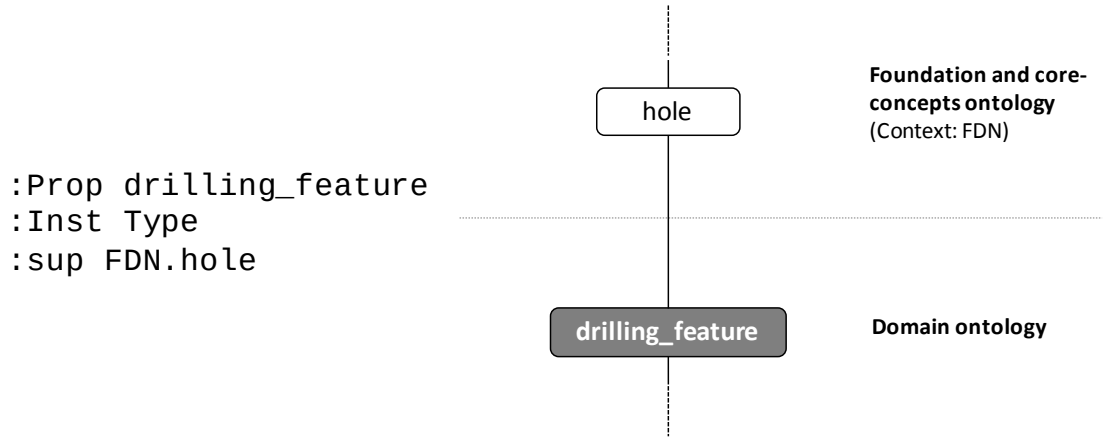


Figure 8.2. Foundation concept subsumption

concept named 'drilling_feature' in the domain ontology is shown to subsume a foundation or core concept of 'hole'. The KFL assertions needed for this subsumption are also shown. The 'sup' directive in these assertions is responsible for the establishment of a link between the domain and foundation concept. In this case it subsumes the domain concept under 'FDN.hole'. The prefix FDN here represents the context in which concepts in the foundation and core are defined. The inclusion of this prefix shows that the class named 'hole' exists in the foundation and core-concepts ontology. The verification mediator works on the assumption that a 'sup' directive for every concept in the domain ontology exists. It is therefore very important that the inclusion of this directive is ensured. This can be done by writing the following axiom.

```
(=> (and (RootCtx.Property ?prop1)
                (RootCtx.Context ?ctx1)
                (RootCtx.contextFor ?prop1 ?ctx1)
                (UserContext ?ctx1)
                (not (= ?ctx1 MLO))
```

```
                (not (= ?ctx1 MFG)))
    (exists (?prop2) (and (RootCtx.Property ?prop2)
                          (RootCtx.contextFor ?prop2 MFG)
                          (sup ?prop1 ?prop2))))
:IC hard "Every class in the domain ontology needs to be subsumed to
a class in the foundation and core-concepts ontology."
```

The above axiom says what is shown in the 'IC hard' directive. This axiom makes sure that no class in the domain ontology is left unidentified and thus provides a solid platform for the verification mediator to work from. This is, however, not the only type of inconsistency that may occur during domain ontology building and linking.

One other type of inconsistency can be the incompleteness of specialization. For example, if a meaningful shape feature is to be introduced in an ontology then concepts depicting its dimensional characteristics also need defining. A core-concepts ontology can be made meaningful in this sense by defining all essential concepts related to a field of knowledge. A domain ontology, however, needs a check through axioms and constraints on the completeness of concepts specialized from the foundation and core-concepts ontology. Consider the segment of a core-concepts ontology with a domain specialization as shown in figure 8.3. It can be seen that the concept named 'hole' in the core-concepts ontology is specialized in the domain ontology with the name 'drilling_feature'. For this specialization to be correct and complete, the foundation concept of diameter and depth also need to be specialized. To make sure that this happens during domain ontology building, the following

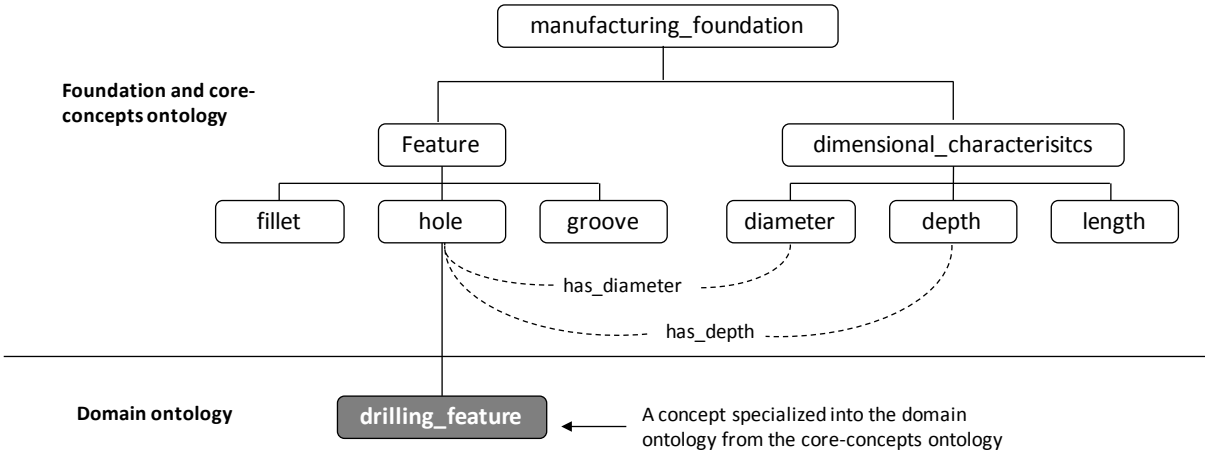


Figure 8.3. A specialization example

axiom can be written.

```
(=> (and (RootCtx.Property ?p1)
                (RootCtx.Context ?c)
                (RootCtx.sup ?p1 MFG.hole)
                (RootCtx.contextFor ?p1 ?c)
                (not (= ?c FDN)))
    (exists (?p2)(and (Property ?p2)
                (sup ?p2 FDN.diameter) (contextFor ?p2 ?c))))
:IC hard "If there is a specialization of the 'hole' concept then a
specialization for the ‘diameter’ should also exist."
```

This axiom says that if there exists a class in the domain ontology and that class is a specialization of the foundation concept ‘hole’, then another class as a specialization of the foundation concept ‘diameter’ should also exist in the domain ontology.

The above example explains just a few of the many possible inconsistencies that may occur during the foundation or core concept specialization in the domain ontology. These examples, however, demonstrate the use of axioms to prevent inconsistencies, which was the main aim of their explanation. The methodology to prevent any known inconsistency, therefore, is the same. Axioms can specifically be written for a possible error and hence inconsistencies can be prevented.

8.4.4.2. Axioms for the knowledge base

When a domain ontology is constructed, the knowledge base is built though the population of concepts introduced in the domain ontology. There can, however, be a case where concepts from the foundation and core-concepts ontology are directly used to build a knowledge fact. To prevent inconsistencies in such a circumstance, axioms are proposed to exist. For example to make sure that all the essential dimensional parameters of a circular disc are defined in the knowledge base, the following axiom can be written.

```
(=> (disc ?x)
           (exists (?d ?h)
           (and (has_diameter ?x ?d)
                     (has_height ?x ?h))))
```

```
:IC hard "For a complete description of a ‘circular disc’ both
‘diameter’ and ‘height’ are needed."
```

This axiom says that if there exists a hole, then its diameter and depth should also exist and since it is a hard integrity constraint, it does not allow the user to proceed without correcting the error. Axioms like these can be written to prevent inconsistencies at the knowledge base level.

The axioms presented above specifically target certain inconsistencies, generally however, they attempt to maintain a certain structure of domain ontologies committing to the foundation. It was explained in chapter 3 that an ontology becomes heavyweight when rules and axioms are added to it which govern the definition and interpretation of concepts within them. A foundation and core-concepts ontology, therefore, can have its own axioms and constraints. In the IMKS project, these axioms and constraints not only prevent the subjective interpretation of concepts but also dictate their use by domain ontologists. In such a case, the verification system needs to be designed according to the specialization controlling constraints of the foundation and core-concepts ontology. The verification mediator explained in the next section, works on the assumption that the concepts in the domain ontologies are connected to similar concepts in the foundation or core through the ‘sup’ directive as shown in figure 8.3. It is this link which is used by the verification mediator to establish similarity between independently developed domain ontologies. The working of this mediator is explained next.

8.4.5. The verification mediator

The verification mechanism described in this section is essentially an ontology mediation mechanism that uses the inheritance of domain ontology concepts in the foundation and core-concepts ontology to establish similarities. Keeping in mind the capabilities and shortcomings of existing ontology mediation tools, the new tool needs to have the capability to make the similarity detection process more automatic and accurate when two foundation ontology based domain ontologies are matched. Figure 8.4 (next page) shows the working of this mechanism.

This verification mediator works with the help of four modules named (1) Source ontology inheritance identifier, (2) Target ontology inheritance identifier, (3) Concept matcher and (4)

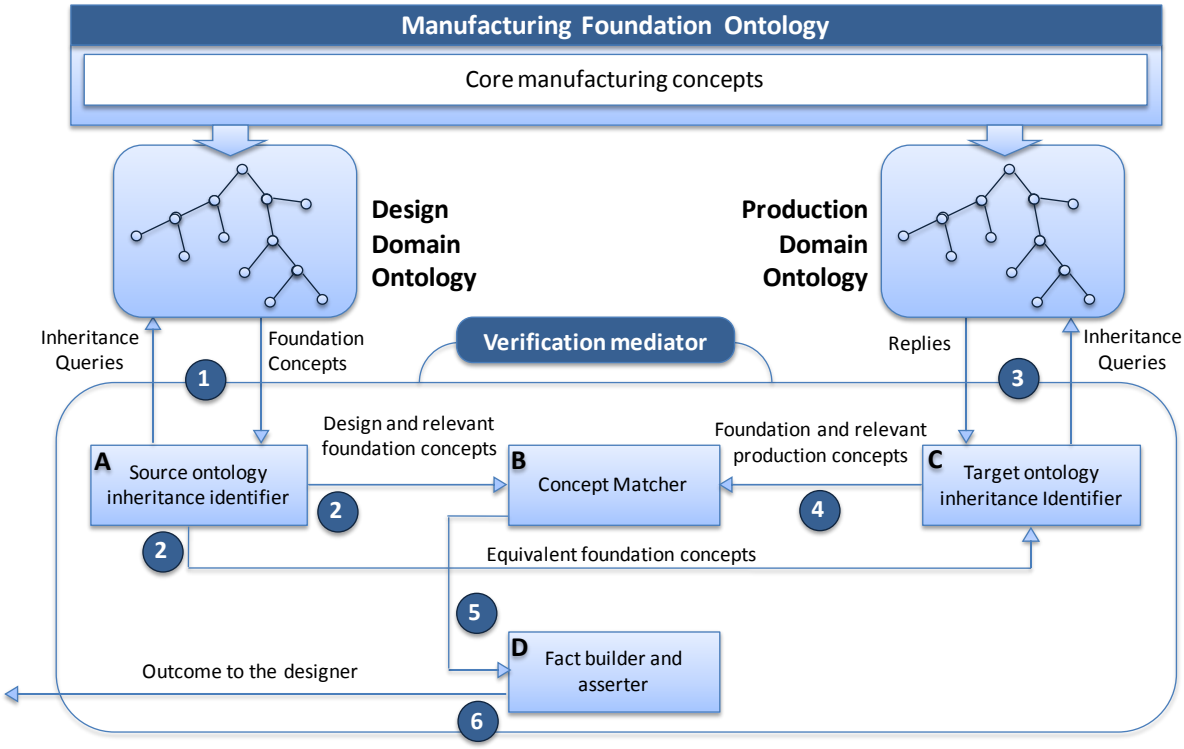


Figure 8.4. Working of the verification mediator

Fact builder and asserter. The role of these modules in ontology mediation and knowledge verification is described in detail below.

8.4.5.1. Source ontology inheritance identifier

The inheritance identifier is responsible for finding the super-class of a domain ontology concept in the foundation and core-concepts ontology. This is done by sending queries to the foundation. It takes a concept from one domain ontology, writes a query to find its foundation inheritance, receives replies and sends the replies to the 'domain concept identifier and concept matcher' module.

8.4.5.2. Target ontology inheritance identifier

The functioning of this module is similar to the inheritance identifier. However, instead of finding the super-class of a domain concept, it searches for the sub-class of a concept in the foundation and core-concepts ontology. This is done by first locating the class names (received from the source ontology inheritance identifier) in the foundation and core-concepts ontology and then querying for their sub-classes in the target domain ontology.

8.4.5.3. The concept matcher

The concept matcher takes input from the inheritance identifier and domain concept identifier modules and determines their similarity by analyzing their foundation inheritances. The rule to follow here is that if the foundation inheritances of two classes in two domain ontologies are the same then these domain ontology classes are also same. Once the similarity is established the similar class names are forwarded to the fact builder and asserter module.

8.4.5.4. The fact builder and asserter

This module starts functioning once the similarities are established. It is, therefore, technically not a part of the ontology mediation system but does play a role in the verification of knowledge. The task of the fact building that it involves is writing a knowledge fact in an ontological formalism and then asserting it in the knowledge base built out of the domain ontology concepts.

The functioning of this module can be better understood when the functioning of the verification mediator is explained. This is done in the next section where, with the help of an industrial scenario, the implementation of the proposed verification framework is explained.

8.5. Implementation of the verification framework

It can be understood from the description of the modules of the verification mediator that the basic principle of its working is based on the exploitation of the link between domain concepts and their equivalents in the foundation or core-concepts ontology. This link, in the case of this research, is the inheritance of concepts in the foundation or core-concepts ontology. The functioning of the verification mediator is therefore explained with the help of an example explaining a real industrial scenario.

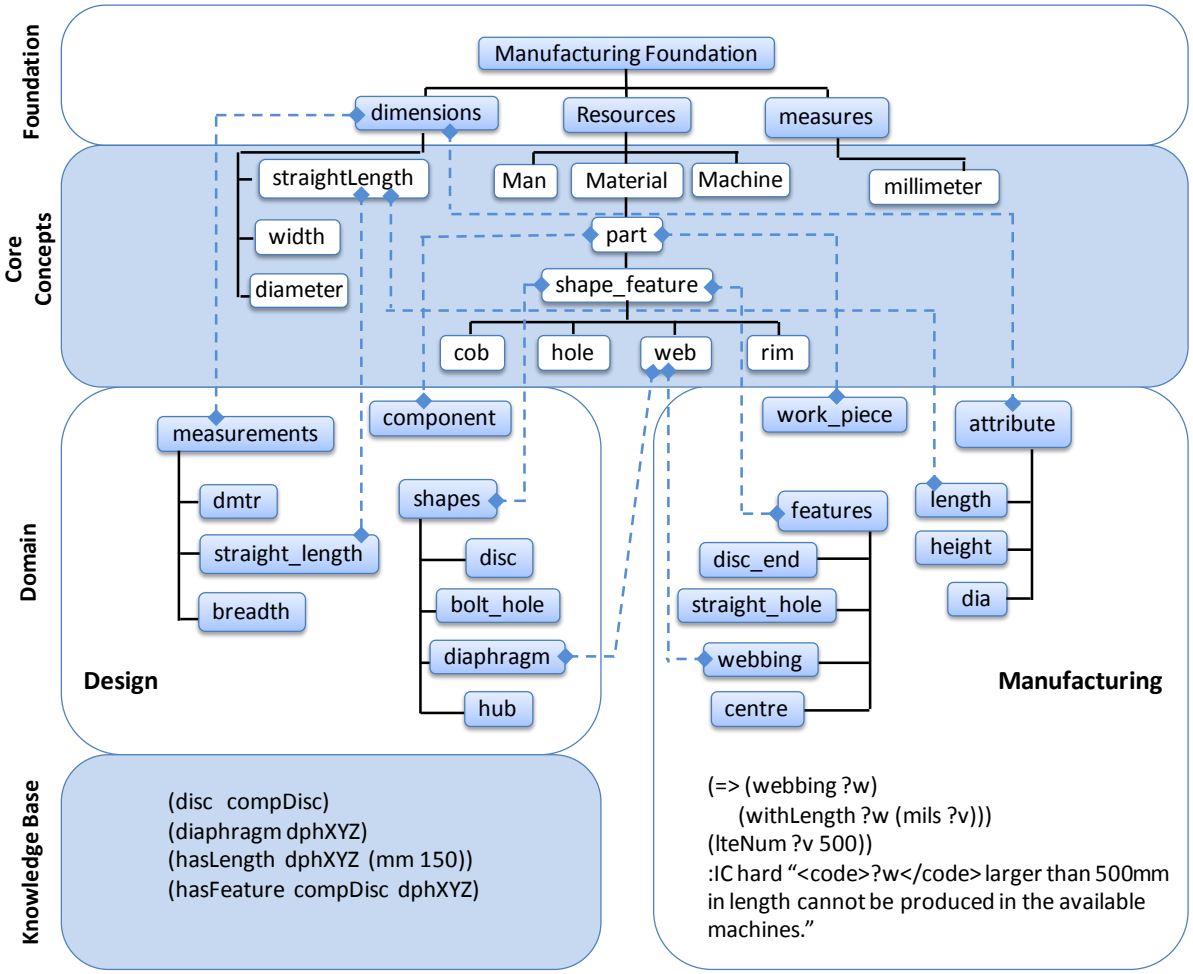


Figure 8.5. A specialization and knowledge population example

8.5.1. The industrial scenario explained

Figure 8.5 illustrates a real industrial scenario where differences exist in two domain ontologies committed to a common foundation and core ontology. The core and domain ontologies bear concepts needed to represent shape features which combine to form the component shown in figure 8.6. The differences in the domain ontologies have been intentionally induced for the sake of explaining the working of the verification mediator.

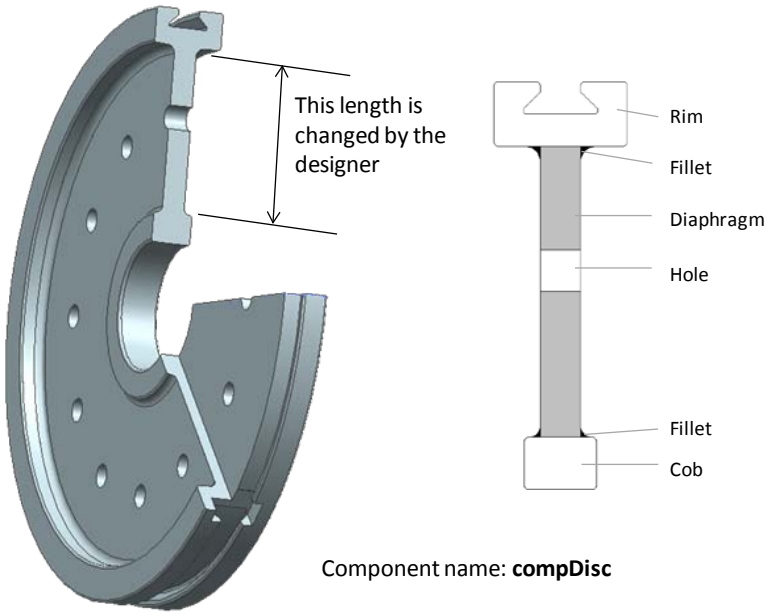


Figure 8.6. The example component

This is done in light of the evidence obtained in the case study explained in chapter 7. It can be seen in figure 8.5 that there are four layers in this model of an ontology based knowledge sharing system. The first layer contains very general foundation concepts while the second layer holds core manufacturing concepts to be committed to by the concepts in the domain ontologies existing in the third layer. The fourth layer is that of the knowledge base which contains ontological models of the components formed by using shape features from the domain ontologies above. The manufacturability rules aimed at governing the creation and modification of these models lie in the manufacturing domain ontology. In figure 8.6, the dotted lines show that the terms in two domain ontologies are connected to their equivalents in the core-concepts ontology. It can be seen that the two domain ontologies use different terminologies to represent the same entity in the core-concepts ontology. This has implications for the facts asserted in the knowledge base. The knowledge facts in the design knowledge base, in such a situation, need to be translated into the manufacturing ontology language for the rules existing there to make sense. This is what the verification mediator does.

The implementation of the framework will now be demonstrated with the help of a formalized example. The ontological formalism used for the ontology formalization is KFL as explained in chapter 5.

It is assumed that the design knowledge base contains the following facts:

```
(component   compDisc)
(diaphragm dphXYZ)
(hasLength  dphXYZ  (mm 150))
(hasFeature  compDisc  dphXYZ)
```

These lines say that an instance of the class ‘disc’ exists with the name ‘compDisc’. This instance has a feature named ‘dphXYZ’ which is an instance of the class ‘diaphragm’ and the length of ‘dphXYZ’ is ‘150mm’. In simple words, these facts say that there is a component which has a diaphragm feature with length 150mm. In an ideal case, these facts, before they are used to model the component in the design knowledge base, are to be inspected by the manufacturability rules existing in the manufacturing ontology. One such rule is shown in figure 8.6 which says that the length of the disc arm (which is diaphragm in the design domain ontology and webbing in the production domain ontology) should not exceed 500mm for it to be produced in the available machining facility. The rule is as follows:

```
(=> (webbing ?w)
      (withLength ?w (mils ?v)))
(lteNum ?v 500))
:IC hard “<code>?w</code> larger than 500mm in length cannot be
produced in the available machines.”
```

In the start of the IC hard statement, it says ‘<code>?w</code>’ which enables this statement to be modified according to the name of the feature represented by the variable ?w. By writing the IC hard statement in this way makes it understandable to the designer as the ‘<code>?w</code>’ is replaced by the name of the feature used by the designer. This will be shown at the end of step 6 of the verification process. The coding of this rule, however, is purely in the manufacturing ontology language and is not comprehensible by the computer system in the design knowledge management system. This industrial scenario poses a problem that needs an ontology mediation mechanism capable of finding similarities across two ontologies. Following is the description of the verification mediator which is designed to do exactly this.

8.5.2. Six steps of verification mediation

As mentioned earlier, in order for the rule to understand the design component models, the facts used to build that model need to be translated into the manufacturing ontology language. This is precisely what the verification mediator is designed to do. It first translates the facts asserted in the design knowledge base into the language of the manufacturing domain ontology, it then asserts the translated facts into the manufacturing knowledge base. Since these facts are now in the manufacturing ontology language, all the manufacturability rules existing in the manufacturing ontology become valid and thus a response according to the validity of changes made by the designer is obtained. The knowledge facts written by the designer for modifying an existing model or for creating a new model are now asserted or denied depending upon the response obtained from the manufacturing ontology. The complete kfl coding of the ontologies shown in figure 8.5 can be seen in Appendix I.

In order to translate the knowledge facts from the design into the manufacturing terminology, the verification mediator first needs to establish similarities between the concepts across two ontologies. Since a typical kfl fact consists of concepts, relations and functions, three types of similarities are to be established named:

1- Concept similarity,
2- Relation similarity and
3- Function similarity.

Once these similarities are established, the facts are translated from one form into another. Figure 8.4 shows the process of verification taking six steps to complete. These six steps are explained next by using the example of figure 8.5.

Step 1 – Inheritance queries and foundation concepts

In this step, the 'source ontology inheritance identifier' module of the verification mediator first decomposes the facts asserted by the designer into its elementary concepts. It then queries the foundation ontology for the inheritances of these concepts one by one. As mentioned earlier, three types of similarities have to be established i.e. the concept, relation and function similarity, the 'source ontology inheritance identifier' module divides these three components of the ontology into its constituent elements or concepts.

In the example of figure 8.6, following design facts are asserted:

```
(component   compDisc)
(diaphragm dphXYZ)
(hasLength  dphXYZ  (mm 150))
(hasFeature  compDisc  dphXYZ)
```

In step 1, these facts are decomposed as follows:

Concept 1: `component`

Concept 2: `diaphragm`

Relation 1: `hasLength`

Relation 2: `hasFeature`

Function 1: `mm`

These relations are further divided into the concepts they hold between.

Relation1: `hasFeature`

Relation 1 - concept 1: `component`

Relation 1 - concept 2: `shapes`

Relation2: `hasLength`

Relation 2 - concept 1: `shapes`

Relation 2 - concept 2: `straight_length`

In the same way, the functions are also divided into the entities they hold between.

Function 1: `mm`

Concept measured: `straight_length`

Once the facts are decomposed, queries are then generated by the inheritance identifier to find the foundation inheritance of these concepts. One such query is shown here which results in the foundation inheritance of the concept ‘component’:

```
(and (sup DSN.component ?f) (contextFor ?f FDN))
```

This query says, ‘find that super-concept of DSN.component which has a context of FDN’. DSN here denote the design ontology context while FDN is the context for foundation ontology as can be seen in the ontologies in appendix I.

Similar queries written for all the concepts return the answers shown in table 8.1.

Table 8.1: Foundation inheritances of design concepts

Design Concept	Foundation Inheritance
component	part
diaphragm	web
shapes	shape_feauture
straight_length	straightLength

Step 2 – Storage of design and relevant foundation concepts

The results obtained in step 1, as shown in table 8.1, are forwarded in this step simultaneously to the two modules of the verification mediator named 'concept matcher' and 'target ontology inheritance identifier'. The 'concept matcher stores these results for later comparing them with the results obtained in step 4 from the 'target ontology inheritance identifier'.

Step 3 – Inheritance queries for the target ontology concepts

In this step, the 'target ontology inheritance identifier' takes the results received from the 'source ontology inheritance identifier' and generates queries to track down the sub-classes of the foundation concepts in the manufacturing domain ontology. One such query is given below which finds the manufacturing ontology subsumption of the foundation concept 'part'.

```
(and (Property ?m) (contextFor ?m MFG) (sup ?m FDN.part))
```

Similar queries are written for other foundation concepts identified in step 1. In case of the example knowledge sharing system presented in figure 8.5, this step will obtain results shown in table 8.2 below.

Table 8.2. Foundation inheritances of manufacturing concepts

Foundation Concepts	Manufacturing Subsumpitons
part	work_piece
web	webbing
shape_feauture	features
straightLength	length

Step 4 – Sending foundation and relevant manufacturing concepts

In this step, the results obtained in step 3 are sent to the 'concept matcher module' for this module to compare the foundation inheritances of concepts in design and manufacturing domain ontologies.

Step 5 – Sending similarity information to the 'fact builder and asserter module'

In the fifth step, the information received from the 'target ontology inheritance identifier' module is used to establish similarities by the 'concept matcher' module. This is done by comparing the foundation inheritances of design concepts received in step 2 from the 'source ontology inheritance identifier module' with those received in step 4. In other words, the information in table 8.1 is compared with that in table 8.2. The results obtained are shown in table 8.3.

Table 8.3. Established similarities according to the foundation inheritances.

Design Concepts	Foundation Inheritances	Manufacturing Concepts
component	part	work_piece
diaphragm	web	webbing
shapes	shape_feauture	features
straight_length	straightLength	length

This comparison of two sets of concepts results in the discovery of similarities across two domain ontologies. These similarities are shown in table 8.4 below.

Table 8.4. Established concept similarities

Design Concepts	Equivalent Manufacturing Concepts
component	work_piece
diaphragm	webbing
shapes	features
straight_length	length

Once similarities between concepts are established, relations and functions existing in the manufacturing ontology binding the same concepts as in the design ontology are also declared similar. The relation similarities found in this case are shown in table 8.5.

Table 8.5. Relation similarities across two ontologies

Design relation	Holding between	Manuf. equivalent	Manuf. relation
hasFeature	component	work_piece	hasAttribute
	shapes	features	
hasLength	shapes	features	withLength
	straight_length	length	

In the same way, the function similarity is shown in table 8.6 below.

Table 8.6. Function similarities across two ontologies

Design function	Measures	Manufacturing equivalent	Manufacturing function
mm	straight_length	length	mils

The relations and functions mentioned in this example can be seen in the complete ontology coding presented in appendix I.

The steps explained up to this point fulfill the requirements of ontology mediation and resulting knowledge verification when text-based knowledge is shared between the communicating parties. The case considered here, however, is different. In this case, the manufacturing knowledge to be shared exists in the form of integrity constraints or manufacturability rules in the manufacturing ontology. To make these rules meaningful, the established similarities now have to be used to translate the entire design model in to a manufacturing model to make use of the manufacturing knowledge. The step explained next is aimed at performing this task.

Step 6 – Translating facts and asserting to verify manufacturability

Since all three types of similarities i.e. concept, relation and function, have now been established for the facts to be asserted, the facts are translated into manufacturing language as shown in table 8.7.

Since these facts are now in a comprehensible form for the manufacturing knowledge base, their assertion results in the manufacturability rule getting activated in case the length is changed to a value greater than 500mm. The facts modelling the length of the 'diaphragm' less than 500mm are successfully asserted in the design knowledge base as they have now been approved by the manufacturability rules existing in the manufacturing knowledge

Table 8.7. Design and translated manufacturing facts

Design facts	Translated manufacturing facts
(component compDisc)	(work_piece compDisc)
(diaphragm dphXYZ)	(webbing dphXYZ)
(hasLength dphXYZ (mm 150))	(withLength dphXYZ (mils 150))
(hasFeature compDisc dphXYZ)	(containsFeature compDisc dphXYZ)

base. In case of a length value more than 500mm, the integrity constraint statement is passed on to the designer which says:

"dphXYZ larger than 500mm in length cannot be produced in the available machines."

dphXYZ in the above statement is the name of the feature designer gave while asserting facts. It is to be noted that the process explained above, broadly, is manufacturability verification. The whole procedure to verify manufacturability, however, involves the process of similarity finding and knowledge verification which is done in steps 1 to 5 up to the point where similarities are established.

8.6. Conclusions

In this chapter, the design and implementation of a novel knowledge verification framework is presented which is the main contribution of this research. The examples of manufacturability verification are used to explain the working of the verification mediator. A very simple example of the length of a feature is presented to explain the working and implementation of the proposed verification framework. In a real industrial scenario, however, significantly complex cases exist. In order to validate the proposed framework, one such scenario is chosen from the case study explained in chapter 7. To automatically perform the task of manufacturability verification, a software application is also developed. The validation of the proposed framework with the help of this software is presented in the next chapter.

Chapter 9: Validation of the proposed verification framework

9.1 Chapter overview

The description given in chapter 8 covers all the intricate design aspects of the proposed knowledge verification framework. An implementation scenario was also presented in the last chapter to further clarify the scope and working of the framework. In continuation with that, this chapter presents the validation of this proposed framework. This validation is done through an API developed to mediate between experimental domain ontologies aligned with a central experimental foundation and core-concepts ontology. This API is designed to work on the principles of the proposed verification framework and is tested on real industrial manufacturability issues observed during the case study explained in chapter 7.

9.2 Design of experiment

To validate the workability of the proposed verification framework, an experiment was conducted which involved the development of a few experimental ontologies and the population of these ontologies through a Java API specially designed and developed to automatically perform the tasks of knowledge verification as explained in steps 1 to 6 in the previous chapter. Following is the detailed description of the components, working, and results of this experiment.

9.2.1 Experimental ontologies

The first step in validating the framework is the conversion of the component shown in figure 9.1 into the form of a shape feature based ontological model. A methodology to create such a model has already been explained in chapter 6. The ontological formalism i.e.

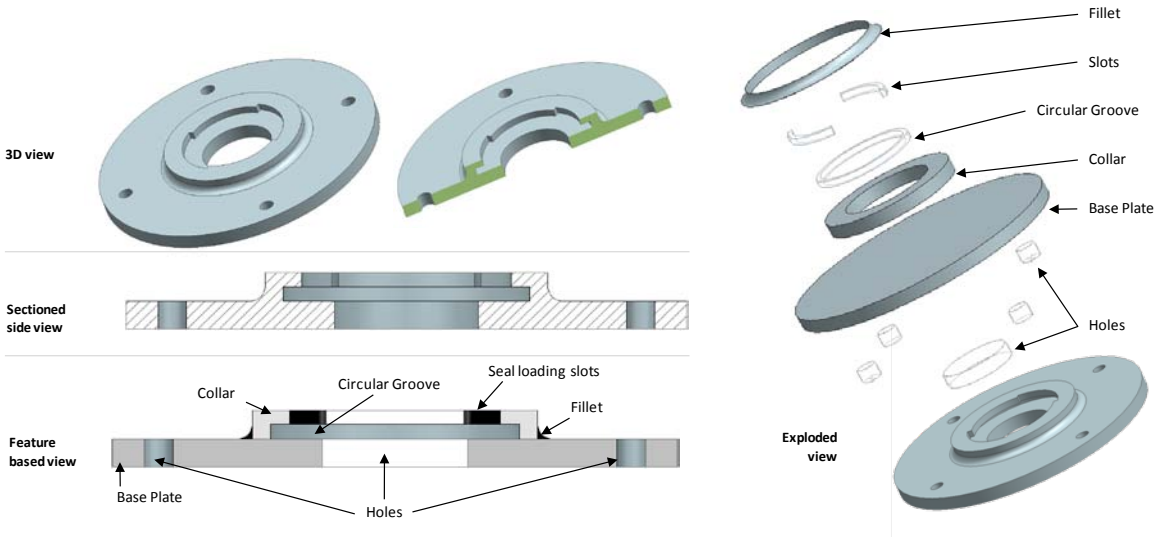


Figure 9.1. A hypothetical component used for the validation of the verification framework

the knowledge frame language is discussed in chapter 5 which is being used to formalize this model. Before the component could be modeled the ontologies containing the building blocks for these models i.e. classes, relations and functions need to be built. Three ontologies were developed for this experiment (1) the foundation and core-concepts ontology, (2) the design domain ontology and (3) the manufacturing domain ontology. The foundation and core-concepts ontology bear all the necessary manufacturing concepts needed for the domain ontologies to take shape. The design domain ontology contains the shape feature concepts from the design point of view. And, finally, the manufacturing domain ontology holds concepts similar to the design ontology but from the manufacturing point of view. Some differences in the structure and content in these ontologies were intentionally implanted to imitate the situation of an actual industrial scenario experienced during the case study explained in chapter 7. The coding of these ontologies can be seen in Appendix II. Concepts in the ontologies were created such that they are enough to build the ontological model of the component shown in figure 9.1. A noticeable feature of the domain ontologies is that the concepts contained by these ontologies bear a link to similar concepts in the foundation ontology. As mentioned earlier, in the case of this research, this link is the subsumption relation i.e. the concepts in the foundation ontology are declared as parents of similar concepts in the domain ontologies. For example, consider a small segment of the design domain ontology taken from Appendix II and shown below.

```
129   :Prop featureOrientationLine
130   :Inst Type
131   :sup orientationCharacteristics
132   :sup FDN.referenceLine
```

It can be seen that the concept ‘featureOrientationLine’ has two subsumption directives one in line 131 and the other in line 132. The first one places this concept within the hierarchy of the manufacturing domain ontology itself while the second one links this concept with an equivalent concept in the foundation and core-concepts ontology. The prefix FDN before the concept ‘referenceLine’ in line 132 denotes the foundation and core-concepts ontology context. ‘FDN.referenceLine’ means that this is a foundation ontology concept and ‘sup’ means that this concept is the parent of ‘featureOrientationLine’. It is this subsumption relation that is being used by the verification mediator to establish similarities between the concepts in two domain ontologies.

9.2.2 Manufacturing knowledge to be shared

In the scenario considered for the validation of the verification framework, as in the previous example, the manufacturing knowledge exists in the manufacturing ontology in the form of manufacturability rules. To develop such a rule, an example is taken from the case study. During the case study it was found that machining tools have a certain life and this limited life has implications for the design of a component. Consider a case where an elongated surface has to be turned in one single turning operation as shown in figure 9.2.

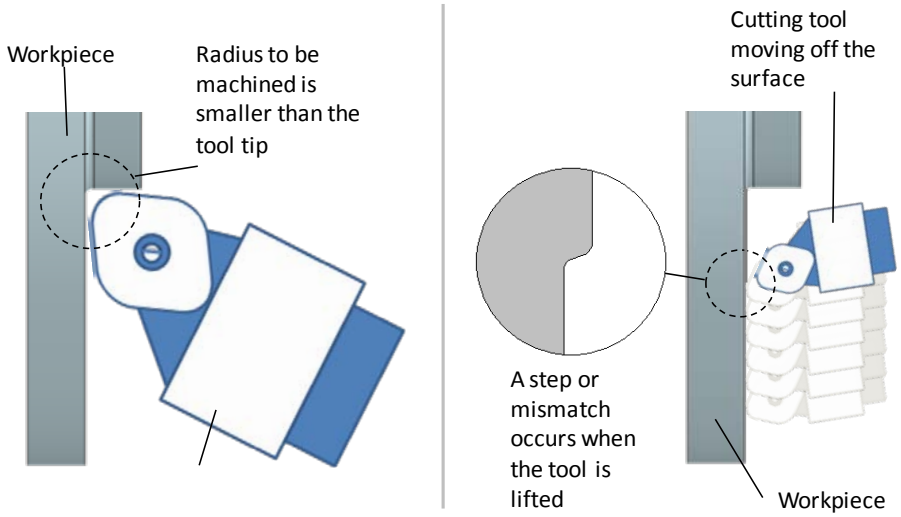


Figure 9.2. Modelled manufacturability problem

Since, the cutting tool tip only has a certain life, the turning operation needs to be interrupted for tool tip re-sharpening or changing. This disruption in the turning operation leaves a mark on the surface of the part being turned. Here the whole surface is considered as a turning feature by the manufacturer while for the designer it contains a base plate and a round (called a fillet by the manufacturers). In this situation a major manufacturability limitation is the overall surface profile tolerance. If the plate arm is too long, the tool will have to be lifted from the surface for sharpening or replacement during the operation which limits the value of surface profile tolerance that can be achieved. If a bigger and heavier tool with longer life is used it limits the value of fillet radius that can be achieved in an uninterrupted single operation. When all of these situations occur simultaneously i.e. a long arm length, a small fillet radius and a small value of overall surface profile tolerance, it becomes impossible to manufacture component.

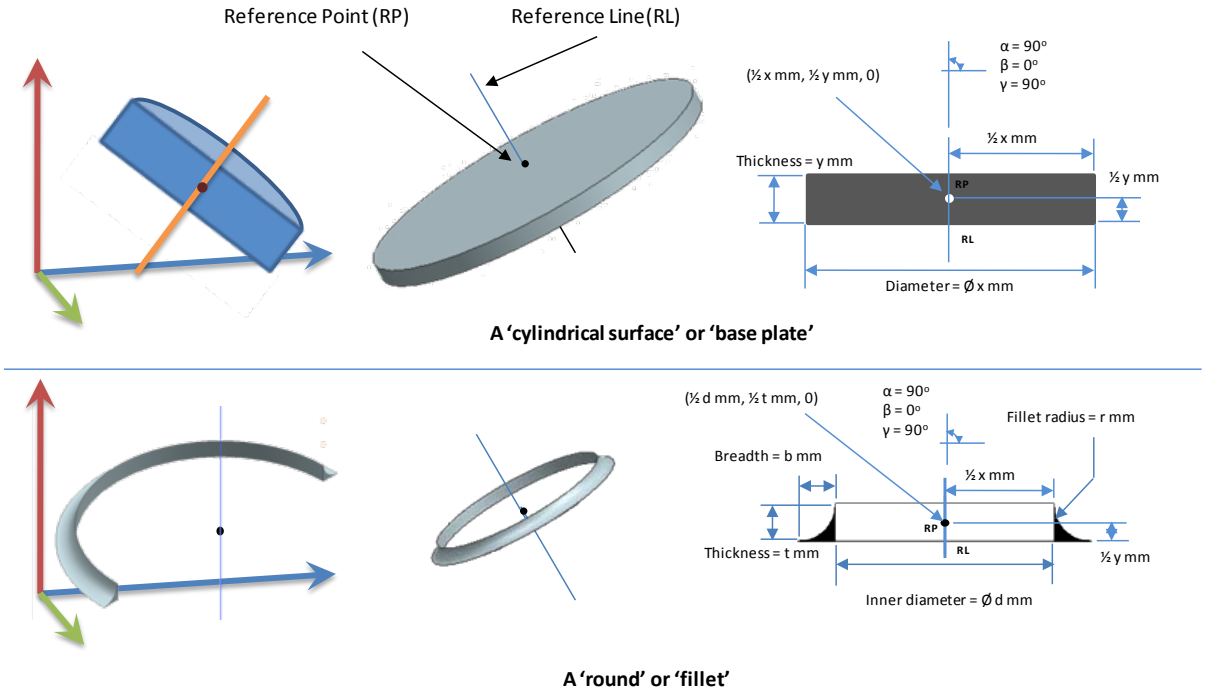


Figure 9.3. Feature dimensional and positional characteristics

Table 9.1 below shows the integrity constraint which defines this manufacturability limitation. Figure 9.3 can be referred to for a description of the different parameters of the two shape features.

Table 9.1. The manufacturability rule

Description	Code
IF statement starts with an AND condition for the declarations which follow.	1 (=> (and
The first condition is that an instance of the class 'cylindrical surface' should exist.	2 (cylindricalSurface ?cs)
These conditions state that for this rule to be valid, the three dimensional characterisitcs of the cylindrical surface should also exist. These three characteristics are diameter, height and profile tolerance.	3 (withExternalDia ?cs (externalDia_mm ?csexd)) 4 (withHeight ?cs (height_mm ?csh)) 5 (withSurfaceProfileTolerance ?cs (surfaceProfile_mm ?csp))
These conditions are showing the requirements of a reference point with its coordinates.	6 (datumPoint ?dp1) 7 (onXpoint ?dp1 (xPoint_mm ?xdp1)) 8 (onYpoint ?dp1 (yPoint_mm ?ydp1)) 9 (onZpoint ?dp1 (zPoint_mm ?zdp1))
These conditions are showing the requirements of a reference line with its angles.	10 (datumLine ?dl1) 11 (withAlpha ?dl1 (alpha_deg ?xdl1)) 12 (withBeta ?dl1 (beta_deg ?ydl1)) 13 (withGamma ?dl1 (gamma_deg ?zdl1))
Says that the instance of the cylindrical surface should have a reference point and line.	14 (hasDatumPoint ?cs ?dp1) 15 (hasDatumLine ?cs ?dl1)

Table 9.1 continued...

Instance of a fillet should also exist with three of it's dimensional characteristics. These include its diameter, radius and height.	16	`(interfaceRound ?ir)`
	17	`(withExternalDia ?ir (externalDia_mm ?irexd))`
	18	`(withInterfaceRoundRadius ?ir (interfaceRoundRadius_mm ?irr))`
	19	`(withHeight ?ir (height_mm ?irh))`
	20	`(withWidth ?ir (width_mm ?irw))`
In the same manner as in case of the cylindrical surface, the fillet should also have a reference point and a reference line with all their coordinates and angles respectively defined. These reference points are assumed to lie in the geometric centre of both of these featues. These assumptions are made when features are formalized in the foundation ontology.	21	`(datumPoint ?dp2)`
	22	`(onXpoint ?dp2 (xPoint_mm ?xdp2))`
	23	`(onYpoint ?dp2 (yPoint_mm ?ydp2))`
	24	`(onZpoint ?dp2 (zPoint_mm ?zdp2))`
	25	`(datumLine ?dl2)`
	26	`(withAlpha ?dl2 (alpha_deg ?xdl2))`
	27	`(withBeta ?dl2 (beta_deg ?ydl2))`
	28	`(withGamma ?dl2 (gamma_deg ?zdl2))`
	29	`(hasDatumPoint ?ir ?dp2)`
	30	`(hasDatumLine ?ir ?dl2)`
For this rule to be valid, the x, y and z angles of the reference lines of the two features should be equal.	31	`(= ?xdl1 ?xdl2) (= ?ydl1 ?ydl2) (= ?zdl1 ?zdl2)`
States that the x and z coordinates of the reference points of two features should also be equal.	32	`(= ?xdp1 ?xdp2) (= ?zdp1 ?zdp2)`
The difference between the diamters of fillet and cylindrical surface is assigned to variable ?result1	33	`(numMinus ?csexd ?irexd ?result1)`
Difference calculated above is greater than 400.	34	`(gtNum ?result1 400)`
Dividing cylindrical surface height by 2.	35	`(numDivide ?csh 2 ?result2)`
Dividing fillet height by 2.	36	`(numDivide ?irh 2 ?result3)`
Adding the two quotients and assignign to ?result4	37	`(numPlus ?result2 ?result3 ?result4)`
Taking the difference between the y coordinates of reference points of the two features.	38	`(numMinus ?ydp2 ?ydp1 ?result5)`
Checking if the addition of heights and difference of y coordinates are equal. An equality here suggests that the fillet and cylindrical surface features touch each other.	39	`(= ?result4 ?result5)`
Checking if the cylindrical surface profileTolerance is less than 0.1.	40	`(ltNum ?csp 0.1)`
For the rule to be valid the two features defined above must belong to a single part. These two lines therefore make sure that they both exist in an instance of a 'part' ?prt.	41	`(workPiece ?wp)`
	42	`(containsFeature ?wp ?cs)`
	43	`(containsFeature ?wp ?ir)`
'IF' statement closes and 'THEN' statement starts.	44	`)`
If all the above conditions are met then the fillet radius (?fr) needs to be greature than 10mm.	45	`(gteNum ?irr 10))`
This statement shows that the above rule is a hard integrity constraint. The statement in quotations appears for the system users to read and understand the manufacturability limitation.	46	`:IC hard "Such a small radius at the disc-collar interface will cause mismatches."`

Lines 1 to 43 in table 9.1 define the inter-feature dependency for which the condition in line 45 should be fulfilled. The first line states that the first condition for this constraint to be true is that a basePlate instance should exist. Lines 2 to 5 add the conditions of the necessary dimensional characteristics of the basePlate instance. Lines 6 to 15 define a reference point and a reference line and associate these positional parameters with the basePlate instance. In the same way lines 16 to 30 define the conditions of the existence of a fillet along with its dimensional and positional characteristics. Up to this point the two features are defined separately. Now in order to make it an integrated turning feature, some positional conditions are to be stated for the constraint to be valid. This is done in lines 31 to 39. If the coordinates of RP1 which is the reference point of the basePlate are set to 0,0,0 and that of the fillet reference point RP2 to 0,25,0 then the two features are aligned, as shown in figure 9.4, provided their reference lines are perpendicular to the x-z plane. If it is fixed in the foundation ontology that the reference points of these features lie exactly in their geometrical centers then for these two features to be perfectly aligned the x and z coordinates of RP1 and RP2 should be equal while the y coordinate of RP2 has to be the sum of half of the height of the basePlate and half of the height of the fillet. This sum comes out to be 25 as shown in figure 9.4 and this is exactly what is being stated in lines 31 to 39. Line 35 declares the condition of base plate profile tolerance to be less than 0.1mm and lines 41 to 43 define the final condition of these two features belonging to a single part

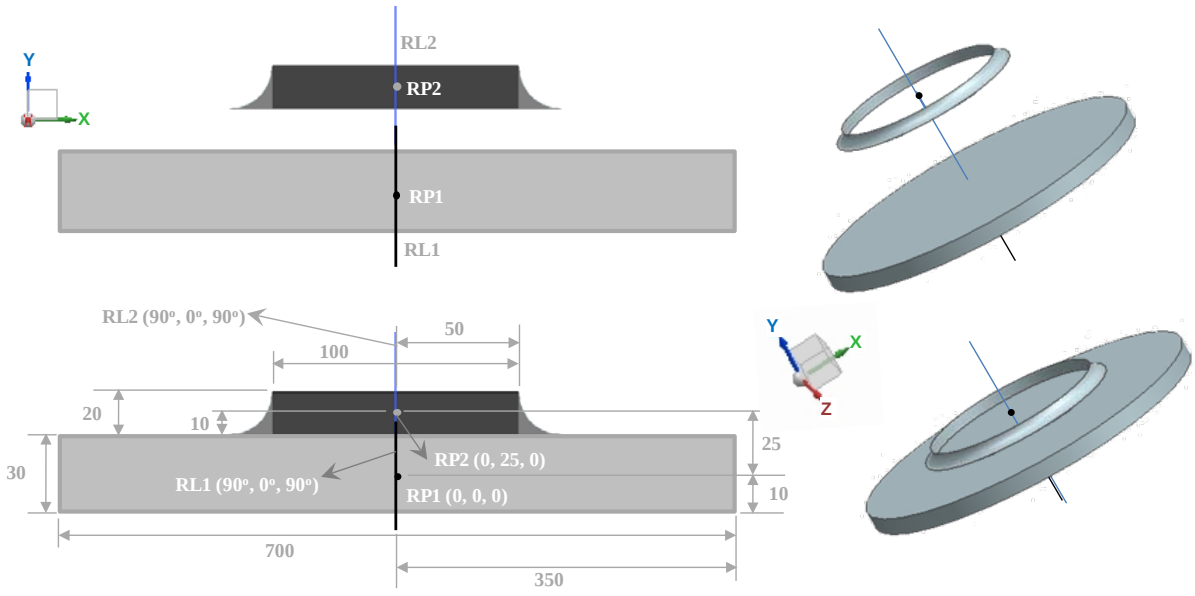


Figure 9.4. Position, orientation and size of the base plate and fillet feature

or component. When all the conditions up to line 44 are met, the system then checks for the fillet radius and fires the integrity constraint in case it falls below 10mm. In an interoperable knowledge sharing system this constraint cautions the designer of the manufacturability limitations.

The rule presented in table 9.1 is in its simplest form. This form, however, requires a large amount of computer memory and in reality needs to be divided into small complementary parts to be easily interpreted and processed. The actual form in which this rule was used in this experiment can be seen at the end of the manufacturing domain ontology in appendix II. After the development of the experimental ontologies and the manufacturability rule, a software application was required capable of automatically conducting the task of knowledge verification. The description of this application or the API is given next.

9.2.3 Design of the Java API

This API or the Application User Interface is specifically developed to just deal with one design scenario but it demonstrates how a more comprehensive software application may work to verify the knowledge being shared which in this case is the checking of the manufacturability of a product being designed. However, the more important thing from this research point of view is the capability of this API to automatically detect similarities across two ontologies. It does this by working on the principles of the verification framework explained earlier. This API is designed to take input from the design user in the form of mathematical values for the geometrical dimensions and positional parameters of a component. Once these values are obtained, it first finds similarities across two ontologies, it then translates the design facts into the manufacturing ontology language and asserts these facts into the manufacturing knowledge base to check their validity and finally sends the result back to the designer. Figure 9.5 gives some snapshots of the main API window and its components.

To explain the working of this API, an example is considered here taken from the case study detailed in chapter 7. Figure 9.4 illustrates the setting in which the two features exist in a component. This situation is used here to validate the working of the verification framework and the API. Following is the description of the main API window and its components.

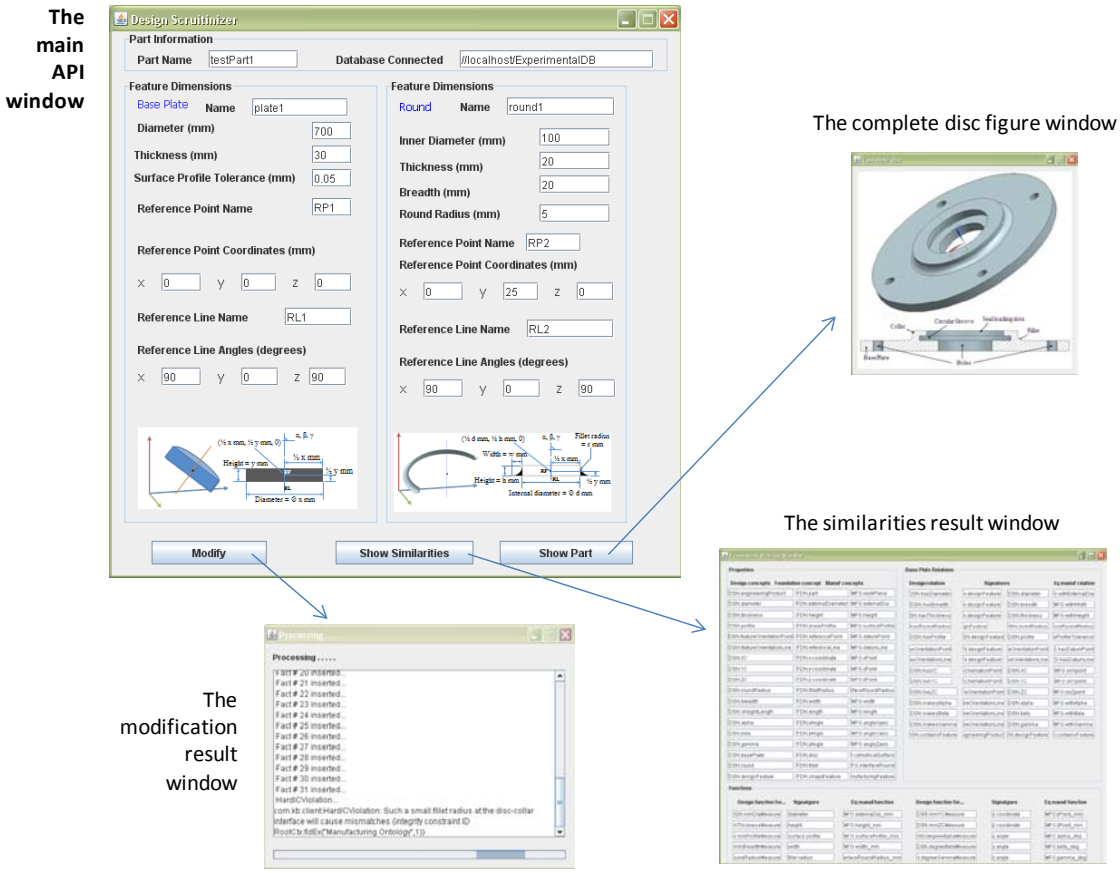


Figure 9.5 The API and its components

9.2.3.1 The main API window

Once the ontologies are available, the concepts contained by them can be used to build the ontological models of the example component. An organized way of doing this is to create individual models of its shape features first and then combine them together to form the whole component. To do this, however, some mathematical values for the geometrical dimensions and positional parameters are needed. The main API window has the task of gathering these values and using them in the knowledge facts created to build the product ontological model. Text boxes can be seen in the main window which are provided to input the information about the part and the knowledge base where its model is to be created. The mathematical values of the dimensional and positional parameters assumed here are shown in figure 9.5. As already mentioned, the API developed for the validation of the verification framework only deals with one design scenario, therefore only two shape features i.e. the base plate (called the cylindrical surface by the manufacturer) and the round (called fillet by the manufacturer) are modeled here. Figure 9.4 depicts the position

of these two features with respect to each other. The main window contains three clickable buttons labeled as ‘Modify’, ‘Show Similarities’ and ‘Show Part’. The description of these three windows comes next.

9.2.3.2 The modification result window

When the button labeled ‘Modify’ is clicked after entering all the required values, the modification result window is launched and the API performs the six steps of knowledge verification as explained in chapter 8. The completion of this process either results in the validation or rejection of the ontological model created though the input values. In the case of validation, the window informs the user of its acceptance and in the case of the rejection, an error message taken from the integrity constraint in the manufacturing ontology is displayed as shown in figure 9.6. This window is also designed to shows the step-by-step progress as can be seen in figure 9.6.

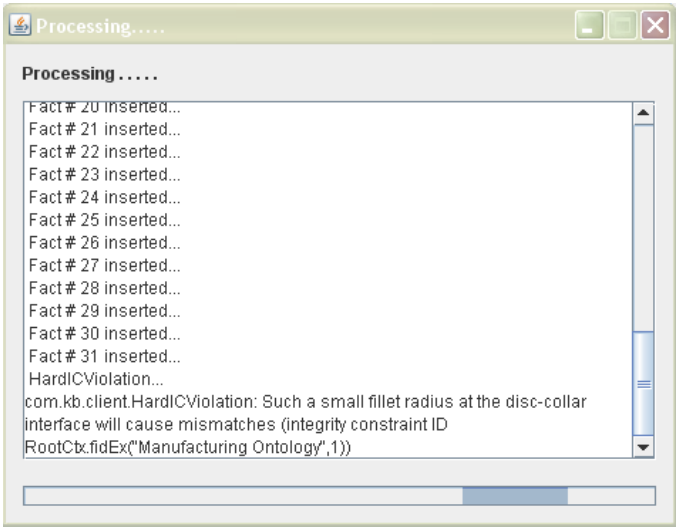


Figure 9.6. The modification result window

9.2.3.3 The similarities results window

The similarities results window can be launched by clicking the ‘Show Similarities’ button in the main window. As it is understood, that design knowledge facts are comprised of certain concepts. For the verification of the shared knowledge, these concepts first need to find their equivalents in the manufacturing ontology. This window shows these equivalents as can be seen in figure 9.7. It was explained in the earlier sections of this chapter that three

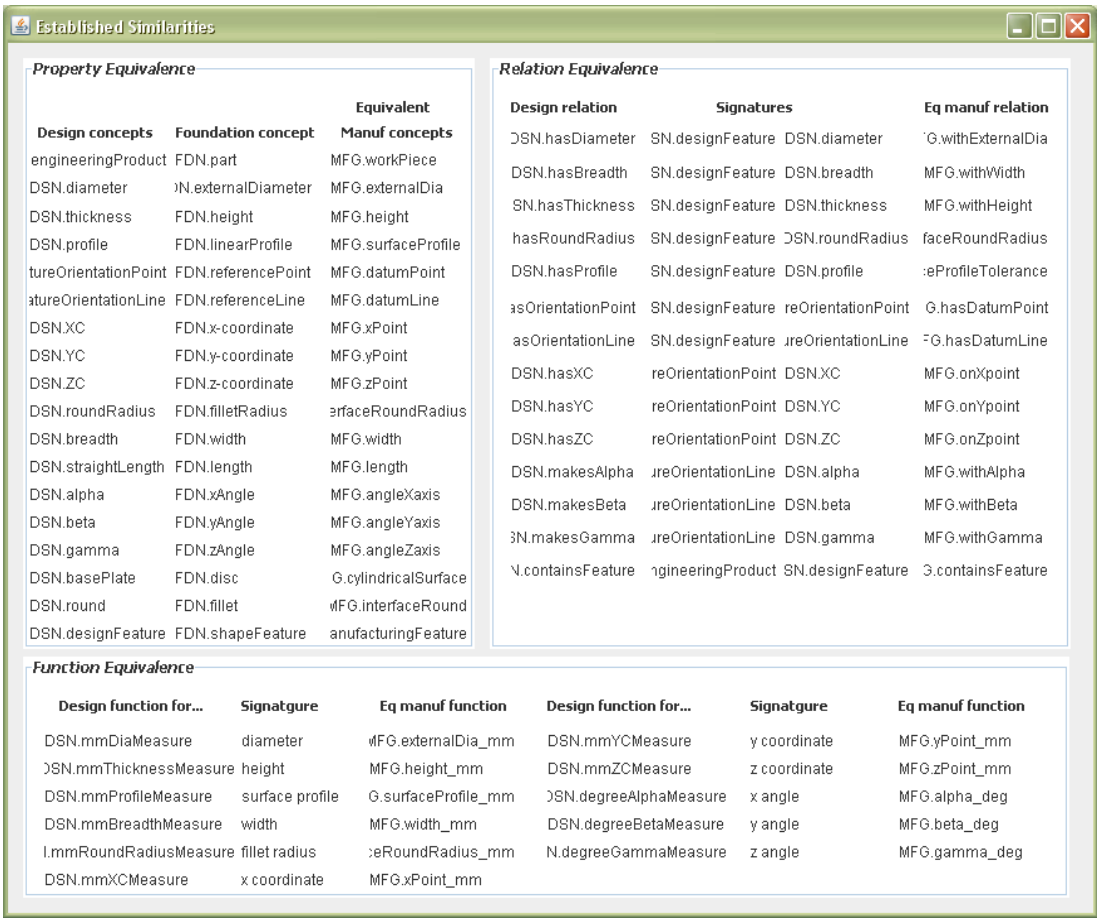


Figure 9.7. The 'Established Similarities' window

types of similarities need to be established i.e. the property, relation and function similarity. This window displays these similarities in separate sections as can be seen in figure 9.7. The ontologies to which these concepts belong can be viewed in appendix II.

Apart from these two windows, the third window which gets launched through the main API is the 'Complete disc' window which is there to just show the complete disc drawing and therefore does not need a separate explanation.

9.3 The validation experiment – functioning of the API

The API is capable of automatically performing all the six steps of knowledge verification detailed in chapter 8. The main code is executed when the 'Modify' button is clicked. The code is written to take the design concepts used to build the intended model of the component being designed. The formalized form of the ontological models of the two shape features is given in tables 9.2 and 9.3.

Table 9.2. Base Plate ontological model

Description	Code
Defining an instance of basePlate named plate1.	(basePlate plate1)
Defining the dimensional parameters of basePlate i.e. height, diameter and surface profile tolerance.	(hasThickness plate1 (mmThicknessMeasure 30)) (hasDiameter plate1 (mmDiaMeasure 700)) (hasProfile plate1 (mmProfileMeasure 0.05)
Defining an instance of referenceLine	(featureOrientationLine RL1)
Defining an instance of referencePoint	(featureOrientationPoint RP1)
Associating the reference point and line with basePlate	(hasOrientationPoint plate1 RP1) (hasOrientationLine plate1 RL1)
Defining the x, y and z coordinates of the reference point. Since this point is now associated with basePlate, defining its coordinates actually positions the basePlate in 3D space. In this case it lies on the origin i.e. 0,0,0.	(hasXC RP1 (mmXCMeasure 0)) (hasYC RP1 (mmYCMeasure 0)) (hasZC RP1 (mmZCMeasure 0))
Defining the angles of the reference line of basePlate and hence orienting the basePlate in 3D space.	(makesAlpha RL1 (degreeAlphaMeasure 90)) (makesBeta RL1 (degreeBetaMeasure 0)) (makesGamma RL1 (degreeGammaMeasure 90))

Table 9.3. Fillet ontological model

Description	Code
Defining an instance of 'round' named round1.	(round round1)
Defining the compulsory dimensional characterisitcs of round1. This includes the thickness, breadth, inner diameter and radius. For a description of these dimensions please refer to figure 9.2b.	(hasThickness round1 (mmThicknessMeasure 20)) (hasBreadth round1 (mmBreadthMeasure 20)) (hasDiameter round1 (mmDiaMeasure 100)) (hasRoundRadius round1 (mmRoundRadiusMeasure 5))
Instantiating a reference point	(featureOrientationPoint RP2)
Instantiating a reference line	(featureOrientationLine RL2)
Associating the instantiated reference point and line with round1.	(hasOrientationPoint round1 RP2) (hasOrientationLine round1 RL2)
Positioning round1 in the 3D space by giving coordinates to its reference point. In this case it exists 25mm above the x-z plane which places it on top of the basePlate defined earlier. Since its x and z coordinates are same as that of the basePlate, it's centre matches with that of the basePlate as shown in figure 9.2a.	(hasXC RP2 (mmXCMeasure 0)) (hasYC RP2 (mmYCMeasure 25)) (hasZC RP2 (mmZCMeasure 0))
These lines orient round1 reference line in the 3D space. Being 90,0,90 it's reference line stays parallel to that of the basePlate as shown in figure 9.2a.	(makesAlpha RL2 (degreeAlphaMeasure 90)) (makesBeta RL2 (degreeBetaMeasure 0)) (makesGamma RL2 (degreeGammaMeasure 90))

Table 9.2 presents the KFL coding with a description of every line for the ontological model of the base plate. It can be seen that first the instance of the base plate is created and then its geometrical dimensional and positional parameters are created and associated with it. Similarly, the ontological model of the 'round' shape feature is presented in table 9.3. It follows the same way of creating the instance of the shape feature itself first, followed by the creation and association of its geometrical dimensional and positional parameters. In addition to the individual feature models above, a part also needs to be created to which these features are then declared to be belonging. This is done in the design ontology terminology as follows:

```
(engineeringProduct wp1)
(containsFeature wp1 plate1)
(containsFeature wp1 round1)
```

If these models are created in the manufacturing knowledge in the existing design terminology, they are not interpretable by the manufacturing ontology and thus will not

Table 9.4. Design and equivalent manufacturing facts for the 'base plate' feature

Line	Design facts	Manufacturing equivalence
1	(basePlate plate1)	(cylindricalSurface plate1)
2	(hasThickness plate1 (mmThicknessMeasure 30))	(withHeight Plate1 (height_mm 20))
3	(hasDiameter plate1 (mmDiaMeasure 700))	(withExternalDia Plate1 (externalDia_mm 700))
4	(hasProfile plate1 (mmProfileMeasure 0.05)	(withSurfaceProfileTolerance Plate1 (surfaceProfile_mm 0.05)) (workPiece wp1)
5	(featureOrientationLine RL1)	(datumLine RL1)
6	(featureOrientationPoint RP1)	(datumPoint RP1)
7	(hasOrientationPoint plate1 RP1)	(hasDatumPoint Plate1 RP1)
8	(hasOrientationLine plate1 RL1)	(hasDatumLine Plate1 RL1)
9	(hasXC RP1 (mmXCMeasure 0))	(onXpoint RP1 (xPoint_mm 0))
10	(hasYC RP1 (mmYCMeasure 0))	(onYpoint RP1 (yPoint_mm 0))
11	(hasZC RP1 (mmZCMeasure 0))	(onZpoint RP1 (zPoint_mm 0))
12	(makesAlpha RL1 (degreeAlphaMeasure 90))	(withAlpha RL1 (alpha_deg 90))
13	(makesBeta RL1 (degreeBetaMeasure 0))	(withBeta RL1 (beta_deg 0))
14	(makesGamma RL1 (degreeGammaMeasure 90))	(withGamma RL1 (gamma_deg 90))

trigger the manufacturability constraint. To verify the manufacturability, therefore, the API finds equivalences across two ontologies, as shown in figure 9.7, and reforms the knowledge facts in the manufacturing ontology language. These translated design facts for the 'base plate' feature can be seen in table 9.4. This translation is based on the equivalence taken from the 'Established Equivalence' window.

Similarly, the translated design facts for the 'fillet' feature are given in table 9.5. Notice that in both of these translations, the values and the instance names do not change. This is done because the designers need to get the result back in the language they are using and by using the same instance names, the error message generated by the manufacturing integrity constraint or rule uses these names to describe the manufacturability limitation.

Table 9.5. Design and equivalent manufacturing facts for the 'fillet' feature

Line	Design facts	Translated manufacturing facts
1	(round round1)	(interfaceRound round1)
2	(hasThickness round1 (mmThicknessMeasure 20))	(withHeight round1 (height_mm 20))
3	(hasBreadth round1 (mmBreadthMeasure 20))	(withWidth round1 (width_mm 20))
4	(hasDiameter round1 (mmDiaMeasure 100))	(withExternalDia round1 (externalDia_mm 100))
5	(hasRoundRadius round1 (mmRoundRadiusMeasure 5))	(withInterfaceRoundRadius round1 (interfaceRoundRadius_mm 5))
6	(featureOrientationPoint RP2)	(datumPoint RP2)
7	(featureOrientationLine RL2)	(datumLine RL2)
8	(hasOrientationPoint round1 RP2)	(hasDatumPoint round1 RP2)
9	(hasOrientationLine round1 RL2)	(hasDatumLine round1 RL2)
10	(hasXC RP2 (mmXCMeasure 0))	(onXpoint RP2 (xPoint_mm 0))
11	(hasYC RP2 (mmYCMeasure 25))	(onYpoint RP2 (yPoint_mm 25))
12	(hasZC RP2 (mmZCMeasure 0))	(onZpoint RP2 (zPoint_mm 0))
13	(makesAlpha RL2 (degreeAlphaMeasure 90))	(withAlpha RL2 (alpha_deg 90))
14	(makesBeta RL2 (degreeBetaMeasure 0))	(withBeta RL2 (beta_deg 0))
15	(makesGamma RL2 (degreeGammaMeasure 90))	(withGamma RL2 (gamma_deg 90))

After the individual feature models translation, the association of these features with a part is translated as follows in table 9.6.

Table 9.6. Design and translated manufacturign facts for feature association

Line	Design facts	Translated manufacturing facts
1	`(engineeringProduct wp1)`	`(workPiece wp1)`
2	`(containsFeature wp1 plate1)`	`(containsFeature wp1 plate1)`
3	`(containsFeature wp1 round1)`	`(containsFeature wp1 round1)`

Once these translations are made, the translated models are asserted in the manufacturing knowledge base to check if they are acceptable. It can be seen that the values of the base plate diameter, the base plate surface profile and the fillet radius being 700mm, 0.05mm and 5mm respectively violate the conditions imposed in the manufacturability rule in table 9.1 and therefore the integrity constraint is triggered. The error message of this integrity constraint is relayed by the API to the designer through the modification result window as shown in figure 9.6.

The API explained in this section is very specific and only deals with the situation it has been designed for i.e the validity of the verification framework proposed in this research. A general application on a bigger scale, however, can be developed using the same principles.

9.4 Discussion and conclusions

This chapter presented the test and validation of the proposed knowledge verification framework which is the main contribution of this research. It was shown that the process of knowledge verification can be made automatic if domain ontologies are aligned with the foundation and core concept ontology in a standard way, compliance with which is ensured through specific axiomatizations. The description of an API specifically developed to validate the working of the verification framework was also given, the successful working of which to resolve semantic differences experienced during the case study, proves that the proposed verification framework is practicable and a useful step towards achieving industrial computer based knowledge systems interoperability.

Before this chapter is concluded, however, some important points need more elaboration. The knowledge verification framework works on certain assumptions. First of all, it is

possible that domain experts decide to directly use the foundation and core-concepts to build their knowledge bases. In these cases, the types of inconsistencies that may occur during concept specialization will be different and the process of knowledge verification proposed in this research will have to be modified accordingly. This verification method, therefore, only deals with the situations where a difference in the use of terminologies exists. This creates the need to have separate domain ontologies before a knowledge base is developed.

Secondly, this verification method only deals with one-to-one correspondences between the foundation and domain concepts. One-to-many and many-to-one correspondences are not interpretable by the verification mediator in its current design state. The inconsistency preventing axiomatizations will need to be modified accordingly to address these other types of correspondences. More on correspondences will be explained in chapter 10. Chapter 10 also looks into certain possible extensions of the proposed framework and what further research is needed to develop them. As a closing remark, therefore, it can be said that this verification framework is a novel contribution and a significant step towards achieving automatic ontology matching for the purpose of developing interoperable computer-based knowledge sharing systems.

Chapter 10: Conclusions and further research

10.1. Chapter overview

This chapter aims to consolidate the findings previously presented in this thesis. The success of the research is analyzed through a brief review of the aims and objectives set in section 2.2. After highlighting the contributions to the field of study, this chapter also looks into some possible further research directions.

10.2. A brief review of research findings

It was found after reviewing the literature that the tools available for ontology matching and reconciliation lack automation and accuracy and require a fair amount of human intervention. A solution to this problem has been devised in a way that it prevents the ontological mismatches from occuring in the first place. This is done by proposing that the domain ontology builders are provided with ways to align their ontologies with a common foundation ontology, thus making it easier for the ontology matching and knowledge verification system to overcome semantic mismatches and thereby find similarities in the two independently developed domain ontologies. Once the domain ontologies are aligned with the common foundation and core concepts ontology, the proposed knowledge verification framework uses these alignments to translate one ontology builder's language into the others and thus make knowledge across the two domains understandable. A brief analysis of the research presented in this thesis is now given so that its success in achieving its goals can be assessed.

10.2.1. Research findings analyzed

In chapter 1, following aims of this research were set:

1. To explore the application of ontology matching and mapping for the verification of knowledge shared between ontology based knowledge bases.
2. To develop an understanding of the application of ontologies for product modelling by using concepts from a library of design and manufacturing concepts i.e. a foundation and core-concepts ontology.
3. To find methods of knowledge verification when manufacturing knowledge associated with ontological product models is shared across domains.
4. To test these methods using IODE as the ontology editor and Common Logic as the ontology development formalism.

Chapter 4 explored the ontology matching and mapping techniques thus fulfilling the first aim. This was done by first reviewing the literature on ontological mismatches that have been identified by the researchers so far. Secondly, those ontology matching techniques were reviewed in detail which have been most frequently quoted in the relevant journal papers. These techniques were then analyzed for their capability to address ontological mismatches and thus verify knowledge that is shared through ontologies. The result of this analysis provided a set of requirements for this research to consider when developing verification methods.

Chapter 6 outlines an ontological product and knowledge modelling technique to meet the requirement mentioned in point 2. This technique is based on shape feature based design and manufacturing and in addition to the development of an understanding of the application of ontologies in product modeling it also provided a way to validate the proposed knowledge verification framework. This methodology on one hand utilizes the benefits of shape feature based design and manufacturing for manufacturability analysis and on the other it makes use of the interoperable nature of ontologies which is useful in cross domain knowledge sharing as is the case in this research.

Chapter 8 explicates the proposed knowledge verification framework to meet the third aim. This explanation first includes the detailed design description of the framework and then the implementation of this framework in a manufacturing scenario is presented. This description clarifies the suitability of the proposed framework for cross domain manufacturing knowledge sharing for the purpose of manufacturability analysis.

Finally, chapter 8 explains and chapter 9 validates this proposed verification framework through a Java API thus fulfilling the last aim. The programming language of Java was learnt for this purpose and a code was written which performs the six steps of knowledge verification and sends the verified responses to the designer as described in chapter 9

This API serves two purposes. Primarily it is used in this research for validating the proposed knowledge verification framework and for showing how this verification can be performed automatically. Secondly, it also provides a simple and user friendly interface for the design engineer to use. This is useful from a manufacturing company point of view because a product designer is usually not familiar with ontologies and knowledge bases and thus this

interface solves this problem and serves as a mediator between the designer and the ontology browsing and handling software which was IODE in the case of this research.

In the light of the above discussion, it can be said that the presented research successfully meets the aims and objectives that were set in the beginning. The contributions this research makes in the field of study are now briefly reviewed.

10.2.2. Contributions to the field of study

The main contribution of this research is the proposed novel knowledge verification framework. This framework is specifically designed to work for a setting in which domain ontologies are committed to a common foundation and are built by using the concepts from a core-concepts ontology. This framework is proposed after a comprehensive study of the available literature and a case study at an aerospace components manufacturing company. The validity of this framework was tested through the development of a prototype user interface and afterwards a more detailed version was used to demonstrate more complex industrial examples, and to verify the design of the proposed framework. The novelty and uniqueness of this work in comparison with the exisiting tools and techniques of ontology mediation and knowledge verification lies firstly in its use of axiomatizations at the foundation and core-concepts level in order to maintain the consistency of domain ontologies, secondly, in its unique use of these consistency principles (laid by the axiomatizations) to mediate between two diverse domain ontologies for knowledge verification, and thirdly, in the unique mechanism of similarity finding. The proposed framework is also a significant step towards increasing the level of automation and accuracy of the process of ontology mediation and knowledge verification. This is evident from the fact that no human intervention was required during the six steps of knowledge verification. There are, however, certain assumptions on which this framework works. These assumptions can be relaxed one by one during the further development of this work. Some possible extentions of this work are discussed later in this chapter.

In addition to the above described contributions, a novel dimensioning and orientation system was also defined to demonstrate the use of ontologies as geometric product models. Being a simple method, it gives a convenient way of representing product models in the form of ontologies. The benefits of these models are twofold. Firstly, being constructed

through shape features, they provide an efficient means of attaching manufacturability knowledge to the shape features. Secondly, being in the form of ontologies, they make it easier to maintain the semantic and syntactic interoperability across the knowledge bases existing in different domains. This work is not the main contribution of this research but nonetheless is a unique step towards achieving a more complete and thorough ontological modelling technique for engineering components.

10.3. Further research

The ontology matching and knowledge verification technique presented in this research utilizes the benefits of a foundation and core-concepts ontology and proves that, during the ontology building process, if the domain concepts are enriched with enough traceability to their origin in the foundation ontology, the techniques of ontology mediation and knowledge verification can be made totally automatic and very accurate. The proposed knowledge verification framework, however, works under certain assumptions and within some defined boundaries. Further research is therefore needed to extend these boundaries and to reduce the number of assumptions the proposed system works under. Some possible research directions in which further development can be done are now given.

10.3.1. Broader specialization and concepts correspondences

A closer look at the proposed knowledge verification technique reveals that the backbone of the whole process is the connection of domain ontology concepts with their parents or origin in the core-concepts ontology. For this research, the parent-child relationship or subsumption is utilized. To make these connections richer and more comprehensive, several different kinds of relationships can be used. For example, a relationship between the domain concepts and core concepts may exist which tells the mediation system that these two concepts are actually similar or that a certain concept is a specialization of a concept in the core concept ontology. If these options are provided in the core concepts and the verification system is designed accordingly the same accuracy and automation can be obtained while making the application of the system broader.

A further extension of this idea is the capability of the system to allow multiple correspondences between the concepts in the foundation and domain ontologies. To further explain this point, take the example of a concept 'hole'. The concept 'hole' in the foundation or core-concepts ontology might get specialized as 'bolt_hole' or as 'straight_hole' in the domain ontology. This is the case of one-to-one correspondence. Other cases may occur where one-to-many or many-to-one correspondences may be required. For example, the same concept 'hole' in the foundation ontology is to be defined as a 'counter_bore' in the domain ontology as shown in figure 10.1. In this case the newly

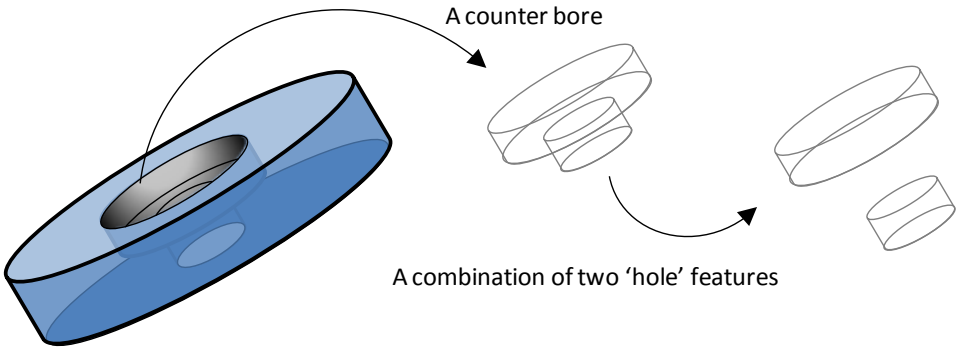


Figure 10.1. A component shown as an aggregation of shape features

defined concept 'counter_bore' is a combination of two 'hole' features. A standard language is needed in these types of cases to empower the domain ontology builders to develop domain concepts by combining two or more concepts from the foundation and core-concepts ontology. This is an example of many-to-one correspondence because more than one shape feature from the foundation and core-concepts ontology are used here to form a single concept of 'counter_bore' in the domain ontology. Similar examples can be taken for one-to-many correspondence where more than one shape features in a domain ontology corresponds to a single concept in the foundation and core-concepts ontology. Further research is therefore needed to develop such a standard vocabulary and an accordingly modified knowledge verification system.

10.3.2. The Verification Meta Ontology (VMO)

The proposed verification framework features a set of inconsistency preventing axiomatizations to control the specialization of concepts in the domain ontology and the

knowledge base building in the knowledge base. One can argue that although these axioms provide a standard way of developing the domain ontologies and knowledge base, they deprive the domain ontology and knowledge base builders of freedom to use the concepts as they want which in turn may stifle the creativity of users. Further development and research, therefore, is needed to develop ways of standardized but with more flexible concept specialization. One such avenue of research can be the development of the Verification Meta Ontology (VMO).

It has been shown in this research that axioms in the form of KFL integrity constraints can be used to dictate the use of concepts from the foundation and core-concepts ontology. Similar options exist in other formalisms for ontology building. The important issue, however, is the way these rules are used. In the work presented here, these rules are used to make sure that the concepts from the core are used in a way which is suitable for the mediation system to function. To make things simpler and more manageable, a rule base in the form of a plug-in ontology can be developed which is in accordance with the needs of the mediation and verification system. This ontology is called the Verification Meta Ontology (VMO). The term 'meta' is used to indicate that this ontology controls the development of domain ontologies at the meta level. More on meta levels can be seen in chapter 3 where MDA and its components are described.

Inspired by the process specification language (PSL), VMO aims to establish a controlled way of specializing ontological concepts from the foundation or core concept ontology. This means a core concepts ontology is built separately without worrying too much about the way concepts are used by the domain ontology builders. Then, to meet the requirements of the verification system, a separate optionally attachable ontology is developed in the form of VMO. This optional attachability of the VMO also relieves the domain ontology builders from unnecessary restrictions on the use of core concepts to model objects. This is because certain levels of verifiability can be introduced in the VMO for the domain ontology builders to choose from. So the more they want their ontology verifiable and the knowledge contained by it to be shareable, the less freedom they are granted in using the concepts in their own way and vice versa. This way, a balance between the flexibility and verifiability of the domain models would be achievable depending upon the personal preference of an

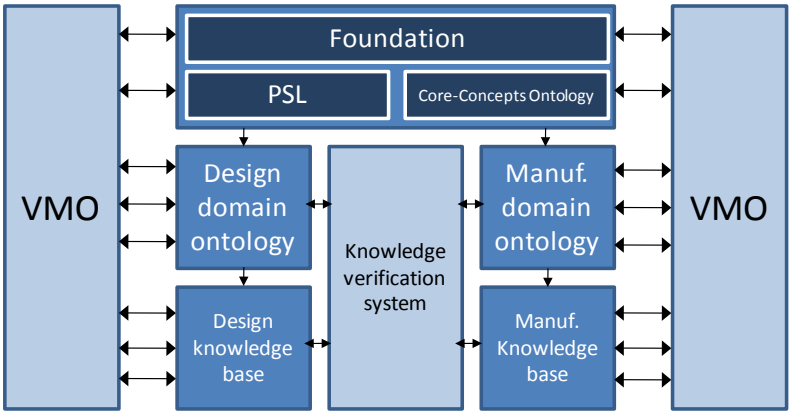


Figure 10.2. Knowledge verification architecture with a verification meta ontology

ontology builder. Figure 10.2 illustrates the way vertical layers of VMO can be used to control the horizontal layers of domain ontology and if needed the knowledge base building.

The VMO has to be built by the ontology mediation and knowledge verification system builders in accordance with the techniques this verification system is going to use to find similarities. It may contain a few classes but mainly a set of rules which govern the use of concepts from the core concept and foundation ontology. In this way when a VMO is loaded with the foundation and core concept ontology in an ontology editor, it cautions the domain ontology and knowledge base builders on the incorrect and untraceable use of core concepts. In a way VMO is a further developed and more flexible form of the inconsistency preventing axiomatizations proposed and demonstrated in this research. A useful further research direction is therefore the development of VMO or a similar rule-base ontology.

10.3.3. Research on exploring the possible inconsistencies

It has already been argued in chapter 8 that no major work in the foundation ontology literature can be found which gives details of the inconsistencies that have been experienced in the domain ontologies developed out of a common foundation. Some research is therefore needed in exploring the possible inconsistencies that domain ontology builders make when developing their ontologies from an available set of concepts from a foundation or core-concepts ontology.

10.4. Closing remarks

The research presented in this thesis brings the current scientific understanding a step closer to developing solutions for interoperable knowledge sharing systems. The

improvement, however, is always continuous and further work, therefore, needs to be done not only in the directions identified in this chapter but in other directions promising useful developments.

Publications

Anjum, N.A., Harding, J.A., Young, R.I.M. and Case, K. Manufacturability verification through feature based ontological product models. Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture. [submitted for review 4 March 2011].

Anjum, N.A., Harding, J.A., Young, R.I.M. and Case, K. Mediation of foundation ontology based knowledge sources. Computers in Industry. [submitted for review 24 September 2010].

Anjum, N.A., Harding, J.A. and Young, R.I.M., 2011. Shape feature based ontological engineering product models. In: 3rd International IFIP Working Conference on Enterprise Interoperability (IWEI). Stockholm, Sweden. March 23-24.

Anjum, N.A., Harding, J.A. and Young, R.I.M., 2010. Cross domain knowledge verification: Verifying knowledge in foundation based domain ontologies. In: Proceedings of the International Conference on Knowledge Engineering and Ontology Development (KEOD) Valencia, Spain. October 25-28.

Anjum, N.A., Harding, J.A., Young, R.I.M. and Case, K., 2010. Gap analysis of ontology mapping tools and techniques. In: Popplewell, K., Harding, J.A, Poler, R. and Chalmeta, R., eds. Enterprise interoperability IV: Making the Internet of the future for the future of enterprise – Proceedings of the 6th International Conference on Interoperability for Enterprise Software and Applications (I-ESA). Coventry, UK. April 14-15. pp. 303-312. DOI: 10.1007/978-1-84996-257-5_28.

Young, R.I.M, Chungoora, N., Usman, Z., Anjum, N.A, Gunendran, G., Palmer, C., Harding, J.A, Case, K. and Cutting-Decelle, A.-F., 2011. Reference ontologies for manufacturing based ecosystems. In: 3rd International IFIP Working Conference on Enterprise Interoperability (IWEI). Stockholm, Sweden. March 23-24.

Young, R.I.M, Chungoora, N., Usman, Z., Anjum, N.A, Gunendran, G., Palmer, C., Harding, J.A, Case, K. and Cutting-Decelle, A.-F., 2010. An exploration of foundation ontologies and verification methods for manufacturing knowledge sharing. In: Workshop on Interoperability for Enterprise Software and Applications (I-ESA). Coventry, UK: University of Coventry. April 13.

References

[Anonymous], 2010. KFL Reference.

Aleksovski, Z., Klein, M., Kate, W.T. and Harmelen, F.V., 2006. Matching Unstructured Vocabularies using a Background Ontology, *In: Proceedings of Knowledge Engineering and Knowledge Management (EKAW,* 2006, Springer-Verlag, pp. 182-197.

Alexiev, V., Breu, M., De Bruijin, J., Fensel, D., Lara, R. and Lausen, H., 2005. *Information Integration with Ontologies: Experiences from an Industrial Showcase.* England: Wiley.

Anjum, N., Harding, J., Young, B. and Case, K., 2010. Gap Analysis of Ontology Mapping Tools and Techniques. *Enterprise Interoperability IV,* , pp. 303-312.

Antoniou, G. and Van Harmelen, F., 2008. *A semantic Web primer.* 2nd edn. Cambridge, Mass.: MIT Press.

Appukuttan, B., Clark, T., Reddy, S., Tratt, L. and Venkatesh, R., 2003. A Pattern based model driven approach to model transformations, *Metamodelling for MDA First International Workshop York, UK, November 2003 Proceedings,* 2003, , pp. 110-128.

Awad, E., M. and Ghaziri, H., M., 2004. *Knowledge Management.* NJ: Prentice Hall.

Bechhofer, S., 2000. *Ontology Language Standardisation Efforts.* IST-2000-29243. Information Management Group, Department of Computer Science, University of Manchester: OntoWeb.

Belaunde, M., Poivre, S. and Dupé, G., 2008. QVT: language, tools and usages.

Bezivin, J. and Gerbe, O., 2001. Towards a Precise Definition of the OMG/MDA Framework, *ASE '01: Proceedings of the 16th IEEE international conference on Automated software engineering,* 2001, IEEE Computer Society, pp. 273.

Bock, C., Zha, X., Suh, H. and Lee, J., 2010. Ontological product modeling for collaborative design. *Adv.Eng.Inform.,* **24**(4), pp. 510-524.

Boehm, ,B.W., 1984. Verifying and Validating Software Requirements and Design Specifications. *IEEE Softw.,* **1**(1), pp. 75-88.

Boyd-graber, J., Fellbaum, C., Osherson, D. and Schapire, R., 2008. Adding dense, weighted connections to wordnet.

Bruijn, J.d., Ehrig, M., Feier, C., Martíns-Recuerda, F., Scharffe, F. and Weiten, M., 2006. Ontology Mediation, Merging, and Aligning. , pp. 95-113.

Catalano, C., Camossi, E., Ferrandes, R., Cheutet, V. and Sevilmis, N., 2009. A product design ontology for enhancing shape processing in design workflows. *Journal of Intelligent Manufacturing,* **20**(5), pp. 553-567.

Chalupsky, H., 2000. OntoMorph: A Translation System for Symbolic Knowledge, *In Principles of Knowledge Representation and Reasoning,* 2000, Morgan Kaufmann, pp. 471-482.

Chandrasekaran, B., Josephson, J.R. and Benjamins, V.R., 1999. What are ontologies, and why do we need them? *Intelligent Systems and their Applications, IEEE,* **14**(1), pp. 20-26.

Changoora, N. and Young, R.I.M., 2010. The configuration of design and manufacture knowledge models from a heavyweight ontological foundation. *International Journal of Production Research,* **25**(1),.

Chungoora, N. and Young, R.I.M., 2008. Ontology Mapping to Support Semantic Interoperability in Product Design and Manufacture.

Coenen, F., Bench-Capon, T., Boswell, R., Dibie-Barthélemy, J., Eaglestone, B., Gerrits, R., Grégoire, E., Lige, C., Za, A., Laita, L., Owoc, M., Sellini, F., Spreeuwenberg, S., Vanthienen, J., Vermesan, A. and Wiratunga, N., 2000. Validation and verification of knowledge-based systems: report on EUROVAV99. *The Knowledge Engineering Review,* **15**(02), pp. 187-196.

Colomb, R.M., Gerber, A. and Lawley, M., 2004. Issues in Mapping Metamodels in the Ontology Development Metamodel, *1st International Workshop on the Model-Driven Semantic Web (MSDW 2004) Monterey, California, USA. 20-24 September, 2004.* 2004, .

Corcho, O. and Gomez-Perez, A., 2000. A Roadmap to Ontology Specification Languages, *EKAW '00: Proceedings of the 12th European Workshop on Knowledge Acquisition, Modeling and Management,* 2000, Springer-Verlag, pp. 80-96.

Corcho, O., Fernandez-Lopez, M. and Gomez-Perez, A., 2003. Methodologies, tools and languages for building ontologies: where is their meeting point? *Data Knowl.Eng.,* **46**(1), pp. 41-64.

Cranefield, S. and Pan, J., 2007. Bridging the gap between the model-driven architecture and ontology engineering. *Int.J.Hum.-Comput.Stud.,* **65**(7), pp. 595-609.

Cycorp Inc., 2009, 2009-last update, Overview of OpenCyc [Homepage of Cycorp Inc.], [Online]. Available: http://www.cyc.com/cyc/opencyc [4/2009, 2009].

Delcam plc, 2010-last update, FeatureCAM, [Homepage of Delcam plc], [Online]. Available: http://www.featurecam.com/ [24/12, 2010].

Delugach, H.S., 2008. Towards Conceptual Structures Interoperability Using Common Logic.

Deng, J., Dong, W., Socher, R., Li, L.-., Li, K. and Fei-Fei, L., 2009. ImageNet: A Large-Scale Hierarchical Image Database, *CVPR09,* 2009, .

Devedzic, V., 2002. Understanding ontological engineering. *Commun.ACM,* **45**(4), pp. 136-144.

Djuric, D., Gasevic, D., Damjanovic, V., Devedzic and V Chang, S.K., 2005a. MDA-Based Ontological Engineering. In: S.K. CHANG, ed, *Handbook of Software Engineering and Knowledge Engineering, Vol.3 - Recent Advances.* World Scientific Publishing Co., Singapore, pp. 203-231.

Djuric, D., Gasevic, D. and Devedzic, V., 2005b. Ontology Modeling and MDA. *Journal of Object Technology,* **4**(1), pp. 109-128.

Doan, ,AnHai, Madhavan, ,Jayant, Domingos, ,Pedro and Halevy, ,Alon, 2002. Learning to map between ontologies on the semantic web, *WWW '02: Proceedings of the 11th international conference on World Wide Web,* 2002, ACM, pp. 662-673.

Draft Federal Information, 1993. Integration Definition for Function Modeling (IDEF0), Federal Information Processing Standards.

Dutra, M., Ghodous, P., Kuhn, O. and Nguyen Minh Tri, , 2010. A Generic and Synchronous Ontology-based Architecture for Collaborative Design. *Concurrent Engineering,* **18**(1), pp. 65-74.

Ehrig, M. and Staab, S., 2004. QOM – Quick Ontology Mapping. pp. 683-697.

Ehrig, M. and Sure, Y., 2004. Ontology Mapping - An Integrated Approach, 2004, Springer Verlag, pp. 76-91.

European Communities, 2004. *European Interoperability Framework v1.* European Communities.

Fensel, D., Horrocks, I., Harmelen, F.v., McGuinness, D.L. and Patel-Schneider, P.F., 2001. OIL: An Ontology Infrastructure for the Semantic Web. *IEEE Intelligent Systems,* **16**(2),.

Fernandez, M., Gomez-Perez, A. and Juristo, N., 1997. METHONTOLOGY: from Ontological Art towards Ontological Engineering, *Proceedings of the AAAI97 Spring Symposium Series on Ontological Engineering,* March 1997, , pp. 33-40.

Frankel, D., Hayes, P., Kendall, E. and McGuinness, D., 2004. The Model Driven Semantic Web, *1st International Workshop on the Model-Driven Semantic Web (MDSW2004) Enabling Knowledge Representation and MDA Technologies to Work Together,* 2004 2004, .

Frankel, D., Hayes, P., Kendall, E. and McGuinness, D.L., 2005. Simple Common Logic: A Constraint Language for the ODM, *First International Workshop on Model-Driven Semantic Web,* 2005, Knowledge Systems Laboratory Stanford University.

Gangemi, A., Guarino, N., Masolo, C., Oltramari, A. and Schneider, L., 2002. Sweetening Ontologies with DOLCE, *13th International Conference on Knowledge Engineering and Knowledge Management (EKAW02),* 2002, , pp. 166-182.

Gangemi, A., Fisseha, F., Keizer, J., Lehmann, J., Liang, A., Pettman, I., Sini, M. and Taconet, M., 2004. A Core Ontology of Fishery and its use in the Fishery Ontology Service Project, *EKAW-CorOnt,* 2004, .

Gardner, T., Griffin, C., Koehler, J. and Hauser, R., 2003. A review of OMG MOF 2.0 Query / Views / Transformations Submissions and Recommendations towards the final Standard, *Metamodelling for MDA First International Workshop York, UK, November 2003 Proceedings,* 2003, .

Gimenez, D.M., Vegetti, M., Leone, H.P. and Henning, G.P., 2008. PRoduct ONTOlogy: Defining product-related concepts for logistics planning activities. *Comput.Ind.,* **59**(2-3), pp. 231-241.

Gomez-Perez, A., Fernandez-Lopez, M. and Corcho, O., 2004. *Ontological Engineering: with example from the areas of Knowledge Manaement, e-Commerce and the Semantic Web.* London: Springer-Verlag.

Gruber, T.R., 1993a. *A translation approach to portable ontology specifications.* Technical Report KSL 92-71. Stanford, California 94305: Knowledge System Laboratory, Computer science depatrment, Stanford University.

Gruber, T.R., 1993b. A translation approach to portable ontology specifications. *Knowl.Acquis.,* **5**(2), pp. 199-220.

Gruber, T.R., 1992. Ontolingua: A Mechanism to Support Portable Ontologies.

Gruninger, M. and Fox, M., 1995. Methodology for the Design and Evaluation of Ontologies, *IJCAI'95, Workshop on Basic Ontological Issues in Knowledge Sharing, April 13, 1995,* 1995, .

Gruninger, M., Atefi, K. and Fox, M.S., 2000. Ontologies to Support Process Integration in Enterprise Engineering. *Computational & Mathematical Organization Theory,* **6**, pp. 381-394(14).

Gruninger, M., 2004. Ontology of the Process Specification Language. In: S. STAAB and R. STUDER, eds, *Handbook on Ontologies.* 1 edn. Germany: Springer, .

Guarino, N., 1998. Some Ontological Principles for Designing Upper Level Lexical Resources, *Proc. of the First International Conference on Lexical Resources and Evaluation,* 1998, , pp. 527-534.

Gupta, U.G., 1993. Validation and verification of knowledge-based systems: A survey. *Applied Intelligence,* **3**(4), pp. 343-363.

Gupter, J.N.D. and Sharma, S.K., 2004. *Normal 0 false false false EN-GB X-NONE X-NONE st1\:*{behavior:url(#ieooui) } Creating knowledge based organizations.* London: Idea Group Publishing.

Hameed, A., Preece, A. and Sleeman, D., 2004. Ontology Reconciliation. In: S. STAAB and R. STUDER, eds, *Handbook on Ontologies.* Springer, pp. 231-250.

Harold, E.R. and Means, W.S., 2004. *XML in a nut shell: A desktop quick reference.* O'Reilly.

Hausmann, J.H., 2003. Metamodeling Relations - Relating metamodels, *Metamodelling for MDA First International Workshop York, UK, November 2003 Proceedings,* 2003 2003, .

Horrocks, I., Fensel, D., Broekstra, J., Decker, S., Erdmann, M., Goble, C., Harlemen, F.v., Klein, M., Staab, S., Studer, R., Motta, E. and Horrocks, I., 2000. The Ontology Inference Layer OIL.

Horváth, I., Vergeest, J.S.M. and Kuczogi, G., 1998. Development and application of design concept ontologies for contextual conceptualization, *Proceedings of ASME DETC,* 1998, , pp. 1-16.

ISO/DIS 10303-224:2003(E), 2003. Product data representation and exchange: Application protocol: Mechanical product definition. .

ISO/IEC 24707:2007(E), 2007. Information technology — Common Logic (CL): a framework for a family of logic based languages.

Jarrar, M. and Meersman, R., 2009. Advances in Web Semantics I. In: T.S. DILLON, E. CHANG, R. MEERSMAN and K. SYCARA, eds, Berlin, Heidelberg: Springer-Verlag, pp. 7-34.

Kalfoglou, ,Yannis and Schorlemmer, ,Marco, 2003. Ontology mapping: the state of the art. *Knowl.Eng.Rev.,* **18**(1), pp. 1-31.

Kaza, ,Siddharth and Chen, ,Hsinchun, 2008. Evaluating ontology mapping techniques: An experiment in public safety information sharing. *Decis.Support Syst.,* **45**(4), pp. 714-728.

Khilwani, N., Harding, J.A. and Choudhary, A.K., 2009. Semantic web in manufacturing. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture,* **223**(7), pp. 905-924.

Klein, M., 2001. Combining and Relating Ontologies: An Analysis of Problems and Solutions.

Li, J., 2004. LOM: A Lexicon-based Ontology Mapping Tool, *Proceedings of the Performance Metrics for Intelligent Systems (PerMIS,* 2004, , pp. 2004.

Li, W.D., Ong, S.K. and Nee, A.Y.C., 2002. Recognizing manufacturing features from a design-by-feature model. *Computer-Aided Design,* **34**(11), pp. 849-868.

Li, W. and Shen, W., 2009. Editorial: Collaborative engineering: From concurrent engineering to enterprise collaboration. *Comput.Ind.,* **60**(6), pp. 365-366.

Li, Z., Raskin, V. and Ramani, K., 2007. A methodology of Engineering Ontology Development for Information Retreaval, 28-31 August, 2007 2007, Purdue University, West Lafayette IN, USA.

Lin, H.K. and Harding, J.A., 2007. A manufacturing system engineering ontology model on the semantic web for inter-enterprise collaboration. *Computers in Industry,* **58**(5), pp. 428-437.

Lin, H.K., Harding, J.A. and Shahbaz, M., 2004. Manufacturing system engineering ontology for semantic interoperability across extended project teams. *International Journal of Production Research,* **42**(24), pp. 5099-5118.

Lourdusamy, R. and Ganapathy, G., 2008. Feature Analysis of Ontology Mediation Tools. *Journal of Computer Science,* **4**(6), pp. 437-446.

Lucanu, ,Dorel, Li, ,Yuan Fang and Dong, ,Jin Song, 2005. Soundness proof of Z semantics of OWL using institutions, *WWW '05: Special interest tracks and posters of the 14th international conference on World Wide Web,* 2005, ACM, pp. 1048-1049.

Ma, Y.S., Tang, S.H., Au, C.K. and Chen, J.Y., 2009. Collaborative feature-based design via operations with a fine-grain product database. *Comput.Ind.,* **60**(6), pp. 381-391.

Maedche, A.,D, 2002. *Ontology learning for the semantic Web.* Boston: Kluwer Academic Publishers.

Maedche, A., Motik, B., Silva, N. and Volz, R., 2002. MAFRA - A MApping FRAmework for Distributed Ontologies, *EKAW '02: Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web,* 2002, Springer-Verlag, pp. 235-250.

Mascardi, V., Locoro, A. and Rosso, P., 2010. Automatic Ontology Matching via Upper Ontologies: A Systematic Evaluation. *IEEE Trans.on Knowl.and Data Eng.,* **22**(5), pp. 609-623.

Mascardi, V., Rosso, P. and CordÃ¬, V., 2008. Enhancing Communication inside Multi-Agent Systems â‹† An Approach based on Alignment via Upper Ontologies.

Masolo, C., Borgo, S., Gangemi, A., Guarino, N., Oltramari, A. and Schneider, L., 2001. *The WonderWeb Library of Foundational Ontologies and the DOLCE ontology.*

Matsokis, A. and Kiritsis, D., 2010. An ontology-based approach for Product Lifecycle Management. *Computers in Industry,* **61**(8), pp. 787-797.

Matuszek, C., Cabral, J., Witbrock, M. and DeOliveira, J., 2006. An Introduction to the Syntax and Content of Cyc. *AAAI Spring Symposium,* .

McGuinness, D.L., 2003. Ontologies come of age. In: SPINNING THE SEMANTIC WEB: BRINGING THE WORLD WIDE WEB TO ITS FULL POTENTIAL, ed, *Fensel,D.;Hendler,J.;Lieberman,H;Wahlster,W.* Cambridge, MA: MIT Press, .

McGuinness, D., Fikes, R., Rice, J. and Wilder, S., 2000. An Environment for Merging and Testing Large Ontologies, *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning (KR-2000),* 2000, .

Meseguer, P. and Preece, A.D., 1996. Assessing the Role of Formal Specifications in Verification and Validation of Knowledge-Based Systems, *Procedings of the Third International Conference on Achieving Quality in Software,* 1996, Chapman & Hall, pp. 317-328.

Miller, J. and Mukerji, J., 2003. *MDA Guide Version 1.0.1.* omg/2003-06-01. Object Management Group.

Mitra, P. and Wiederhold, G., 2002. Resolving Terminological Heterogeneity In Ontologies.

Mizoguchi, R., 2004. Ontology development, tools and languages. *New Generation Computing,* **22**(1), pp. 61-96.

Mizoguchi, R., Vanwelkenhuysen, J. and Ikeda, M., 1995. Task Ontologies for Reuse of Problem Solving Knowledge. In: N. MARS, ed, *Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing.* Netherlands: IOS Press, pp. 46-57.

Neches, R., Fikes, R., Finin, T., Gruber, T., Patil, R., Senator, T. and Swartout, W.R., 1991. Enabling technology for knowledge sharing. *AI Mag.,* **12**(3), pp. 36-56.

Niles, I. and Pease, A., 2001. Towards a standard upper ontology, *FOIS '01: Proceedings of the international conference on Formal Ontology in Information Systems,* 2001, ACM, pp. 2-9.

Niles, I. and Terry, A., 2004. The MILO: A General-purpose, Mid-level Ontology, *IKE,* 2004, , pp. 15-19.

Nirenburg, S. and Raskin, V., 2004. *Ontological Semantics.* Cambridge, Massachusetts, London: The MIT Press.

NIST, 15/4/2008, 2008-last update, Process Specification Language (PSL) [Homepage of NIST], [Online]. Available: http://www.mel.nist.gov/psl/index.html [4/21, 2009].

Nonaka, I., 1994. A Dynamic Theory of Organizational Knowledge Creation. *Organization Science,* **5**(1), pp. pp. 14-37.

Noy, N. and McGuinness, D., 2001. Ontology Development 101: A Guide to Creating Your First Ontology.

Noy, N.F., 2004. Semantic Integration: A Survey Of Ontology-Based Approaches. *SIGMOD Record,* **33**, pp. 2004.

Noy, N.F. and Musen, M.A., 2003. The PROMPT Suite: Interactive Tools for Ontology Merging and Mapping. *International Journal of Human-Computer Studies,* **59**, pp. 2003.

OMG, 2008a. *Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification.* formal/2008-04-03. OMG.

OMG, 2008b. *Ontology Definition Metamodel: OMG Adopted Specification.* ptc/2008-09-07. OMG.

Ouellet, R. and Ogbuji, U., January 30, 2002, 2002-last update, Introduction to DAML: Part I [Homepage of O'Reilly], [Online]. Available: http://www.xml.com/pub/a/2002/01/30/daml1.html [April/20, 2009].

Panetto, H. and Molina, A., 2008. Enterprise integration and interoperability in manufacturing systems: Trends and issues. *Comput.Ind.,* **59**(7), pp. 641-646.

Patil, L., Dutta, D. and Sriram, R., 2005a. Ontology-based exchange of product data semantics. *Automation Science and Engineering, IEEE Transactions on,* **2**(3), pp. 213-225.

Patil, L., Dutta, D., Nistir, R.D.S., Patil, L., Dutta, D., Sriram, R.D., Gutierrez, C.M., Patil, L., Dutta, D. and Sriram, R., 2005b. Ontology formalization of product semantics for Product Lifecycle Management.

Pease, A., Niles, I. and Li, J., 2002. The Suggested Upper Merged Ontology: A Large Ontology for the Semantic Web and its Applications, *In Working Notes of the AAAI-2002 Workshop on Ontologies and the Semantic Web,* 2002, , pp. 2002.

Preece, A., 2001. Evaluating verification and validation methods in knowledge engineering. *Micro-Level Knowledge Management,* **2001**, pp. 123-145.

Pulido, J.R.G., Ruiz, M.A.G., Herrera, R., Cabello, E., Legrand, S. and Elliman, D., 2006. Ontology languages for the semantic web: A never completely updated review. *Know.-Based Syst.,* **19**(7), pp. 489-497.

Qadir, M.A., Fahad, M. and Noshairwan, M.W., 2007. On Conceptualization Mismatches Between Ontologies. *Granular Computing, IEEE International Conference on,* **0**, pp. 275.

Schlenoff, C., Gruninger, M., Tissot, F., Valois, J., Lubell, J. and Lee, J., 2000. The Process Specification Language (PSL) Overview and Version 1.0 Specification. *National Institute of Standards and Technology,* .

Schorlemmer, M. and Kalfoglou, Y., 2005. Progressive ontology alignment for meaning coordination: an information-theoretic foundation, *AAMAS '05: Proceedings of the*

fourth international joint conference on Autonomous agents and multiagent systems, 2005, ACM, pp. 737-744.

Silvert, W., 2001. Modelling as a Discipline. *International Journal of General Systems,* **30**(3), pp. 261.

Smart, P.R. and Engelbrecht, P.C., 2008. An Analysis of the Origin of Ontology Mismatches on the Semantic Web, *EKAW,* 2008, , pp. 120-135.

Staub-French, S., Fischer, M., Kunz, J., Paulson, B. and Ishii, K., 2002. A Feature Ontology to Support Construction Cost Estimating.

Studer, R., Benjamins, V.R. and Fensel, D., 1998. Knowledge engineering: Principles and methods. *Data & Knowledge Engineering,* **25**(1-2), pp. 161-197.

Stumme, G. and Maedche, A., 2001. FCA-Merge: Bottom-Up Merging of Ontologies, *Proceedings of the 7th International Conference on Artificial Intelligence (IJCAI'01),* 2001, , pp. 225-230.

SUMO, 22/4/2009, 2009-last update, Suggested Upper Merged Ontology [Homepage of CIM3.NET], [Online]. Available: http://www.ontologyportal.org/ [4/2009, 2009].

SUO WG, 28/12/2003, 2003-last update, Standard Upper Ontology [Homepage of SUO Working Group], [Online]. Available: http://suo.ieee.org/ [4/22, 2009].

Swartout, B., Ramesh, P., Knight, K. and Russ, T., 1997. Towards Distributed Use of Large Scale Ontologies, A. FARQUHAR, M. GRUNIGER, A. GOMEZ-PEREZ, M. USHOLD and P. VAN-DER-VET, eds. In: *AAAI'97 Spring Symposium on Ontological Engineering,* 1997, , pp. 138-148.

Tursi, A., Panetto, H., Morel, G. and Dassisti, M., 2007. Ontology-Based Products Information Interoperability in Networked Manufacturing Enterprises, IFAC, ed. In: *IFAC Conference on Cost Effective Automation in Networked Product Development and Manufacturing, IFAC-CEA'07; IFAC Conference on Cost Effective Automation in Networked Product Development and Manufacturing , IFAC-CEA'07,* 2007-10-02 2007, Elsevier, pp. CDROM.

Uschold, M. and Gruninger, M., 1996. Ontologies: Principles, methods and applications. *Knowledge Engineering Review,* **11**(2), pp. 93-155.

Uschold, M. and Grüninger, M., 1996. Ontologies: Principles, Methods and Applications. *Knowledge Engineering Review,* **11**, pp. 93-136.

Uschold, M. and Jasper, R., 1999. A Framework for Understanding and Classifying Ontology applications, V.R. BENJAMINS, ed. In: *IJCAI'99 Workshop on Ontology and Problem Solving Methods: Lessons Learned and Future Trends,* 1999, CEUR Workshop Proceedings, pp. 11.1-11.12.

Uschold, M., 1996. Building Ontologies: Towards a Unified Methodology, *In 16th Annual Conf. of the British Computer Society Specialist Group on Expert Systems,* 1996, , pp. 16-18.

van Heijst, G., Schreiber, A.T. and Wielinga, B.J., 1997. Using explicit ontologies in KBS development. *International Journal of Human-Computer Studies,* **46**(2-3), pp. 183-292.

Vegetti, M., Henning, G.P. and Leone, H.P., 2005. Product ontology: definition of an ontology for the complex product modelling domain, *4th Mercosur Congress on Process Systems Engineering,* 2005, .

Visser, P.R.S., Jones, D.M., Bench-Capon, T.J.M. and Shave, M.J.R., 1997. An analysis of ontological mismatches: Heterogeneity versus interoperability, *AAAI 1997 Spring Symposium on Ontological Engineering,* 1997, .

W3C, 2004. *OWL Web Ontology Language Reference: W3C Recommendation 10 February 2004.* ref-20040210. W3C, http://www.w3.org/TR/2004/REC-owl-ref-20040210/.

Wiederhold, G., 1994. An Algebra for Ontology Composition, *In Proceedings of 1994 Monterey Workshop on Formal Methods,* 1994, , pp. 56-61.

WordNet, 2006, 2006-last update [Homepage of Princeton University 2006], [Online]. Available: http://wordnet.princeton.edu/ [4/2009, 2009].

Yang, Q.Z. and Zhang, Y., 2007. Semantic Interoperatbility to Support Collaborative Product Development. In: W.D. LI, S.K. ONG, A.Y.C. NEE and C. MCMOHAN, eds, *Collaborative Product Design and Manufacturing Methodologies and Applications.* Springer, .

Yoo, S.B. and Suh, H.W., 1999. Integrity Validation of Product Data in a Distributed Concurrent Engineering Environment. *Concurrent Engineering,* **7**(3), pp. 201-213.

Young, R., Chungoora, N., Usman, Z., Anjum, N., Gunendran, G., Palmer, C., Harding, J., Case, K. and Cutting-Decelle, A., 2010. An exploration of foundation ontologies and verification methods for manufacturing knowledge sharing, *Workshop on Interoperability for Enterprise Software and Applications (I-ESA Workshop, 2010),* 2010, .

Zack, M., H., 1999. *Knowledge and strategy.* Boston, Mass: Butterworth-Heinemann.

Appendices

Appendix I – Formalized ontologies for figure 8.5

1- Foundation and core-concepts ontology

```
:Name "Example foundation and core-concepts ontology"
:Description "Ontologies for the example illustrated in figure 8.6
in chapter 8.”

:Ctx FDN
:Inst UserContext
:supCtx MLO

:Use FDN


;----------------------------- Properties -------------------------

:Prop ManufacturingFoundation
:Inst Type
:sup Top

      :Prop dimensions
      :Inst Type
      :sup ManufacturingFoundation

            :Prop straightLength
            :Inst Type
            :sup dimensions

            :Prop width
            :Inst Type
            :sup dimensions

            :Prop diameter
            :Inst Type
            :sup dimensions

      :Prop Resources
      :Inst Type
      :sup ManufacturingFoundation

            :Prop Man
            :Inst Type
            :sup Resources

            :Prop Material
            :Inst Type
            :sup Resources


                  :Prop shape_feature
```

```
                  :Inst Type
                  :sup Material

                        :Prop cob
                        :Inst Type
                        :sup shape_feature

                        :Prop hole
                        :Inst Type
                        :sup shape_feature

                        :Prop web
                        :Inst Type
                        :sup shape_feature

                        :Prop rim
                        :Inst Type
                        :sup shape_feature

            :Prop Machine
            :Inst Type
            :sup Resources

      :Prop measures
      :Inst Type
      :sup ManufacturingFoundation

            :Prop millimeter
            :Inst Type
            :sup measures
```

2- Design domain ontology

```
:Name "Example design domain ontology"
:Description "Design domain ontology for the example illustrated in
figure 8.6 in chapter 8.”

:Ctx DSN
:Inst UserContext
:supCtx MLO

:Use DSN


;----------------------------- Properties -------------------------

:Prop measurements
:Inst Type
:sup Event
:sup FDN.dimensions

      :Prop dmtr
      :Inst Type
      :sup measurements

      :Prop straight_length
      :Inst Type
      :sup measurements
      :sup FDN.strightLength

      :Prop breadth
      :Inst Type
      :sup measurements

:Prop component
:Inst Type
:sup Object
:sup FDN.part

:Prop shapes
:Inst Type
:sup ConcreteEntity
:sup FDN.shape_feature

      :Prop disc
      :Inst Type
      :sup shapes

      :Prop bolt_hole
      :Inst Type
      :sup shapes

      :Prop diaphragm
      :Inst Type
```

```
52         :sup shapes
53         :sup FDN.web
54
55         :Prop hub
56         :Inst Type
57         :sup shapes
58
59
60   :--------------------------- Relations --------------------------
61
62   :Rel hasFeature
63   :Inst BinaryRel
64   :Sig component shapes
65
66   :Rel hasLength
67   :Inst BinaryRel
68   :Sig shapes straight_length
69
70   ;------------------------ Functions --------------------------
71
72   :Fun mm
73   :Inst UnaryFun
74   :Sig RealNumber -> straight_lengt
```

3- Manufacturing domain ontology

```
1   :Name "Example design domain ontology"
2   :Description "Design domain ontology for the example illustrated in
3   figure 8.6 in chapter 8.”
4
5   :Ctx DSN
6   :Inst UserContext
7   :supCtx MLO
8
9   :Use DSN
10
11
12  ;----------------------------- Properties -------------------------
13
14  :Prop work_piece
15  :Inst Type
16  :sup Object
17  :sup FDN.part
18
19  :Prop features
20  :Inst Type
21  :sup ConcreteEntity
22  :sup FDN.shape_feature
23
24        :Prop disc_end
25        :Inst Type
26        :sup features
27
28        :Prop straight_hole
29        :Inst Type
30        :sup features
31
32        :Prop webbing
33        :Inst Type
34        :sup features
35        :sup FDN.web
36
37        :Prop centre
38        :Inst Type
39        :sup features
40
41  :Prop attributes
42  :Inst Type
43  :sup ConcreteEntity
44  :sup FDN.dimensions
45
46        :Prop length
47        :Inst Type
48        :sup attributes
49        :sup FDN.straightLength
50
51        :Prop height
```

```
      :Inst Type
      :sup attributes

      :Prop dia
      :Inst Type
      :sup attributes



:--------------------------- Relations --------------------------

:Rel hasAttribute
:Inst BinaryRel
:Sig work_piece features

:Rel withLength
:Inst BinaryRel
:Sig features length

;------------------------ Functions -------------------------

:Fun mils
:Inst UnaryFun
:Sig RealNumber -> length


;------------------------ Rules -----------------------------

(=> (webbing ?w)
      (withLength ?w (mils ?v)))
(lteNum ?v 500))
:IC hard “<code>?w</code> larger than 500mm in length cannot be
produced in the available machines.”
```

Appendix II - The Experimental Ontologies

1- The foundation and core-concepts ontology

```
:Name "Foundation Ontology and Core Concepts Ontology"
:Description "An ontology with foundation and core concepts"

:Ctx FDN
:Inst UserContext
:supCtx MLO

:Use FDN


;----------------------------- Properties -------------------------

:Prop foundationTop
:Inst Type
:sup Top


:Prop dimensions
:Inst Type
:sup foundationTop

     :Prop linearDimensions
     :Inst Type
     :sup dimensions

          :Prop length
          :Inst Type
          :sup linearDimensions

          :Prop externalDiameter
          :Inst Type
          :sup linearDimensions

          :Prop internalDiameter
          :Inst Type
          :sup linearDimensions

          :Prop filletRadius
          :Inst Type
          :sup linearDimensions

          :Prop width
          :Inst Type
          :sup linearDimensions
```

```
45
46              :Prop height
47              :Inst Type
48              :sup linearDimensions
49
50              :Prop coordinates
51              :Inst Type
52              :sup linearDimensions
53
54                   :Prop x-coordinate
55                   :Inst Type
56                   :sup coordinates
57
58                   :Prop y-coordinate
59                   :Inst Type
60                   :sup coordinates
61
62                   :Prop z-coordinate
63                   :Inst Type
64                   :sup coordinates
65
66              :Prop geometricalTolerance
67              :Inst Type
68              :sup linearDimensions
69
70                   :Prop concentricity
71                   :Inst Type
72                   :sup geometricalTolerance
73
74                   :Prop flatness
75                   :Inst Type
76                   :sup geometricalTolerance
77
78                   :Prop linearProfile
79                   :Inst Type
80                   :sup geometricalTolerance
81
82                   :Prop circularProfile
83                   :Inst Type
84                   :sup geometricalTolerance
85
86         :Prop nonLinearDimensions
87         :Inst Type
88         :sup dimensions
89
90              :Prop angle
91              :Inst Type
92              :sup nonLinearDimensions
93
94                   :Prop xAngle
95                   :Inst Type
96                   :sup angle
97
98                   :Prop yAngle
99                   :Inst Type
100                  :sup angle
```

```
101
102                  :Prop zAngle
103                  :Inst Type
104                  :sup angle
105
106  :Prop positionalReferences
107  :Inst Type
108  :sup foundationTop
109
110       :Prop referencePoint
111       :Inst Type
112       :sup positionalReferences
113
114       :Prop referenceLine
115       :Inst Type
116       :sup positionalReferences
117
118  :Prop part
119  :Inst Type
120  :sup foundationTop
121
122       :Prop shapeFeature
123       :Inst Type
124       :sup part
125
126            :Prop baseFeature
127            :Inst Type
128            :sup shapeFeature
129
130                 :Prop disc
131                 :Inst Type
132                 :sup baseFeature
133
134
135            :Prop additionFeature
136            :Inst Type
137            :sup shapeFeature
138
139                 :Prop collar
140                 :Inst Type
141                 :sup additionFeature
142
143                 :Prop fillet
144                 :Inst Type
145                 :sup additionFeature
146
147            :Prop subtractionFeature
148            :Inst Type
149            :sup shapeFeature
150
151                 :Prop hole
152                 :Inst Type
153                 :sup subtractionFeature
154
155
156                 :Prop chamfer
```

```
                    :Inst Type
                    :sup subtractionFeature


;------------------------------  Relations  ----------------------

:Rel hasInternalDiameter
:Inst BinaryRel
:Sig shapeFeature internalDiameter

:Rel hasExternalDiameter
:Inst BinaryRel
:Sig shapeFeature externalDiameter

:Rel hasLength
:Inst BinaryRel
:Sig shapeFeature length

:Rel hasWidth
:Inst BinaryRel
:Sig shapeFeature width

:Rel hasHeight
:Inst BinaryRel
:Sig shapeFeature height

:Rel hasFilletRadius
:Inst BinaryRel
:Sig shapeFeature filletRadius

:Rel isReferenceFeatureFor
:Inst BinaryRel
:Sig shapeFeature part

:Rel hasRefPoint
:Inst BinaryRel
:Sig shapeFeature referencePoint

:Rel hasRefLine
:Inst BinaryRel
:Sig shapeFeature referenceLine

:Rel hasX-coordinate
:Inst BinaryRel
:Sig referencePoint x-coordinate

:Rel hasY-coordinate
:Inst BinaryRel
:Sig referencePoint y-coordinate

:Rel hasZ-coordinate
:Inst BinaryRel
:Sig referencePoint z-coordinate

:Rel hasXangle
```

```
213   :Inst BinaryRel
214   :Sig referenceLine angle
215
216   :Rel hasYangle
217   :Inst BinaryRel
218   :Sig referenceLine angle
219
220   :Rel hasZangle
221   :Inst BinaryRel
222   :Sig referenceLine angle
223
224   :Rel hasFeature
225   :Inst BinaryRel
226   :Sig part shapeFeature
227
228
229
230
231   ;----------------------  Functions  -----------------
232
233   :Fun inDia_MilliMeter
234   :Inst UnaryFun
235   :Sig RealNumber -> internalDiameter
236
237   :Fun height_MilliMeter
238   :Inst UnaryFun
239   :Sig RealNumber -> height
240
241   :Fun width_MilliMeter
242   :Inst UnaryFun
243   :Sig RealNumber -> width
244
245   :Fun filletRadius_MilliMeter
246   :Inst UnaryFun
247   :Sig RealNumber -> filletRadius
248
249   :Fun xCoordinate_MilliMeter
250   :Inst UnaryFun
251   :Sig RealNumber -> x-coordinate
252
253   :Fun yCoordinate_MilliMeter
254   :Inst UnaryFun
255   :Sig RealNumber -> y-coordinate
256
257   :Fun zCoordinate_MilliMeter
258   :Inst UnaryFun
259   :Sig RealNumber -> z-coordinate
260
261   :Fun degree
262   :Inst UnaryFun
263   :Sig RealNumber -> angle
264
265   :Fun radian
266   :Inst UnaryFun
267   :Sig RealNumber -> angle
```

2- The design domain ontology

```
:Name "Design Ontology"
:Description "An ontology with design concepts"

:Ctx DSN
:Inst UserContext
:supCtx MLO

:Use DSN

;------------------------------ Properties -------------------------

:Prop designTop
:Inst Type
:sup Top

     :Prop engineeringProduct
     :Inst Type
     :sup designTop
     :sup FDN.part

          :Prop shapeCharacterisitcs
          :Inst Type
          :sup engineeringProduct

               :Prop designFeature
               :Inst Type
               :sup shapeCharacterisitcs
               :sup FDN.shapeFeature

                    :Prop stressRelievingFeature
                    :Inst Type
                    :sup designFeature

                         :Prop round
                         :Inst Type
                         :sup stressRelievingFeature
                         :sup FDN.fillet

                    :Prop foundationFeature
                    :Inst Type
                    :sup designFeature

                         :Prop basePlate
                         :Inst Type
                         :sup foundationFeature
                         :sup FDN.disc

                    :Prop joiningFeature
                    :Inst Type
                    :sup designFeature

```

```
52
53                              :Prop boltHole
54                              :Inst Type
55                              :sup joiningFeature
56                              :sup FDN.hole
57
58
59
60          :Prop dimensionalCharacteristics
61          :Inst Type
62          :sup designTop
63
64                :Prop straightLength
65                :Inst Type
66                :sup dimensionalCharacteristics
67                :sup FDN.length
68
69                :Prop roundedLength
70                :Inst Type
71                :sup dimensionalCharacteristics
72
73                :Prop diameter
74                :Inst Type
75                :sup dimensionalCharacteristics
76                :sup FDN.externalDiameter
77
78                :Prop internalDia
79                :Inst Type
80                :sup dimensionalCharacteristics
81
82                :Prop thickness
83                :Inst Type
84                :sup dimensionalCharacteristics
85                :sup FDN.height
86
87                :Prop breadth
88                :Inst Type
89                :sup dimensionalCharacteristics
90                :sup FDN.width
91
92                :Prop roundRadius
93                :Inst Type
94                :sup dimensionalCharacteristics
95                :sup FDN.filletRadius
96
97                :Prop tolerances
98                :Inst Type
99                :sup dimensionalCharacteristics
100
101                     :Prop flatness
102                     :Inst Type
103                     :sup tolerances
104                     :sup FDN.flatness
105
106
107
```

```
108
109                  :Prop profile
110                  :Inst Type
111                  :sup tolerances
112                  :sup FDN.linearProfile
113
114                  :Prop concentricity
115                  :Inst Type
116                  :sup tolerances
117                  :sup FDN.concentricity
118
119
120         :Prop orientationCharacteristics
121         :Inst Type
122         :sup designTop
123
124               :Prop featureOrientationPoint
125               :Inst Type
126               :sup orientationCharacteristics
127               :sup FDN.referencePoint
128
129               :Prop featureOrientationLine
130               :Inst Type
131               :sup orientationCharacteristics
132               :sup FDN.referenceLine
133
134               :Prop XC
135               :Inst Type
136               :sup orientationCharacteristics
137               :sup FDN.x-coordinate
138
139               :Prop YC
140               :Inst Type
141               :sup orientationCharacteristics
142               :sup FDN.y-coordinate
143
144               :Prop ZC
145               :Inst Type
146               :sup orientationCharacteristics
147               :sup FDN.z-coordinate
148
149               :Prop alpha
150               :Inst Type
151               :sup orientationCharacteristics
152               :sup FDN.xAngle
153
154               :Prop beta
155               :Inst Type
156               :sup orientationCharacteristics
157               :sup FDN.yAngle
158
159               :Prop gamma
160               :Inst Type
161               :sup orientationCharacteristics
162               :sup FDN.zAngle
163
```

```
164
165
166   ;------------------------- Relations ------------------------------
167
168   :Rel hasDiameter
169   :Inst BinaryRel
170   :Sig designFeature diameter
171
172   :Rel hasInternalDia
173   :Inst BinaryRel
174   :Sig designFeature internalDia
175
176   :Rel hasStraightLength
177   :Inst BinaryRel
178   :Sig designFeature straightLength
179
180   :Rel hasBreadth
181   :Inst BinaryRel
182   :Sig designFeature breadth
183
184   :Rel hasThickness
185   :Inst BinaryRel
186   :Sig designFeature thickness
187
188   :Rel hasRoundRadius
189   :Inst BinaryRel
190   :Sig designFeature roundRadius
191
192   :Rel hasProfile
193   :Inst BinaryRel
194   :Sig designFeature profile
195
196   :Rel isDatumFeatureOf
197   :Inst BinaryRel
198   :Sig designFeature engineeringProduct
199
200   :Rel hasOrientationPoint
201   :Inst BinaryRel
202   :Sig designFeature featureOrientationPoint
203
204   :Rel hasOrientationLine
205   :Inst BinaryRel
206   :Sig designFeature featureOrientationLine
207
208   :Rel hasXC
209   :Inst BinaryRel
210   :Sig featureOrientationPoint XC
211
212   :Rel hasYC
213   :Inst BinaryRel
214   :Sig featureOrientationPoint YC
215
216   :Rel hasZC
217   :Inst BinaryRel
218   :Sig featureOrientationPoint ZC
219
```

```
220
221   :Rel makesAlpha
222   :Inst BinaryRel
223   :Sig featureOrientationLine alpha
224
225   :Rel makesBeta
226   :Inst BinaryRel
227   :Sig featureOrientationLine beta
228
229   :Rel makesGamma
230   :Inst BinaryRel
231   :Sig featureOrientationLine gamma
232
233   :Rel containsFeature
234   :Inst BinaryRel
235   :Sig engineeringProduct designFeature
236
237
238   ;---------------------------  Functions  -------------------------
239
240   :Fun mmDiaMeasure
241   :Inst UnaryFun
242   :Sig RealNumber -> diameter
243
244   :Fun mmInternalDiaMeasure
245   :Inst UnaryFun
246   :Sig RealNumber -> internalDia
247
248   :Fun mmBreadthMeasure
249   :Inst UnaryFun
250   :Sig RealNumber -> breadth
251
252   :Fun mmStraightLengthMeasure
253   :Inst UnaryFun
254   :Sig RealNumber -> straightLength
255
256   :Fun mmThicknessMeasure
257   :Inst UnaryFun
258   :Sig RealNumber -> thickness
259
260   :Fun mmRoundRadiusMeasure
261   :Inst UnaryFun
262   :Sig RealNumber -> roundRadius
263
264   :Fun mmProfileMeasure
265   :Inst UnaryFun
266   :Sig RealNumber -> profile
267
268   :Fun mmXCMeasure
269   :Inst UnaryFun
270   :Sig RealNumber -> XC
271
272   :Fun mmYCMeasure
273   :Inst UnaryFun
274   :Sig RealNumber -> YC
275
```

```
276
277   :Fun mmZCMeasure
278   :Inst UnaryFun
279   :Sig RealNumber -> ZC
280
281   :Fun degreeAlphaMeasure
282   :Inst UnaryFun
283   :Sig RealNumber -> alpha
284
285   :Fun degreeBetaMeasure
286   :Inst UnaryFun
287   :Sig RealNumber -> beta
288
289   :Fun degreeGammaMeasure
290   :Inst UnaryFun
291   :Sig RealNumber -> gamma
```

3- The manufacturing domain ontology

```
:Name "Manufacturing Ontology"
:Description "An ontology with manufacturing concepts"

:Ctx MFG
:Inst UserContext
:supCtx MLO

:Use MFG


;------------------------------ Properties -------------------------

:Prop manufacturingTop
:Inst Type
:sup Top

     :Prop workPiece
     :Inst Type
     :sup manufacturingTop
     :sup FDN.part

          :Prop manufacturingFeature
          :Inst Type
          :sup workPiece
          :sup FDN.shapeFeature

               :Prop millingFeature
               :Inst Type
               :sup manufacturingFeature



                    :Prop straightGroove
                    :Inst Type
                    :sup millingFeature

                    :Prop pocket
                    :Inst Type
                    :sup millingFeature

               :Prop turningFeature
               :Inst Type
               :sup manufacturingFeature

                    :Prop cylindricalSurface
                    :Inst Type
                    :sup turningFeature
                    :sup FDN.disc

                    :Prop interfaceRound
                    :Inst Type
```

```
                    :sup turningFeature
                    :sup FDN.fillet

                    :Prop circularGroove
                    :Inst Type
                    :sup turningFeature

               :Prop drillingFeature
               :Inst Type
               :sup manufacturingFeature

                    :Prop hole
                    :Inst Type
                    :sup drillingFeature
                    :sup FDN.hole


      :Prop featureDefinitionalAttributes
      :Inst Type
      :sup manufacturingTop

           :Prop dimensionalAttributes
           :Inst Type
           :sup featureDefinitionalAttributes

                :Prop length
                :Inst Type
                :sup dimensionalAttributes
                :sup FDN.length

                :Prop width
                :Inst Type
                :sup dimensionalAttributes
                :sup FDN.width



                :Prop height
                :Inst Type
                :sup dimensionalAttributes
                :sup FDN.height

                :Prop externalDia
                :Inst Type
                :sup dimensionalAttributes
                :sup FDN.externalDiameter

                :Prop innerDia
                :Inst Type
                :sup dimensionalAttributes
                :sup FDN.internalDiameter


                :Prop interfaceRoundRadius
                :Inst Type
                :sup dimensionalAttributes
```

```
                    :sup FDN.filletRadius

                    :Prop surfaceProfile
                    :Inst Type
                    :sup dimensionalAttributes
                    :sup FDN.linearProfile

                    :Prop flatness
                    :Inst Type
                    :sup dimensionalAttributes
                    :sup FDN.flatness


               :Prop positionalAttributes
               :Inst Type
               :sup featureDefinitionalAttributes

                    :Prop datumPoint
                    :Inst Type
                    :sup positionalAttributes
                    :sup FDN.referencePoint

                         :Prop xPoint
                         :Inst Type
                         :sup datumPoint
                         :sup FDN.x-coordinate

                         :Prop yPoint
                         :Inst Type
                         :sup datumPoint
                         :sup FDN.y-coordinate

                         :Prop zPoint
                         :Inst Type
                         :sup datumPoint
                         :sup FDN.z-coordinate


                    :Prop datumLine
                    :Inst Type
                    :sup positionalAttributes
                    :sup FDN.referenceLine

                    :Prop orientationAngles
                    :Inst Type
                    :sup positionalAttributes
                    :sup FDN.angle

                         :Prop angleXaxis
                         :Inst Type
                         :sup orientationAngles
                         :sup FDN.xAngle

                         :Prop angleYaxis
                         :Inst Type
                         :sup orientationAngles
```

```
                        :sup FDN.yAngle

                        :Prop angleZaxis
                        :Inst Type
                        :sup orientationAngles
                        :sup FDN.zAngle


;------------------------  Relations  -----------------------

:Rel withInnerDia
:Inst BinaryRel
:Sig manufacturingFeature innerDia

:Rel withExternalDia
:Inst BinaryRel
:Sig manufacturingFeature externalDia

:Rel withLength
:Inst BinaryRel
:Sig manufacturingFeature length

:Rel withWidth
:Inst BinaryRel
:Sig manufacturingFeature width

:Rel withHeight
:Inst BinaryRel
:Sig manufacturingFeature height

:Rel withInterfaceRoundRadius
:Inst BinaryRel
:Sig manufacturingFeature interfaceRoundRadius



:Rel withSurfaceProfileTolerance
:Inst BinaryRel
:Sig manufacturingFeature surfaceProfile

:Rel isDatumFeatureFor
:Inst BinaryRel
:Sig manufacturingFeature workPiece

:Rel hasDatumPoint
:Inst BinaryRel
:Sig manufacturingFeature datumPoint

:Rel hasDatumLine
:Inst BinaryRel
:Sig manufacturingFeature datumLine

:Rel onXpoint
:Inst BinaryRel
```

```
:Sig datumPoint xPoint

:Rel onYpoint
:Inst BinaryRel
:Sig datumPoint yPoint

:Rel onZpoint
:Inst BinaryRel
:Sig datumPoint zPoint

:Rel withAlpha
:Inst BinaryRel
:Sig datumLine angleXaxis

:Rel withBeta
:Inst BinaryRel
:Sig datumLine angleYaxis

:Rel withGamma
:Inst BinaryRel
:Sig datumLine angleZaxis

:Rel containsFeature
:Inst BinaryRel
:Sig workPiece manufacturingFeature



;--------------------  Functions  -----------------

:Fun innerDia_mm
:Inst UnaryFun
:Sig RealNumber -> innerDia

:Fun externalDia_mm
:Inst UnaryFun
:Sig RealNumber -> externalDia

:Fun height_mm
:Inst UnaryFun
:Sig RealNumber -> height

:Fun width_mm
:Inst UnaryFun
:Sig RealNumber -> width

:Fun length_mm
:Inst UnaryFun
:Sig RealNumber -> length



:Fun interfaceRoundRadius_mm
:Inst UnaryFun
:Sig RealNumber -> interfaceRoundRadius
```

```
276
277   :Fun surfaceProfile_mm
278   :Inst UnaryFun
279   :Sig RealNumber -> surfaceProfile
280
281   :Fun xPoint_mm
282   :Inst UnaryFun
283   :Sig RealNumber -> xPoint
284
285   :Fun yPoint_mm
286   :Inst UnaryFun
287   :Sig RealNumber -> yPoint
288
289   :Fun zPoint_mm
290   :Inst UnaryFun
291   :Sig RealNumber -> zPoint
292
293   :Fun alpha_deg
294   :Inst UnaryFun
295   :Sig RealNumber -> angleXaxis
296
297   :Fun beta_deg
298   :Inst UnaryFun
299   :Sig RealNumber -> angleYaxis
300
301   :Fun gamma_deg
302   :Inst UnaryFun
303   :Sig RealNumber -> angleZaxis
304
305
306
307
308   ;------------------------ Rules --------------------
309
310
311   :Rel conditionalRel_csdp
312   :Inst QuinaryRel
313   :Sig cylindricalSurface datumPoint RealNumber RealNumber RealNumber
314
315
316   :Rel conditionalRel_csdl
317   :Inst QuinaryRel
318   :Sig cylindricalSurface datumLine RealNumber RealNumber RealNumber
319
320   :Rel conditionalRel_irdp
321   :Inst QuinaryRel
322   :Sig interfaceRound datumPoint RealNumber RealNumber RealNumber
323
324   :Rel conditionalRel_irdl
325   :Inst QuinaryRel
326   :Sig interfaceRound datumLine RealNumber RealNumber RealNumber
327
328
329   :Rel conditionalRel_csdim
330   :Inst QuaternaryRel
331   :Sig cylindricalSurface RealNumber RealNumber RealNumber
```

```
332
333   :Rel conditionalRel_irdim
334   :Inst QuinaryRel
335   :Sig interfaceRound RealNumber RealNumber RealNumber RealNumber
336
337   :Rel conditionalRel_wp
338   :Inst TernaryRel
339   :Sig workPiece cylindricalSurface interfaceRound
340
341   :Rel conditionalRel_eqNum1
342   :Inst QuaternaryRel
343   :Sig datumPoint datumPoint RealNumber RealNumber
344
345   :Rel conditionalRel_relNum2
346   :Inst QuinaryRel
347   :Sig datumPoint datumPoint RealNumber RealNumber RealNumber
348
349   :Rel conditionalRel_eqNum3
350   :Inst QuaternaryRel
351   :Sig datumPoint datumPoint RealNumber RealNumber
352
353   :Rel conditionalRel_eqNum4
354   :Inst QuaternaryRel
355   :Sig datumLine datumLine RealNumber RealNumber
356
357   :Rel conditionalRel_eqNum5
358   :Inst QuaternaryRel
359   :Sig datumLine datumLine RealNumber RealNumber
360
361   :Rel conditionalRel_eqNum6
362   :Inst QuaternaryRel
363   :Sig datumLine datumLine RealNumber RealNumber
364
365   :Rel conditionalRel_numCalc1
366   :Inst QuinaryRel
367   :Sig cylindricalSurface interfaceRound RealNumber RealNumber
368   RealNumber
369
370   :Rel conditionalRel_numCalc2
371   :Inst TernaryRel
372   :Sig cylindricalSurface RealNumber RealNumber
373
374   :Rel conditionalRel_numCalc3
375   :Inst TernaryRel
376   :Sig interfaceRound RealNumber RealNumber
377
378   :Rel conditionalRel_numCalc4
379   :Inst QuaternaryRel
380   :Sig RealNumber RealNumber RealNumber RealNumber
381
382
383
384
385   (<= (conditionalRel_csdp ?cs ?csdp ?csx ?csy ?csz)
386         (and (datumPoint ?csdp)
387              (onXpoint ?csdp (xPoint_mm ?csx))
```

```
            (onYpoint ?csdp (yPoint_mm ?csy))
            (onZpoint ?csdp (zPoint_mm ?csz))
             (hasDatumPoint ?cs ?csdp)))


(<= (conditionalRel_irdp ?ir ?irdp ?irx ?iry ?irz)
     (and (datumPoint ?irdp)
          (onXpoint ?irdp (xPoint_mm ?irx))
          (onYpoint ?irdp (yPoint_mm ?iry))
          (onZpoint ?irdp (zPoint_mm ?irz))
           (hasDatumPoint ?ir ?irdp)))


(<= (conditionalRel_csdl ?cs ?csdl ?csa ?csb ?csc)
     (and (datumLine ?csdl)
          (withAlpha ?csdl (alpha_deg ?csa))
          (withBeta ?csdl (beta_deg ?csb))
           (withGamma ?csdl (gamma_deg ?csc))
           (hasDatumLine ?cs ?csdl)))


(<= (conditionalRel_irdl ?ir ?irdl ?ira ?irb ?irc)
     (and (datumLine ?irdl)
          (withAlpha ?irdl (alpha_deg ?ira))
          (withBeta ?irdl (beta_deg ?irb))
           (withGamma ?irdl (gamma_deg ?irc))
           (hasDatumLine ?ir ?irdl)))


(<= (conditionalRel_csdim ?cs ?csxd ?csh ?cssp)
     (and (cylindricalSurface ?cs)
          (withExternalDia ?cs (externalDia_mm ?csxd))
          (withHeight ?cs (height_mm ?csh))
          (withSurfaceProfileTolerance ?cs (surfaceProfile_mm
?cssp))
           (ltNum ?cssp 0.1)))


(<= (conditionalRel_irdim ?ir ?irr ?irxd ?irh ?irw)
     (and (interfaceRound ?ir)
           (withInterfaceRoundRadius ?ir (interfaceRoundRadius_mm
?irr))
          (withExternalDia ?ir (externalDia_mm ?irxd))
          (withHeight ?ir (height_mm ?irh))
          (withWidth ?ir (width_mm ?irw))))


(<= (conditionalRel_wp ?wp ?cs ?ir)
     (and (workPiece ?wp)
          (cylindricalSurface ?cs)
```

```
            (interfaceRound ?ir)
            (containsFeature ?wp ?cs)
            (containsFeature ?wp ?ir)))

(<= (conditionalRel_eqNum1 ?csdp ?irdp ?csx ?irx)
     (and (datumPoint ?csdp)
                (datumPoint ?irdp)
                (onXpoint ?csdp (xPoint_mm ?csx))
                (onXpoint ?irdp (xPoint_mm ?irx))
                (eqNum ?csx ?irx)))

(<= (conditionalRel_relNum2 ?csdp ?irdp ?csy ?iry ?yDistance)
     (and (datumPoint ?csdp)
                (datumPoint ?irdp)
                (onYpoint ?csdp (yPoint_mm ?csy))
                (onYpoint ?irdp (yPoint_mm ?iry))
                (numMinus ?iry ?csy ?yDistance)))

(<= (conditionalRel_eqNum3 ?csdp ?irdp ?csz ?irz)
     (and (datumPoint ?csdp)
                (datumPoint ?irdp)
                (onZpoint ?csdp (zPoint_mm ?csz))
                (onZpoint ?irdp (zPoint_mm ?irz))
                (eqNum ?csz ?irz)))

(<= (conditionalRel_eqNum4 ?csdl ?irdl ?csa ?ira)
     (and (datumLine ?csdl)
                (datumLine ?irdl)
                (withAlpha ?csdl (alpha_deg ?csa))
                (withAlpha ?irdl (alpha_deg ?ira))
                (eqNum ?csa ?ira)))

(<= (conditionalRel_eqNum5 ?csdl ?irdl ?csb ?irb)
     (and (datumLine ?csdl)
                (datumLine ?irdl)
                (withBeta ?csdl (beta_deg ?csb))
                (withBeta ?irdl (beta_deg ?irb))
                (eqNum ?csb ?irb)))

(<= (conditionalRel_eqNum6 ?csdl ?irdl ?csc ?irc)
     (and (datumLine ?csdl)
                (datumLine ?irdl)
                (withGamma ?csdl (gamma_deg ?csc))
                (withGamma ?irdl (gamma_deg ?irc))
                (eqNum ?csc ?irc)))

(<= (conditionalRel_numCalc1 ?cs ?ir ?csxd ?irxd ?diaDifference)
     (and (cylindricalSurface ?cs)
                (interfaceRound ?ir)
                (withExternalDia ?cs (externalDia_mm ?csxd))
                (withExternalDia ?ir (externalDia_mm ?irxd))
                (numMinus ?csxd ?irxd ?diaDifference)
                (gtNum ?diaDifference 400)))

(<= (conditionalRel_numCalc2 ?cs ?csh ?CSmidHeight)
     (and (cylindricalSurface ?cs)
```

```
                (withHeight ?cs (height_mm ?csh))
                (numDivide ?csh 2 ?CSmidHeight)))

(<= (conditionalRel_numCalc3 ?ir ?irh ?IRmidHeight)
     (and (interfaceRound ?ir)
                (withHeight ?ir (height_mm ?irh))
                (numDivide ?irh 2 ?IRmidHeight)))

(<= (conditionalRel_numCalc4 ?CSmidHeight ?IRmidHeight ?yDistance
?MPdistance)
     (and (numPlus ?CSmidHeight ?IRmidHeight ?MPdistance)
                (eqNum ?yDistance ?MPdistance)))

(=> (and
                (conditionalRel_csdp ?cs ?csdp ?csx ?csy ?csz)
                (conditionalRel_irdp ?ir ?irdp ?irx ?iry ?irz)
                (conditionalRel_csdl ?cs ?csdl ?csa ?csb ?csc)
                (conditionalRel_irdl ?ir ?irdl ?ira ?irb ?irc)
                (conditionalRel_csdim ?cs ?csxd ?csh ?cssp)
                (conditionalRel_irdim ?ir ?irr ?irxd ?irh ?irw)
                (conditionalRel_wp ?wp ?cs ?ir)
                (conditionalRel_eqNum1 ?csdp ?irdp ?csx ?irx)
                (conditionalRel_relNum2 ?csdp ?irdp ?csy ?iry
?yDistance)
                (conditionalRel_eqNum3 ?csdp ?irdp ?csz ?irz)
                (conditionalRel_eqNum4 ?csdl ?irdl ?csa ?ira)
                (conditionalRel_eqNum5 ?csdl ?irdl ?csb ?irb)
                (conditionalRel_eqNum6 ?csdl ?irdl ?csc ?irc)
                (conditionalRel_numCalc1 ?cs ?ir ?csxd ?irxd
?diaDifference)
                (conditionalRel_numCalc2 ?cs ?csh ?CSmidHeight)
                (conditionalRel_numCalc3 ?ir ?irh ?IRmidHeight)
                (conditionalRel_numCalc4 ?CSmidHeight ?IRmidHeight
?yDistance ?MPdistance)
            )
            (gteNum ?irr 10))

:IC hard "Such a small fillet radius at the disc-collar interface
will cause mismatches
```

Appendix III – The requirements document - Interoperable Manufacturing Knowledge Systems (IMKS)

IMCRC Project:	253
Project:	Interoperable Manufacturing Knowledge Systems (IMKS)
Project deliverable:	2
Deliverable:	System's Requirements Document
Dated	12/05/20 10
Authors	Dr. Bob Young
	Mr. Zahid Usman
	Mr. Najam AkbarAnjum
	Dr. George Gunendran

1--- Introduction

This document provides a description of the user and system requirements for the knowledge sharing system to be developed in the Interoperable Manufacturing Knowledge Systems (IMKS) project. The IMKS project contributes to research in the area of knowledge management software interoperability within the specific area of product design and manufacture, but with a specific focus on manufacturing knowledge sharing. Thus, the requirements document for this project outlines the problems that can be found in sharing manufacturing knowledge across design and manufacture and presents the user needs and system requirements for the resolution of these problems.

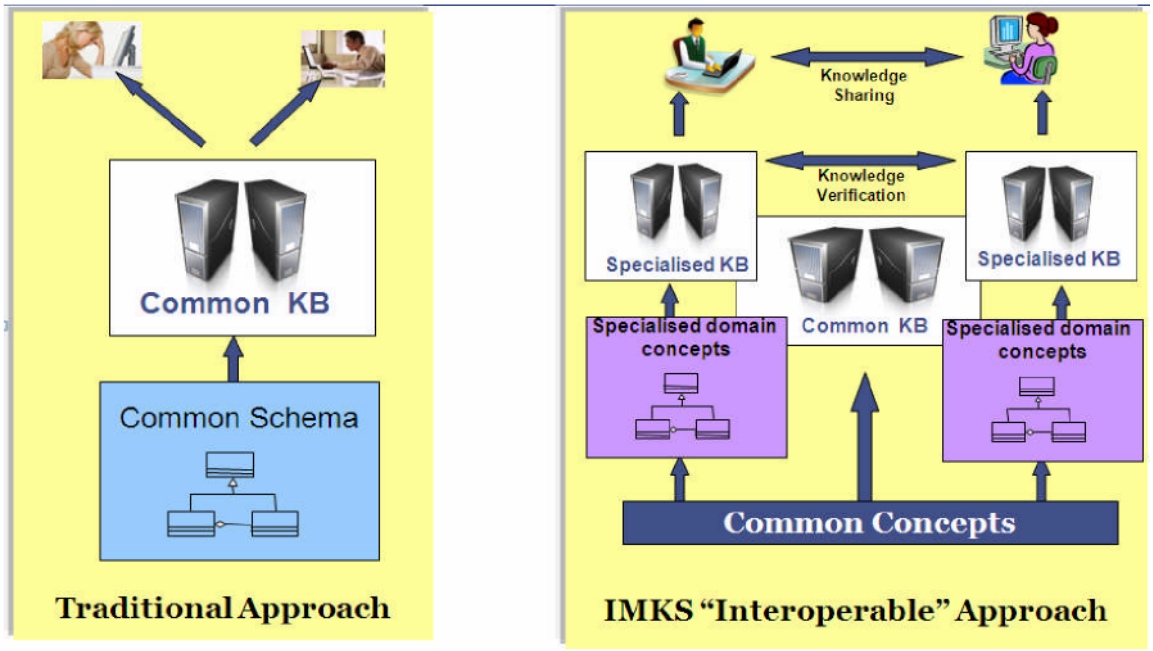
This requirements document is primarily needed to structure the efforts that are going to be made during the development of the interoperable system. It helps in providing a reference document for the research team and potentially for others involved in interoperability research, by explaining the support at which the IMKS experimental system being researched is targeted in terms of both user and system requirements.

A system requirements specification (SyRS) defines the functionality and performance of the system, function partitioning (software/hardware etc) and the user interaction. Some of the functionality can be performed in hardware and some in software. It is the job of the SyRS to define the functionality and performance requirements and to allocate these functions to different disciplines. (IEEE 1998a) A software requirements specification (SRS), on the other hand, addresses the functionality, external interfaces, performance, attributes, design constraints and implementation issues for the software. This document is principally a SRS focused on the conceptual approach of the IMKS project.

1.1--- IMKS Context

To ensure efficient manufacturability, reparability, use and disposal, the designer is required to take into account all the respective lifecycle stages while designing the product. To simplify the complexity of this design methodology, concepts like concurrent engineering have emerged where experts from different domains sit together to mould the design according to their needs. But in a busy industrial environment, people with specialized domain knowledge are not always available to provide their input. As a result, to facilitate the sharing of necessary knowledge and information, computer based knowledge management solutions are being

Figure 1: A comparison of knowledge sharing approaches



researched. Solutions are required for software systems that have the potential to help the designer in complex tasks like design for manufacturing.

Derived from the industrial needs, the top level knowledge management requirement view may comprise of a set of knowledge repositories and a computer based system supporting the dissemination and sharing of knowledge. More specifically an approach is needed to support the containment of knowledge in an organized form and a mechanism to overcome the interoperability problems when knowledge is shared between different domains or knowledge sources. Figure 1 provides a simplified comparison of traditional approaches to systems development and the approach being researched through the IMKS approach.

Within the scope of manufacture, IMKS is targeted specifically at machining knowledge and how this can be shared across the design and manufacturing phases of the product lifecycle. The research thus essentially needs to include:

i. Methods for the formation of sharable specialized design and manufacturing knowledge bases containing commonly used manufacturing concepts and
ii. Verification mechanisms for the authentication of the knowledge being shared between these knowledge bases.

1.2--- Users and Uses

The users of the IMKS system are engineering designers and manufacturing process planners. Designers are in need of assessing the manufacturability of the components they design and manufacturing process planners need to identify the best possible route for the product manufacturing flow. The assumptions made here about the potential users of this system are:

i--- The design users: are assumed to be skilled individuals with some understanding of manufacturing.

ii--- The manufacturing planners: are assumed to have the requisite knowledge of manufacturing planning and an understanding of the available facilities and their capability.

Hence, for the required product quality, the main use of the developed system in design will be to inform the designer of the implications of the decisions in choosing a design solution. These implications will relate to an unacceptably increased cost and time for manufacturing of the product or some technical issues due to the limitations of manufacturing processes. Within the IMKS project the aim from the designer's perspective is be able to inform the designer of situations when the design parameters will require manufacturing to go outside its normal bounds. Similarly the use for a manufacturing process planner is to obtain a guideline in choosing the preferred manufacturing process plan.

2--- User Requirements

Two types of users are discussed in this section: the designer and the manufacturing planner.

2.1---Design User Requirements

Before the design user requirements are outlined, it would be appropriate to first list the typical design objectives. Roughly considered, every design process has an objective of meeting the customer requirements i.e. attaining the intended quality and producing the cheapest and quickest possible manufacturable design using the available facilities. The designer therefore may need the preferred standards, previous design history and the product lifecycle knowledge. Figure 2 illustrates these general high level design requirements.

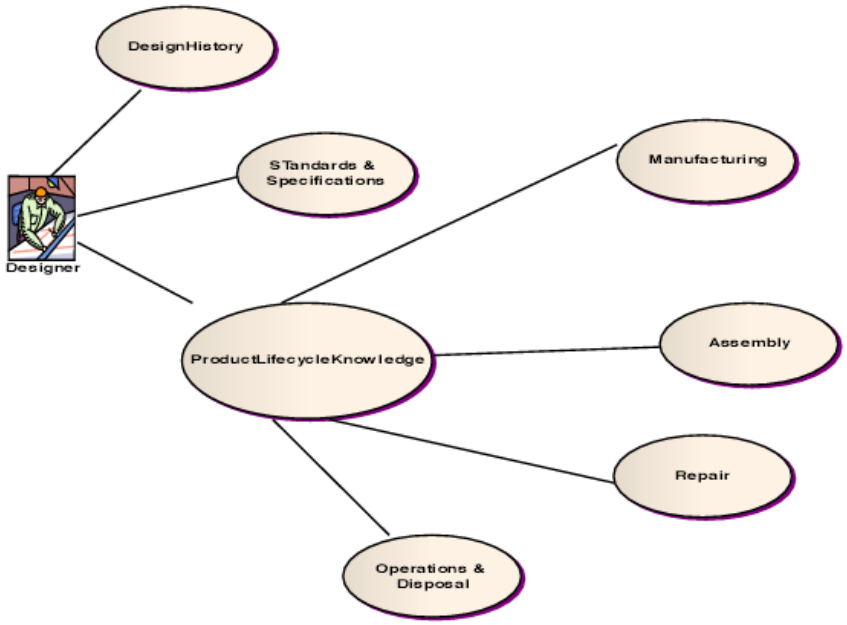


Figure 2: Design User Requirements

2.1.1--- Design History

The design history is needed to prevent the repetition of mistakes and the reinvention of wheel. If the previous history is made available to the designer, the follow up of some earlier design decisions may also lead to manufacturability limitations of the existing facility.

2.1.2--- Standards and Specifications

Standards are needed to ensure that the design meets the environmental, technical and legal requirements both of the company and the user. Some of these standards are possibly dictated by the limitations of manufacturing, assembly, repair, operations and disposal of the product.

2.1.3--- Product Lifecycle Knowledge

To ensure that the designed product is simple to manufacture, assemble and repair, easy to use and its disposal is managed efficiently, the designer needs to be aware of the requirements and limitations of the lifecycle stages. In the IMKS scope, the lifecycle stage of manufacturing and within it machining of the product is important. It is therefore necessary to explore in detail the design user knowledge requirements to understand the machining capabilities and limitations of the manufacturing facility at hand.

2.1.3.1--- Manufacturability Knowledge

The types of manufacturability knowledge that is to be made available to the designer are:

i. Knowledge about the manufacturability of a product.

ii. Archival knowledge which may comprise of best design for manufacture practices and lessons learnt.

The manufacturability understanding requires knowledge of the limitations and feasibility of a design from

a manufacturing standpoint. A designer is therefore in need of the experience and the manufacturing capability understanding possessed by the manufacturer. This knowledge is required during the design process, to ensure that the product is efficiently manufacturable within the available facility.

Although the main aim of every engineering improvement effort is to make the product better and its manufacturing faster and cheaper, the industrial experience shows that most of the time the impediments in the manufacturing of the product are technical in nature. That is to say that the things a designer may need to be aware of, from the manufacturing point of view, are more related to the achievable dimensions and other product parameters than to the business side of the product.

The link between the designer and manufacturing engineer is traditionally an engineering drawing. An engineering drawing can be viewed as on output from the design process which translates the functional needs of the product into manufacturing requirements. These manufacturing requirements include the production of geometry, tolerances, material specification, and the possible addition of production volume. The fulfilment of all these design requirements depends upon the available manufacturing facilities and the knowledge which is in manufacturing engineer's head. When the drawing is studied, a review by the manufacturer may reveal some of the design requirements are difficult or impossible to achieve and this in turn may start an iterative process of design modifications. This design for manufacture iteration could be much faster if the relevant manufacturing knowledge can be conveniently available.

2.2--- Design User Queries

A further way of considering designer requirements is to consider the types of manufacturing related question which a designer may ask during the design process. Typical examples of design queries are

i. Is the designed surface finish achievable on this particular material at the specified angle?

ii. Is the required internal radius obtainable between two surfaces at the specified angle?

iii. Is it possible to obtain the required concentricity between a specific size bore and a cylinder containing that bore?

iv. Is the needed straightness achievable for a thin and long machined surface of the specified material?

v. Is the designed parallelism obtainable between two slots of certain length and width on this specific material?

These queries illustrate design user requirements and give a reference point for the definition of system requirements. Hence the system designed should essentially be capable of answering these sorts of queries.

2.2--- Manufacturing User Requirements

The Manufacturing user is the manufacturing planner.

2.2.1 Manufacturing user queries

As in the case of the designer a useful way of considering requirements is to consider some of the types of queries that a planner might want to answer. The following provides examples of some of the types of queries a manufacturing planner may come up with during the planning process.

ii. What is the likely tool life when machining this material?

iii. What is the Standard Time for machining this feature in this material?

iv. Can this tolerance be achieved on a section this thin?

v. What is the preferred manufacturing method for manufacturing this feature on this kind of part made in this material?

vi. Can a standard fixture be used on this part?

vii. Are there any environmental hazards involved in machining this material?

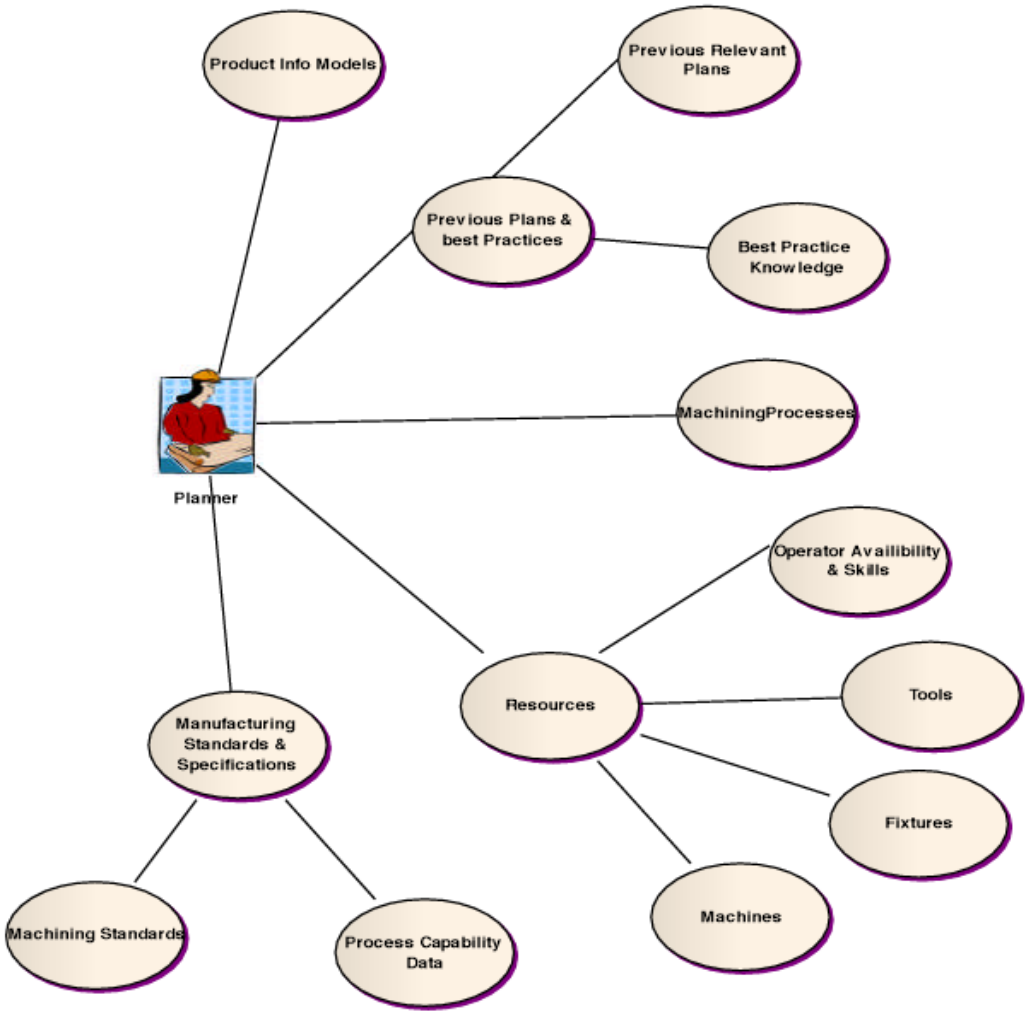


Figure 3: Manufacturing Planner requirements

From these kinds of query the categories of information required by a process planner have been identified in terms of product information, knowledge of previous plans and best practice, knowledge of manufacturing process capability and resource availability plus an understanding of general and company specific standards and specifications. These are shown as a use case diagram in figure 3 below.

2.2.2--- Product Information Models

The first thing which a manufacturing planner needs to start working on the process plan is product information which is normally captured in engineering drawings and 3D models. 3D models provide a complete geometric description while the engineering drawing contains:

i. All the geometric and dimensional tolerance requirements
ii. Material information
iii. Critical features information

2.2.3--- Previous plans & best practices knowledge

The planning activity relies on previous knowledge being applied to a new set of product requirements. This knowledge can relate to (i) previous relevant plans and how they have best been used to produce parts (ii) previous knowledge of how best to produce specific shapes or features given access to specific resources.

A manufacturing planer uses the previous plans as a guide and reference to come with a manufacturing plan. The Information and knowledge about all the best previous plans which suit the present part or can be helpful in planning it should be used. As knowledge about new processes, machines and materials becomes available then there is a requirement to develop an understanding of its applicability and relevance to best practice

2.2.4 Manufacturing Processes

Again the focus of our work is on the machining process so the focus on the requirements of the planner is considered with respect to machining processes.

2.2.4.1 Machining processes

The Planner requires information about all the processes

1. Which can be performed in house in the plant where the machining is to be performed
2. Which are available through potential outsourcing and
3. About any potential new machining processes which might be incorporated.

2.2.5--- Resource Requirement

2.2.5.1--- Operator Availability & Skills Knowledge

Operators' availability and skill level are very important to finalize any manufacturing plans. In a highly dynamic industrial environment particularly in the automobile sector the availability of labour is a key factor which can affect both the production as well as quality of product. It is vital to have the right person at right place at the right time. The Operators' skill levels can is required by the planner in following ways

1. Operator Skill w.r.t available machines
2. Operator skills w.r.t. processes Information on availability of the workers is

required for planning the process.

2.2.5.2--- Machine Availability & Capability

Machines availability, Machine capability are the two key issues for any manufacturing planner in coming up with a rough plan. Machine capability is not about the process capability but is related to the following 1. the size of the parts the machine can hold, 2. the machine's inherent holding device size e.g. chuck or table, 3. the

numbers of cutting tools the machine can have in an automatic tool changer, 4. the number of axes, 5. the type of machine and operations which can be performed.

Availability and capability are important mainly in outlining the overall process and to start detailed planning of specific machines and operations. Machine availability & capability are used against the work piece to be manufactured before finalizing the machine sequence.

Machine availability requirement is only explored in the sense of

1. The existing equipment and machines available in the plant
2. The regular outsourced machines and equipment available.
3. The potential new evolving technology which might be needed for manufacturing new designs.

2.2.5.3--- Cutting Tools and Fixtures Available

These relate directly to the machines as well as processes on machines. They can be listed with respect to machines and processes. Some of the tools or fixtures might be used on more than one machine. Some tools are specially made to meet the design requirement which can be achieved with standard and conventional tooling.

Tool and fixtures availability requirement are of following main types

1. The Tools available in the machine tool magazine & fixtures available with machine
2. The tools or fixtures available in plant store
3. Tools and fixture available as standard market products
4. Tools/fixtures required to be ordered as specialised/customised products.

2.2.6---Standards and Specification

2.2.6.1--- Statistical Process Capability Index

The process capability is required by the planner in finalizing a process in the process plan. It is also used in the detailed process plan of each process. Process capability index Cpk is the measure of process capability and is a dynamic standard which can change with the number of parts being machined and can change as process understanding improves.

2.2.6.2. Machining Standard information

These requirements are of information from general standards like Machining Handbooks, British and ISO standards for machining. These relate to the materials which a tool can machines, tool wear rate, recommended speed, feeds, environmental hazards, coolants to be used, standard procedure and safety measures etc.

2.2.6.3 Company Specific Standards and Specification

Organisations typically have their own standards and specifications established to formalise methods within the company. These requirements are of two types.

1. Standard manufacturing procedures and methods
2. Standard manufacturing information
   Standard manufacturing information is required which represent the limitations of machines and equipment e.g. the angles which can be machined, the minimum dimensions which can be machined an inspected etc.

3. System requirements

Given that the IMKS project is focused on researching manufacturing foundation ontologies and verification methods this section of the document focuses on the requirements in these two areas. The following figure

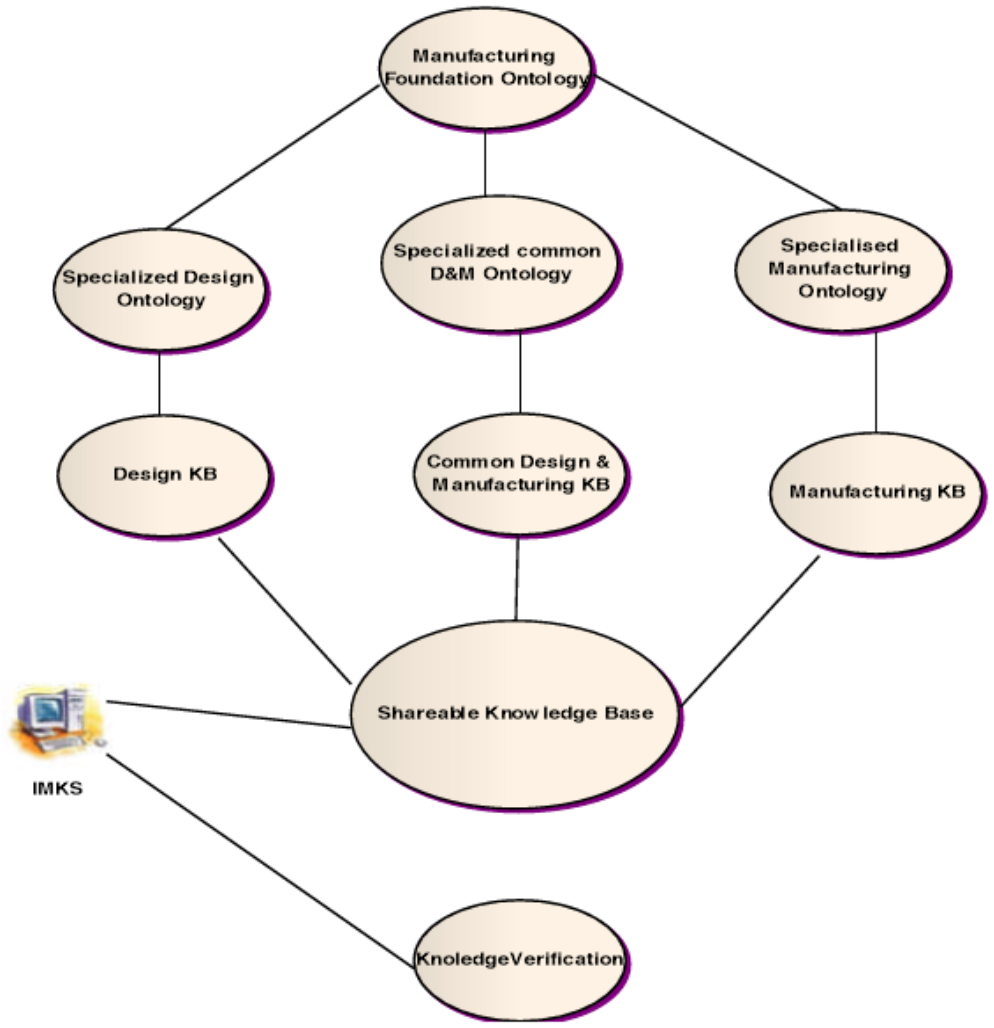


Figure 4: IMKS systems requirements

shows the systems requirements in context and connection of the design and planning user requirements. As the

diagram shows there are two main system requirements

i. Shareable Manufacturing Knowledge Bases

ii. Verification Methods

3.1--- Shareable Manufacturing Knowledge Bases

The IMKS requires to build manufacturing knowledge bases which are sharable across product design and manufacturing enginering activities. In addition to knolwedge there is a requirment to capture information related to the manufacturing facilities being used and the types of product which are to be manufactured.

Other information of relevance would be in the form of Inspection sheets generated from CMM, Process capability, Maintenance and tool wear.

The knowledge which needs to be captured will be in the form of rules & axioms and can be of the following types.

i. Machine availability and capability
ii. Tools & Fixtures availability and capability
iii. Tool accessibility
iv. Achievable Tolerance & surface finish rules
v. Cost association rules with all the various alternatives
vi. Measurement and Inspection rules.

Within IMKS the sharability of design and manufacturing concepts is being researched through the exploration of sets of manufacturing foundation ontology concepts as described in the next section.

3.1.1--- Manufacturing Foundation Ontology

The KBs need a foundation through which they can interoperate and share. This common foundation requirement is met by using set of common concepts, relations, function and axioms defined through the development of heavyweight ontology. Ontology is a lexicon of specialised terminology along with some specification of the meanings of the terms involved (ISO---18629---1) while the IMKS project requires a common set of manufacturing terms which can be specialised to suit the needs of both product design and manufacturing planning.

There is a requirement to identify an ontology to suit the needs of product design and an ontology to suits the needs of manufacturing planning, both based upon a common manufacturing foundation ontology. It is anticipated that some terms at eh foundation level may also be commonly used outside of the manufacturing domain. The requirement is therefore to identify a structured approach to the definition of manufacturing ontologies which can be used to support the development of knowledge bases which have the potential to be assessed in terms of the level to which their knowledge can be shared. shared The manufacturing ontology will require concepts, relations, functions and axioms at varying level of specialization from being very generic to very specific. For example “Feature” is a concept which is generic but It can be specialised in its meaning for a “product Feature” and further specialised as “Design feature” and “Manufacturing Feature”.

A broad range of concepts need to be formalised within the ontology and includes manufacturing terms typically used in industry such as the following list of concepts from our academic & industrial exploration.

Some Concepts from Academic and Industrial Exploration

MfgPlan Part PartFamily MfgPartFamily DesignPartFamily Standard Specificcation Material Resource Operation Inspection InspectionHistory BatchCard Batch CuttingTool Setup Fixture Quality Concession Wear Life CriticalFeature StandardFeature MfgFeature DesignFeature Dimension Tolerance SurfaceFinish Function Stress Thermal Analysis MachineTool Turning TwinTurning Cob Rim Web Diaphragm Hub PartProgram Forge Model MasterModel EnggDwg WPMaterial ToolMaterial CAPP CriticalPartsPlan FixingProcess MASBuyoff Commodity ReqDoc FunctionalDesignInfo Engine Turbine Compressor Pressure HorsePower Fan Blade Slot Groove CircumGroove FirtreeSlot LoadingSlot Locking Slot DefenderSlot InspectionHistroryCard OperatorInstruction InspectionOperatorCard ToolOrder ConditionofSupply NumberofBlades Force WorkingLoad Load Impact Load 2DFeature 3DFeature Axis Measure ProcessCapability Cpk SurfaceTreatment Etching BalanceLand BenchInspection Milling Drilling BinocularInspection Washing ShotPeening NonDestructiveTesting DwgChangeRequest MfgDesignBuyoff Interface Speed Feed DepthofCut Repeatability Roughness StandaloneDisc Drum SpigotEdge ConnectingArm Welding BoltHoles Fins ToolAccess ToolAvailibility WeldArm InternalRadius CMMInspection

CMMProgram ManufacturingKNowledge DesignKnowledge Governance SubSystemreq EnggCangeRequest DimensionalCharacteriticTable ManufacturingEngr OperationDwg TargetDate Tempearure Pressure Humidity ProductionPlan Equipment ProcessOwner CommodityOwner MachiningPlan InspectionPlan Stock Greasing Degreasing Capacity NewKNowledge Validation Balancing Definition DetailCell Cell

3.2--- Verification Methods

One of the important considerations in the development of an interoperable system is a knowledge sharing mechanism. This mechanism should be capable of facilitating the sharing of knowledge between diverse domains without any misinterpretation. Furthermore, this sharing has to be done by using different knowledge bases developed out of a common set of concepts as explained earlier in the IMKS context section of this document. This may give rise to the problem of interoperability between the two knowledge bases and ontologies. This is because although the ontologies are developed from the same set of concepts, they may have a different architecture for concepts definition and knowledge organization.

To prevent any misinterpretation of concepts, verification methods are needed which certify the understanding of a concept being shared in a certain context. A major system requirement is therefore a mechanism which authenticates the veracity of similarities in two ontologies. Within the IMKS scope, two domains selected are design and manufacturing. The verification system is therefore required to be specially effective in detecting and resolving any misunderstandings in the interpretation of concepts on manufacturing

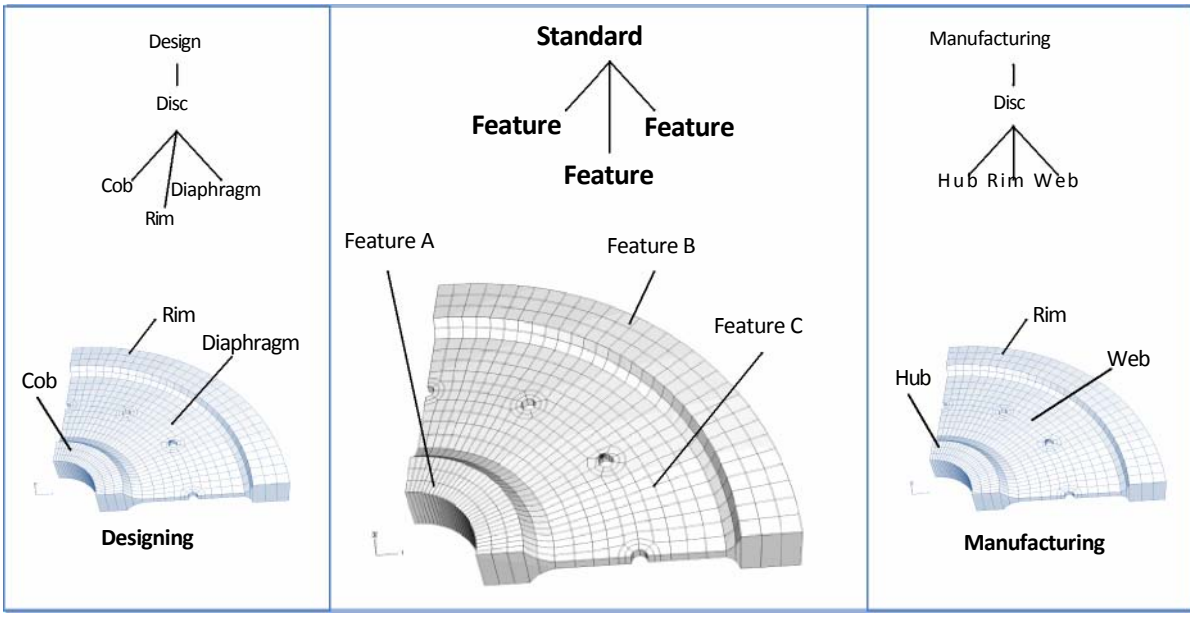


Figure 5: Design and Manufacturing differences in terminologies and a need for a common set of concepts

or design side.

Experience from the industry shows that different terminologies are used in design and manufacturing domains to refer to the same thing. For example, to refer to the portion of a disc connecting the outer and inner edges, designers might use the term *diaphragm* while on the shop floor, the word *web* is used for the same part of the component. Similarly the terms of cob and hub are used to refer to the same feature in design and manufacturing respectively as shown in figure 2. Another possibility is of different features having similar names across the two domains. Apart from these terminological mismatches, there can be certain concepts which are referred to in a particular domain in a specific way. For example, designers may name different features from the functional point of view while the manufacturer may categorize them and refer to them from the machine tool or manufacturing method perspective. So there can be a conceptual difference in

the interpretation of things across the two domains. Due to these problems, if a designer wants to get some knowledge about the manufacturing method of a certain part of the component, the knowledge system might not be able to find a match on the manufacturing side unless similar terms or concepts used in two domains are mapped. A requirement is therefore to have a set of common concepts, as shown in figure 2, to which the concepts in design and manufacturing can be mapped. It is to be noted that present mapping technologies are not fully automatic and a manual input is needed through human intervention. Since manual mapping is a very cumbersome and time consuming process, there needs to be a way through which this mapping and later verification are done automatically. When the mapping is attempted to be made automatic, the next hurdle to overcome is the problem of mismatches. This can happen when during the similarity finding stage, the knowledge system interprets two different concepts in two ontologies to be similar and vice versa.

Reference:

IEEE, 1998a. IEEE guide for developing system requirements specifications. IEEE,

1998b. IEEE recommended practice for software requirements specifications.

ISO 18629---1, ISO TC184/SC4/JWG8, 2004, Industrial Automation System and Integration—Process Specification Language: Part 1. Overview and Basic Principles.

ISO (CEN/ISO)11354 ,V8.3, (Dec, 2008), Requirements for establishing manufacturing enterprise process interoperability, PART I:Framework for Enterprise Interoperability.

Young, R.I.M., Harding, J.A., Case, K., 2009, Interoperable Manufacturing Knowledge Systems (IMKS)---State of the Art Review: Ontological Formalisms and Model Driven Approaches