Capture and Maintenance of Constraints in Engineering Design

A thesis presented for the degree of

Doctor of Philosophy

at the University of Aberdeen

by

Suraj Ajit

(B. E. Computer Science and Engineering, Bangalore University, India)

Department of Computing Science

University of Aberdeen

United Kingdom

2009

Abstract

The Designers' Workbench is a system, developed by previous research to support designers in large organizations, such as Rolls-Royce, to ensure that the design is consistent with the specification for the particular design, as well as with the company's design rule book(s). The evolving design is described against a jet engine ontology. Design rules are expressed as constraints over the domain ontology. To capture the constraint information, a domain expert (design engineer) has to work with the knowledge engineer to identify the constraints, and it is then the task of the knowledge engineer to encode these into the Workbench's knowledge base. This is an error-prone and time-consuming task. It is highly desirable to relieve the knowledge engineer of this task, and so this thesis proposes a novel approach to facilitate domain experts in capturing and maintaining constraints. The approach has been embodied by developing a system, ConEditor that facilitates domain experts in combining selected entities from the domain ontology with keywords and operators of a constraint language to form a constraint expression. Further, this thesis reports that in order to appropriately apply, maintain and reuse constraints, it is important to know the assumptions and context in which each constraint is applicable. This is referred to as the "application condition" and this forms a part of the rationale associated with the constraint. The central hypothesis of this thesis is that an explicit representation of constraints together with the corresponding application conditions and the appropriate domain ontology can be used to support the maintenance of constraints. The thesis investigates two domains, initially the kite domain and then part of a more demanding Rolls-Royce domain (jet engine design). Four main types of refinement rules that use the associated application conditions and domain ontology to support the maintenance of constraints are proposed. The refinement rules have been implemented in ConEditor and the extended system is known as ConEditor+. With the help of ConEditor+, the thesis demonstrates that an explicit representation of application conditions together with the corresponding constraints and the domain ontology can be used to detect inconsistencies, redundancy, subsumption and fusion, reduce the number of spurious inconsistencies and prevent the identification of inappropriate refinements of redundancy, subsumption and fusion between pairs of constraints.

Notes

Parts of the research work reported in this thesis have been published previously in:

Book Chapter:

- Ajit, S, Sleeman, D, Fowler, D.W., Knott, D and Hui, K (2005): Capture and Maintenance of Engineering Design Constraints, Advanced Knowledge Technologies (Selected Papers 2005), Nigel Shadbolt and Yannis Kalfoglou (ed), pages 309-322.

Journals:

- Ajit, S, Sleeman, D, Fowler, D.W., Knott, D and Hui, K (2008): Constraint Capture and Maintenance in Engineering Design, *Journal of Artificial Intelligence in Engineering Design and Manufacturing (AIEDAM),* Special Issue on Design Rationales, Rob Bracewell and Janet Burge (ed), Volume 22, Issue No. 4, pages 325-343.

- Sleeman, D, Ajit, S, Fowler, D.W. and Knott, D (2008): The role of ontologies in creating & maintaining corporate knowledge: a case study from the aero industry, *Journal of Applied Ontology*, Roberta Cuel and Roberta Ferrario (ed), Volume 3, Issue No. 3, pages 151-172.

Conferences/Workshops:

- Ajit, S, Sleeman, D, Fowler, D.W., Knott, D and Hui, K: ConEditor+ (2007): Capture and Maintenance of Constraints in Engineering Design, Rose Dieng and Nada Matta (ed), *IJCAI-07 Workshop on "Knowledge Management & Organizational Memories",* Hyderabad, India, pages 6-11.

- Sleeman, D, Ajit, S, Fowler, D.W. and Knott, D (2006): The role of ontologies in creating & maintaining corporate knowledge: a case study from the aero industry, Roberta Cuel and Roberta Ferrario (ed), *FOMI-06 Workshop Proceedings,* Laboratory for Applied Ontology, Trento, Italy.

- Ajit, S, Sleeman, D, Fowler, D.W., Knott, D and Hui, K (2005): Capture and Maintenance of Engineering Design Constraints, *Proceedings of the 2nd AKT Doctoral Symposium*, January 2006, Aberdeen, pages 4-13.

- Ajit, S, Sleeman, D, Fowler, D.W., Knott, D and Hui, K (2005): Capture and Maintenance of Engineering Design Constraints, The Twenty-fifth SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence, CD *Proceedings of AI 2005*, Cambridge, UK.

- Ajit, S, Sleeman, D, Fowler, D.W., Knott, D and Hui, K (2005): Acquisition and Maintenance of Constraints in Engineering Design, Third International Conference on Knowledge Capture, *Proceedings of KCAP 2005*, Banff, Canada, pages 173-174.

- Ajit, S, Sleeman, D, Fowler, D.W. and Knott, D (2004): ConEditor: Tool to Input and Maintain Constraints, 14th International Conference on Engineering Knowledge in the Age of the Semantic Web, *Proceedings of EKAW 2004*, Northampton, UK, pages 466 - 468.

Declaration

I declare that this thesis has been composed by myself and describes my own work. It has not been accepted in any previous application for a degree. All sources of knowledge have been specifically acknowledged.

Suraj Ajit
27th May, 2009

Department of Computing Science
University of Aberdeen
Aberdeen
United Kingdom

Acknowledgement

This work is supported under the EPSRC's grant number GR/N15764, and the Advanced Knowledge Technologies Interdisciplinary Research Collaboration, which comprises of the Universities of Aberdeen, Edinburgh, Sheffield, Southampton and the Open University.

I would like to thank Mr. David Knott (Head of Design Technology), Dr. Michael Moss and other people at the Rolls-Royce plc, Derby, UK for all their support. In particular, I would like to specially thank Mr. Colin Cadas (Head of Knowledge Management) and Mr. Stephen Docherty in the Transmission and Structures division of Rolls-Royce plc, Derby, UK for all the important discussions and contributions, relevant to the work reported in this thesis.

This thesis would not have been possible without the support of my colleagues, friends and family. In particular, I would like to thank my supervisor Professor Derek Sleeman for providing me the opportunity to do a PhD. I deeply appreciate his continuous support, encouragement and guidance over the years. I would like to thank Dr. David Fowler for all his support. I am grateful to Dr. Wamberto Vasconcelos and Professor Peter Gray for reviewing parts of my research work and giving me useful comments. I have received support from my former colleague Dr. Kit Hui and am extremely grateful to him for providing the translator that converts CoLan into CIF. I would like to thank all the subjects involved in the evaluation of ConEditor+ for taking part and contributing. I would also like to thank my examiners (both internal and external) for their constructive comments in improving the thesis.

Lastly (but not in any way least), on a personal note, I would like to thank my parents and sister for their love and support.

Contents

Contents

Contents

Contents

List of Figures

List of Tables

List of Acronyms & Abbreviations

AI	Artificial Intelligence
AKT	Advanced Knowledge Technologies
API	Application Programming Interface
Auto	Automatic
CAD	Computer Aided Design
CIF	Constraint Interchange Format
CommonKADS	Common Knowledge Acquisition and Design support
CRLM	Configurable Role Limiting Method
CVO	Constraint Version Object
DEC	Digital Equipment Corporation
DFX	Design for X
DR	Design Rationale
DRed	Design Rationale editor
DTI	Department of Trade and Industry
FO	Feature Oriented
FOL	First Order Logic
GT	Generic Task
GUI	Graphical User Interface
HCI	Human-Computer Interaction
HVAC	Heat, Ventilation and Air Conditioning
ICARE	Illustration, Constraint, Activity, Rule, Entity
ID	Identification Number
IDA	Institute for Defence Analysis
IPD	Integrated Product Development
IPAS	Integrated Products and Services
JDS	Joint Design Standards (specific to Rolls-Royce)
KA	Knowledge Acquisition
KB	Knowledge Base
KBE	Knowledge Based Engineering
KBS	Knowledge Based System
KE	Knowledge Engineering
MAKE	Maintenance Assistance for Knowledge Engineers
MOKA	Methodology and Tools Oriented to Knowledge Based Engineering Applications
OKBC	Open Knowledge Base Connectivity
OWL	Web Ontology Language
PO	Process Oriented
PSM	Problem Solving Method
RDF	Resource Description Framework
RDQL	RDF Query Language
RLM	Role Limiting Method
UI	User Intervention
UK	United Kingdom
UML	Unified Modelling Language
W3C	World Wide Web Consortium

Chapter 1
Introduction

> 'Knowledge Management is the Major Enabler of Enterprise Performance.'
>
> **- Karl Wiig**

This thesis presents original research in the field of knowledge management with engineering design as an application domain. The research proposes a novel approach to facilitate domain experts in capturing and maintaining constraints in engineering design. The thesis further embodies this approach with the design and construction of a system. This chapter provides a background on the topics relevant to this thesis, describes the motivation for the research work, outlines the research questions and also provides an overview of the thesis. The chapter is organised as follows: Section 1.1 provides a background to knowledge management including ontologies and the semantic web. Section 1.2 introduces engineering design, and describes a system developed by previous research (Fowler *et al*., 2004) to support engineering designers in large organisations such as Rolls-Royce. Section 1.3 describes the motivation for the research work reported in this thesis. Section 1.4 outlines the research questions that the thesis aims to address. Section 1.5 provides an overview of the thesis. The chapter concludes with Section 1.6 describing the thesis structure.

1.1 Knowledge Management

We live in a world where there has been an explosion of data, information and knowledge. However, knowledge is only of value when it can be used effectively and efficiently. The management of knowledge is increasingly being recognised as a key element in the organization of companies and institutions (Dieng *et al*., 1999; Dieng & Corby, 2000). The forms of knowledge have grown in terms of both complexity and applications. People often work for a number of employers during their lifetime. Loss

of knowledge can be a major factor in reducing an organisation's productivity and effectiveness. Organisations have experienced many changes to the way they operate. The nature of work has changed enormously with the shift from an industrial economy (where commercial products were the main business focus) to a knowledge economy (where service and expertise are the main business outcomes) (Debowski, 2006). The shift in focus from products to services has encouraged greater recognition of the importance of the knowledge held within an organisation. Knowledge management is concerned with the acquisition, modelling, use, reuse, retrieval, publishing and maintenance of knowledge. Knowledge engineering techniques have been known to bring significant benefits to knowledge management (Preece *et al*., 2001). More details of the various knowledge engineering techniques can be found in Chapter 2.

The challenges relevant in the context of this thesis are knowledge acquisition and maintenance. Knowledge acquisition is about extracting knowledge from sources of expertise and transferring it to a knowledge base (KB). Knowledge acquisition is well known to be a "critical bottleneck" in expert system development. The traditional approach to knowledge acquisition is mainly an interaction process involving the domain expert and knowledge engineer. This approach can be laborious, time-consuming and error-prone, especially if the knowledge engineer is unfamiliar with the domain. The challenge here is to develop tools and methodologies that facilitate domain experts in capturing and maintaining knowledge. In other words, the challenge is to eliminate or minimize the role of a knowledge engineer.

Knowledge maintenance is concerned with the process of controlling change in a knowledge-based system. Knowledge maintenance normally involves the following activities:

- *Verification and validation of knowledge based systems*: Verification and validation of the content of knowledge repositories is at the heart of knowledge maintenance. Verification is a process of ensuring that the knowledge base is consistent and complete within itself. Validation is the process of determining if a KBS meets its users' requirements (Meseguer & Preece, 1995).

- *Updating/refining of knowledge bases:* The challenge is to keep the knowledge repository functional and consistent. This may involve the regular updating/refining of content as it changes (e.g., as price lists are revised). Updating/refinement of KBs can make them inconsistent and further they can accumulate redundant knowledge. It is important to discard the redundant knowledge and make sure that the KB remains consistent.

- *Dealing with the obsolescence of knowledge*: Certain sections of the knowledge may be based on assumptions/conditions, which later become untrue. One has to identify and shelve/remove such sections, when necessary. This may involve a deeper analysis of the knowledge content. Some content has a considerable longevity, while other knowledge dates very quickly. If a repository of knowledge is to remain active over a period of time, it is essential to know which (and when) parts of the knowledge base must be discarded.

1.1.1 Ontologies and the Semantic Web

An ontology is a core element in knowledge management. The word ontology has been taken from Philosophy, where it is used to describe the existence of beings in the world and referred to as the *theory of existence*. The most commonly used definition of ontology in Artificial Intelligence (AI) is that of Gruber (1993): "An ontology is an explicit specification of a conceptualization". Borst (1997) and Borst *et al*. (1997) slightly modified Gruber's definition saying that: "Ontologies are defined as a formal specification of a shared conceptualization." Both the above definitions have been explained by Studer *et al*. (1998) as: "A 'conceptualisation' refers to an abstract model of some phenomenon in the world by having identified the relevant concepts of that phenomenon. 'Explicit' means that the type of concepts used, and the constraints on their use are explicitly defined. 'Formal' refers to the fact that the ontology should be machine readable, which excludes natural language. 'Shared' reflects the notion that an ontology captures consensual knowledge, that is, it is not private to some individual, but accepted by a group."

Large organizations are more likely to face the problem of integrating heterogeneous and distributed information expressing the specificity of the sub-

communities which, altogether constitute the organization itself (Sanghee *et al*., 2008). The integration problem is due to the lack of shared and globally consistent terminologies. Ontologies facilitate knowledge sharing and reuse by providing a commonly agreed domain model. The main differences between an ontology and a database schema, as listed in Fensel (2004) are:

- A language for defining ontologies is syntactically and semantically richer than common approaches for databases.
- The information that is described by an ontology consists of semi-structured natural language texts and not tabular information.
- An ontology must be a shared and consensual terminology because it is used for information sharing and exchange.
- An ontology provides a domain theory and not the structure of a data container.

Ontologies provide the backbone technology for the semantic web (Fensel, 2004). The semantic web is an evolving extension of the world wide web in which web content can be expressed in a form that can be understood, interpreted and used by computers to find, share and integrate information more easily (Berners-Lee *et al*., 2001; Shadbolt *et al*., 2006). According to the World Wide Web Consortium (W3C)[1], the semantic web is about two things: "It is about common formats for integration and combination of data drawn from diverse sources. It is also about language for recording how the data relates to real world objects. That allows a person, or a machine, to start off in one database, and then move through an unending set of databases which are connected not by wires but by being about the same thing."

The main uses of ontologies and semantic web technologies can be summarized as follows:

- To enable integration of heterogeneous data sources. A common task is to pose queries that require data from more than one source.

[1] http://www.w3.org/2001/sw/. Accessed online on 12 May 2008.

- To ensure people and software agents have a shared understanding of the terms and relationships used in a domain.
- To annotate documents and other resources with terms from the ontology, and then to use these annotations to retrieve documents. Using the structure of the ontology, documents that are related to those originally sought can be explored. For example, a document may be about one engine part, and by using the ontology, documents about parts that are similar to, or adjacent to, that part may be found.
- To allow reasoning to take place (deduce new statements that were not explicitly stated) and reveal inconsistencies in the data.
- To enable reuse of domain knowledge.
- To make domain assumptions explicit.
- To separate domain knowledge from the operational knowledge.

Ontologies are now in widespread use as a means of formalizing domain knowledge in a way that makes it accessible, shareable and reusable (Darlington & Culley, 2008). The research work reported in this thesis uses ontologies and semantic web technologies for knowledge management in engineering design. Engineering design is used as an application domain and this topic is discussed in the next section.

1.2 Engineering Design

"Knowledge management has been identified as one of the key enabling technologies for distributed engineering enterprises in the 21st Century. Central to the application and exploitation of knowledge in engineering is the engineering design process" (McMahon *et al*., 2004). Engineering Design is constraint-oriented and much of the design process involves the recognition, formulation and satisfaction of constraints (Serrano & Gossard, 1992; Lin & Chen, 2002). A constraint here refers to a rule that a successful design must satisfy. Constraints are continually being added, removed and modified throughout the development of a new product.

Engineering design is an important phase in product development that is known to have a significant impact on the life cycle characteristics (e.g. cost, reliability) of the product (Newnes *et al*., 2008; Salonen *et al*., 2008). Design begins with a functional specification of the desired product: a description of properties and

conditions that the product should satisfy (i.e. constraints). Engineering designers typically have to find a configuration of parts that implements a particular function. To assist them, most organizations have built up a large number of design rules and standards, usually held as large volumes of text. Designers must try to ensure that their configurations satisfy these constraints, but it is easy to overlook some. Novice designers may have a hard task in finding and appreciating relevant constraints. Additionally, in a collaborative environment, where many designers are working on subsections of a common component, it is common for changes made by one designer to affect the options available to another, and for this to go unnoticed until much later, thus causing expensive and time-consuming redesigns. It would clearly be useful to have some way of automating the design checking process, so that all applicable constraints are checked, without the designer having to manually initiate a search and check if all the constraints are satisfied. Hence previous research has developed a system known as the Designers' Workbench (Fowler *et al.*, 2004) to support engineering designers in large organisations such as Rolls-Royce. The following section introduces the Designers' Workbench.

1.2.1 Designers' Workbench

The Designers' Workbench uses an ontology to describe elements in a configuration task. Configurations are composed of features, which can be geometric or non-geometric, physical or abstract. The design rules are expressed as constraints over the ontology. The system allows the designers to build a configuration, and to check that all the constraints hold. In a real engineering situation, there may be many thousands of constraints, which means that it is easy to overlook some of them. Constraints are often defined generically, in that they apply to particular types of sub-configurations of features, rather than to specific features. Therefore, it is not necessary to have any actual features specified in the design before defining a constraint. For example, one may need to define a constraint that applies to all pairs of neighbouring features such that if one feature is made of copper and the other feature is made of zincalume® steel then the features are incompatible[2]. This constraint could be added without any knowledge that such a pair of features exists in a design. Constraint checking becomes

[2] http://www.bluescopesteel.com.au/go/howto/avoid-incompatible-metals. Accessed online on 7 May 2009.

a process of finding such sub-configurations by posing a query and checking that they satisfy the constraints.

The system has been implemented so that the human designer is free to use his or her engineering expertise to override constraints that are not deemed applicable to the current situation. A graphical user interface (GUI) enables the designer to import a drawing, annotate it with features, assign property values, and perform constraint checks. When a constraint is violated, the designer is presented with a list of features involved in the violation and a link to the source document that contains the design rule. The reader is encouraged to read Section 3.1 in this thesis and Fowler *et al.* (2004) for a more detailed description of the Designers' Workbench. The issues concerning acquisition and maintenance of knowledge (design rules) for systems such as the Designers' Workbench are the main topics of this thesis. The problems faced by the Designers' Workbench have been the motivation for the research work reported in this thesis and the following section describes this in some detail.

1.3 Problem Description and Motivation

The motivation for this thesis has been largely inspired by the observation of problems faced by the Designers' Workbench (Fowler *et al*., 2004), developed to support designers in large organizations, such as Rolls-Royce, by ensuring that the design is consistent with the specification for the particular design, as well as with the company's design rule book(s). The process of acquiring design rules for the Designers' Workbench's KB consists of the following phases:

(i) A domain expert (design engineer) works with a knowledge engineer to identify the design rules.

(ii) The knowledge engineer encodes the constraints in the Designers' Workbench's KB as a query in RDQL (RDF Query language) (Seaborne, 2004), and a predicate in Sicstus[3] Prolog.

[3] Swedish Institute of Computer Science, version 3.10, Accessed online on 29 May 2008 at http://www.sics.se/sicstus/

This process is laborious, error-prone and time-consuming. As design rules are described succinctly in the design rule book(s), a non-expert in the field finds it very difficult to understand the context and formulate constraints directly from the design rule book(s), and so a design engineer has to help the knowledge engineer in the process. It is highly desirable to relieve the knowledge engineer of this task and to facilitate domain experts themselves inputting design rules into the Designers' Workbench's KB. It would be useful if a new constraint could be formulated in an intuitive way, by selecting classes and properties from the ontology, and somehow combining them using a predefined set of operators. This would enable designers to have control over the definition and refinement of constraints, and presumably, to be able to have greater trust in the results of constraint checks. This thesis proposes a novel approach to facilitate domain experts in capturing and maintaining constraints. The approach involves the use of a graphical interface to facilitate domain experts in selecting classes and properties from the appropriate domain ontology and combining them with predefined keywords and operators from a high–level constraint language to form a constraint. The approach has been embodied by developing a system known as ConEditor (Ajit *et al*., 2004; Ajit *et al*., 2005a, 2005b, 2005c; Ajit *et al*., 2006). The thesis provides a detailed description of the adopted approach and the implemented system, ConEditor, in Chapters 3 and 4.

The engineering design process has an iterative nature as designed artefacts often develop through a series of changes before a final solution is achieved. A common problem encountered during the design process is that of constraint evolution, which may involve the identification of new constraints or the modification or removal of existing constraints. The reasons for such changes include development in the design and manufacturing technology, changes to improve performance and changes to reduce development time and costs. The evolutionary nature of constraints establishes the need to constantly update, revise, and maintain them. Maintenance of constraints involves various issues/problems. An overview of the issues and problems encountered during maintenance is provided below:

The constraints formulated by experts are generally applicable only in particular contexts, as the constraint may be based on specific assumptions. These contexts and assumptions are often *implicit* to the expert who formulates them and are not well documented or represented explicitly. When the experts who have formulated the constraints leave the company or become unavailable, it becomes extremely

difficult for other experts to maintain the knowledge base. One needs to identify all the constraints that require modification and make sure that all the constraints are applied in the right contexts. After making any change(s) to the KB, one has to make sure the KB is consistent. In addition, constant addition/revision of constraints can result in considerable redundancy in the KB. It is important to prevent/remove such redundancies as part of the maintenance of the KB. Maintenance is an important task that can be both complicated and expensive (Barker & O'Connor, 1989).

In order to reduce/overcome the various maintenance issues/problems, the thesis proposes a methodology and incorporates it into ConEditor to support the maintenance of constraints. The methodology involves: (i) the capture of the context in which a constraint is applicable as an *application condition* (Ajit *et al*., 2008a; Sleeman *et al*., 2008) together with the constraint in a machine-interpretable format and (ii) the use of the application condition together with the constraint and the domain ontology to support the maintenance of constraints. The thesis proposes four main types of knowledge refinement rules to detect redundancy, subsumption, inconsistency and fusion between pairs of constraints using the associated application conditions and domain ontology. The term "application condition" is used throughout the thesis to refer to the context and underlying assumptions associated with a constraint. The application conditions form a part of the rationales associated with the constraint. Further discussion of application conditions with examples, the proposed methodology, knowledge refinement rules and the support provided for the maintenance of constraints can be found in Chapters 3, 5 and 6. The following section describes the research aims and hypotheses of the research work.

1.4 Research Aims and Hypotheses

One of the aims of the knowledge engineering community has been to minimize/eliminate the role of a knowledge engineer. "Enabling domain experts to maintain the knowledge in a knowledge-based system has long been an objective of the knowledge engineering community" (Bultman *et al*., 2000b). This thesis identifies a situation where it is highly desirable to eliminate the knowledge engineer from doing a laborious, error-prone and time-consuming task. The thesis aims to explore how the design and construction of a system can facilitate domain experts in capturing and maintaining constraints. Further, the thesis reports that, in order to appropriately

apply, maintain and reuse constraints, it is important to capture the context in which a constraint is applicable in a machine- interpretable format. The thesis hypothesises that this context information, referred to as application conditions, together with the corresponding constraints and the domain ontology can be used to support the maintenance of constraints. Maintenance of constraints includes (i) detecting inconsistencies, redundancy, subsumption and fusion (ii) reducing the number of spurious inconsistencies and (iii) preventing the identification of inappropriate refinements of redundancy, subsumption and fusion, between pairs of constraints. It is also important to ensure that the speed of the system for realistic tasks is viable for domain experts to use. Hence, the main research aims and hypotheses of the thesis can be posed as the following research questions:

Research Question I:

1. Examine whether it is possible to design and construct a system to facilitate (domain) experts in capturing and maintaining constraints in engineering design. This question can be detailed into the following smaller questions:
    a) Can (domain) experts successfully perform the allocated tasks within acceptable time limits?
    b) Did the (domain) experts perform the tasks accurately? What kind of mistakes did they make? Can the system's GUI be modified to eliminate or minimize these errors?
    c) How easy and intuitive did (domain) experts find the system to use?
    d) Is the speed of the system on realistic tasks viable for (domain) experts to use?

Research Question II:

2. Examine whether capturing application conditions associated with constraints, in a machine-interpretable format can provide significant benefits to the maintenance of constraints in engineering design. In particular, can an explicit representation of application conditions together with the corresponding constraints and the domain ontology be used to:
    a) Detect inconsistencies, redundancy, subsumption and fusion,

b) Reduce the number of spurious inconsistencies, and
c) Prevent the identification of inappropriate refinements of redundancy, subsumption and fusion between pairs of constraints?

The next section provides an overview of the research work reported in this thesis.

1.5 Thesis Overview

The context for the research work reported in this thesis is the Designers' Workbench system that has been developed by previous research to support designers in large organisations, such as Rolls-Royce, to ensure that the design is consistent with the specification for the particular design, as well as with the company's design rule book(s). The knowledge engineering process to capture and maintain constraints for the Designers' Workbench is tedious, error-prone and time-consuming. It is highly desirable to relieve the knowledge engineer from the above task. The thesis proposes a novel approach to facilitate domain experts in capturing and maintaining constraints in engineering design. The thesis embodies the proposed approach with the design and construction of a system known as ConEditor. ConEditor facilitates basic maintenance by enabling domain experts to detect and resolve syntax errors, edit, delete and store constraints. The thesis reports on a preliminary evaluation of ConEditor conducted at Rolls-Royce. Further, the thesis reports that in order to appropriately apply, maintain and reuse constraints, it is important to capture the underlying assumptions and context in which each constraint is applicable. These assumptions and context are referred to as the "application conditions". The application conditions form a part of the rationales associated with a constraint. The thesis proposes an approach to capture the use these application conditions in a machine-interpretable format together with the domain ontology to support the maintenance of constraints.

The thesis analyses the kite design domain and proposes four main types of refinement rules to detect inconsistencies, subsumption, redundancy and fusion between pairs of constraints using application conditions and the domain ontology. The refinement rules have been proved to be logically sound. The thesis extends ConEditor to implement the proposed refinement rules and provide additional support to the maintenance of constraints. The extended system that was developed to provide additional support for maintenance became known as ConEditor+. The central

hypothesis of the thesis is that an explicit representation (machine-interpretable format) of application conditions together with the corresponding constraints and the domain ontology can be used to support the maintenance of constraints. Supporting the maintenance of constraints includes detecting inconsistencies, subsumption, redundancy and fusion, reducing the number of spurious inconsistencies, and preventing the identification of inappropriate refinements of subsumption, redundancy and fusion between pairs of constraints. The thesis reports on experiments, usability and scalability studies that apply ConEditor+ to support the capture and maintenance of constraints from a kite design KB. The usability studies demonstrate that ConEditor+ can facilitate domain experts in capturing and maintaining constraints in engineering design. The scalability studies demonstrate that the speed of ConEditor+ on realistic tasks is viable for domain experts to use. Further, the thesis investigates part of the Rolls-Royce domain, and demonstrates that the proposed approach can be applied to a more complex KB consisting of real world design constraints. The logical proofs of refinement rules together with the results of experiments in the kite domain and part of the Rolls-Royce domain demonstrate that an explicit representation (machine-interpretable format) of application conditions together with the corresponding constraints and the domain ontology can be used in: i) detecting inconsistencies, subsumption, redundancy and fusion, ii) reducing the number of spurious inconsistencies, and iii) preventing the identification of inappropriate refinements of subsumption, redundancy and fusion between pairs of constraints.

1.6 Thesis Structure

Theses usually adopt a structure in which they first provide background material to the field(s) of research, i.e., a literature review, and then explain the main problems/issues tackled, before presenting, discussing and evaluating the proposed solution or new approach. This thesis is no exception.

Chapter 1 provides a background to knowledge management including ontologies and the semantic web. This chapter introduces engineering design and describes the motivation for the research work reported in this thesis. The research aims and the hypotheses of the research work are then outlined. The chapter concludes by providing an overview of the thesis and its structure.

Chapter 2 provides a literature review of the principal fields relevant to this thesis. The review highlights some of the key issues in knowledge acquisition, knowledge engineering methodologies, knowledge maintenance (including verification, validation and refinement), constraints and engineering design. In addition, it provides a brief overview of some of the prominent systems that have been developed in these areas over the last couple of decades. The strengths and limitations of systems that have helped motivate the research work reported in this thesis have been indicated wherever appropriate. Finally, the chapter concludes by summarizing the key points from the literature review.

Chapter 3 presents a proposal for the research work reported in this thesis. The chapter starts by describing the Designers' Workbench and the problems faced in the capture of constraints for this system. The chapter then outlines the proposed approach to facilitate domain experts in capturing and maintaining constraints. Further, the chapter describes the issues/problems faced during the maintenance of constraints in an engineering design environment. The chapter outlines the proposed approach to support the maintenance of constraints. The chapter concludes with a summary.

Chapter 4 describes the design, implementation and functionality of ConEditor. The chapter presents an overview of the constraint representation languages (CoLan and CIF) used by ConEditor. The chapter also describes the principles involved in converting the domain ontology in OWL into a Daplex Schema and converting CoLan into CIF. The chapter concludes with a summary.

Chapter 5 introduces the concept of an application condition associated with a constraint. The chapter analyses the kite domain and describes how the application conditions together with the constraints and the corresponding domain ontology can be used to support the maintenance of constraints. Four main types of knowledge refinement rules are described with examples from the kite design KB. Further, the refinement rules are expressed in a formal notation (first order logic), and it is proved that they are logically sound. The chapter concludes with a summary.

Chapter 6 describes the design, implementation and functionality of ConEditor+. The chapter highlights the main changes made in extending ConEditor to ConEditor+. The chapter outlines the algorithm used by ConEditor+ to support the maintenance of constraints. The chapter also describes how ConEditor+ interprets the constraints in CIF to support maintenance. The chapter concludes with a summary.

Chapter 7 describes the evaluations performed during the research work. The chapter reports on a preliminary evaluation performed using ConEditor at Rolls- Royce. The chapter then describes experiments, usability and scalability studies conducted in the kite domain using ConEditor+ together with a discussion of the results obtained. The chapter concludes by describing the application of the proposed approach to part of the more complex Rolls-Royce domain together with the results obtained. The chapter concludes with a summary.

Chapter 8 provides an overview of the results and research contributions of this thesis. It also discusses some limitations of the work. The chapter concludes by presenting possible directions for future work and the significance of the major contributions.

Additionally, there are five appendices. Appendix A presents a list of the constraints obtained from the kite domain together with explanations of the corresponding rationales and application conditions. Appendix B lists the questionnaire that was used to evaluate the usability of ConEditor+. Appendix C presents an annotated walkthrough of constraint capture using screenshots of ConEditor+. Appendix D contains scanned copies of questionnaires that were answered by subjects during the evaluation of ConEditor+. Appendix E presents sample refinements of constraints and application conditions by ConEditor+ in the Rolls-Royce domain.

Chapter 2

Literature Review

> 'The important thing is not to stop questioning.'
>
> **- Albert Einstein**

This chapter presents a literature review of the principal fields relevant to the research work reported in this thesis. The review on knowledge acquisition, engineering and maintenance mainly provides a background and sets the context for the research work reported in the thesis. The sections on constraints in engineering design and design rationales present a review of literature that is more closely relevant to the work reported in the thesis. The chapter is divided into five main sections: Section 2.1 introduces the field of knowledge acquisition, including the various approaches to knowledge acquisition and a brief description of some of the tools developed to support knowledge acquisition. Section 2.2 reviews some of the prominent knowledge engineering methodologies developed over the years. Section 2.3 provides background information on knowledge maintenance with an overview of work done on verification and validation of KBS, and in the area of knowledge refinement. Section 2.4 provides background information on engineering design, and provides an overview of work done in the areas of constraints in engineering design, concurrent engineering and integrated product teams, and design rationales. A discussion of the key points of the review is provided at the end of each main section. Section 2.5 summarizes the literature review presented in this chapter. A brief introduction to knowledge engineering is provided below.

Knowledge Engineering is a field within Artificial Intelligence that refers to the building, maintenance and development of knowledge-based systems (KBSs). Initially, early descriptions of knowledge-based systems claimed that they consist of a knowledge base (usually a set of rules) and an inference engine that executed the rules by forward or backward chaining. This simple structure failed to distinguish the roles of different kinds of knowledge in a KBS, such as defining terms, expressing domain

facts, and supporting inference and problem solving. This confounding of different kinds of knowledge resulted in poorly structured knowledge-based systems and made them difficult to understand and maintain. It became clear that one needs to separate out the different kinds of knowledge in KBSs. A knowledge-based system essentially consists of two main components, a knowledge base and a problem-solving method (PSM).

MYCIN is one of the earliest knowledge-based systems that were developed in the early 1970s at Stanford University to diagnose infectious blood diseases. Its KB comprised of approximately 400 rules relating possible conditions to associated interpretations. MYCIN was highly domain specific and it became difficult to adapt the system for related diagnostic applications. This led to a domain independent version of MYCIN, known as the EMYCIN. EMYCIN allowed the inference engine of MYCIN to be applied to new problem domains and provided an environment for building and debugging knowledge bases. Subsequently, the notion of identifying the general problem solving ability in a domain of expertise was introduced by Hayes- Roth *et al*. (1983).

Clancey's (1985) identification of heuristic classification as the method underlying MYCIN KBS and the analysis of a number of knowledge-based systems led to the discovery of several general-purpose problem-solving components. Considerable emphasis has been placed on the development of knowledge-based systems from sharable and reusable, knowledge components. The development of this type of knowledge-based system requires a knowledge-engineering process where the developer selects, adapts, or constructs an appropriate problem solver, and supplies the system with the knowledge it needs to operate (Puerta & Eriksson, 1996). The two central activities in this type of development are the engineering of reusable components, and the acquisition of domain knowledge. Knowledge Engineering also involves the process of maintaining a KBS after it has been developed. Maintenance of a KBS involves verification, validation and refinement of knowledge. More details follow in subsequent sections of this chapter.

2.1 Knowledge Acquisition

Knowledge Acquisition is a field that deals with approaches to capture expert knowledge, specifically for use in knowledge-based systems. A difficulty that became

prominent during the development of MYCIN, and subsequent complex knowledge-based systems, was the extraction of the necessary knowledge from the human experts in the relevant fields. Knowledge Acquisition can be defined as follows:

'The transfer and transformation of potential problem solving expertise from some knowledge source to a program.' (Buchanan *et al.*, 1983).

Knowledge Acquisition may involve a wide range of sources such as human experts, documents, the World Wide Web, etc. Knowledge Acquisition is referred to as Knowledge Elicitation when the source of the knowledge acquired is specifically a *human* expert. The traditional approach to Knowledge Acquisition involves the following phases:

- Knowledge Engineer learns about the domain: Terminology (Glossary and Structured Glossary) and the dominant problem solving approaches.
- Domain Expert gets a top-level view of Expert Systems technology.
- Domain Expert solves tasks in the presence of the Knowledge Engineer; then the Knowledge Engineer solves same/similar tasks and is corrected (if required) by the Domain Expert.
- Knowledge Engineer encodes knowledge in an Expert System shell and then does gross debugging of the knowledge base.
- Knowledge Engineer and Domain Expert together use the Expert System to solve demanding tasks; debugging and modifying the knowledge base if necessary.

Early attempts to acquire knowledge in this way proved to be so time-consuming and intellectually demanding that knowledge acquisition was labelled the "bottleneck" in building knowledge-based systems (Feigenbaum, 1977). The reasons that can make knowledge acquisition unsuccessful include:

- Miscommunication between the knowledge engineer and the domain expert can make knowledge acquisition an error-prone process. This can happen especially when a knowledge engineer is unfamiliar with the domain or when the domain is too specialised for a knowledge engineer to understand.

- It is not always possible to transfer a domain expert's knowledge directly to a system because the respective representations are too dissimilar. In addition, the facts and principles underlying many domains of interest cannot easily be encoded in the precise mathematical/logical way that is necessary for subsequent processing and inference by a machine.
- Human problem solving expertise often relies on 'common sense' knowledge about the everyday world. Such knowledge is so deeply rooted in our experiences as humans that we may not even realise what we know, or what knowledge we are using in our reasoning. The existence of this *tacit* knowledge can make the knowledge acquisition task formidable.
- The form of questions can affect the answers given by the experts.
- The domain expert may be busy and hence unwilling to cooperate with the knowledge engineer.

There are various methods that can be used for Knowledge Acquisition. These methods can be classified in many ways depending on:

(i) the type of knowledge that is acquired, whether it is procedural or conceptual (e.g., problem solving strategy, classification).

(ii) the type of interaction with the expert (Burge, 1998): Direct methods involve directly questioning or observing a domain expert performing the job (e.g., interviewing). Indirect methods are those where the needed information is not requested directly. Instead, the knowledge acquisition session is analysed to obtain the needed information. (e.g., repertory grid).

(iii) whether knowledge is acquired "manually" or with the help of computer-based tools (e.g., SALT, MORE).

(iv) whether it is uncontrived or contrived (White, 2000): An uncontrived method seeks to observe an expert during problem solving without interfering in the problem solving process. In a contrived method, the knowledge engineer interacts directly with the domain expert, and can therefore steer the knowledge acquisition process towards topics of particular interest.

The type of method chosen can have an effect on the knowledge that is acquired. For example, adopting an indirect method can sometimes obtain additional information than that provided by direct methods. There are many reasons why an indirect method might produce more information. One reason is that the indirect method may end up probing aspects of the problem that the knowledge engineer did not anticipate, and may not have asked in the direct KA session. Another reason is that some subjects are not as verbal as other subjects are and are unlikely to give full and detailed answers to direct questions. A third reason is that some knowledge may be *implicit*. Implicit knowledge is knowledge that was either learned implicitly and cannot be expressed explicitly, or that was once explicit but has become implicit over time as the domain expert has used it repeatedly and it became "automatic" (Berry, 1987).

The behaviour of the knowledge engineer can also play a significant part in the effectiveness of the acquisition exercise, and can even harm the experiment by introducing an unwanted bias. For example, when interviewing a domain expert, the language used by the knowledge engineer can carry connotations, which influence the domain expert's answers. For example, consider the question, do you get headaches *frequently*, and if so how often? as opposed to do you get headaches *occasionally*, and if so how often?

When choosing a method, there should clearly be a good match between the type of knowledge required and the type generally produced by the method. For example, can the proposed method elicit class hierarchies, causal knowledge, examples, constraints, facts, goals, explanations, justifications, preferences, procedures, or relations? Cordingley (1989) states that although *interviewing* (see section 2.1.1) is a good technique for eliciting conceptual structures, facts, and causal knowledge, its efficacy for eliciting rules and assessments of weight of evidence is questionable. Similarly, the *repertory grid method* (see section 2.1.5) is good for eliciting conceptual structures, rules and weights of evidence, but bad for eliciting causal knowledge, procedures, and an expert's problem solving strategy. A brief review of some KA methods is given below:

2.1.1 Interviewing

An interview of the domain expert by the knowledge engineer is a common knowledge acquisition technique. There are several different types of interview

(Diaper, 1989). In an unstructured interview, the knowledge engineer asks probing questions and records the responses. The style of interviewing is flexible, so that the domain expert's reaction can be pursued if the direction looks fruitful. One alternative is a focussed interview, which concentrates on a single aspect of problem solving, and covers it in great depth. Another approach is a structured interview, in which the knowledge engineer keeps strictly to an agenda, and prepares for the interview with a list of specific questions.

2.1.2 Protocol Analysis

Protocol Analysis (Ericsson & Simon, 1984) involves asking the expert to perform a task while "thinking aloud." The intent is to capture both the actions performed and the mental process used to determine these actions. As with all the direct methods, the success of the protocol analysis depends on the ability of the experts to describe why they are making their decisions. In some cases, the experts may not remember why they do things a certain way. In many cases, the verbalised thoughts will only be a subset of the actual knowledge used to perform the task. One method used to augment this information is Interruption Analysis (Olson & Reuter, 1987). For this method, the knowledge engineer interrupts the expert at critical points in the task to ask questions about why he/she performed a particular action.

2.1.3 Document Analysis

Document analysis involves gathering information from existing documentation. This may or may not involve interaction with a human expert to confirm or enhance this information. Some document analysis techniques, particularly those that involve a human expert, can be classified as direct. Others, such as collecting artefacts of performance, such as documents or notes, in order to determine how an expert organises or processes information are classified as indirect (Cordingley, 1989). This method has been adopted by systems such as the Designers' Workbench (Chapter 3 of this thesis) to acquire design knowledge (rules).

2.1.4 Card Sorting

Card Sorting is a specialised indirect method, used for eliciting further knowledge about a pre-selected set of concepts. When sorting, each concept of interest is described on a card (the card consists of a picture, name of a concept or a short description), and the domain expert is asked to divide the pack of cards into separate, but meaningful, piles. The knowledge engineer records the separation and asks the domain expert to explain it. Then the process is repeated, and the domain expert is requested to provide a further consistent separation. This continues until the domain expert can think of no more ways to separate the concepts. Often, sorting acquires knowledge about classes, properties and priorities. For example, if the task was sorting pictures of different types of houses, a subject might sort them into groups “brick”, “stone”, “wood”, etc., with the criterion being “main material of construction.” The second time, the subject might divide the cards into groups called “one”, “two”, and “three,” with the criterion being “number of floors in each building.”

2.1.5 Construct Elicitation (Repertory Grid)

Construct Elicitation methods are used to obtain information about how the expert discriminates between entities in the problem domain. The most commonly used construct elimination method is Repertory Grid Analysis. The repertory grid is an indirect method based on personal construct theory (Kelly, 1955). In this method, the domain expert is presented with a list of domain entities and is asked to describe the similarities and differences between them. These similarities and differences are analysed to derive the important attributes of the entities. After completing the initial list of attributes, the knowledge engineer works with the domain expert to assign ratings to each entity/attribute pair. In some cases, attributes are rated as present/not present for each entity, in others a scale is used where the attribute is ranked by the degree to which it is present. The ratings are arranged in the form of a grid/matrix and subsequently analyzed for any existing correlations. Numerical values in cells will allow more complex numerical/statistical analysis to be done. The type of information acquired by this elicitation method may be classification, dependencies/relationships or evaluation.

2.1.6 Laddering

Laddering was first introduced by Hinkle (1965), a clinical psychologist, in order to model the concepts and beliefs of people by an unambiguous and systematic approach. Laddering is a structured questioning method (indirect method), enabling a hierarchy of concepts to be established (Corbridge *et al*., 1994). The knowledge engineer starts with a so-called seed concept and poses questions such that the domain expert justifies the position of the concept in a hierarchy, and at the same time offers further knowledge. For example, given the concept Apple, one might ask 'Can you give examples that belong to the concept Apple?' This should acquire concepts that are lower in the hierarchy (e.g. Cox, Gala). It is also possible to acquire concepts at the same level in the hierarchy by asking for alternatives, e.g., 'What alternative concepts are there that are similar to the concept Apple?' Concepts higher in the hierarchy may be obtained by asking for commonalities, e.g., 'What have Banana, Apple and Orange got in common?' An example of knowledge acquired using the laddering method is shown in Figure 2.1.

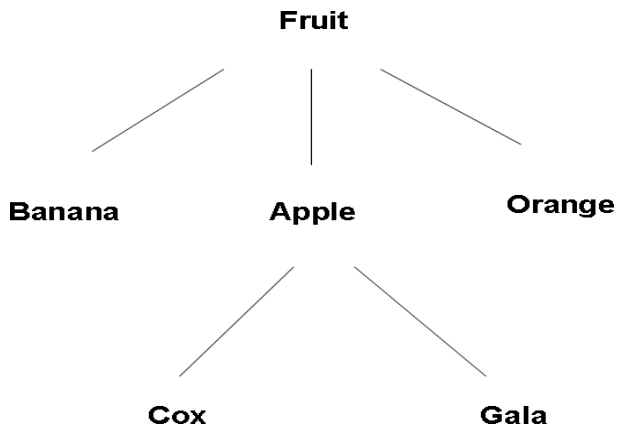


Figure 2.1: Laddering Method

2.1.7 Use of Computer-assisted/Computer-based tools

KA methods can be error-prone, laborious and time-consuming when applied manually. Additionally the acquired knowledge has to be encoded into a computer- based system. Hence, some of the KA methods (e.g., repertory grid) have been incorporated directly into computer programs with the aim of minimizing the role of a knowledge engineer. ETS and AQUINAS (expanded version of ETS), both computerised extensions of the repertory grid method, have been used to derive 'hundreds' of small and medium-sized knowledge-based systems in Boeing (Boose & Bradshaw, 1999) . Researchers have also concentrated on harnessing the synergy of the different KA methods by building a computerised workbench. One of the first was a research prototype called ProtoKEW (Reichgelt & Shadbolt, 1992). This was subsequently re-implemented and has been marketed as a commercial product, called PC-PACK[4] (Goodall, 1996; Milton *et al*., 1999; Milton, 2007, 2008). It contains a suite of integrated tools that allows the user to create, inspect and edit XML knowledge bases. Each tool provides a different way of viewing the knowledge base. The latest version is PCPACK5 and consists of the following five acquisition and modelling tools, and five specialised tools:

Acquisition and Modelling Tools:

Protocol Tool – allows the marking up of interview transcripts, notes and documentation (protocols) to identify and classify knowledge elements to be added to the KB.

Ladder Tool – facilitates the creation of hierarchies of knowledge elements such as concepts, attributes, processes and requirements.

Diagram Tool – allows the user to construct compact networks of relations between knowledge elements such as process maps, concept maps and state-transition diagrams.

Matrix Tool –allows grids to be created and edited that show relations and attributes of knowledge elements.

Annotation Tool – allows sophisticated annotations to be created using dynamic html, which include automatically generated hyperlinks to other resources in the KB.

Specialised Tools:

Admin Tool – used to access and manage KBs.

[4] www.epistemics.co.uk. Accessed online on 16 May 2008.

Publisher Tool – allows creation of websites using a template driven approach.
Diagram Template Tool – used to create templates for use in the Diagram Tool.
Equation Editor – used to create equations for use in the Annotation tool.
Tool Launcher – is a wizard tool allowing easy access to other tools.

PCPACK supports knowledge engineering methodologies such as CommonKADS and MOKA (Milton, 2008). These knowledge engineering methodologies are discussed in the next section. Other examples of computer-based tools include tools driven by PSMs: SALT, S-SALT, MORE, MOLE, OPAL, a grammar-driven tool known as COCKATOO, and a KA tool that generates expectations to develop PSMs, known as EMeD of the EXPECT framework. Tools driven by PSMs such as SALT, S-SALT, MORE, MOLE, etc, use a knowledge engineering methodology called Role-Limiting Methods that is explained in section 2.2.1. A brief review of some of the computer-based KA tools is given below:

MORE: MORE (Kahn *et al*., 1985) is a system that acquires diagnostically significant knowledge from domain experts by formulating a number of questions. MORE uses a model-theoretic approach to the acquisition of diagnostic knowledge. It uses a qualitative model of causal relations together with a theory of how causal knowledge can be used to achieve more accurate diagnostic conclusions to guide the interview process.

MOLE: MOLE (Eshelman *et al*., 1988) is a successor of MORE. It uses a method of heuristic classification known as the Cover-and-Differentiate problem solving method, which makes several heuristic assumptions and constructs an initial knowledge base with the help of domain expert(s). Subsequent interactive problem solving detects errors in the KB, suggests remedies to the domain expert and makes the required changes to the KB. An important aim of MOLE is to minimise the number of questions a domain expert is asked, by making intelligent guesses.

SALT: SALT (Marcus & McDermott, 1989) is a knowledge acquisition tool that uses the Propose-and Revise problem solving method. In essence, three generic roles may be identified for Propose-and-Revise (Studer *et al.*, 1998):

- "design extensions" refer to knowledge for proposing a new value for a design parameter, (a way of upgrading an existing entity)

- "constraints" provide knowledge restricting the admissible values for parameters, and
- "fixes" make potential remedies available for specific constraint violations.

For each type of role, a fixed menu (or schema) is presented to the domain expert to be filled out.

OPAL: OPAL (Musen *et al*., 1988) is a custom-tailored KA tool which is driven by a problem solving method known as skeletal plan refinement. OPAL allows medical specialists to enter and review cancer treatment plans for use by an expert system called ONCOCIN (Shortliffe *et al*., 1981) that provides therapy advice to physicians who take care of cancer patients. The cancer therapy task model has been built into OPAL, and OPAL's user interface primarily consists of graphical forms that facilitate instantiation of the task model.

COCKATOO: COCKATOO (White & Sleeman, 2001) is a grammar-driven KA tool that uses constraint technology to acquire knowledge from the domain experts. The advantages of this approach include: (i) It provides concise specifications of tasks that are more readable and save development time (ii) The required properties of each user input can be checked at acquisition time rather than prior to problem solving or at problem solving time. (iii) It provides a reactive user interface where the choice of a particular value for one input might narrow the options for another.

EXPECT: EXPECT (Kim & Gil, 1999) provides a framework to develop KA tools. EXPECT uses dependencies between factual knowledge and PSMs to find related pieces of knowledge in their KBS and create expectations from them. To give an example of these expectations, suppose that the user is building a KBS for a configuration task that finds constraint violations, and then applies fixes to them. When the user defines a new constraint, the KA tool has the expectation that the user will specify possible fixes for cases when the constraint is violated, and helps the user do so. EMeD (EXPECT Method Developer) is a KA tool that uses such expectations to support users to develop PSMs.

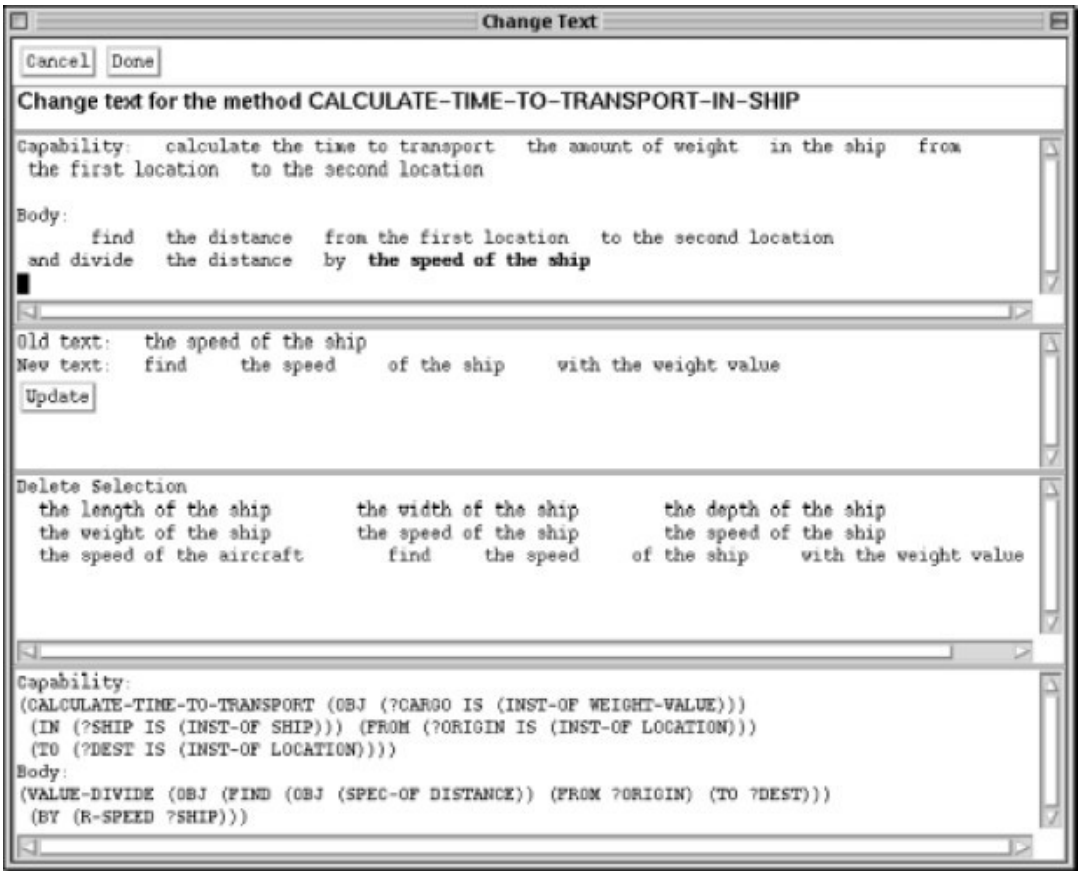


Figure 2.2: A screenshot of the English-based method editor used here to acquire problem solving knowledge to compute the time to transport an item in a ship
Source: Kim & Gil (1999)

An English-based Method Editor (Blythe & Ramachandran, 1999; Blythe *et al*., 2001) has been developed to help a user modify and add problem-solving knowledge to existing KBs. The value of the tool lies in the fact that the user need not understand the syntax of the expert system to make modifications. Two main steps are involved in this approach: Firstly, the problem solving knowledge is converted into an English- like structured text fragment and presented to the user. Secondly, selectable parts of the text are modified by choosing among alternatives that are also presented to the user via an English paraphrase.

A central theme of this KA research has been how KA tools can exploit *Interdependency Models* that relate individual components of the knowledge base in order to develop expectations of what users need to add next. A screenshot of the

English-based Method Editor is shown in Figure 2.2. Figure 2.2 shows an English-based front end that describes the method to compute the time to transport an item in a ship, by dividing the distance to travel by the speed of the ship. The user can alter the method by selecting a part of the sentence ("speed of the ship") and choosing from a set of alternatives provided (shown in the second window from the bottom).

2.1.8 Discussion

Knowledge Acquisition is a critical phase within Knowledge Engineering. The quality (correctness) of the knowledge acquired affects the performance of a KBS. There are various methods that can be used for knowledge acquisition, including manual and computer-assisted tools. There is no single best method for knowledge acquisition. The type of method to be adopted for knowledge acquisition depends on the type of knowledge being acquired. Knowledge Acquisition is referred to as Knowledge Elicitation when the source of knowledge acquired is specifically a human expert. Several methods and tools have been developed with the aim to either minimize or eliminate the role of a knowledge engineer in the Knowledge Acquisition process. The underlying assumption here is that minimizing or eliminating the role of a knowledge engineer would make the Knowledge Acquisition process less error-prone and less time-consuming. The Designers' Workbench uses the KA method of Document Analysis involving a knowledge engineer to acquire design rules. This thesis presents a novel approach to relieve the knowledge engineer from doing the error-prone and time-consuming task of acquiring design rules (expressed as constraints) in the context of the Designers' Workbench. The thesis embodies the proposed approach with the design and construction of a system that has been developed to facilitate domain experts in capturing and maintaining constraints in engineering design. More details about the proposed approach and the developed system can be found in Chapter 3.

2.2 Knowledge Engineering Methodologies

Several methodologies and tools have been developed over the years to efficiently support all the phases of knowledge engineering. A brief review of some of the knowledge engineering methodologies is given below:

2.2.1 Role-Limiting Methods (RLM)

Role-Limiting Methods (Marcus, 1988) was one of the first attempts to support the development of KBSs by exploiting the notion of a reusable problem-solving method (PSM), where a PSM is a model of KBS problem solving behaviour (also known as the inference system). Examples of PSMs are Cover-and-Differentiate (for solving diagnostic tasks) (Marcus, 1988) and Propose-and-Revise (for parametric design tasks) (Marcus & McDermott, 1989). The RLM approach can be characterized as a shell approach. Such a shell comes with an implementation of a specific PSM and thus can only be used to solve a task for which the PSM is appropriate. The given PSM also defines the generic roles that knowledge can play during the problem solving process.

Strong Points:
From the characterization of the PSM (Propose-and-Revise) for SALT, one can see that the PSM is described in generic, domain independent terms. Thus, the PSM may be used for solving design tasks in different domains by specifying the required domain knowledge for the different predefined generic knowledge roles. For example, S-SALT (Leo, 1995) is an enhancement of SALT system and implements the Propose-and-Revise problem solving method. S-SALT has been successfully applied to solve the VT-Sisyphus-II problem, an elevator configuration task that is used in the knowledge acquisition community as a benchmark. With S-SALT, the domain expert uses a form-oriented user interface for entering domain specific design extensions. That is, the generic terminology of the knowledge roles, which is defined by object and relation types, is instantiated with VT-Sisyphus-II specific instances.

Weak Points:
A problem faced with RLMs is how to determine whether a specific task may be solved by a given RLM. Such task analysis is crucial. Moreover, RLMs have a fixed structure and do not provide a good basis when a particular task can only be solved by a combination of several PSMs. The problem-solving strategy is fixed and cannot be adapted or augmented. In order to overcome this inflexibility of RLMs, the concept of configurable RLMs (CRLMs) was developed. CRLMs (Poeck & Gappa, 1993; Fensel & Poeck, 1994) exploit the idea that a complex PSM may be decomposed into several

subtasks. Each of these subtasks may be solved by selecting a method from a predefined set of different methods within the CRLM framework. CRLM provides this kind of flexibility but still comes with a fixed set of knowledge types. Further, there are no clear examples of where CRLM-developed systems have been used to solve complex (real-world) tasks.

2.2.1.1 Generic Tasks and Task Structures

The knowledge engineering literature has identified a number of problem types (Hayes-Roth *et al*., 1983; Clancey, 1985) such as diagnosis, design etc. and identified for each problem type a number of problem solving methods (PSMs). Following the work of Hayes-Roth and Clancey, the notion of a Generic Task (GT) (Chandrasekaran, 1986) evolved. GTs can be viewed as building blocks that can be reused for the construction of different KBSs. The basic idea of GTs may be characterized as follows (Chandrasekaran, 1986; Studer *et al.*, 1998) :

- A GT is associated with a generic description of its input and output.
- A GT comes with a fixed set of knowledge types specifying the structure of domain knowledge needed to solve a task.
- A GT includes a fixed problem solving strategy specifying the inference steps the strategy is composed of and the sequence in which these steps have to be carried out.

Strong Points:
GTs provided a larger vocabulary of task-related terms, and additionally, related the knowledge to how it was going to be used. The task-view provided important points of leverage in the generation of explanations of problem solving. The GTs also appeared to have computational advantages.

Weak Points:
The GT approach is based on the hypothesis that the structure and representation of domain knowledge is completely determined by its use (Bylander & Chandrasekaran, 1987). Analysis of the GT approach in more detail led to identification of two main disadvantages (Chandrasekaran *et al*., 1992):

- No clear distinctions exist between the notion of a task and the notion of the PSM used to solve the task, since each GT includes a pre-determined problem solving strategy.
- The complexities of the proposed GTs are very different, i.e. the appropriate levels of granularity for the building blocks are not clear.

Based on this insight into the disadvantages of the notion of a GT, the so called Task Structure approach was proposed (Chandrasekaran *et al.*, 1992). The Task Structure approach makes a clear distinction between a task, which is used to refer to a type of problem, and a method, which is a way to accomplish a task. In that way a task structure may be defined as follows: a task is associated with a set of alternative methods suitable for solving the task. Each method may be decomposed into several subtasks. The decomposition structure is refined to a level where elementary subtasks can be directly solved by using available knowledge. This basic notion of task, PSM and the decomposition structure are perspectives that are shared among most of the knowledge engineering methodologies in recent years.

2.2.1.2 Overview of RLMs and GTs

RLMs are methods that strongly guide knowledge collection and encoding (McDermott, 1988). They specify the roles various types of knowledge play in the operation of each method. The major difference between the role-limiting method approach and most of the other approaches is the requirement that a RLM be completely specified (i.e., that all tasks and subtasks be pre-specified down to the level of primitive operations). A problem faced with RLMs is how to determine whether a specific task may be solved by a given RLM. Such task analysis is crucial. A GT defines a task of general utility (such as classification), a method for doing the task and the kinds of knowledge needed by the method. Complex tasks are decomposed into generic tasks and the required knowledge is directly described for any domain in which the task is performed. GTs grouped both task and method together with each task having a pre-determined problem solving strategy. The Task Structure approach was then proposed that makes a clear distinction between a task and a method.

2.2.2 The PROTÉGÉ Approaches

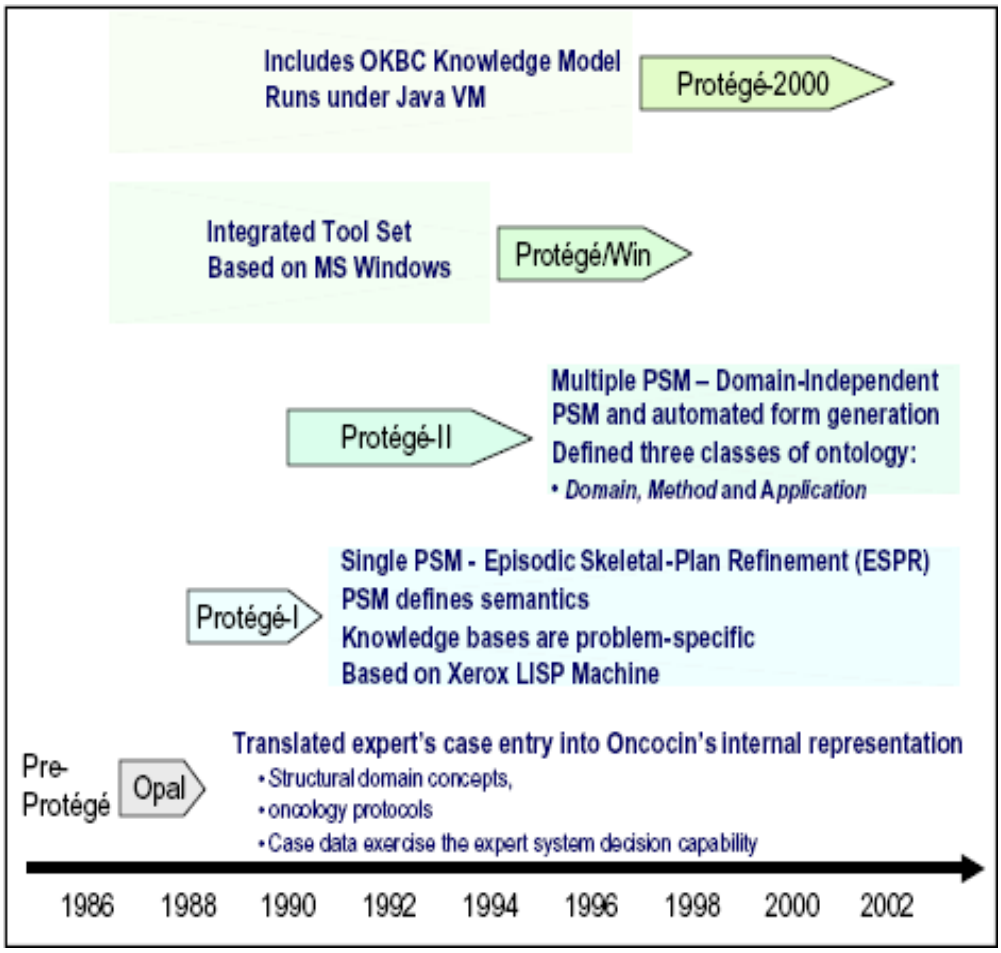


Figure 2.3: The Protégé Approaches

Source: Hengl (2004)

The Protégé approach has evolved over the years (Grosso *et al.*, 1999). Figure 2.3 illustrates the evolution of Protégé approaches. Protégé was developed with the aim to reduce the knowledge-acquisition bottleneck by minimizing the role of the knowledge engineer in constructing knowledge bases. This was achieved by using task-specific knowledge to generate and customize knowledge acquisition tools. The original Protégé was then modified to explicitly separate the problem-solving method from the domain knowledge. This led to the Protégé-II approach. The Protege-II approach (Puerta *et al*., 1992; Musen *et al*., 1993; Rothenfluh *et al*., 1994; Eriksson *et al*.,

1995b; Gennari *et al*., 1995; Rothenfluh *et al*., 1996) aimed at supporting the development of KBSs by the reuse of PSMs and ontologies. In addition, Protege-II laid emphasis on the generation of custom-tailored knowledge acquisition tools from ontologies (Eriksson & Musen, 1993; Eriksson *et al*., 1994; Eriksson *et al*., 1995a). Protege-II relied on the task-method-decomposition structure as followed in Generic Tasks and Task Structures. The Protégé-II approach introduced declarative mappings to enable reuse of both ontologies and PSMs. Mapping relations could be formed to connect the application and method ontologies. In addition, Protégé-II included the "downhill flow" assumption of classes over instances. The assumption is that classes were more durable than instances. It was expected that knowledge engineers would use one tool to define classes and then domain experts would use a separate tool (KA tool) to create and edit instances.

The Protégé-Win approach emerged later with the goals of:

(i) making knowledge bases more reusable and maintainable by splitting them into modular components that can be included in one another.

(ii) making software tools more usable by porting them to a standard platform. Protégé tools became executable in a Windows environment (earlier, they ran on NeXT workstations). Protégé-Win became a useful tool for building models of small domains and experimenting with various types of KBSs. However, it suffered from three limitations:

a) the standard set of user-interface widgets was too limited for many envisioned users.
b) interoperability with other modelling frameworks was limited
c) flexibility was not enough for many domains.

The recent model adopted is that of Protégé-2000 (Grosso *et al*., 1999; Noy *et al*., 2000). However, the most recent implementation (at the time of writing this thesis) is Protégé 3.2.1[5]. The goals here are to make knowledge bases reusable across modelling frameworks by adopting standard representation languages and lay groundwork for addressing scalability issues in knowledge engineering. Protégé-2000 adopts a new OKBC knowledge model that offers three major advantages of greater expressivity, clean model-theoretic semantics and the possibility of reuse with distributed ontology servers. Protégé-2000 provides support for modellers to

[5] Protégé Ontology Editor and knowledge-base framework, version 3.2.1, Accessed online 02 July 2007 at http://protege.stanford.edu/download/registered.html

customise and extend Protégé in task and domain specific ways. Protégé-2000 also introduces the explicit notion of a project. Projects contain knowledge base and configuration information.

A knowledge base is simply a collection of frames (it also contains things like reified slots, facets and axioms). The configuration information contains description of all the widgets that have been added to the project, information about the knowledge base server being used and a list of all the projects that are included by the current project. Protégé-2000 is highly customisable, and has recently been adapted to the new world of semantic web by reusing its user interface, internal representation, and framework (Noy *et al.*, 2001). The most recent version of Protégé 3.2.1 supports the Web Ontology Language (OWL) of the semantic web (Knublauch *et al*., 2004). Protégé 3.2.1 has been used to develop ontologies in OWL for use by the systems Designers' Workbench and ConEditor/ConEditor+, that are described later in this thesis.

2.2.3 The CommonKADS Approach

CommonKADS (Common Knowledge Acquisition and Design Support) (Kingston, 1998; Schreiber *et al*., 2000; Bromby *et al*., 2003) is a methodology to support structured knowledge engineering. It supports most aspects of a KBS development project, such as:

- Project management
- Organisational analysis (including problem/opportunity identification)
- Knowledge acquisition (including initial project scoping)
- Knowledge analysis and modelling
- Capture of user requirements
- Analysis of system integration issues
- KBS design

CommonKADS provides a clear link to modern object-oriented development and uses notations compatible with UML. CommonKADS consists of the following predefined set of models:

- Organization model: The organization model supports the analysis of the major features of an organization. The deficiencies or problems faced by the current business processes are identified with opportunities to improve these processes by introducing KBSs.
- Task model: Tasks are the relevant subparts of a business process. The task model analyzes the global task layout, its inputs and outputs, preconditions and performance criteria, as well as needed resources and competencies.
- Agent model: The agent model specifies the capabilities of each agent involved in the execution of the tasks at hand. In general, an agent can be a human or some kind of software system.
- Knowledge model: The purpose of the knowledge model is to describe in detail the types and structures of the knowledge used in performing a task. It provides an implementation-independent description of the roles that different knowledge components play in problem solving, in a way that is understandable for humans. This makes the knowledge model an important vehicle for communication with experts and users about the problem solving aspects of a KBS.
- Communication model: Here the various interactions between the different agents are specified. Among others, it specifies which type of information is exchanged between the agents and which agent is initiating the interaction.
- Design model: The design model gives the technical system specification in terms of architecture, implementation platform, software modules, representational constructs and computational mechanisms needed to implement the functions laid down in the knowledge and communication models.

The Knowledge Model has three parts, each capturing a related group of knowledge structures. Each part is called a knowledge category. The first category is the domain knowledge; this category specifies the domain specific knowledge and information types required to solve the task at hand. This includes a conceptualization of the domain in a domain ontology, and a declarative theory of the required domain knowledge. The second category is the inference knowledge. The inference knowledge describes the basic inference steps to be made using the domain knowledge. The third category is the task knowledge. Task knowledge describes what

goal(s) an application pursues, and how these goals can be realized through decomposition into subtasks and inferences. This "how" aspect includes a description of the dynamic behaviour of tasks, i.e., their internal control.

2.2.4 The MIKE Approach

In MIKE (Model-based and Incremental Knowledge Engineering) (Fensel & Poeck, 1994; Landes, 1994; Studer *et al*., 1998), the entire development process is divided into the following sub activities (Figure 2.4):

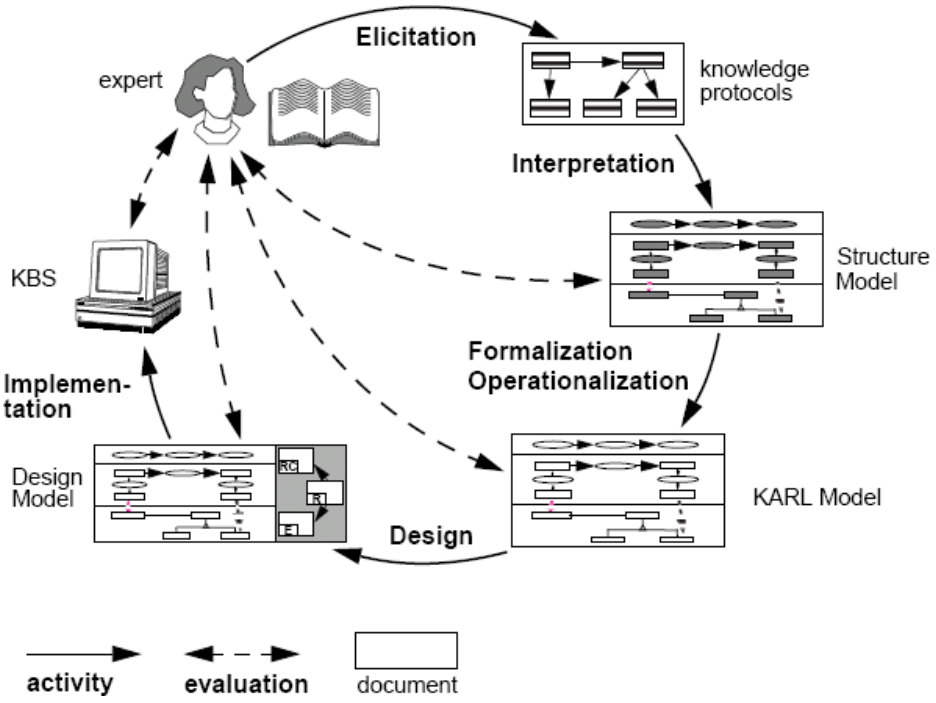


Figure 2.4: The MIKE Approach

Source: Studer *et al.* (1998)

Elicitation: Methods like structured interviews are used for acquiring informal descriptions of the knowledge about the specific domain and the problem solving processes. The resulting knowledge expressed in natural language is stored in so- called knowledge protocols.

Interpretation: During this phase, the knowledge structures identified in the knowledge protocols are represented as the structure model. All structuring information in this model, like the data dependencies between two inferences, is expressed in a fixed, restricted language while the basic building blocks, e.g., the description of an inference, are represented by unrestricted texts. The knowledge engineer and the expert can use this representation to communicate with each other.

Formalization/Operationalization: The structure model is the foundation for the formalization/operationalization process that results in the model of expertise known as the KARL model. The KARL model has the same conceptual structure as the structure model while the basic building blocks represented as natural language texts are now expressed in the formal specification language KARL (Fensel *et al*., 1998). The formal specification describes the functionality of the system precisely, yet abstracting from implementation details.

Design: The Design phase is performed on the basis of the KARL model after it has been evaluated with respect to the required functionality. This phase captures all the functional as well as the non-functional requirements of the KBS. The non-functional requirements include e.g., efficiency and maintainability, and the constraints imposed by target software and hardware environments.

Implementation: This is the final phase in which the design model is implemented in the target hardware and software environment to form the KBS.

2.2.5 The MOKA Approach

MOKA (Callot *et al*., 1999; Klein, 2000; Stokes, 2001) is a methodology that has been developed for knowledge modelling in design and engineering. From a knowledge modelling point of view, there are two key issues that have been identified in knowledge-based design (Klein, 2000): First, there is a close interaction in design between object level knowledge (components, structures, behaviours, functions, etc.) and problem solving knowledge (transformations, constraint solving, search). Second, control of problem solving and strategic reasoning is essential in design. This results in two challenges of knowledge modelling in design (Klein, 2000): first, to develop

general knowledge modelling schemes that are expressive, powerful and flexible enough; and second, to adapt these model requirements to the special requirements of design. This will also allow us to reduce the gap between "general AI" and AI in design (Smithers, 1998).

Knowledge based engineering (KBE) is defined as the use of advanced software techniques to capture and re-use product and process knowledge in an integrated way. KBE systems differ from other knowledge-based systems mainly in terms of geometry and the high degree of iteration within engineering design. The iteration here means that building a design requires processing a little bit of knowledge in one area, then a little in another, then maybe back to the original and so on and the process is far from linear. The linking between the many parts of the process and, as a consequence, the complicated linking with the product objects makes it difficult for a KBE (Stokes, 2001) approach. This led to the development of the MOKA Approach (Methodology and tools Oriented to Knowledge based engineering Applications). Rolls-Royce is currently adopting the MOKA approach.

The main objectives of the MOKA project are:

- Reduce the lead times and associated costs of developing KBE applications by 20-25%
- Provide a consistent way of developing and maintaining KBE applications
- Develop a methodology that will form the basis of an international standard
- Provide software tools to support the methodology

MOKA consists of the following elements:

1. Lifecycle: A description of the lifecycle for a KBE application (whether new or being modified) as a MOKA Route Map to guide you through the life cycle is provided. The life cycle is described by means of the following six steps:
   IDENTIFY – This step aims to investigate the business needs and to determine the type of KBE system that might satisfy those needs.
   JUSTIFY – This step involves the generation of a global Project Plan that is used together with a business case to seek management approval for the steps below.

CAPTURE – This step aims to collect the domain knowledge in a raw form and structure it into an informal model. Engineering design covers a wide variety of knowledge including product specification, general constraints, conceptual design knowledge, physical design knowledge, design rationales, and design process knowledge.

FORMALIZE – This step builds a formal model in two distinct parts: the product model and the design process model.

PACKAGE – This step involves translation of the formal model into code for a working KBE system.

ACTIVATE – This step involves the distribution, installation and use of the KBE application.

2. Representation: A means of representing the knowledge associated with the application using text and graphics is provided. MOKA uses two layers of representation. The first is designed to be very user-friendly and to represent the many different ways in which engineers think about design. This first layer is called the informal model. In this model, the knowledge is classified into five types:

   Illustrations – for recording past experiences, case histories, anecdotal knowledge.

   Constraints – restrictions on the objects or the attributes of an object.

   Activities – the elements of the design process.

   Rules – knowledge used to make choices between activities.

   Entities – the objects that describe the product.

   Each knowledge type has a specific template or form. The set of completed forms, called ICARE (Illustration, Constraint, Activity, Rule, Entity) forms, holds the knowledge description for the KBE application.

   The second layer of representation is the formal model. The knowledge engineer takes the knowledge from the linked ICARE forms and converts it into a UML-style of representation known as MML (MOKA Modelling Language) (Brimble & Sellini, 2000). The formal model has two key elements: the product model and the design process model.

3. Tool: A software tool known as "MOKA tool" that helps users apply the representation and the route map is provided. The tool allows management of the project and module details. It supports creation of both informal and formal models. The tool avoids logical inconsistency when developing the product and process models. The main functions managed by the tool are:
    - Create, modify objects and navigate among the different models (informal model and formal models for product and process)
    - Provide different viewpoints and levels of details
    - Generate a knowledge book

2.2.6 Discussion

The above sections have provided background information on the various knowledge engineering methodologies. This thesis uses Protégé to develop ontologies in OWL for use by systems, namely, Designers' Workbench and ConEditor/ConEditor+. All the knowledge engineering methodologies reviewed in Section 2.2 have placed considerable emphasis on the development of KBSs from sharable and reusable knowledge components using a structured process. The two central activities in this type of development are the engineering of reusable components and the acquisition of domain knowledge. The basic notions of the task, PSM and the decomposition structure from the Task Structure approach have been adopted in recent methodologies such as CommonKADS and MIKE. The entire development process is divided into phases with clearly defined roles in each phase. MOKA has been developed specifically to develop KBE systems in the field of engineering and design. KBE systems differ from KBSs mainly in terms of geometry and the high degree of iteration within engineering design. Rolls-Royce currently adopts the MOKA approach. The knowledge engineering process does not end after one successfully builds a KBS or KBE system. One needs to subsequently maintain the KBS or KBE system throughout its lifecycle. Knowledge Maintenance is discussed further in the next section.

2.3 Knowledge Maintenance

Knowledge Maintenance is concerned with controlling change in a KBS. "Knowledge Maintenance is the process of reflecting over some knowledge-based system in order to handle a new situation" (Menzies, 1999). This process involves updating/refining the contents of the KB so that they are consistent with (a) a set of previously specified task-solution pairs (b) constraints known about the task (c) domain theory/background knowledge. The importance of knowledge maintenance is often underestimated. A brief review of this field is given below.

The issues faced in KB maintenance within engineering were first raised by the XCON[6] configuration system at Digital Equipment Corporation (DEC). "Initially it was assumed that knowledge-based systems could be maintained by simply adding new elements or replacing existing elements. However this simplicity proved to be illusory as indicated by the experience of R1/XCON" (Coenen, 1992). XCON (Soloway *et al*., 1987; Barker & O'Connor, 1989; McDermott, 1993) is a rule-based expert system that configures computer systems. XCON has a very large rule set and underwent constant change (50% of the rules in XCON were changed each year). Given the large number of rules that had complex conditional parts, it became quite difficult to update the rules in the light of new product announcements; it was hard to know if one had found all the rules that need changing. A new methodology called RIME was developed to help in the maintenance of XCON. RIME's philosophy is that complex rules need to be broken down; in particular, multiple tasks need to be factored out and each task needs to be made an explicit process. The objectives of these changes were to reduce the size and complexity of an average rule, and hence better manage the increasing number of rules.

RIME's impact was felt dramatically in the reimplementation of XCON. RIME methodology aided the management of large quantities of rules. When adding new rules, one can now more easily take advantage of existing rules, and thus knowledge reuse results in a major productivity gain. Although the RIME methodology made it easier to maintain, the company's use of XCON was stopped in the early nineties. Maintenance continued to be a major unsolved problem because of

[6] known earlier as 'R1'

the sheer quantity of rules and their size. “It did the work of 75 people but it took 150 to maintain it” was a joke shared at Digital Equipment Corporation (DEC).

A lesson learnt from the XCON system is that: The XCON system did not provide a clear separation between component knowledge and processing knowledge, since constraints on components are often expressed in the production rules. Moreover, it is not clear how a newly added rule would interact with the existing rules in the absence of an explicit problem solving method (Frayman & Mittal, 1987).

Enabling a domain expert to maintain his own knowledge base in a knowledge-based system has long been an ideal for the Knowledge Engineering community. Bultman *et al*. (2000a) report their experience in trying to achieve this ideal in a practical setting, by designing a maintenance tool for a KBS. The KBS considered is a Company Classification System. The task of this KBS is to classify employers into one of fifty-five sectors. Classification of an employer is necessary to determine the level of various insurance contributions for the Dutch social security system, and is based on the primary activity of the employer. Because of a lack of consistency in the classifications various people made, and a decreasing number of experts available in this domain, this KBS was built. The users of this KBS often report bugs and shortcomings of the system and hence, a lot of maintenance is performed on the system. The objective here is to develop a maintenance tool to help domain experts directly implement the required changes in the system without repeated, time-consuming and error-prone interaction with a knowledge engineer. The approach adopted here is to provide domain experts with a conceptual model (comprising both task-model and domain ontology of the system to be maintained) that is close enough to the concepts familiar to them.

Coenen (1992) discusses a methodology for the maintenance of KBSs, which consists of a number of distinct stages. Initially the need for maintenance is passed on to the maintainers in the form of bug reports and change requests. Having established that some maintenance is required, the next stage is to identify, from a global perspective, the section of the KB that will require attention and determine the nature of the maintenance action that will be required. Having determined the immediate nature of the required action, the next stage is to identify, locally, the elements in the KB that will also require attention as a result of the proposed change. The next stage is to consider further maintenance actions required with respect to these elements. For

example, the removal of a rule may require the modification of the rules that call it and are called by it. The next stage is the implementation stage that should be carried out in a consistent and sequential manner. The final stage is the testing phase where the implemented changes are verified and validated. The above methodology was developed as a result of work carried out on MAKE (Maintenance Assistance for Knowledge Engineers) project that was concerned with the specification and development of software tools to support knowledge-based system maintenance. Coenen concludes that the field of KBS maintenance has been sorely neglected and that this is the principal reason why KBSs have failed to gain the general acceptance that was expected when they first came to prominence.

Qian *et al*. (2005) present principles and approaches for knowledge base maintenance in an expert system. Development and implementation of maintenance modules for the expert system for fault diagnosis of an industrial fluid catalytic cracking unit are reported in detail. During the application of the expert system to fluid catalytic cracking unit, new rules need to be added into the existing expert knowledge base from time to time, according to the changes in operating conditions and other circumstances. The new rules added could conflict with the existing rules. Hence, the new rules added are verified and screened by an integrality verification module. Algorithms are proposed for detection of inconsistencies, namely, contradiction, redundancy, subsumption, circulation and reclusion. This improves the efficiency of the knowledge base and ensures that the inference engine works properly and effectively. The following two sub sections present a review of literature, specifically in the fields of verification and validation, and knowledge refinement respectively.

2.3.1 Verification and Validation

Verification and Validation of KBs is at the heart of knowledge maintenance. Knowledge-based systems (KBS) are being used in many application areas where their failures can be costly because of the losses in services, property, or even life (Tsai *et al*., 1999). To ensure their reliability and dependability, it is therefore important that these systems are verified and validated before they are deployed. There is much confusion about the distinction between Validation and Verification but the conventional view is that Verification is a process aimed at demonstrating

whether a system meets its specified requirements; this is often called "building the system right". Validation is a process aimed at demonstrating whether a system meets the user's true requirements; this is often called "building the right system" (Meseguer & Preece, 1995). There have been several systems developed to verify and validate rule-based systems. A brief review of work done in this area is given below:

ONCOCIN: The ONCOCIN Rule Checker (Suwa *et al*., 1982) can be considered as the first verifier referenced in the literature. It detects the following issues in attribute-value rule bases: conflict, redundancy, subsumption and missing rules. Rules are grouped by their concluding attribute, forming a table for each group. Verification issues are tested on each table, by static comparison of rules.

CHECK: The CHECK (Nguyen *et al*., 1985) system was developed to verify the consistency and completeness of knowledge-based systems built using the Lockheed Expert Systems development environment. In addition to conflicts, redundancy and subsumption, the system detected unnecessary if-conditions, circular rules, illegal attribute-values, unreachable conclusions, dead-end if-conditions and goals.

ONCOCIN Rule Checker and CHECK perform only a partial analysis of inconsistency (conflict) and redundancy because they test these issues locally, comparing static pair of rules and ignoring rule chaining. This problem was solved by subsequent systems such as KB-REDUCER (Ginsberg, 1988) and COVADIS (Rousset, 1988). The KBs considered by all these systems were forward-chaining propositional rule bases.

COVER: COVER (Preece *et al*., 1992) was another tool for verification of rule- based systems that detected a wider range of anomalies. COVER carries out seven verification checks: redundancy, conflict, subsumption, unsatisfiable conditions (rules that cannot be fired, missing values), dead-end rules, circularity and missing rules. The rules had to be written in, or converted to, a language based on first-order logic. The worst-case complexity after theoretical analysis for rule checks is $O(n^2)$ for n rules, as every rule in KB is compared against all other rules. This system was applied to many real world KBs and it detected genuine and potentially serious faults in each

KB to which it was applied. This system was extended to verifying multi-agent systems and became known as COVERAGE (Preece, 1999).

2.3.2 Knowledge Refinement

One of the main aims of knowledge refinement is to improve the performance of an imperfect (faulty) KB. There have been various tools developed to enable knowledge refinement. Some examples are given below:

TEIRESIAS: TEIRESIAS (Davis, 1979) helped domain experts detect shortcomings and also make refinements in the KB of one of the earliest expert systems, MYCIN (Shortliffe, 1981). If the expert (physician) did not agree with the output (diagnosis) of MYCIN, TEIRESIAS enabled the expert to identify the discrepancies by systematically tracing the line of reasoning.

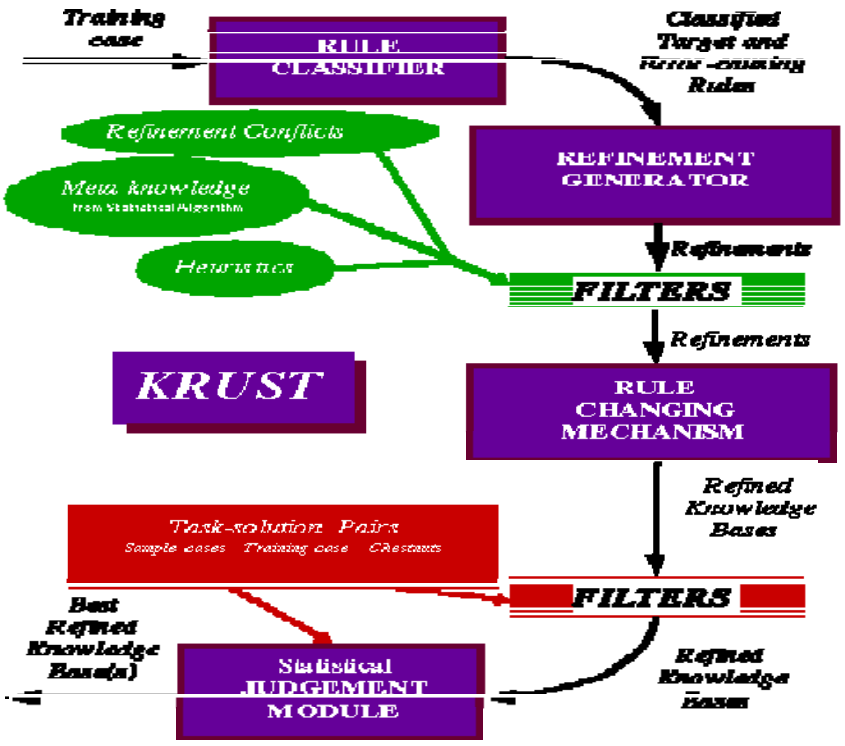


Figure 2.5: KRUST Refinement System

Source: Craw & Sleeman (1995)

KRUST: KRUST (Craw & Sleeman, 1990, 1995) is an automated refinement system for knowledge-based systems. The system is presented with a training case, where the expert's conclusion conflicts with the KBS's conclusion. KRUST implements a set of possible refinements to the KB so that the KBS now suggests the expert's conclusion. Various filters use evidence suggested by other task-solution pairs to remove ineffective refinements. When KRUST terminates, the expert is usually given a single refined KB that KRUST has judged to be the best. A flowchart showing the process in KRUST is given in Figure 2.5. An important assumption is that the KB needs only minor "tweaking" rather than a major overhaul.

STALKER: STALKER (Carbonara & Sleeman, 1999) is an extension of KRUST. It has two major enhancements. Firstly, the refinement suggested has been augmented by the introduction of inductive refinement operators. Secondly, the testing phase has been greatly speeded up by using a Truth Maintenance System. STALKER was tested on two real-world rule bases and proved to be 50 times faster than KRUST.

CONREF: CONREF (Winter *et al*., 1998) is a system that was developed to help British Aerospace make efficient use of their inventory of fasteners. Constraint satisfaction techniques are used to determine which fasteners are suitable for a particular application, given a design KB. Additionally knowledge refinement techniques are used to refine the KB, if the domain expert (an Airbus designer) disagrees with the retrieved fasteners. The system is also able to generate reports, describing the frequency of retrieval of specific fasteners and the contexts of their use.

TIGON: TIGON (Sleeman & Mitchell, 1996) is a system that helps in the diagnosis of turbine faults by providing diagnostic information which helps an engineer detect the nature and location of faults. The system consists of four co-operating subsystems – a Learning Module which learns the fault detection and diagnosis models; a Monitoring Module that monitors the turbine's behaviour and detects when it is behaving abnormally; a Diagnosis Module that tries to determine what is causing the abnormality; and a Transformation Module that modifies the knowledge bases so that they are applicable to further turbines. If any inconsistencies are reported by the system, the expert is asked to suggest changes to the set of cases, the causal graph or the descriptors in the data set.

REFINER++: REFINER++ (Aiken & Sleeman, 2003) is a system that has been developed to help domain experts classify data, and has largely been applied in the medical domain. The domain expert is required to specify which category each case belongs to; Refiner++ then infers a description for each of the categories and reports inconsistencies that exist in the dataset. An inconsistency occurs when a case matches a category other than the one to which the expert has assigned it. If inconsistencies have been detected, the system suggests ways of dealing with the inconsistencies by refining the dataset; however, it is the domain expert who selects the actual refinements to be applied.

ReTAX++: ReTAX++ (Lam *et al*., 2005; Lam *et al*., 2008) is a system that has been developed to help knowledge engineers browse and resolve inconsistencies present in ontologies. The system uses graph-based algorithms to detect which relationships among concepts cause inconsistencies and provides the knowledge engineer with various options to correct them.

2.3.3 Discussion

The review of literature in the field of knowledge maintenance has reported on issues faced during maintenance and also on some systems that have been developed to support the verification, validation and refinement of rule-based systems. Verification, validation and refinement are three important activities in knowledge maintenance. An important lesson that can be learnt is that the initial phases of knowledge acquisition and knowledge modelling in knowledge engineering have considerable effects on the maintenance phase. This is particularly evident from the problems faced by the R1/XCON configuration system. It is important to explicitly record the contexts in which each rule is applicable, during the KA phase. Recording the contexts should help identify all the rules that need to be updated during maintenance. This thesis investigates how an explicit representation of contexts together with the engineering design rules can help in the maintenance of a KB. Knowledge modelling also plays an important role in the maintenance phase. As indicated in the R1/XCON system, if no clear separation is provided between component knowledge and processing knowledge, it can cause serious problems during the maintenance of a system.

The following section provides a review of work in the field of engineering design. The thesis has used engineering design as an application domain.

2.4 Engineering Design

In engineering design literature, three phases of design are generally identified: conceptual design, embodiment design and detailed design (Pahl & Beitz, 1995; O'Sullivan, 2002b; Ullman, 2003). During conceptual design, the designer searches for a set of broad solutions to a design problem, each of which satisfies the fundamental requirements for the desired product. The embodiment phase of design is traditionally regarded as the phase in which an initial physical design is developed. This initial physical design requires the determination of component arrangements, initial forms and other part characteristics. The detailed phase of design is traditionally regarded as the phase during which the final physical design is developed. The final physical design requires the specification of every detail of the product in the form of engineering drawings and production plans.

2.4.1 Constraints in Engineering Design

Most decisions in daily life involve considering some form of restriction on the choices that are available. For example, the destination to which someone travels has a direct impact on their choice of transport and route: some destinations may only be accessible by air, while others can be reached using any mode of transport. Formulating decision problems in terms of parameters and the restrictions that exist between them is an intuitive approach to modelling them. These general restrictions can be referred to as "constraints" (O'Sullivan, 2002b).

Engineering Design is constraint-oriented and much of the design process involves the recognition, formulation and satisfaction of constraints (Serrano & Gossard, 1992; Lin & Chen, 2002). A constraint here refers to a design rule that needs to be satisfied. Constraints are continually being added, deleted and modified throughout the development of a new product. Design begins with a functional specification of the desired product: a description of properties and conditions that the product should satisfy (i.e. constraints). The original set of functional requirements are augmented, changed and/or refined as the design solution evolves. The resulting

constraint set may contain conflicting and/or unrealizable requirements. The management of these constraints throughout the evolving design involving all the phases is a non-trivial task. The constraints are often numerous, complex and contradictory.

Particularly, in more complex designs, where form, function and physics interact strongly, it is difficult to keep track of all relevant constraints and parameters, and to understand the basic design relationships and tradeoffs. Constraint-based approaches to supporting conceptual design have been reported in the literature for quite a number of years (Gross *et al*., 1988; Serrano & Gossard, 1992; O'Sullivan, 2002b). Effective tools for constraint management are of great importance in knowledge-based systems for conceptual design. They provide designers with assistance during the early stages of design. In addition, they will help close the gap between novice designers and experienced designers. The interactive constraint-based approach presented in O'Sullivan (2002b) is based upon an expressive and general technique for modelling: the design knowledge which a designer can exploit during a design project; the life-cycle environment which the final product faces; the design specification which defines the set of requirements that the product must satisfy; and the structure of the various schemes that are developed by the designer. A computational reasoning environment based on constraint filtering (Bowen & Bahler, 1992; Bowen, 1997) is proposed as the basis of an interactive conceptual design support tool. Using such a tool, the designer can be assisted in developing and evaluating a set of schemes that satisfy the various constraints that are imposed on the design. In particular, the designer can be assisted in synthesising a number of alternative schemes for the required product. The consistency of each scheme is constantly monitored, as is the consistency of each scheme with respect to the design specification and the other schemes that have been developed.

There have been several constraint-based applications that involve constraint solving during post-conceptual phases of design. The CADET system was developed as a computer tool to support embodiment design (Thorton, 1996; Yao, 1996). CADET consists of a generic database of components that can be used to develop a constraint-based model of the geometry of the product being designed. The IDIOM system uses constraint solving on geometric parameters for floor-planning (Lottaz *et al*., 1998). SpaceSolver uses the notion of solution spaces, defined by sets of constraints on continuous domains, as a basis for supporting interactive design (Lottaz

et al., 2000). Many constraint-based systems reported in the literature have been developed for supporting reasoning about purely geometric aspects of design for use with CAD systems (Bhansali *et al*., 1996; Shimizu & Numao, 1997; Gao & Chou, 1998a, 1998b). These systems have been developed to address aspects of the design process that are too specific to geometric CAD to be reviewed in depth here. However, to solve constraints in design, representation of constraints still remains a challenge facing the design engineers (Lin & Chen, 2002).

One of the first attempts to manage constraints for automation of computation in engineering applications was the work done by Harry (1962) and Steward. Since then there has been considerable amount of work done on the representation, use and management of constraints including the development of rule-based systems. Rule-based (expert) systems have been applied to assist in a variety of engineering design tasks such as: design for VAX computer systems by Digital equipment Corporation (this company was acquired in June 1998 by Compaq, which subsequently merged with Hewlett-Packard in May 2002): R1—(McDermott, 1982); design system for small computers: M1—(Brown & Chandrasekaran, 1985); design of VLSI circuits: VEXED—(Mitchell *et al*., 1985); configuration of microcomputer systems: COSSACK—(Frayman & Mittal, 1987); design of air cylinders: AIR-CYL—(Brown & Chandrasekaran, 1989); design of facilities on construction sites: SightPlan—(Tommelein *et al*., 1991); design of elevators: VT—(Marcus *et al*., 1992), design of buildings: HI-RISE—(Maher, 1988); CONGEN—(Sriram, 1997); design of paper-feeding mechanisms of photocopiers: PRIDE—(Koo *et al*., 1998), design of pneumatic systems: PNEUDES—(Shin & Lee, 1998). Rule based systems have been shown to be very difficult to maintain and in many cases had to be completely rewritten so as to function in a production environment (Soloway *et al.*, 1987).

Frayman & Mittal (1987) classified constraints into explicit constraints and implicit constraints. Explicit constraints enumerate a set of possibilities to be selected from, for example, the word processing package WRITER requires the operating system DOS version 2.1 or 3.1. Implicit constraints do not contain explicit enumeration of alternatives but contain enough information to reconstruct such a set of all currently available components, for example, the word processing package WRITER requires the operating system DOS version 2.1 or later versions. They pointed out that processing of implicit constraints is more complicated than the

processing of explicit constraints, but has benefits for the maintainability of the system.

It became important to represent the defaults and preferences declaratively as constraints, rather than encoding them in the procedural parts of the program (Borning *et al*., 1989). In most cases, domain-oriented or method-oriented tools (in the form of templates) were provided to capture constraints/rules from the domain experts. The cost of developing such tools was high and became an issue, especially when their restricted scope is taken into account (Eriksson *et al*., 1995a).

The use of constraint processing techniques for supporting configuration design has been widely reported in the literature (Barker & O'Connor, 1989; Wielinga & Schreiber, 1997; McGuinness & Wright, 1998b, 1998a; Sabin & Weigel, 1998; Carnduff & Goonetillake, 2004). Configuration can be regarded as a special case of engineering design. The key feature of configuration is that the product being designed is assembled from a fixed set of predefined components that can only be connected in predefined ways. The core of the configuration task is to select and arrange a collection of parts in order to satisfy a particular specification. The growing interest in configuration systems is reflected by the level of interest reported from industry. The role of constraint-based configurators has been reported in a number of reviews (Sabin & Weigel, 1998). The configuration problem can be naturally represented as a CSP. In general, a configuration problem can be formulated as a CSP by regarding the design elements as variables, the sets of predefined components as domains for each of the design elements and the relationships that must exist between the design elements as constraints.

Constraints can also be used to state the compatibility of particular arrangements of components and connections. One of the earliest works in the field of constraint-based support for configuration was based on dynamic constraint satisfaction (Mittal & Falkenhainer, 1990). The key characteristic of dynamic constraint satisfaction problems is that not all variables have to be assigned a value to solve the problem. Depending on the value of particular variables, other variables and constraints may be introduced into the network. Inspired by this approach, the use of constraint processing for configuration problems in complex technical domains emerged (Haselbock & Stumptner, 1993; Fleischanderl *et al*., 1998). The Designers' Workbench mainly deals with problems that lie in the domain of configuration (Fowler *et al.*, 2004). The Designers' Workbench uses an ontology to represent

elements in a configuration task. The Designers' Workbench has concentrated on checking that the constraints are satisfied by a configuration produced by a human designer, rather than finding a solution. This has implications for tractability, in that solving a CSP is a NP-complete problem, whereas checking a solution can be done in polynomial time. The system has been implemented so that the human designer is free to use his or her engineering expertise to override constraints that are not deemed applicable to the current situation.

Description logics have been used to develop commercial configurators in telecommunication and automotive industries (McGuinness & Wright, 1998b, 1998a; Rychtyckyj & Reynolds, 2000). Concepts can be defined corresponding to the classes of an ontology and individuals correspond to instances. Forward chaining rules can be defined, which are associated with concepts but are applied only to individuals. These rules are used to enforce constraints that are generic, i.e. defined on classes of objects, rather than to specific individual objects. Description logics provide logical completion of information and can detect any inconsistencies formed in the knowledge base. However, description logics have limited expressive power.

Some of these description logic-based systems (Prose, DLMS) have been used in industries since the 1990s. One such system is Ford's Direct Labor Management System in the very dynamic domain of process planning for vehicle assembly. The maintainability of the systems can become very difficult over time due to changes in the following areas: the external business environment, the processes and physical concepts being modelled, and the underlying hardware and software architecture. The experience of using DLMS indicated that user editing of the knowledge base has not been very successful either from the user viewpoint or from the developer side. The editing of the KB requires a deeper understanding of the knowledge representation scheme than is needed for updating a spreadsheet or database. This necessitated the creation of a complex user interface that many users found difficult to master. In addition, most of the user changes to the system consisted of lexical information, which required properties such as parts of speech to be specified. This was often done incorrectly and introduced errors into the system. This meant that the developers had to spend time reviewing and correcting user edits in order to catch these types of errors. Other problems were caused by users adding misspelled terms, alternate spellings, and different abbreviations for the same terminology. The process of checking this kind of errors was manually intensive.

Another approach to develop configurators was to combine constraints and ontologies. Junker & Mailharro (2003) describe a system, ILOG Configurator, that combines the power of description logic (to describe the parts used in a configuration), with constraint programming (to solve the configuration problem). The description logic uses classes that are either abstract or concrete. Concrete classes correspond to actual parts (e.g., bolt) while abstract classes represent features (e.g., hole). Properties are used to describe the instances of a class. Generic constraints can be defined in a constraint language that allows numeric and symbolic constraints. To solve a configuration problem, a description logic representation of the class hierarchy and the constraints are converted into a constraint satisfaction problem. Laburthe (2003) extends CSPs to cases where variables have domains that are taken from a hierarchy. This differs from the approach of other systems such as the Designers' Workbench, ILOG (Junker & Mailharro, 2003) and Prose (McGuinness & Wright, 1998b, 1998a) in that these systems are concerned with constraints over values of properties of instances. Laburthe's approach aims to find the entities in a hierarchy that will satisfy the constraints.

Increased complexity and size of configurator knowledge bases can make the user of a configuration system increasingly challenged to find the source of the problem whenever it is not possible to produce a working configuration, i.e., the configuration process is aborted. Ultimately, the cause of an abort is either an incorrect knowledge base or unachievable requirements. Automated support of the debugging process of such KBs is a necessary prerequisite for effective development of configurators. Felfernig *et al*. (2004) show that this task can be achieved by consistency-based diagnosis techniques. They basically employ model-based diagnosis techniques using positive and negative examples for this purpose. This means that positive configuration examples should be accepted by the configurator whereas negative examples should be rejected. The examples therefore play a role much like what is called a test case in software engineering, i.e. they provide an input such that the generated output can be compared to the tester's expectations. Once a test has failed, diagnosis can be used to locate the parts of the KB responsible for the failure. Such parts will typically be constraints that specify legal connections between components, or domain declarations that limit legal assignments to attributes. These constraints and declarations, written as logical sentences will serve as diagnosis components when the problem is mapped to the model based diagnosis approach.

A second type of situation where diagnosis can be used is the support of the actual end user where the user's requirements are not satisfied even though the knowledge base is correct, e.g., because she/he placed unrealistic restrictions on the system to be configured. An algorithm has been proposed for computing diagnoses. The overall time for diagnosing a problem is split into time needed for consistency checking (solution search for the configuration problem), time for conflict generation and diagnosis time. The experimental results showed the suitability of the approach to commercial configurator development environments. It has to be noted that systems such as the Designers' Workbench differ from the configurators used in the above approach because Designers' Workbench performs constraint checking and do not involve constraint solving (solution search for the configuration problem). An interesting outcome of their experiments is that in typical declarative configuration knowledge bases, there are only few interdependencies among constraints, i.e. the size of the minimal conflicts is typically very small (up to three or four constraints).

Goonetillake & Wikramanayake (2004) propose a framework for the management of evolving constraints in a computerized engineering design environment. The evolving constraints are embedded in a class definition. There is a facility to incorporate constraint evolution. The framework is based on a Constraint Version Object (CVO). Each CVO contains a set of integrity constraints. CVOs are affected by (i) modification(s) to existing constraints (ii) introduction of new constraints (iii) omission of previously used constraints (iv) any combination of (i) – (iii). A new CVO (child) contains only the changes made to its parent CVO constraint set. There is a mechanism in the child CVO to inherit constraints from the parent, redefine and alter constraints that were already defined in the parent and leave out constraints defined in the parent. Thus, a chain of CVOs is generated with the latest CVO usually becoming the default CVO. This facilitates the maintenance of constraint evolution history. Automatic validation is performed when a new CVO is produced. One can retrieve the set of constraints applicable to a particular version. The versions are stored and managed by a DBMS. Thus, the framework to manage the evolving constraints in an engineering design environment is proposed. Constraints are updated and not overwritten when they evolve. However, the framework has limited expressivity. One cannot express declarative first-order logic quantified constraints and it is highly domain specific. A considerable amount of work would have to be invested to adapt the framework to another domain. No information about

the context in which the constraints are applicable is recorded by the system. This could lead to problems during maintenance and may result in inappropriate constraints being applied. Also, there is no maintenance support provided to detect any conflicts, redundancy or subsumption between constraints. The research work reported in this thesis aims to address such problems.

2.4.2 Concurrent Engineering and Integrated Product Teams

Concurrent Engineering which is sometimes called Simultaneous Engineering or Integrated Product Development (IPD) was defined by the Institute for Defense Analysis (IDA) in its December 1988 report 'The Role of Concurrent Engineering in Weapons System Acquisition" as

"Concurrent Engineering is a systematic approach to the integrated, concurrent design of products and their related processes, including manufacture and support. This approach is intended to cause the developers, from the outset, to consider all elements of the product life cycle from conception through disposal, including quality, cost, schedule and user requirements." (Winner *et al*., 1988; Cleland, 2004)

Increasingly, it is being realised that success of product development in industry requires integration between the various phases of the product life cycle. One of the key aspects of this integration is that, during the design of an artefact, due consideration should be given to facilitating the down-stream phases of the life cycle. This is frequently known as "Design for X" (or DFX), where the X ranges over such issues as manufacturability, serviceability, assembly and so on (Bowen, 2001). For example, the design for manufacture (or DFM) is defined as establishing the shape of components to allow for efficient, high-quality manufacture. For any component, many manufacturing processes could be used in its manufacture. For each manufacturing process, there are design guidelines that, if followed, result in consistent components and little waste. A detailed literature survey conducted on the state of the art of the concurrent engineering technique in automotive industry revealed that the technique is very powerful in achieving successful products in the automotive industry (Sapuan *et al*., 2006). Sapuan and his colleagues stated that the

companies who adopted this technique have gained tremendous benefit in terms of reduced time-to-market, low cost and improved quality.

Concurrent Engineering attempts to maximise the degree to which design activities are performed in parallel. A number of researchers in the constraint processing community have developed constraint-based technologies that support integrated approaches to product development (Bowen & Bahler, 1992; Bowen, 2001; O'Sullivan, 2002a). The analogy between the computational concept of a constraint and the concurrent engineering concept of mutually constraining influences between different phases of the product life cycle suggests that constraint networks may be the right basis on which to develop a generic architecture for software to support concurrent engineering. Constraints can be used to express in an explicit way the mutual restrictions exerted on each other by artefact functionality, component/material properties, and life-cycle processes (Bowen, 2001).One of the critical issues that must be addressed in supporting integrated design is the issue of conflict resolution and negotiation. Constraint-based approaches to managing conflict in collaborative design systems have been reported (Bahler *et al*., 1994; Haroud *et al*., 1995; Abdalla, 1997, 1998; Lottaz *et al*., 2000). Traditional conflict resolution techniques in constraint-based models of the design process use backtracking and constraint relaxation.

The Designers' Workbench has been developed with a view to support concurrent engineering. In the Designers' Workbench, a domain ontology can be used to incorporate different aspects of a product life-cycle. Design rules are expressed as constraints over a domain ontology. Typically, complex engineering artefacts are designed by teams who may not be located in the same building or even city. Designers in Rolls-Royce, as in many large organizations, work in teams. Thus, it is important when a group of designers are working on aspects of a common project, that the subcomponent designed by one designer is consistent with the overall specification, and with those designed by other members of the team. Additionally, all designs have to be consistent with the company's design rule book(s). Making sure that these various constraints are complied with is a complicated process and so the Designers' Workbench seeks to support these activities. Constraint violations are reported to the human designer together with a link to the source document describing the constraint. The designer could then adjust the appropriate property values using the GUI to resolve the constraint violations. The system has been implemented so that

the human designer is free to use his or her engineering expertise to override constraints that are not deemed applicable to the current situation (Fowler *et al.*, 2004). Hence, the main difference between Designers' Workbench and other previously reviewed constraint-based systems to support concurrent engineering is that Designers' Workbench does not perform constraint solving or employ any conflict resolution strategies. The Designers' Workbench performs constraint checking, reports any constraint violations and facilitates the human designer to resolve the constraint violations.

Collaborative engineering design activities are influenced not only by the technological factors, but also by the social interactions among various stakeholders with different perspectives. An article by Lu & Cai (2001) describes a generic collaborative design process model based on a socio-technical design framework that is suitable to represent, analyse and evaluate the collaborative design activities. Lu and Cai describe collaborative design process as a perspective evolution process. They emphasise that while the technical decisions are dealing with "what" and "how", the social interaction, which is about "why" and "who" is indispensable to the negotiations among the collaborative design decisions. They point out that most of the conflicts in the collaborative design are caused by the discord among the stakeholders' perspectives. Hence, in collaborative design processes, the influence of one's decision making in a specific domain to others' decision making in different sub problems should be represented, analysed and evaluated. They use Petri nets as topological process representation tools and adapt them for collaborative design process modelling. A methodology of design conflict management is developed with the design process representation model. After that, a prototype collaborative design support system, which is a computer implementation of the methodology, is discussed. Similarly, the paper by Veeke *et al*. (2006) defines a conceptual interdisciplinary model that can be used by all domains involved in the design of an industrial system. The model serves as a common frame of reference to support communication and decision making by different mono disciplinary approaches. The model is also used to record conditions, decisions and assumptions that lead to the final design.

The article by Crowder *et al*. (2003) presents a future socio-technical scenario to capture, share and reuse knowledge within the engineering design environment. In the scenario, it is assumed that the technical elements of the future design

environment have been embodied in an application termed KTfD (Knowledge Tools for Designers). KTfD includes tools such as Tablet PCs with handwriting recognition software and software to resolve sketches. KTfD also provides interfaces to specific engineering packages. KTfD is able to access information including the full range of office and data analysis tools from anywhere in the design office through the local wireless network. The use of KTfD would increase accountability by making the input of a designer visible to other designers and allow decisions to be traceable. However, the presumption that all processes in the future should be based on IT systems was strongly resisted during their discussions with designers. It was felt that there is a preference for face-to-face interaction and social support, rather than using technology, such as teleconferencing. One of the key issues for them was for any system to be accurate and reliable. In addition, in many cases the designer may not fully understand exactly what is required and therefore may not know what type of expertise or information is required to resolve the problem. With a human based system, the question and problem can be discussed and interpreted for the user, making it more likely to proceed with maximum trust. Wallace & Ahmed (2003) and Aurisicchio *et al*. (2006) have performed studies on how engineering designers obtain information. Two main questions are addressed: how do designers currently obtain their information and what is the best way to help novice designers obtain appropriate information. The studies showed that documents were very seldom used as a source of design information and for around 90% of information requests designers contacted another person. In addition, novice designers were unaware of the strategies adopted by experienced designers and failed to ask the right questions to the right people.

Recent work done by Fruchter *et al*. (2007) at Stanford present an integrated framework that enables collaborative design exploration, knowledge reuse and decision making. A working prototype, called CoMem-iRoom that leverages and integrates two software environments, CoMem and iRoom is presented. CoMem (Fruchter & Demian, 2002) is a collaboration technology that facilitates context-based reuse of corporate knowledge in a single-user setting for the architecture, engineering, construction teams and individuals in the design process. CoMem allows for context based visualisation and exploration of large hierarchical project databases. CoMem uses a map metaphor for the overview. The area on the map allocated to each item is based on a measure of how much knowledge this item encapsulates, that is, how

richly annotated it is, how many times it is versioned, how much external data is linked to it. Each item on the map is colour coded by a measure of relevance to the designer's current task. Currently, this relevance measure is based on textual analysis of the corporate memory using the latent semantic indexing (LSI) algorithm (Landauer & Dumais, 1995; Demian & Fruchter, 2005). The iRoom architecture (Johanson *et al*., 2002) is a technology that enables communication between discipline-specific control applications running on multiple machines. By making CoMem the nodal application of the iRoom architecture, they extend the contextual visualisation and exploration functionality provided by CoMem from a single-user to a multi-user interactive setting, thereby enabling collaborative exploration in project group meetings and knowledge reuse discussions.

Other recent work includes a general type net-based collaborative product design support system called CoDesign Space system designed by Tian *et al*. (2007). The system aims to satisfy the requirements of geographically dispersed collaborative design by integrating several collaborative design support tools that can be used independently. The several collaborative design support tools that can be integrated include a collaborative virtual assembly tool, a collaborative viewing and markup tool, a conflict-management tool, a visual document-management tool, a collaborative task management tool and a collaborative design resource repository management tool. The sharing and visualisation of product information are the foundation of Internet-based collaborative design and manufacturing (Zhang *et al*., 2004). CoDesign Space uses XML and VRML technologies to resolve the sharing and integration problem of heterogeneous product model information. VRML is a language that enables information sharing and integration among geometry models from heterogeneous CAD systems. VRML is more suitable for transfer over the internet when compared to STEP based CAD model files that are often very large. Collaborative work can also be realised by the communication and management mechanism of agents (Cutkosky *et al*., 1993; Anumba *et al*., 2001; Wu *et al*., 2006).

Ontologies are increasingly becoming important in the fields of intelligent searching on the web, knowledge sharing, reuse and management. There has been an increasing number of research projects applying ontological techniques in the context of product development (Moore *et al*., 1999; Roche, 2000; Ciocoiu *et al*., 2001; Lin & Harding, 2003; Lee *et al*., 2009). The paper by Cheung *et al*. (2006) reports on utilizing ontologies to share manufacturing knowledge during product development in

a collaborative and distributed manner. Ontologies are particularly useful in a collaborative and distributed environment because they provide a shared and common understanding (or agreed vocabulary) of a domain that can be communicated between people and application systems. Apart from providing a common understanding, Valarakos *et al*. (2004) states that ontologies can be used to facilitate dissemination and reuse of information and knowledge. The research work reported in this thesis uses an ontology to represent domain knowledge. Design rules are expressed as constraints over the domain ontology. Inferencing over the domain ontology is done to detect various refinements (inconsistency, subsumption, redundancy and fusion) between pairs of consraints. Thus, ontologies play an important role in supporting the maintenance of constraints. More details regarding the use of ontologies in supporting the maintenance of constraints can be found in subsequent chapters of this thesis.

2.4.3 Design Rationales

A large amount of design information that is generated during design does not get recorded in formal design documentation. Some of this information is often referred to as design rationale, but can include any sort of knowledge of the who, what , when, where, why, and how of design (Richter & Abowd, 1999). Rationale can include assumptions made about the system, the alternatives considered and the reasoning behind decisions. Some other definitions of design rationale from literature are as follows:

"Design rationale means information that explains why an artefact is structured the way that it is and has the behaviour that it has" (Conklin & Begeman, 1988)

"A design rationale is an explanation of how and why an artefact, or some portion of it, is designed the way it is" (Gruber & Russell, 1991)

"A design rationale is a representation of the reasoning behind the design of an artefact" (Shum & Hammond, 1994)

"Design rationale means statements of reasoning underlying the design process that explain, derive and justify design decisions" (Fischer *et al*., 1995)

"Design rationales include not only the reasons behind a design decision but also the justification for it, the other alternatives considered, the tradeoffs evaluated, and the argumentation that led to the decision" (Lee, 1997)

While all these definitions have their merits, Richter & Abowd (1999)'s description covers all aspects of design rationale. The study of design rationale spans a number of diverse disciplines, touching on concepts from research communities in mechanical design, software engineering, artificial intelligence, civil engineering, human factors and human-computer interaction research (Hu *et al*., 2000). It is commonly accepted that the IBIS (Issue-Based Information System), proposed by Rittel (1972) is the first formal presentation of design rationale (Shum, 1991). The initial IBIS was based upon planning and social policy formulation problems. However, the demand for a formal method of system analysis and design from the "software engineering" and "human computer interaction" communities appeared to be driving much of the design rationale research and its application since 1980s (Conklin & Burgess, 1991). It was subsequently introduced into the engineering design community due to the demand for computer support in engineering design (Guihua *et al*., 2002).

The paper by Clarkson & Hamilton (2000) discusses the need for computer support in aerospace design. They propose a parameter-based model of design that has been founded on the assumption that a design process can be constructed from a predefined set of tasks. They have stressed the importance of capturing the implicit knowledge that refers to the order in which the information is processed: "In developing a knowledge-based system to support the engineer in aerospace design, the capture and modelling of explicit knowledge is itself not sufficient. The context in which the knowledge should be applied is of equal importance. A knowledge based system must include not only the explicit knowledge required but also provide guidance on the order in which the information is used." (Clarkson & Hamilton, 2000).

Various tools have been developed to capture design rationales. This information is valuable for design evaluation, reuse and maintenance. A brief review of work done in the area of design rationales is given below:

Regli *et al*. (2000) provide a survey of recent research in the area of design rationale. This survey has discussed design rationale systems from five perspectives: knowledge representation, rationale capture, rationale retrieval, technical approach and application domain. A number of recent design rationale systems, including IBIS, JANUS, COMET, ADD and REMAP are analyzed. A table providing a summary of

the description of some of these systems is shown in Figure 2.6. Issue-based representation involves articulating issues as questions, with each issue followed by one or more positions that respond to the issue.

System Name Acronym	Knowledge Representation	Knowledge Capture	Knowledge Retrieval	Approach	Design Domain	Year
IBIS [17]	Issue-based	UI	Navigate	PO	Generic	1970
PHI [20]	Extending IBIS	UI	Navigate	PO	Generic	1987
QOC [5]	Design Space Analysis	UI	Navigate	PO	Generic	1990
DRL [1]	Representing elements of decision making	UI	Navigate	PO	Generic	1991
CRACK [16]	N/A	Auto	Trigger	FO	Kitchen	1989
VIEWPOINTS [16]	IBIS	N/A	Navigate	FO	Kitchen	1989
JANUS [21]	PHI	Auto	Hybrid	FO	Kitchen	1989
IBIS-style browser [22]	IBIS	Auto	Navigate	PO	Generic	1991
COMET [23]	LOOM	UI	Navigate	FO	Sensor-based tracker software	1992
ADD [24]	Argumentation & Model-based	UI	Trigger	FO	HVAC	1992
REMAP [25]	IBIS	UI	Query	PO	Generic	1992
REMAP/MM [26]	IBIS	Auto	Query	PO	Generic	1995
ADD+ [27]	Rhetorical Structure	UI	Query	PO	HVAC	1997
HOS [18]	PHI	Auto	Trigger	PO	Generic	1997
PHIDIAS [18]	PHI	Auto	Trigger	FO PO	2D, 3D	1997
KBDS-IBIS [14,15]	IBIS	UI	Query & Navigate	FO PO	Chemical Plant	1997
DRIVE [28]	PDN	UI	Query	FO	Building	1997
DRARS [29]	QOC	N/A	N/A	FO	Building	1995
KRITIK [30,9]	SBF	UI	Query	FO	Mechanical	1993
IDIS [31]	IBIS	UI	Navigate	FO	Chemical Plant	1998
RCF [32]	N/A	Auto	N/A	PO	N/A	1999

Capture Method: User-Intervention (UI) or Automatic (Auto) *Representation Method*: Feature-Oriented (FO) or Process-Oriented (PO) *Retrieval Method*: Navigate, Query, Trigger or Hybrid.

Figure 2.6: Summary of a survey of Design Rationale systems

Source: Regli *et al.* (2000)

The capture methods are user-intervention (UI) (in which designers are required to input or record the design discussions, decisions and reasoning themselves) and secondly automatic (auto). The different retrieval mechanisms involve:

(a) navigation: allowing designers to explore design rationale by traversing from one node to another through existing links.

(b) automatic triggering: detecting or monitoring certain conditions according to the design context and retrieving design rationales automatically.

(c) query-based: allowing designers to pose queries and retrieve the required design rationales.

(d) hybrid: providing a combination of automatic triggering and navigation mechanisms.

The two main approaches to building design rationale systems are:

(a) process-oriented (PO): emphasize the design rationale as a history of the design process; design rationales are merely descriptive and generally informal; concerned with the initial design stage, as design progresses from the requirements to a conceptual design.

(b) feature-oriented (FO): representation of artefacts and the body of established rules governing the design process; design rationales have a logical structure and are generally formal; concerned with the detailed design stage, when the design process is more constrained by the rules in the field or domain knowledge.

Justification

an ancillary divider for the oscillator for the cpu should exist
because of the following constraint(s)
(49) every crystal oscillator must satisfy the following:
ideally, the oscillation frequency of the crystal oscillator =<
the maximum frequency testable at the facility where the board will be tested;
otherwise, an ancillary divider for the crystal oscillator
must exist and must be a frequency divider;
(50) every test_facility must satisfy the following:
the maximum clock frequency testable at the test_facility =
the maximum clock speed that can be handled by the equipment at the test_facility;
and because of the following parameter value(s):
the equipment at the facility where the board will be tested = Erdsys TX;
the oscillation frequency of the oscillator for the cpu = 250.

the maximum clock speed that can be handled by Erdsys TX = 98.

the equipment at the facility where the board will be tested
was established according to the perspective taken by test engineers.

(96) the oscillation frequency of the oscillator for the cpu = 250
because you said so.

Figure 2.7: An example of a rationale generated by KLAUS4

[Source: Bowen (2001)]

Garcia & Howard (1992) discussed design rationale approaches that divides the process-oriented approach into two categories: action-based (e.g., RCF (Myers *et al*., 2000)) and argumentation-based (e.g., DRed (Bracewell & Wallace, 2003; Aurisicchio *et al*., 2006)). When focusing on each component or phase of the design process, the former corresponds to how it is done, and the latter corresponds to why it is done. The advantages and shortcomings of the different design rationale systems depend on the trade off between ease of capture and the explanatory power of the rationale. Action-based design rationales are easy to capture and do not require much intervention of the designer while argumentation-based design rationales are difficult to capture.

Constraints can form a part of the rationales associated with the design decisions taken by designers. A typical rationale is of the form: "A component X exists in the design because of the need to satisfy constraint Y." The ability to capture and use this type of design rationale in concurrent engineering has been referred to as Design Rationale Management by Bahler & Bowen (1992) and Bowen (2001), who describe a constraint-based design advice system that generates machine-generated suggestions to support coordination among multiple design engineers. An example of this type of rationale is shown in Figure 2.7. The design advice system called KLAUS4 is written in a generic language, Galileo2, to assist in the concurrent engineering design of printed wiring boards. The system captures perspectives of several members of the design team, including designers, manufacturing engineers, test engineers, and maintains a set of dependency records to support negotiation between various members of a problem solving team. The protocol for negotiation is based on identifying alternative ways in which conflicts can be overcome and suggesting these alternatives to the parties involved, the suggestions being ranked in accordance with the relative preferences (priorities) of the constraints involved in the conflict. By choosing among the suggestions offered, the designer can disable a particular constraint. Whenever a designer disables a constraint other than the one he/she previously asserted, he/she is required to enter a free-text rationale for his/her action, which is saved for possible use in a design review.

Bracewell & Wallace (2003) and Aurisicchio *et al*. (2006) describe DRed (Design Rationale editor), an IBIS-based software tool that allows designers to record their design rationales at the time the design issues are being considered. DRed

consists of a graphical structure to present the issues addressed, options considered and associated arguments for and against each one. Figure 2.8 shows an example of a DRed document capturing the design rationale of an aero-engine internal gearbox. The design rationale is displayed in a document as a graph of nodes linked with directed arcs. The user creates the nodes by choosing from a predefined set of element types including the issue, answer, pro and con argument. Any element on a work plane can be linked without restriction to any other, and any element can easily be converted from its existing type to another. Each element type has a predefined set of statuses, signified by changes in colour and geometry of the background shape or font style of the text. There is only a single type of link, a unidirectional arrow, which represents a dependency of some sort. The precise meaning of that dependency is inferred from the types of the elements at each end of the arrow.

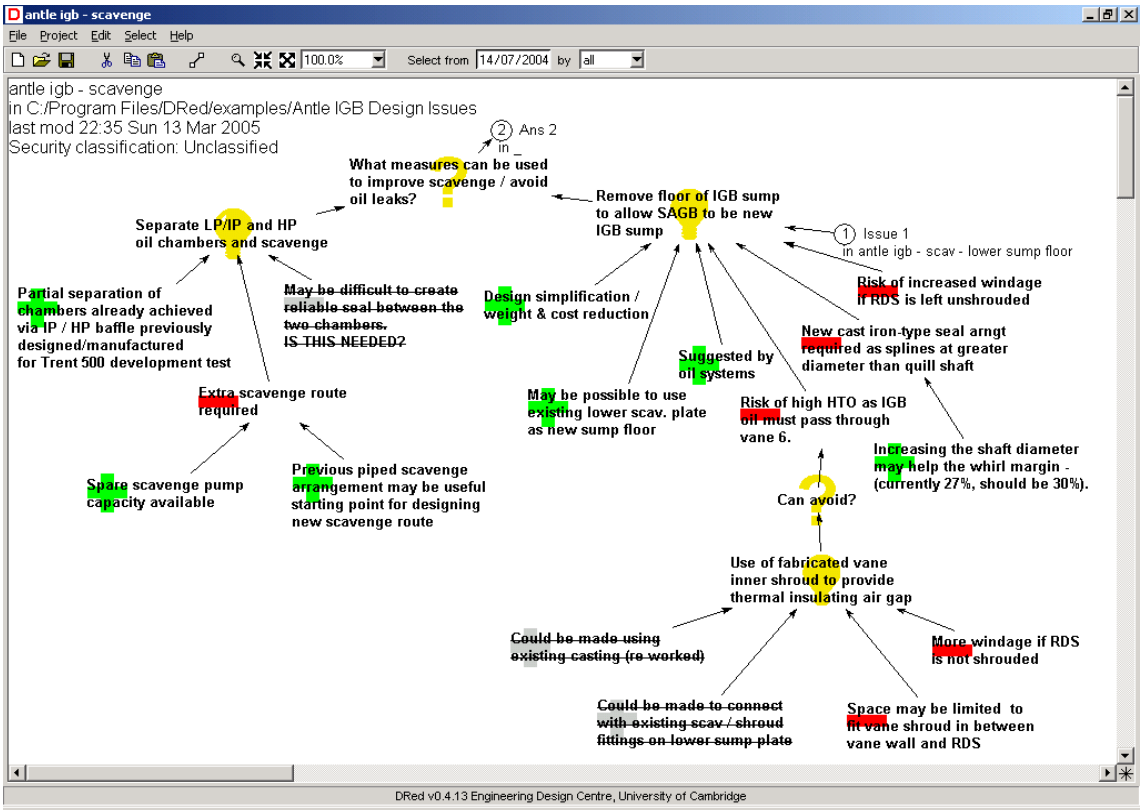


Figure 2.8: An example of DRed document capturing the design rationale of an aero-engine internal gearbox

Source: Aurisicchio *et al.* (2006)

RCF (Rationale Construction Framework) (Myers *et al.*, 2000) acquires rationale information automatically for the detailed design process without disrupting a designer's normal activities. The underlying approach involves monitoring designer interactions with a commercial CAD tool to produce a process history. This history is subsequently structured and interpreted, relative to a background theory of design that enables explanation of certain aspects of the process. RCF extracts two different types of rationale-related information. The first is a series of hierarchical abstractions of the design history: what the designer did and when. In addition, RCF reasons about intent as to why the designer performed certain actions. A set of design metaphors, which describe temporally extended sets of designer operations that constitute meaningful episodes of activity, drives the extraction of rationale related to designer intent. Design metaphors provide the basis for inferring intent on the part of the designer by linking observed activities to explanations for them. However, the authors report that automatic generation of complete rationale for all aspects of a design is clearly infeasible. Certainly, designers make many critical decisions and assumptions that are not explicit in the designs or in the design process. The work reported by (Myers *et al.*, 2000) seeks to automate documentation of important but low level aspects of the design process in a time and cost effective manner, thus freeing designers to focus their documentation efforts on the more creative and unusual aspects of the design. Ideally, the methods presented by them would be complemented by interactive rationale acquisition methods that would enable designers to extend or correct automatically generated information.

Burge & Brown (2000) investigated the use of design rationales by building InfoRat, a prototype system that draws inferences on a design's rationale to detect inconsistencies in the decisions made and to assess the impact of changes. The approach can be described as follows: The process begins with a set of requirements for the system being designed. These requirements are then mapped to goals and, if required sub goals. Goals and sub-goals can then be satisfied by one or more alternatives. Each alternative then maps to an artefact, or a requirement for the next design stage. The rationale for each choice is represented as arguments, expressed as claims, for or against each alternative. Figure 2.9 from Burge & Brown (2000) shows an overview of the use of design rationale in the design process. The verification

involves ensuring that the design is consistent and complete, i.e., all requirements correspond to goals and all goals have selected alternatives.

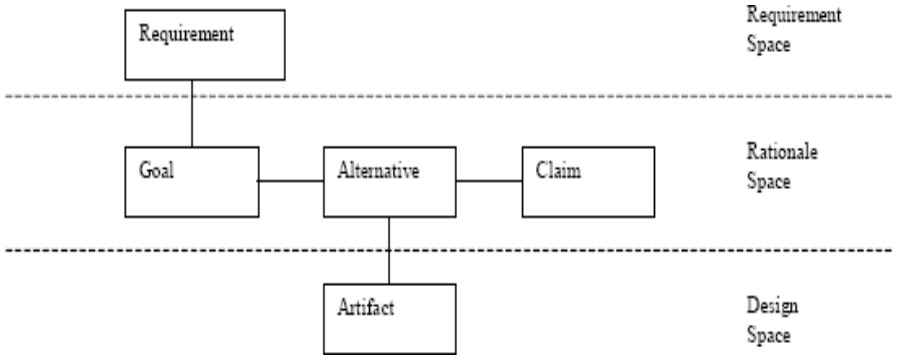


Figure 2.9: The use of Design Rationale in the design process by InfoRat
Source: Burge & Brown (2000)

Design rationales are invaluable in the reuse of design information. Design reuse can make an important contribution towards design efficiency (Sanghee *et al*., 2007). Given the competitive pressures in business environments, the reuse of previous designs has significant value for shorter delivery times and lower production costs. For example, research has identified that up to 90% of all design activities are based on the variants of existing designs (Fletcher & Gu, 2005). However, design information is often difficult to retrieve (Sanghee *et al*., 2007). There is limited support in recognising the existence of the reusable information and designers often do not attempt to reuse. Sanghee *et al*. (2007) propose a task model based approach that systems could adopt to suggest recommendations and aid reuse of past design information. They have used DRed to demonstrate the approach. A task model is acquired by observing a designer's activities. The design rationales captured by DRed are represented as a directed graph of elements. The elements are chosen from a predefined menu of types, at the core of which are Issue (I), Answer (A), ProArgument (PA) and Con Argument (CA). Each element is associated with a label that is a textual description in natural language. A DRed path is the list of links starting from a specific element and finishing at a specific element. In the context of a design process, the designer uses the DRed path for exploring solutions for a given task. Such a DRed path is a task model and the proposed approach recommends the

next likely element that the designer will employ. The proposed approach recommends using two strategies: (1) a DRed path similarity: The strategy examines the sequence in which a current designer has invoked particular elements and uses this as a basis of calculating the prediction of a new element. (2) Content similarity: the strategy uses shallow Natural Language Processing (NLP) techniques to analyse the DRed document. The NLP techniques include term identification, part-of-speech tagging and term normalization. Terms are identified as words lying between two spaces including a full stop. Although Sanghee *et al*. (2007)'s approach claims to improve design reuse by assuming relevance between tasks and suggesting recommendations, the approach fails to enable a system to understand and interpret the textual content of the rationales. Representing rationales in a machine- interpretable format should enable a system in making recommendations that are more accurate, detecting inconsistencies among rationales and design decisions, etc.

Burge & Brown (2003) researched the benefits of reusing design rationales for a large-scale maintenance task. They report that one of the chief difficulties in maintaining a large system is knowing the reasons behind the choices made by the developers during design and implementation. The presence of rationale would serve as a "corporate memory" by capturing design information that would be lost if the developers left the company or if they were inaccessible to the maintainers. Karensty (1996) also showed the importance of reusing rationales, i.e. over 50% of designer's information needs are related to the questions that could be answered by reusing the rationales. Thus, design rationale would enable both easier maintenance of artefacts over their lifecycles and more effective reuse of designs by making it easier for downstream engineers to understand how a design works (Myers *et al.*, 2000). For example, Brazier *et al*. (1997) present an example of stored rationale being used in the redesign of a model passenger aircraft to accommodate changes in the overall design requirements.

2.4.4 Discussion

Engineering design is constraint-oriented and constraint-based systems are applicable in all phases of design. Constraints have been used to assist in a variety of engineering design tasks including the development of rule-based systems. Maintainability of rule-based systems in industries became very difficult because of the need to constantly

make changes to the knowledge base. Since rules were encoded into the procedural parts of the program, it was hard to determine which rules needed changing. Description logic and ontology based systems have been used in industries, particularly in configuration-based design tasks. These systems have made the maintenance task easier when compared to rule-based systems. However, they are still faced with maintenance issues. Constraint management systems have been developed mainly to detect conflicts among constraints during constraint solving. In Designers' Workbench, design rules are expressed as constraints over the domain ontology. Designers' Workbench performs constraint checking instead of constraint solving. This has implications for tractability, in that constraint solving is a NP-complete problem, whereas checking a solution can be done in polynomial time. This thesis proposes a methodology and reports on a system that has been developed to detect inconsistencies and suggest appropriate refinements between pairs of constraints prior to constraint solving or constraint checking by systems such as the Designers' Workbench.

Concurrent Engineering and Integrated Product Development have become increasingly important in the success of product development within industries. They provide tremendous benefits in terms of reduced time-to-market, low cost, considering the entire product lifecycle and improved quality. By considering the effects of all the other phases in the product lifecycle such as manufacturing, maintenance, etc. during the design phase, one can optimise the cost, quality and time of product development. Collaborative design support systems play a key role in concurrent engineering. There are different aspects to collaborative design such as conflict detection and resolution, sharing, social interactions, integration and visualisation of information. The approaches adopted to tackle these aspects include constraint-based, agent-based, model-based and ontologies. Constraint-based systems are widely used and particularly useful in collaborative design for conflict detection and resolution. Collaborative engineering design activities are influenced not only by the technological factors, but also by the social interactions among various stakeholders with different perspectives. These perspectives of various stakeholders constitute a part of the design rationale. It is important to capture these perspectives (rationales) of various stakeholders and analyse them in concurrent engineering.

The knowledge of the who, what, when, where, why, and how of design constitute the design rationales. Rationale can include assumptions made about the system, the alternatives considered and the reasoning behind decisions. Recording design rationales is useful for both current and future designers. The process of capturing design rationales supports the designer in clarifying decision-making. It may also relieve the designer from the burden of retrospectively documenting the design at the end of a task. Research has indicated that most of the design activities involve reuse of previous design. Hence, capturing design rationales would be invaluable for future designers. Although design rationales are useful, they are often extremely hard to capture, mainly because the process is very intrusive and requires a lot of the designers' time. Various design rationale systems have been developed to enable the capture of rationales. The advantages and shortcomings of the different design rationale systems depend on the trade off between ease of capture and the explanatory power of the rationale. Action-based design rationales are easy to capture and do not require much intervention from the designer while argumentation-based design rationales are difficult to capture. However, argumentation-based design rationales provide more useful explanation when compared to action-based rationales. Most design rationale systems represent the rationales in a human readable format (natural language). Although the information may have some structure, the information cannot be understood, interpreted and used by systems to provide immediate benefits to the designers. In addition, design rationale systems have not concentrated on capturing information pertaining to when a particular section of the design knowledge is applicable. Design rationales are also often difficult to retrieve and hence rarely used. This thesis investigates the capture of information pertaining to when a particular constraint is applicable (referred as application conditions). The thesis argues that it is important to concentrate on representing design rationales (application conditions) in a machine-interpretable format. This would enable systems to use the rationales and provide immediate benefits to the designers by detecting inconsistencies and suggesting refinements among design decisions taken by the designers. The immediate benefits provided by the system should encourage designers to capture design rationales. In particular, the thesis investigates how an explicit representation of rationales (referred to as application conditions) together with the corresponding constraints and the domain ontology can be used to support the maintenance of constraints in engineering design.

2.5 Summary

This chapter provides a review of the work done in the area of knowledge acquisition, the issues involved and the different types of tools that have been developed to support knowledge acquisition. This is followed by a review of some of the prominent knowledge engineering methodologies. Taken together, the review describes the issues/problems faced by knowledge-based systems over the past few decades and how the latest methodologies and tools have dramatically changed the way in which knowledge-based systems are developed. Building knowledge-based systems now focuses on reusing and adapting existing resources, rather than building them from scratch. Moreover, the emphasis has been in facilitating domain experts to build and maintain knowledge bases, and hence minimize or eliminate the role played by a knowledge engineer. This thesis reports on the design and construction of a system that has been developed to facilitate domain experts in capturing and maintaining constraints in engineering design.

Further, the chapter reviews work done in the area of knowledge maintenance that involves verification, validation and refinement of knowledge. This is followed by a review of engineering design. Maintenance is a critical phase in knowledge engineering that can be complex and time-consuming. It is important to explicitly record the contexts in which each rule is applicable, during the KA phase. Recording the contexts should help identify all the rules that need to be updated during maintenance. This thesis investigates issues in maintenance by using engineering design as an application domain.

Engineering design is constraint-oriented and involves the identification of new constraints or the modification or deletion of existing constraints. The evolutionary nature of constraints establishes the need to provide support for maintenance. Constraint management systems have been developed mainly to detect conflicts among constraints during constraint solving. It would be useful to develop tool(s) that can detect inconsistencies among constraints prior to constraint solving, suggest appropriate refinements and help in the maintenance of constraints.

Concurrent Engineering and Integrated Product Development have become increasingly important within industries by providing tremendous benefits in terms of reduced time-to-market, low cost, considering the entire product lifecycle and

improved quality. Collaborative engineering design activities are influenced not only by the technological factors, but also by the social interactions among various stakeholders with different perspectives. The perspectives of various stakeholders constitute a part of the design rationale. It is important to capture these perspectives (rationales) and analyse them in concurrent engineering. Rationales can include assumptions made about the system, the alternatives considered and the reasoning behind decisions. Although design rationales are useful, they are often extremely hard to capture, mainly because the process is very intrusive and requires considerable amount of the designers' time. This thesis investigates how an explicit representation of rationales (referred to as application conditions) together with the corresponding constraints and the domain ontology can be used to support the maintenance of constraints in engineering design. More details about the investigation can be found in subsequent chapters.

Chapter 3

Constraint Capture and Maintenance in Engineering Design: A Proposal

> 'Most of the effort in the software business goes into the maintenance of code that already exists.'
> **- Wietse Venema**

This chapter sets the scene for the research work reported in this thesis. The chapter is divided into four main sections. Section 3.1 introduces the Designers' Workbench, and describes the problems encountered while capturing knowledge (design rules) for this system. Section 3.2 describes the proposed approach to capturing constraints to address the problems faced by systems such as the Designers' Workbench. Section 3.3 describes the issues/problems faced during the maintenance of constraints in an engineering design environment. Finally, Section 3.4 describes the proposed approach to tackle the various issues/problems faced during the maintenance of constraints. The chapter concludes by summarising the key points in Section 3.5.

3.1 Introduction to the Designers' Workbench

Typically, complex engineering artefacts are designed by teams who may not be located in the same building or even city. Designers in Rolls-Royce, as in many large organizations, work in teams. Thus, it is important when a group of designers are working on aspects of a common project, that the sub-component designed by one designer is consistent with the overall specification, and with those designed by other members of the team. Additionally, all designs have to be consistent with the company's design rule book(s). Making sure that these various constraints are complied with is a complicated process, and so previous research has developed the Designers' Workbench (Fowler *et al.*, 2004) which seeks to support these activities.

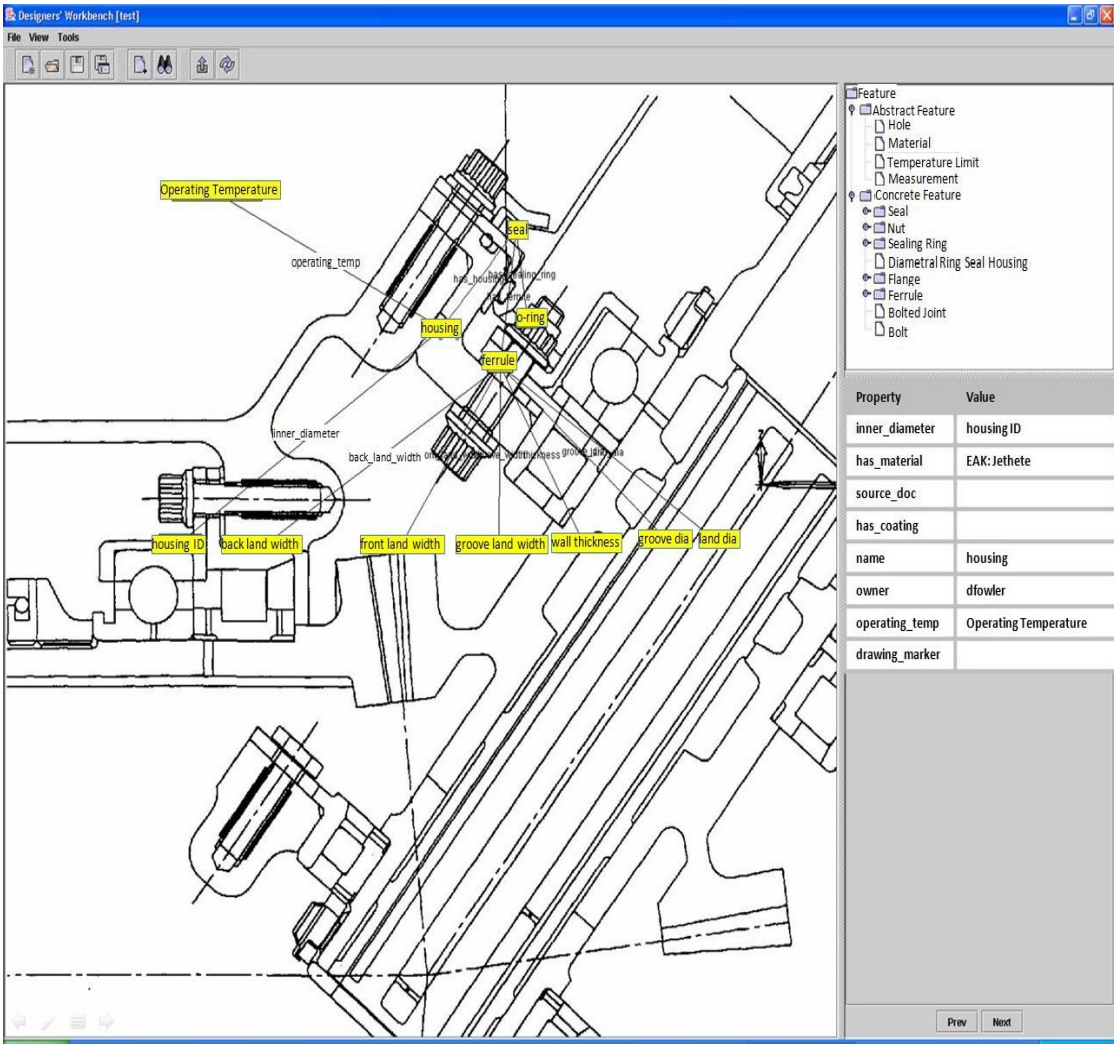


Figure 3.1: A screenshot of the Designers' Workbench

The Designers' Workbench (Figure 3.1) uses an ontology to describe elements in a configuration task. The system supports human designers by checking that their configurations satisfy both physical and organizational constraints. Configurations are composed of features, which can be geometric or non-geometric, physical or abstract. The following example from Fowler *et al*. (2004) illustrates the use of an ontology to describe a configuration.

The class hierarchy of a simple ontology is shown in Figure 3.2. The concept 'Feature' is the root of that ontology. The concept 'Feature' is divided into 'Concrete feature' (a physical sub-component) and 'Abstract Feature' (holes, temperature, etc.).

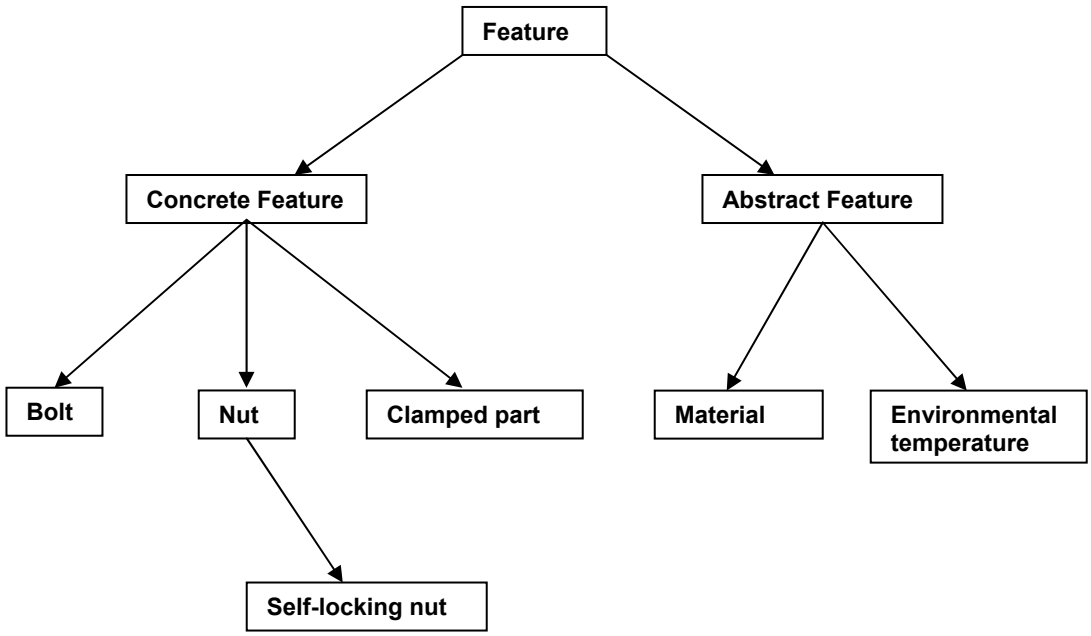


Figure 3.2: The class hierarchy of a simple configuration ontology

[Source: Fowler *et al*. (2004)]

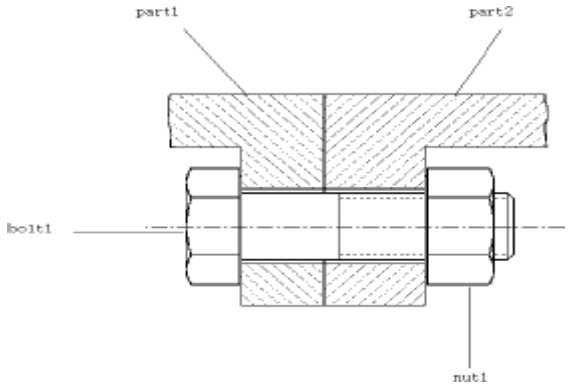


Figure 3.3: A bolted joint

[Source: French *et al.* (1993)]

'Concrete Feature' is further divided into 'Bolt', 'Nut' and 'Clamped Part' while 'Abstract Feature' is divided into 'Material' and 'Environmental Temperature'. 'Self-

locking Nut' is a specific type of 'Nut'. Figure 3.3 (above) shows a simple arrangement of a bolted joint, subject to a particular environmental temperature and Figure 3.4 (below) shows a configuration of the bolted joint, described using an ontology.

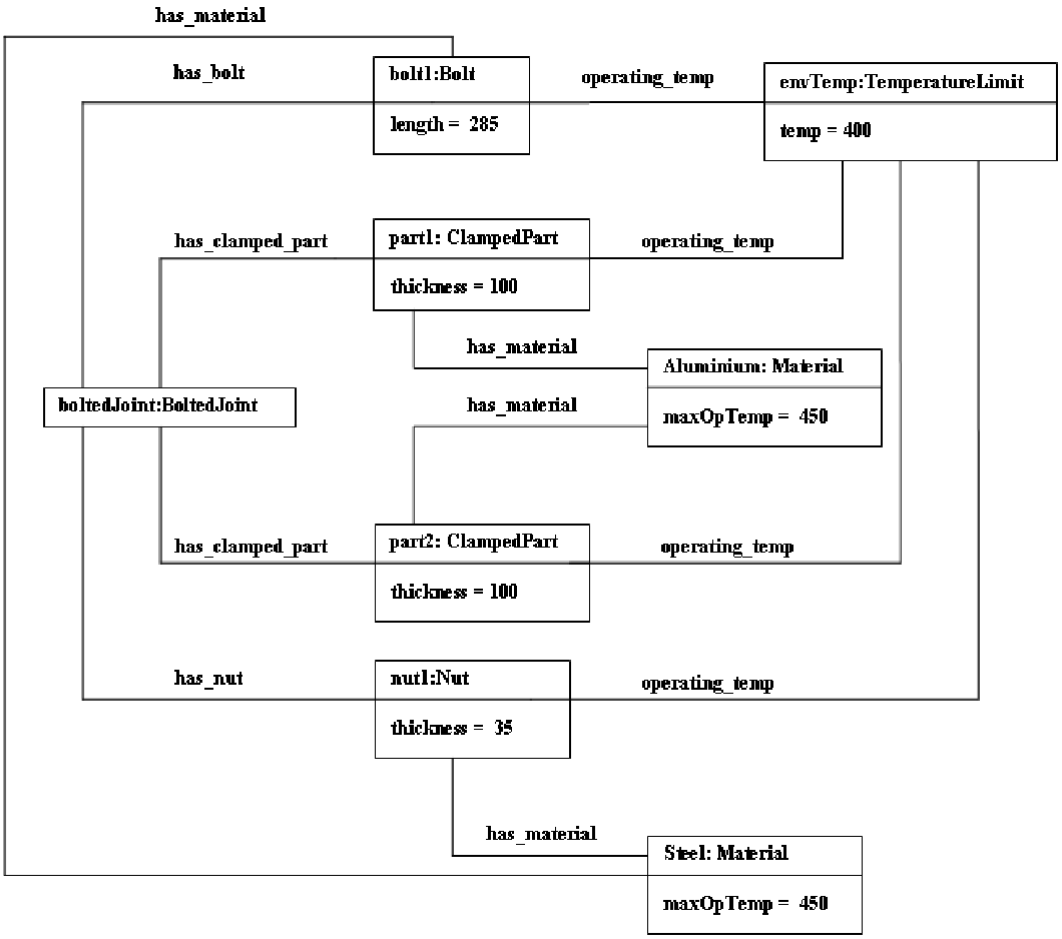


Figure 3.4: A configuration of the bolted joint in Figure 3.3 described using an ontology

Constraints defined over this ontology (Figure 3.4) include:

- The value of the maximum operating temperature of the material of each concrete feature must be greater than the prevailing environmental temperature;
- The length of the bolt in a bolted joint must exceed the sum of the thicknesses of the clamped parts, plus the thickness of the nut. For simplicity, issues such

as tolerances of dimensions have been ignored. Tolerances and dimensions can be dealt with, for example, by defining a 'Measurement' class with properties 'dimension' and 'tolerance' containing real values.

The first constraint above applies to all concrete features that have a 'has_material' property and an 'environmental_temperature' property defined. The second constraint is more complicated, and applies to all bolts, nuts, and clamped parts that are parts of a bolted joint.

3.1.1 Functionality of Designers' Workbench

In the Designers' Workbench, the designer can select a feature class from the ontology and create an instance of that class. The property values of the instance can then be filled with: (i) datatype values by literals of the appropriate type, and (ii) object type values by selecting an instance from a list of instances of the appropriate type. Constraints are handled in a two stage process:

- Identify feature values that should be constrained;
- Formulate a tuple(s) of values for each set of feature values, and check that the constraint is satisfied by these values.

The constraint processing uses RDQL to find the constrained features and values. After using RDQL to extract the constrained features and values, Sicstus Prolog is used to check that the constraints hold. For example, the RDQL query that locates features affected by the material temperature constraint is:

```
SELECT ?arg1,?arg2 WHERE
(?feature,<dwOnto:has_material>,?mat),
(?mat,<dwOnto:max_operating_temp>,?arg1),
(?feature,<dwOnto:operating_temp>,?optemp),
(?optemp,<dwOnto:temperature>,?arg2)
USING dwOnto FOR <namespace>
```

The values of the returned variables ?arg1 and ?arg2 are the material's maximum operating temperature, and the environmental operating temperature, respectively. The check that the values must satisfy is represented by the Sicstus predicate:

```
op_temp_limit(MaterialMaxTemp, EnvironTemp) :-
EnvironTemp =< MaterialMaxTemp.
```

Using the values of ?arg1 and ?arg2, the predicate op_temp_limit(MaterialMaxTemp, EnvironTemp) is formed, and checked. This process is repeated for each set of values returned by the RDQL query, and for each constraint that has been specified.

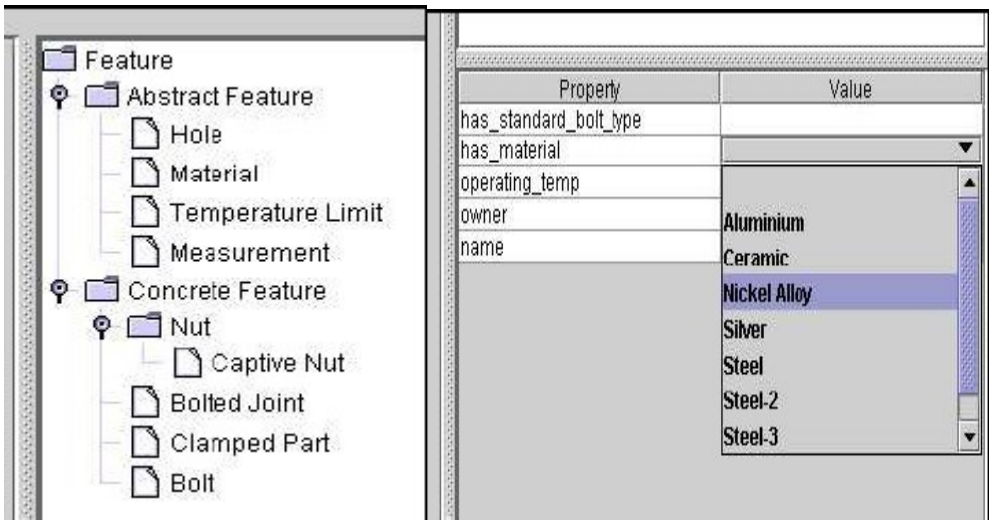


Figure 3.5: Close-ups of the Designers' Workbench panels: the feature ontology (left), and properties of selected feature (right)

[Source: Fowler *et al.* (2004)]

Additional features of the Designers' Workbench are as follows:

(i) Graph-based display of configuration: A graphical user interface enables the designer to import a drawing, annotate it with features, assign property values, and perform constraint checks. The drawing is actually a visual aid i.e. the designer can mark up an existing drawing or construct a configuration without a drawing. Features can be selected from an ontology. Features that are added by the designer are shown

as labels overlaying the background drawing. Properties that connect features are represented by arcs. Features can be selected, and their properties viewed and modified using the table displayed beneath the ontology. Datatype properties are set by typing values into the field, whereas object properties are set using a drop down list of values representing the valid possibilities for the property. For example, if the property has_bolt is specified to have range of class Bolt, the list will consist only of instances of Bolt.

If a constraint is violated, the affected features are highlighted and a report is generated. The report gives the designer a short description of the constraint that is violated, the features affected by that violation, and a link to the source document that contains the design rule. The designer can often resolve the violations by adjusting the property values of the affected features. On selecting the affected feature from the ontology, a table is displayed with the corresponding properties and values (as shown in Figure 3.5). These property values can then be adjusted to resolve the constraint violations.

(ii) Checking incomplete configurations: Before checking constraints, it is not necessary to specify values for every defined property of each feature. Instead, the designer can fill in values for whichever properties he or she desires, and request a constraint check. The RDQL query will only return results for the features that have sufficient values specified, so that only certain constraints will be checked. This allows designers to operate in an exploratory way, defining small parts of a configuration, checking them, and then gradually extending the configuration until it is complete.

(iii) Constraint rationales: Each constraint has an associated rationale (currently a short text string, but which in future may have more structure), and an (optional) URI for a source document explaining the rationale in more depth. When a constraint violation is reported, the designer is presented with a list of the features involved in the violation, the rationale, and the link that can be clicked on to read the source document. In this way, the designer can learn more about the constraint, and decide if it is in fact appropriate. As the constraint checking proceeds, an experienced designer may decide to override the constraint.

9.3.2 Internally trapped nuts (see Fig 4 Table 4)

TABLE 4

PCD ABOVE mm	TO mm	2M mm
150 - 180		1,00
180 - 300		0,80
300		0,60

ø N MIN. = PCD (NOM) + 2M + MAX. NUT WIDTH (SEE TABLE 5)

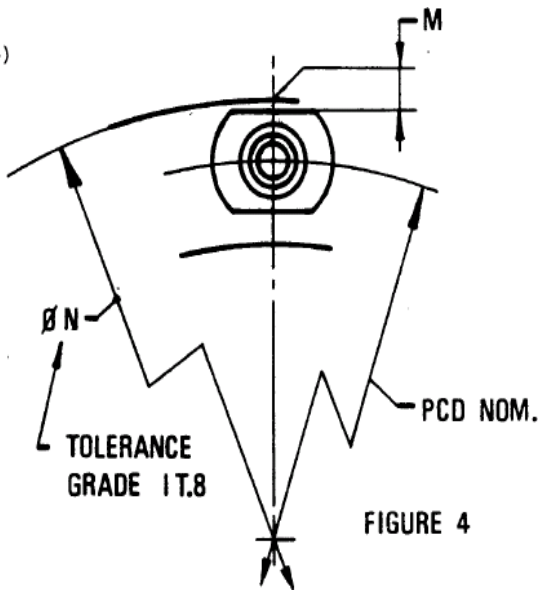


FIGURE 4

TABLE 5

THREAD SIZE	MAX. NUT WIDTH mm
0.1900-32	10,13
0.2500-28	11,56
0.3125-24	12,33
0.3750-24	15,37
0.4375-20	17,53
0.5000-20	19,05
0.5625-18	20,78
0.6250-18	23,02

Figure 3.6: Constraint as expressed in a rule book

[Source: Joint Design Standards (JDS) No: 805.04, Rolls-Royce]

3.1.2 Capturing the knowledge in the design rule book(s)

As noted above, the Designers' Workbench needs access to the various constraints, including those inherent in the company's design rule book(s). To capture this information, a design engineer (domain expert) works with a knowledge engineer to identify the constraints, and it is then the task of the knowledge engineer to encode these into the Workbench's KB as a query in RDQL, and a predicate in Sicstus Prolog. This is a laborious, error-prone and time-consuming task. The constraints are formulated succinctly in the design rule book(s) and hence a non-expert in the field often finds it very difficult to understand the context and formulate constraints directly from the design rule book, and so a design engineer has to help the knowledge engineer in this process. An example of such a constraint is shown in Figure 3.6. The design rule book(s) gives the description of constraints, in the form of tables and figures in most cases.

3.2 A Proposed Approach to the Capture of Constraints

As noted in the previous section, there are many issues/problems faced when a knowledge engineer seeks to capture knowledge from the design rule book(s) and encodes them as constraints in the Designers' Workbench.

The thesis's proposed approach to the capture of constraints is to facilitate domain experts in formulating a constraint by selecting classes and properties from the domain ontology, and combining them with predefined keywords and operators from a high-level constraint language. This should relieve the knowledge engineer from the error-prone and time-consuming process of capturing constraints. This would also enable designers to have greater control over the definition and refinement of constraints, and presumably, to have greater trust in the results of the constraint checking process. In order to embody the proposed approach, the thesis outlines the following tasks:

- Development of a system comprising of the following components/features:

  (i) A graphical user interface that enables a user to formulate a constraint by means of a few mouse clicks. The graphical user interface contains the following sub components:

a) A scrollable list of keywords from a high level constraint language.
b) A scrollable tree structure of classes and properties from the domain ontology.
c) A tool bar containing appropriate arithmetic, logical and relational operators.
d) A result panel to display the constraint being formulated and the results of a syntax check.

The user formulates a constraint by selecting entities from (a), (b) and (c) for display in the result panel.

(ii) Use a high-level constraint language with good expressivity to represent the constraint.

(iii) Perform syntax checking of the formulated constraint.

(iv) Display details of any syntactical errors.

(v) Facilitate the user in editing a constraint, creating a table of constraints, and reading/writing constraints from/to a text file.

(vi) Allocate each constraint with a unique identification number that also denotes its version number.

(vii) Provide a search facility to retrieve constraints from the KB.

(viii) Convert the constraint into a standard (semantic web enabled) format that enables other systems such as the Designers' Workbench, constraint solvers, agents, etc. to process the constraint.

- Perform a preliminary evaluation by demonstrating the system to domain experts (Rolls-Royce design engineers).

- Run an experiment to evaluate the usability of the system.

More details on the system developed are provided in the subsequent chapter (Chapter 4). Information regarding the preliminary evaluation and experiments carried out are

provided in Chapter 7. The research question that the proposed approach aims to address is as follows:

Research Question I:

- Examine whether it is possible to design and construct a system to facilitate (domain) experts in capturing and maintaining constraints in engineering design. This question can be detailed into the following smaller questions:

    a) Can (domain) experts successfully perform the allocated tasks within acceptable time limits?
    b) Did the (domain) experts perform the tasks accurately? What kind of mistakes did they make? Can the system's GUI be modified to eliminate or minimize these errors?
    c) How easy and intuitive did (domain) experts find the system to use?
    d) Is the speed of the system on realistic tasks viable for (domain) experts to use?

The thesis aims to examine whether it is possible to design and construct a system to facilitate domain experts in capturing and maintaining constraints in engineering design. Systems such as the Designers' Workbench should then be able to process these constraints captured by the domain experts. This would eliminate the knowledge engineer from the error-prone and time-consuming task of capturing design rules for the Designers' Workbench's KB. The next two sections describe the issues/problems faced during maintenance of these constraints and the proposed approach to address these issues/problems.

3.3 Maintenance of Constraints in Engineering Design

The engineering design process has an iterative nature as designed artefacts often develop through a series of changes before a final solution is achieved. A common problem encountered during the design process is that of evolution, which may

involve the identification of new constraints or the modification or removal of existing constraints. The reasons for such changes include developments in the technology, changes to improve performance, and changes to reduce development time and costs. Typically, maintenance involves various issues/problems:

- Original experts are unlikely to be available: The transient nature of modern organizations and workforces, and the rapid flow of knowledge and experience out of companies due to staff leaving, make it difficult for new designers to effectively use stored design knowledge and subsequently to maintain it.

- Insufficient documentation provided: Some constraints may be applicable only in particular contexts. These contexts are often implicit to the designer formulating them but are not documented. In addition, many constraints will be based on assumptions that may not be true later on. These assumptions are often not made explicit.

- Maintenance is time-consuming and complex: Maintenance of constraints in an engineering design environment is a complicated process and is very difficult to do manually. Thus, there is a pressing need for tools to support maintenance of this kind of knowledge.

- The evolutionary nature of constraints establishes the need to constantly update, revise, and maintain them. One needs to identify the constraints that require modification. In addition, one needs to make sure the knowledge base is consistent after making a change.

Verification in KBSs plays a very important role. As we automate more processes, the need for verification becomes even more critical. Many automated processes perform incorrectly for a long time, as no person is responsible for checking the process (Hicks, 2003). Additionally, as the KB evolves, constant addition/revision of rules can result in high levels of redundancy. It is important to prevent/minimize redundant rules in the KB. Removing/reducing redundancy in a KB will make it easier to maintain the KB.

Constraints are continually being added, deleted and modified throughout the development of a new product. Design begins with a functional specification of the desired product: a description of properties and conditions that the product should satisfy (i.e. constraints). Constraints themselves form a rationale associated with the design decisions taken by designers. A typical rationale is of the form: "A component X exists in the design because of the need to satisfy constraint Y." The ability to capture and use this type of design rationale in concurrent engineering has been referred to as Design Rationale Management by Bahler & Bowen (1992), who describe a constraint-based design advice system that generates machine-generated suggestions to support coordination among multiple design engineers. The Designers' Workbench (Fowler *et al.*, 2004) provides similar functionality by checking if the design satisfies all the relevant constraints, providing details of the violated constraints and enabling the designers to resolve them.

Much research has been done to develop systems that capture and represent the rationales associated with design knowledge. Design rationales considered so far refer to the information containing either one or all of the following:

a) the reasons behind the design decisions taken (why a decision was taken).
b) the design alternatives considered and rejected with reasons for rejection.
c) how certain design actions are performed.

However, design rationale systems have not concentrated on capturing information pertaining to when a particular section of the design knowledge is applicable. Constraints may be formulated based on a number of assumptions and may be relevant only in certain contexts. Designers often tend to assume "normal" situations (Brown, 2006). They tend to make assumptions about the match between the current design situation and one where their chosen technique worked well before. They tend to make abstractions across all the situations where particular techniques worked well before, by assuming that some key detail is relevant or irrelevant. These assumptions are not deliberate, but form the tacit knowledge underlying expert skill. In order to support maintenance of design knowledge, it is important to make these assumptions visible. One needs to find ways to capture the assumptions and contexts as part of the rationale associated with a constraint. The thesis refers to this type of rationale as the *application conditions* associated with a constraint (Ajit *et al*., 2008a; Sleeman *et al*.,

2008). A recent article by Hooey & Foyle (2007) reported on the requirements for design rationale capture tool to support all the design phases of NASA's complex systems. They stressed the need to capture the assumptions and contexts as the rationale for a given design element, particularly in the conceptual design phase. The paper describes how this information is rarely captured in a systematic and usable format because there are no tools that adequately facilitate and support the capture and use of this critical information. An example quoted in the paper is: "The minimum volume for the Crew Exploration Vehicle cockpit is based on an assumption of a specific crew size". The above example is a constraint together with its application condition. If a design element or a constraint is modified, there is no easy way to propagate that change to understand the implications and consequences of those changes.

Thus, it is important to capture information pertaining to when a particular section of the design knowledge is applicable and enable systems to use this information to support maintenance. This thesis proposes an approach to capture application conditions associated with constraints and use these application conditions together with the constraints and domain ontology to support the maintenance of constraints. The next section (section 3.4) describes the proposed approach with an example.

3.4 A Proposed Approach to the Maintenance of Constraints

Due to restricted availability of Rolls-Royce designers' time and because it is a simpler domain, the kite domain was initially investigated to elicit equations and constraints together with the corresponding application conditions. The sources referred to study the kite domain include Yolen (1976), Streeter (1980), Eden (1998), AKA (2006), CEKS (2006), Leigh (2006), Lords (2006) and Wardley (2006).

For a successful kite design, one has to make sure that the design complies with all the appropriate rules/constraints. There are different types of constraints associated with the design of a kite. The analysis of kite domain showed that several constraints were applicable only to particular types of kites and under specific conditions. Appendix A provides a list of equations and constraints elicited from the kite domain together with the corresponding application conditions and sources..

The context in which a constraint is applicable is referred to as an application condition in this thesis. Application conditions form a part of the rationales associated with the constraint. Consider the following example of a constraint together with its associated application condition:

Constraint[7] – "The density of the cover material of the kite must be greater than 21.9 kilograms per square metre"

Application condition – "This is applicable only when there is a requirement to produce low cost kites for beginners. Kites for experts use lighter materials that are of higher quality and hence costlier."

As shown in the example above, the application condition specifies the context in which the constraint is applicable. Often, the information of application conditions is implicit to the person who formulates the constraint. The assumptions/conditions on which a constraint is based may no longer be true and in such cases, it becomes necessary to deactivate or remove those constraints from the KB. Further, an application condition may not be relevant to a particular design task. Hence, in order to apply constraints appropriately and support maintenance, it is important to make the application conditions explicit.

Although design rationales can provide a lot of information about the reasoning involved in making a design decision, rationales are extremely hard to collect mainly because the process is very intrusive and requires a lot of the designers' time. Not much work has been carried out on how this information can be used by machines. Although the information may have some structure, the information cannot be understood, interpreted and used by machines to provide immediate benefits to the designers. Capturing large amounts of detailed rationales is not useful if it is never looked at again. If rationales are useful to the designers, there is a greater incentive for designers to assist in the capture of the information, particularly if the designer who is recording it can immediately use the rationale. As Grudin (1996) and Brown (2006) have pointed out, there cannot be a disparity between who invests effort in a groupware system, and who benefits. No designer can be expected to altruistically enter quality design rationale solely for the possible benefit of a possibly unknown person at an unknown point in the future for an unknown task. There must be immediate value. In addition, knowing how the information will be used provides

[7] http://www.cuttingedgekites.com/faq.htm. Accessed on 28 June 2006.

guidance about what information should be captured and how it should be represented. Thus, it is important to concentrate on the use of such information. Representing rationales in a machine-interpretable format would enable systems to immediately use the rationales to detect inconsistencies, redundancy, subsumption, fusion and suggest appropriate refinements between constraints.

The thesis hypothesises that an explicit representation of the context information (referred to as application conditions) together with the corresponding constraints and the domain ontology can be used to support the maintenance of constraints. In order to tackle the various maintenance issues/problems effectively, the thesis's proposed approach can be summarized as follows:

- Capture the "context" in which a constraint is applicable, in a machine-interpretable form, as an application condition and associate this information (rationale) with the constraint.
- Use the application condition together with the constraint and the corresponding domain ontology to support maintenance.

Maintenance of constraints includes (i) detecting inconsistencies, redundancy, subsumption and fusion (ii) reducing the number of spurious inconsistencies and (iii) preventing the identification of inappropriate refinements of redundancy, subsumption and fusion, between pairs of constraints. More details regarding the proposed approach to capture and use application conditions together with the corresponding constraints and the domain ontology can be found in Chapter 5. The proposed approach should encourage the designers to capture the application conditions together with the constraints because the system can immediately use them to provide benefits to the designers. If application conditions are useful to the designers, there is a greater incentive for designers to assist in the capture of the information, particularly if the designer who is recording it can immediately use the application condition. It is also important to ensure that the speed of the system for realistic tasks is viable for domain experts to use. The research question that the proposed approach aims to address is as follows:

Research Question II:

- Examine whether capturing application conditions associated with constraints, in a machine-interpretable format can provide significant benefits to the maintenance of constraints in engineering design. In particular, can an explicit representation of application conditions together with the corresponding constraints and the domain ontology be used to:
    a) Detect inconsistencies, redundancy, subsumption and fusion,
    b) Reduce the number of spurious inconsistencies, and
    c) Prevent the identification of inappropriate refinements of redundancy, subsumption and fusion between pairs of constraints?

Application conditions are captured in the same language as that of the constraints. More details about the representation of these application conditions together with the constraints are explained in Chapter 5. The thesis investigates the kite design domain and proposes four main types of knowledge refinement rules, namely, redundancy, subsumption, inconsistency and fusion. The rules make use of the application condition together with the constraint and the domain ontology to detect inconsistencies, suggest refinements (subsumption, redundancy and fusion), and hence support the maintenance of constraints. In addition, the knowledge refinement rules are expressed in a formal notation and it has been proved that they are logically sound. In order to embody the proposed approach and implement the refinement rules, the thesis outlines an algorithm and reports on a system developed to implement the algorithm. More details regarding the outlined algorithm and the system developed can be found in Chapter 6.

3.5 Summary

This chapter describes the proposal for the research work reported in this thesis. The chapter provides a description of the Designers' Workbench, a system developed by previous research to support designers in large organizations, such as Rolls-Royce, to ensure that the design is consistent with the specification for the particular design, as well as with the company's design rule book(s). The problems faced by the knowledge engineer during the capture of constraints for the Designers' Workbench

have motivated the author to propose an approach to facilitate domain experts themselves in capturing and maintaining constraints. The proposed approach involves providing an intuitive way to facilitate domain experts formulating a constraint by selecting classes and properties from the domain ontology, and combining them with predefined keywords and operators from a high-level constraint language. The tasks that need to be done to embody the above approach have been outlined.

Further, the chapter describes the various issues/problems faced during maintenance of constraints. The chapter reports that it is important to capture the context in which a constraint is applicable and refers to this context as an application condition associated with the constraint. The thesis hypothesises that an explicit representation of application conditions together with the corresponding constraints and the domain ontology can be used to support the maintenance of constraints. In particular, supporting the maintenance of constraints refers to: (i) detecting inconsistencies, redundancy, subsumption and fusion, (ii) reducing the number of spurious inconsistencies and (iii) preventing the identification of inappropriate refinements of redundancy, subsumption and fusion, between pairs of constraints. It is also hypothesised that the speed of the system for realistic tasks is viable for domain experts to use. The following chapter describes the design and construction of the system that has been developed to facilitate domain experts in capturing and maintaining constraints in engineering design.

Chapter 4

ConEditor

'The true creator is necessity,
who is the mother of invention.'
- Plato

This chapter describes the design, implementation and functionality of ConEditor. The chapter also presents an overview of the constraint representation languages (CoLan and CIF) used by ConEditor. The chapter is structured as follows: Section 4.1 provides an overview of the high-level constraint language (CoLan) used for the research work reported in this thesis. Parts of the description in Section 4.1 have been extracted from Bassiliades & Gray (1995) and Gray *et al*. (2001). Section 4.2 describes the design of ConEditor's GUI. Section 4.3 describes the implementation and functionality of ConEditor. Section 4.4 describes the principles involved in the conversion of OWL ontology into a Daplex Schema. Section 4.5 provides an overview of the XML Constraint Interchange Format used by ConEditor and the principles involved in converting CoLan into CIF. Section 4.6 summarises the chapter.

4.1 Overview of CoLan

CoLan (Bassiliades & Gray, 1995; Gray *et al*., 2001) is a constraint language based on an Object Data Model. Fully quantified constraints can be expressed in a very readable form of first order logic, including functions, which can be computed over data values expressed in the ER diagram (or UML class diagram). Hence, the underlying data model is called the Functional Data Model (FDM). The FDM, P/FDM (Prolog/Functional Data Model) is a semantic data model based on Shipman's original data model (Shipman, 1981). The semantics of the objects referred to in CoLan constraints are described in terms of this extended ER data model, which is of the kind in widespread use in UML and in database schemas. CoLan has features of both

first-order logic and functional programming, and is intended for scientists and engineers to express constraints.

```
constrain each t in tutor
    such that astatus(t)="research"
no s in advisee(t) has grade(s) =< 30;

constrain each r in residue to have
    distance(atom(r,"sg"),
      atom(disulphide(r),"sg")) < 3.7;
```

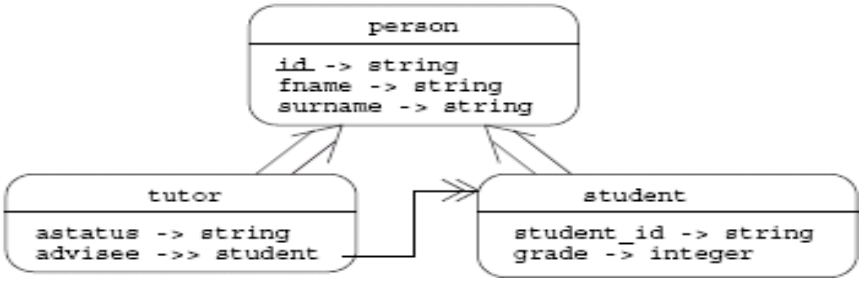


Figure 4.1: Examples of CoLan constraints from different application domains. The ER diagram models the relationships between entity classes in the first constraint

[Source: Gray *et al.* (2001)]

Two example constraints from different application domains are shown in Figure 4.1. An ER diagram that models the relationships between entity classes in the first constraint is also shown. The first example shows a CoLan constraint on a university database containing student records. The same constraint language is applicable to the domain of protein structure modelling, as shown by the second example restricting bond lengths. In the first example, a variable `t` ranges over the entity type `tutor` that is populated with stored object instances. Each of these instances may be related to instances of student entities through the relationship `advisee`, which delivers a set of related entities as in an object-oriented language. These entities can be restricted by the values of attributes such as `grade`. There are also other entity types such as residue (representing parts of protein chains) which have method functions for determining distances by computation. Thus, functions may also represent a derived relationship, or method. The entity classes can form part of a subtype hierarchy, in which case all properties and methods on the superclass are

inherited by each subclass. Method definitions may be overridden, but not constraints. This is significant for semantic web applications, since it means that information represented in this way is not restricted to human inspection. It can be proof-checked mechanically, transformed by symbol manipulation, or sent to a remote constraint solver. Moreover, given a standardised interchange format, data and attached constraints from multiple sources can be gathered together, checked for compatibility, and used to derive new information. Because the P/FDM data model is an extended ER model, it maps very easily onto the RDF schema specification.

CoLan is as expressive as the subset of first-order logic that is useful for expressing integrity constraints: namely, range-restricted constraints. This class of constraints includes those first-order logic expressions in which each variable is constrained to be a member of some finite set of values. CoLan provides a precise denotation for constraints but it does not force us to evaluate them as integrity checks. The constraint expresses a formula of logic which is true when applied to all the instances in a database, but it is also applicable to instances in a solution database which is yet to be populated with constructed solutions by a solver process (Gray *et al*., 1999a; Gray *et al*., 1999b). Here, it is behaving more like a specification than as an integrity check. The power of this in the context of the semantic web is that constraints can be passed as a form of mobile knowledge between agents and processes and they are no longer tied to a piece of database software. For more details of P/FDM, CoLan and related work, the reader is encouraged to visit www.csd.abdn.ac.uk/~pfdm or refer the relevant technical papers that have been referenced above.

4.2 ConEditor's GUI

ConEditor has been designed to facilitate domain experts in capturing and maintaining constraints. A screenshot of ConEditor's GUI is shown in Figure 4.2. A constraint expression can be created by selecting entities from a domain ontology and combining them with a pre-defined set of keywords and operators from the high-level constraint language, CoLan. An example of a simple constraint expressed in CoLan, against the domain ontology (a jet engine ontology) used by the Designers' Workbench is as follows:

constrain each f **in** ConcreteFeature
to have max_operating_temp(has_material(f)) >= operating_temp(f)

The above constraint states that for every instance of the class ConcreteFeature, the value of the maximum operating temperature of its material must be greater than or equal to the environmental operating temperature.

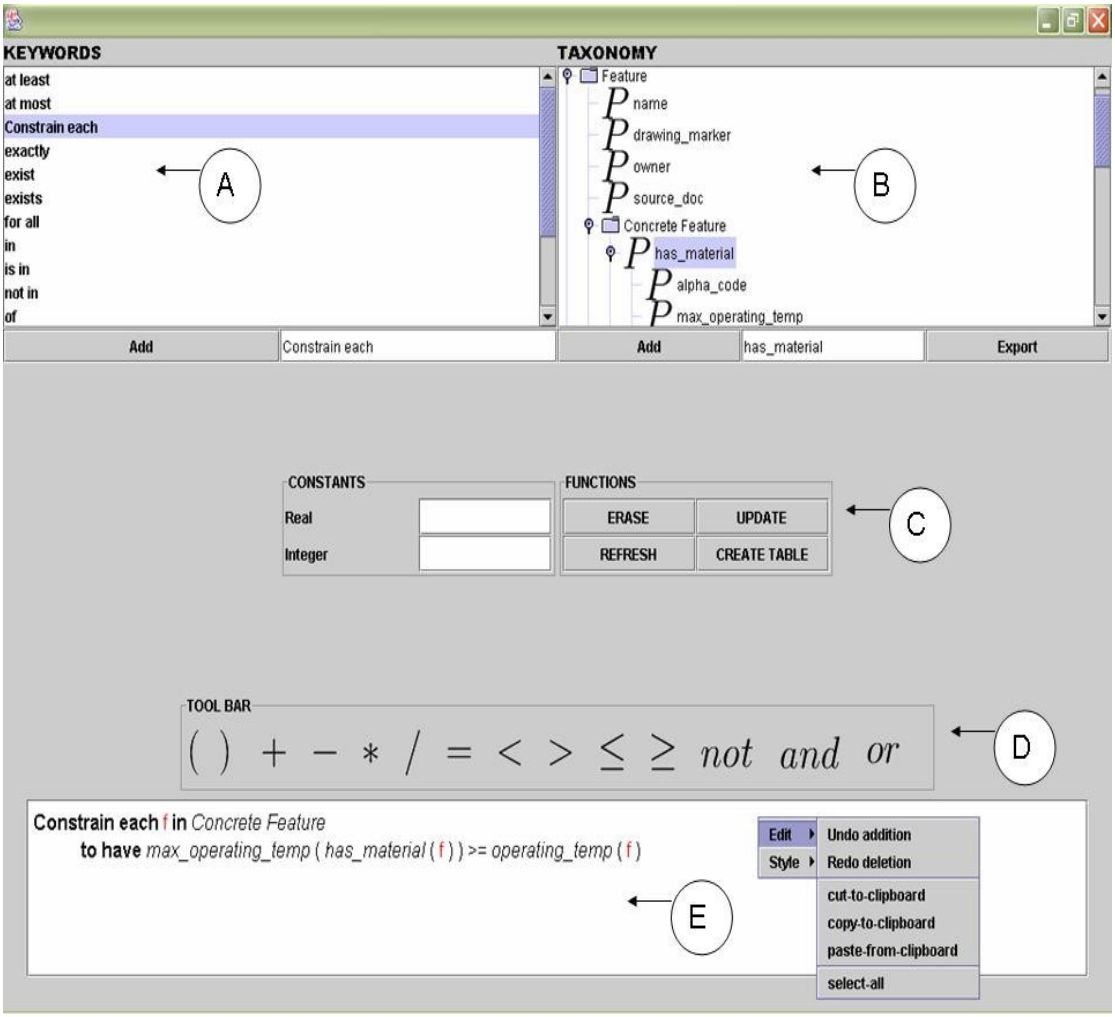


Figure 4.2: A screenshot of ConEditor's GUI

ConEditor's GUI essentially consists of five components, namely: (A) Keywords Panel, (B) Taxonomy Panel, (C) Central Panel, (D) Tool Bar and (E) Result Panel. These components provide the user with entities required to form a constraint

expression. The user can then choose the appropriate entities by clicking the mouse and so form a constraint expression. The process of formulating a constraint using ConEditor is explained further by considering the example of the constraint reported earlier in this section.

(A) Keywords Panel: This panel consists of a list of keywords from the CoLan language. In the example considered, the keywords *constrain each*, *in*, *to have* can be selected from this panel. Clicking the "Add" button in the panel appends the selected keyword to the text area in the result panel.

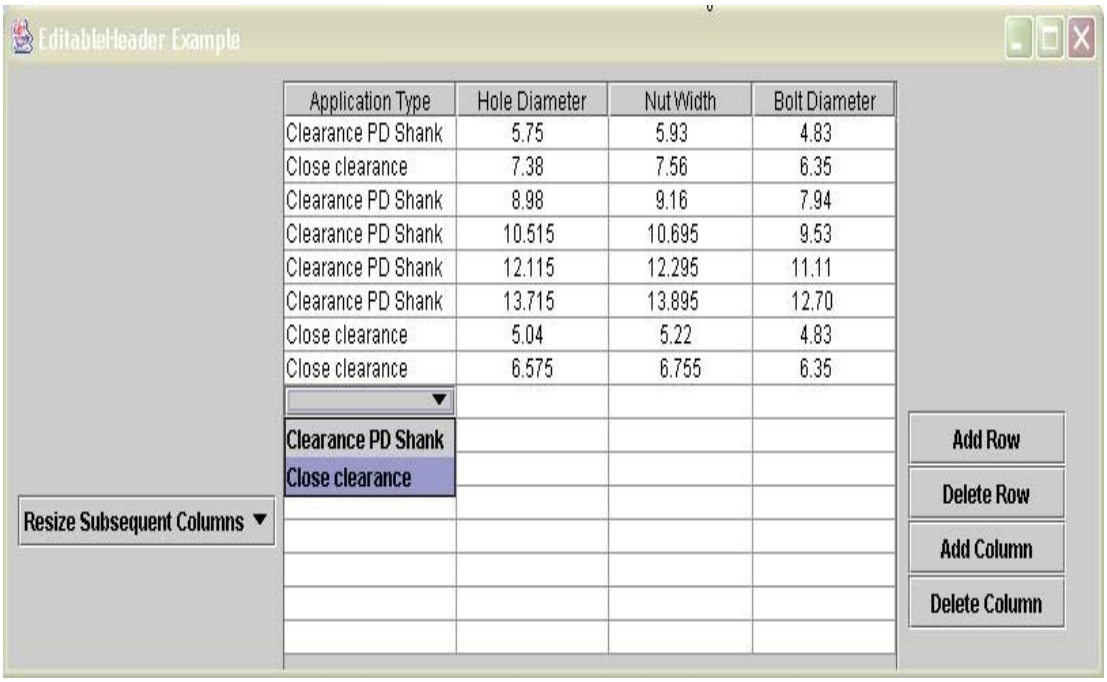


Figure 4.3: A screenshot showing constraints expressed in tables using ConEditor

(B) Taxonomy Panel: The taxonomy panel displays all the top level classes (i.e. classes having its parent as "Thing" in the OWL ontology) in the domain ontology together with their subclasses, properties (both object and datatype), and properties of the range classes of object properties and so on, as a taxonomy in a tree structure. Each class or object property can be expanded by a double mouse click to list all its subclasses and properties below it in the

taxonomy. Nodes represented by letter 'P' denote properties of a class. Clicking the "Add" button appends the selected node to the constraint expression being formed in the result panel. In the example considered, the entities *ConcreteFeature*, *max_operating_temp*, *has_material*, and *operating_temp* can be selected from this panel.

(C) Central Panel: This panel has two sections, namely constants and functions. In the constants section, two text fields are provided for inputting real and integer constants. Clicking the "update" button appends the constant to the constraint expression being formed. In the functions section, function buttons are provided for editing the constraint expression, adding a constant, refreshing the panel and creating a table. ConEditor provides a mechanism for inputting tables and exporting entities from the taxonomy panel to the table using the "Export" button. Figure 4.3 shows an example of constraints expressed in a table using ConEditor.

(D) Tool Bar: The tool bar displays the operators (arithmetic, relational and logical) and delimiters. A single mouse click on the selected operator will append it to the text area in the result panel. In the example considered, the operator '>=' and the delimiters '(', ')' can be selected from the tool bar.

(E) Result Panel: The result panel consists of a text area, displaying the constraint expression formulated by the user. Edit/Style menus are provided, on right-clicking the mouse in the result panel, which allow the user to undo/redo actions, cut, copy, paste text, specify the font and size of the text.

The implementation and functionality of ConEditor is described in the following section.

4.3 Functionality of ConEditor

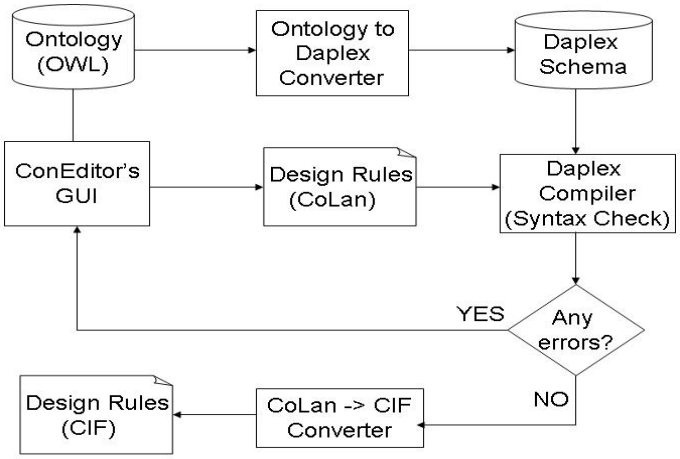


Figure 4.4: Process Flow within ConEditor

ConEditor has been developed in the Java programming language. The domain ontology is represented in the Web Ontology Language (OWL) (McGuinness & Harmelen, 2004) and has been developed using Protégé (Noy *et al.*, 2000). The ontology is parsed using Jena to extract classes and properties for display in the taxonomy panel. Design rules are captured by the domain experts as CoLan constraints over the domain ontology in OWL using ConEditor's GUI. The domain ontology in OWL is converted into a Daplex (Shipman, 1981) schema within ConEditor. This conversion process is described in Section 4.4. ConEditor makes use of the Daplex schema together with a Daplex compiler to verify the syntax of each constraint and report any syntactical errors. Constraints that are syntactically correct are then converted to a semantic web enabled XML Constraint Interchange Format (CIF) using a translator developed by Gray *et al*. (2001). The translator makes use of the Daplex schema for this conversion. An overview of the XML CIF format and the principles involved in the conversion of CoLan to CIF are provided in Section 4.5. Hence, the input to ConEditor is a CoLan constraint and the output is a constraint in CIF, provided the constraint is syntactically correct. Any syntactic errors are reported to the user. The process flow within ConEditor is as shown in Figure 4.4. The

constraints in XML Constraint Interchange Format (CIF) are then converted into Sicstus predicates and RDQL queries for processing by the Designers' Workbench. Both ConEditor and the Designers' Workbench use the same domain ontology represented in OWL. The framework of ConEditor and Designers' Workbench is as shown in Figure 4.5.

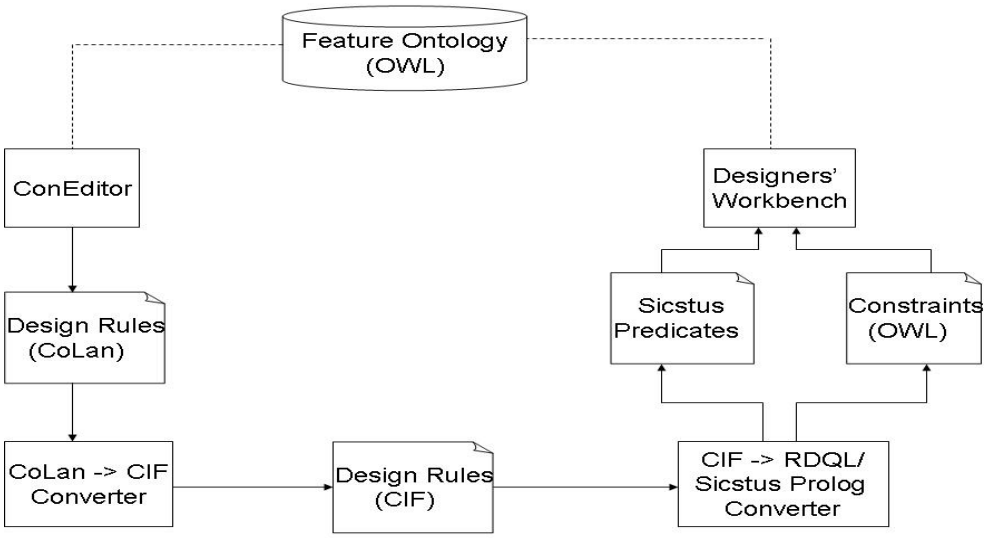


Figure 4.5: Framework of ConEditor and Designers' Workbench

4.4 Conversion of OWL ontology into Daplex Schema

ConEditor uses Jena (HP, 2004) to convert the domain ontology in OWL into an appropriate Daplex schema. Jena provides a set of APIs to read, create and manipulate ontologies. The Daplex schema is used by both the Daplex compiler and the CoLan to CIF translator. A similar transformation program to convert a XML DTD specification into Daplex schema has been implemented previously in Selpi (2004). This section describes the representation of inheritance hierarchies in P/FDM and the principles involved in conversion of OWL ontology into a Daplex Schema.

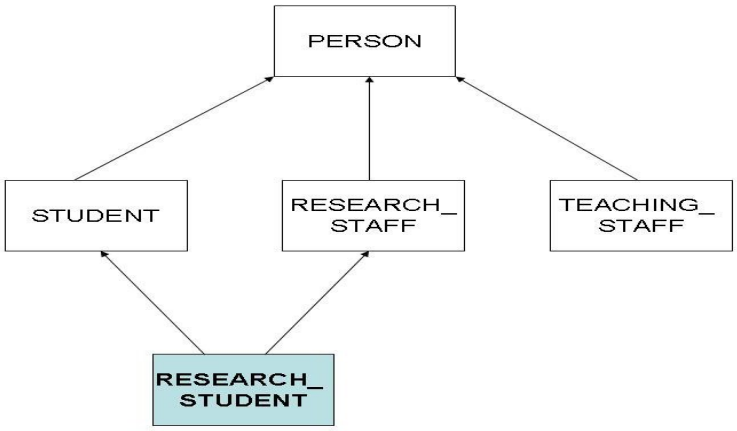


Figure 4.6 (a): Modelling research staff who are also students using multiple inheritance

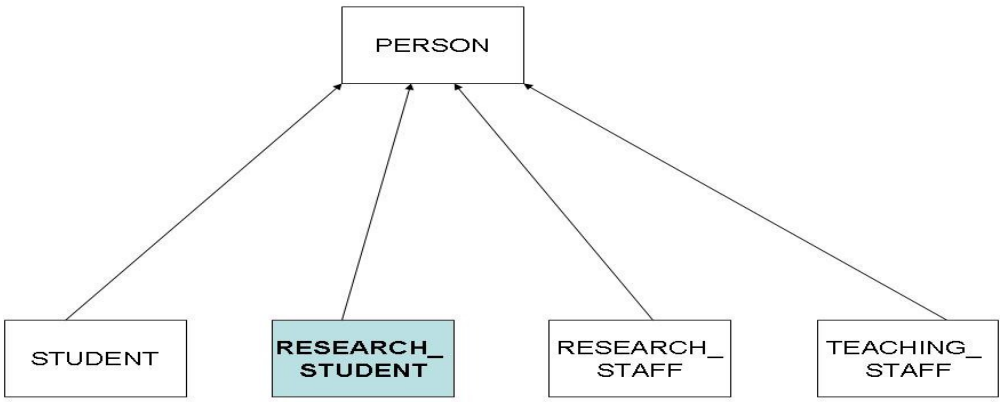


Figure 4.6 (b): Modelling research staff who are also students without using multiple inheritance

As in many semantic data models, entity classes in P/FDM may be arranged into inheritance hierarchies so that functions defined at higher levels in the hierarchy may be inherited by lower level classes. For example, one can abstract the common attributes of the 'teacher' and 'student' classes into a 'person' class:

```
declare person ->> entity                    %Subclass declaration
declare surname(person) -> string            %Function declaration
declare forename(person) -> string           %Function declaration
```

```
key_of person is surname, forename;
```

and can then make the original classes as subclasses of the ‘person’ class:

```
declare teacher ->> person                  % Subclass declaration
declare subject(teacher) -> string          %Function declaration
declare student ->> person                  % Subclass declaration
declare age(student) ->> integer;           %Function declaration
```

A key has been declared for the root class ‘person’. This ensures that the key is the same for all classes in the hierarchy, and that is defined in terms of functions that are inheritable by all classes in the hierarchy. Each class may have only one immediate superclass, so multiple inheritance is not supported. On the other hand, OWL ontologies support multiple inheritance. In practice, however, this is less of a restriction than might be supposed, due to the way in which P/FDM handles subclasses. Unlike many other data models, P/FDM supports overlapping subclasses, where an instance of a class may simultaneously belong to any number of its subclasses. ConEditor uses Jena to parse an OWL ontology and extract all the classes, subclasses, properties and their relations. These relations are then represented in an appropriate Daplex schema. A class that is a direct subclass of two or more parent classes is expressed as a subclass of the nearest common superclass of the parent classes. For example, if ‘Research_Student’ is a subclass of ‘Student’ and ‘Research_Staff’ as shown in Figure 4.6 (a) then ‘Research_Student’ is expressed as an immediate subclass of ‘Person’ as shown in Figure 4.6 (b).

4.5 XML Constraint Interchange Format (CIF)

This section provides an overview of CIF and the principles involved in the conversion of CoLan to CIF, as described in Gray *et al*. (2001).

XML-CIF is an open interchange format that supports a range of applications in which information needs to be moved across a network with rich metalevel information describing how the information can be used. XML-CIF is based on a well-established semantic data model (P/FDM) with an associated expressive constraint language (CoLan). To allow data instances to be transported across a network, the P/FDM data model has been mapped to a RDF Schema. RDF Schema is the simplest and most universal of the proposed semantic web data representations. To

allow constraints to be transported across a network, CIF has been developed in the form of a RDF Schema for CoLan. The basic design principles adopted are as follows:

- the CIF would need to be serialisable into XML, to make it maximally portable and open;
- constraints should be represented as resources in RDF, so that RDF statements can be made about the constraints themselves;
- there must be no modification to the existing RDF Schema specifications, so that the CIF would be layered cleanly on top of RDF;
- it must be possible for constraints to refer to terms defined in any RDF Schema, with such references made explicit.

```
% objmet - superclass of entity and prop-
erty metaclasses
declare objmet ->> entity
declare oname(objmet) -> string

% entmet - the metaclass of all en-
tity classes
declare entmet ->> objmet
declare super(entmet) -> entmet
declare rdfname(entmet) -> string
% link to RDF Schema

% propmet - the metaclass of all proper-
ties (functions)
declare propmet ->> objmet
declare fname(propmet) -> string
declare firstargtype(propmet) -> entmet
declare resulttype(propmet) -> entmet
declare has_inv(propmet) -> boolean
declare rdfname(propmet) -> string
% link to RDF Schema
```

Figure 4.7: P/FDM Daplex definitions for entity and property metaclasses

Source: Gray *et al.* (2001)

The entity-relational basis of both P/FDM data model and RDF makes it relatively straightforward to map from the former to the latter. The RDF Schema for CIF has been guided by the existing grammar for CoLan (Gray *et al*., 1999a) that relates

constraints to entities, attributes and relationships present in the ER model. This grammar serves as a metaschema for the CoLan constraints. The implementation of the P/FDM semantic data model makes use of an 'entmet' class that holds information on all entity classes, and a 'propmet' class that holds information on relationships (functions), both stored and derived (Embury & Gray, 1995). The P/FDM Daplex definitions of the 'entmet' and 'propmet' classes are shown in Figure 4.7, together with their superclass 'objmet'. The property 'rdfname' on the 'entmet' and 'propmet' classes holds the unique URI for a RDF resource, and thus provides an explicit link to the RDF Schema definition for the corresponding RDF Schema class or property. Thus, constraints carry explicit relationships to the domain ontology (as represented by a RDF Schema) for the terminology to which they refer.

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

<rdfs:Class rdf:ID="objmet">
  <rdfs:subClassOf rdf:resource=
  "http://www.w3.org/2000/01/rdf-schema#Class"/>
</rdfs:Class>

<rdf:Property rdf:ID="oname">
  <rdfs:domain rdf:resource="#objmet"/>
  <rdfs:range rdf:resource=
  "http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>

<rdfs:Class rdf:ID="entmet">
  <rdfs:subClassOf rdf:resource="#objmet"/>
</rdfs:Class>

<rdf:Property rdf:ID="super">
  <rdfs:domain rdf:resource="#entmet"/>
  <rdfs:range rdf:resource="#entmet"/>
</rdf:Property>

<rdf:Property rdf:ID="entmet_rdfname">
  <rdfs:domain rdf:resource="#entmet"/>
  <rdfs:range rdf:resource=
  "http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>
...
</rdf:RDF>
```

Figure 4.8: RDF Schema definitions for the objmet and entmet classes

Source: Gray *et al.* (2001)

Figure 4.8 shows the RDF Schema definitions corresponding to the Daplex definitions of the 'objmet' and 'entmet' classes in Figure 4.7. It is worth noting that, because properties in RDF are global, some of the original local P/FDM property names must be renamed (for example, 'entmet_rdfname' in Figure 4.8 is renamed from 'rdfname' in Figure 4.7). The basic rules used when mapping the P/FDM declarations to RDF Schema are as follows:

- A P/FDM class c defined as an 'entity' (declared as c->> entity) maps to an RDF resource of type rdfs:Class (where rdfs is the namespace prefix for the RDF Schema descriptions);
- A P/FDM class c declared to be a subtype of another class s (declared as c->> s) maps to a RDF resource of type rdfs:Class, with an rdfs:subClassOf property, the value of which is the class named s;
- A P/FDM function f declared on entities of class c, with result type r (declared as f(c) -> r) maps to a RDF resource of type rdf:Property with a rdfs:domain of c and a rdfs:range of r.

```
<rdfs:Class rdf:ID="variable">
  <rdfs:subClassOf rdf:resource="#singleton"/>
</rdfs:Class>

<rdf:Property rdf:ID="varname">
  <rdfs:domain rdf:resource="#variable"/>
  <rdfs:range rdf:resource=
  "http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>

<rdfs:Class rdf:ID="setmem">
  <rdfs:subClassOf rdf:resource="#boolprim"/>
</rdfs:Class>

<rdf:Property rdf:ID="var">
  <rdfs:domain rdf:resource="#setmem"/>
  <rdfs:range rdf:resource="#variable"/>
</rdf:Property>

<rdf:Property rdf:ID="set">
  <rdfs:domain rdf:resource="#setmem"/>
  <rdfs:range rdf:resource="#setexpr"/>
</rdf:Property>
```

Figure 4.9: RDF Schema definitions relating to the 'setmem' metaclass

Source: Gray *et al.* (2001)

A fundamental notion in CoLan is that a variable is always introduced in conjunction with a set that it ranges over. Thus, terms such as '(p in pc)' and '(e in employee)' are common, as in the example expressions:

(p in pc) such that name (p) = "xxx"
(e in employee) such that salary(e) > 5000 and age(e) < 50

This is represented in the syntax by the 'setmem' metaclass, while variables themselves are described by the 'variable' class, both defined as shown in Figure 4.9. An instance of the 'variable' class is a legal instance of an 'expr' class (representing an expression) by virtue of a series of subclass relationships. An example XML-CIF fragment corresponding to the CoLan fragment '(p in pc)' is shown in Figure 4.10.

```
<cif:setmem>
  <cif:var>
    <cif:variable ID="#p">
      <cif:varname>p</cif:varname>
    </cif:variable>
  </cif:var>
  <cif:set>
    <cif:entset>
      <cif:entclass>
      http://www.aktors.org/domain/pc_config#pc
      </cif:entclass>
    </cif:entset>
  </cif:set>
</cif:setmem>
```

Figure 4.10: XML-CIF fragment corresponding to the CoLan fragment (p in pc)

Source: Gray *et al.* (2001)

In summary, the CIF RDF Schema serves the purpose of describing what are called valid constraints, themselves expressed at an instance level in RDF. It combines the information in a grammar, which is normally used by a syntax checker or a parser, with information normally held in a database schema. It should be noted that the metaschema makes a clean separation between the description of constraints (both universal and existential) and expressions. Constraints and their boolean components are a representation of first-order logic, with the usual connectives. Any

knowledge source that uses FOL should be able to understand this. Expressions refer to facts about entities, their subtypes, attributes and relationships, and is based on the concepts of an ER model, which are very widely used. The ER model abstracts over relational storage, flat files and object-oriented storage, following the principle of data independence. It does not tie to any particular system, such as Oracle or P/FDM.

4.6 Summary

This chapter has described the design, implementation and functionality of ConEditor. The chapter has also provided an overview of the constraint representation languages used by ConEditor. CoLan is the high-level constraint language used by ConEditor to capture constraints. A constraint expression can be created using ConEditor's GUI by selecting entities from a domain ontology and combining them with a pre-defined set of keywords and operators from CoLan. The domain ontology in OWL is converted into a Daplex Schema by ConEditor. The principles involved in the conversion have been described in this chapter. The constraint captured in CoLan is checked for any syntax errors by the Daplex compiler, which uses the Daplex schema. If there are no errors in the constraint, it is converted into a semantic web enabled Constraint Interchange Format using a translator, which also uses the Daplex schema. An overview of the Constraint Interchange Format and the principles involved in the conversion of CoLan to CIF has been described in this chapter. The following chapter describes the verification and refinement of constraints captured by ConEditor.

Chapter 5

Verification and Refinement of Constraints

'Logic is the technique by which
we add conviction to truth.'
- Jean de la Bruyere

This chapter introduces the concept of an application condition associated with a constraint. The chapter analyses the kite domain and describes how the application conditions together with the constraint and the corresponding domain ontology can be used to support the maintenance of constraints. Four main types of knowledge refinement rules are described with examples from the kite design KB. Further, the refinement rules are expressed in a formal notation (in first order logic) and proved that they are logically sound. The chapter is structured as follows: Section 5.1 provides an analysis of the kite domain and introduces the concept of an application condition. Section 5.2 describes the proposed refinement rules with examples. Section 5.3 provides a formal representation of the refinement rules in first-order logic together with logical proofs. Section 5.4 provides a summary of the chapter.

5.1 Analysis of the Kite Domain

Due to restricted availability of Rolls-Royce designers' time and because it is a simpler domain, the kite domain was initially investigated. The sources referred to study the kite domain include Yolen (1976), Streeter (1980), Eden (1998), AKA (2006), CEKS (2006), Leigh (2006), Lords (2006) and Wardley (2006). There are different types of constraints associated with the design of a kite. Several constraints are applicable only to particular types of kites and under specific conditions The studies helped elicit constraints and their application conditions to form a knowledge base for the kite domain. Appendix A provides a list of equations and constraints elicited from the kite domain together with the corresponding application conditions and sources.

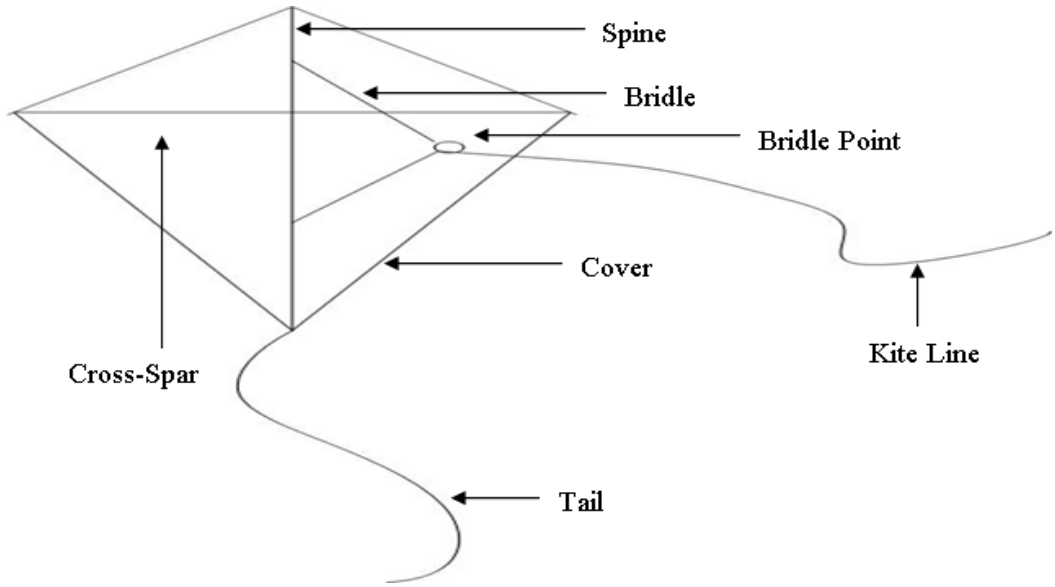


Figure 5.1: Basic Parts of a flat diamond kite

Consider the design of a flat diamond kite. Figure 5.1 shows this type of kite with its basic parts labelled. There are various constraints involved in the design of a kite depending on the type of kite, wind conditions, etc. For a successful design (design that leads to a kite that can fly), one has to ensure that the appropriate constraints are satisfied. A sample constraint with its application condition is given below in CoLan:

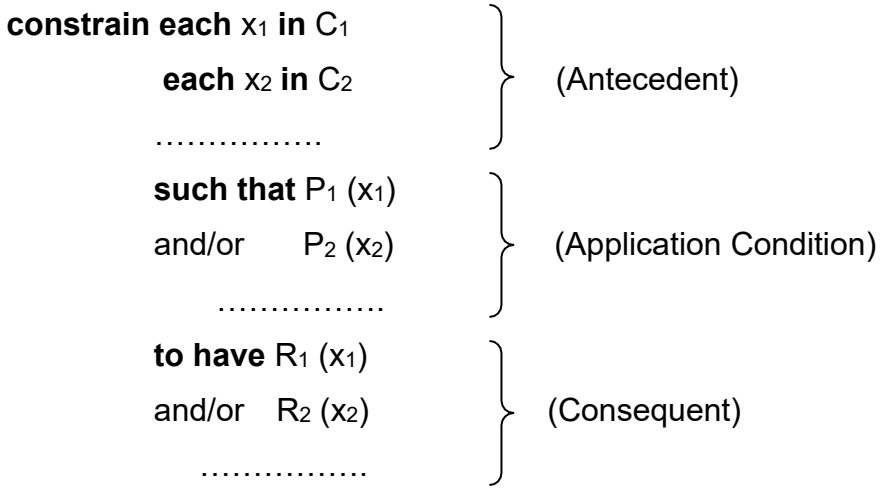
constrain each k **in** Kite
such that *has_type(k) = "Flat" and has_shape(k) = "Diamond"*
to have tail_length(has_tail(k)) = 7 * spine_length(has_spine(k))

The above constraint represented in first-order logic is :

$\forall$ k [(Kite(k) ^ (has_type(k) = "Flat") ^ (has_shape(k) = "Diamond")) $\rightarrow$
(tail_length(has_tail(k)) = 7 * spine_length(has_spine(k)))]

The context in which a constraint is applicable is referred to as an application condition in this thesis. The application condition states the condition when a particular constraint is applicable and forms a part of the rationale associated with the constraint. In the CoLan version of the above constraint, the application condition (in italics) is introduced by the clause "such that". The clause "such that" is already a part of CoLan language and can be used to express conditional statements. The author aims to make use of this clause "such that" to express an application condition associated with the constraint. The above constraint states that "For every instance of the class Kite, *when the type of the kite is flat and shape of the kite is diamond*, the length of the tail of the kite needs to be seven times the length of the spine of the kite". The phrase in italics (above) is the application condition of the constraint.

In order to make it clearer, the thesis divides a constraint represented in CoLan into three parts namely antecedent, application condition and consequent. Thus, a constraint can be represented by the following general structure in CoLan:



where x_1, x_2 represent variables that belong to classes C_1 and C_2 respectively; P_1 (x_1), P_2 (x_2) , R_1 (x_1), R_2 (x_2) represent properties.

Thus, the example constraint from the kite domain consists of:

Antecedent: **constrain each** k **in** Kite

Application condition: **such that** *has_type(k) = "Flat"*
and has_shape(k) ="Diamond"

Consequent: tail_length(has_tail(k)) = 7 * spine_length(has_spine(k))

A constraint in CoLan, in general, can be represented by a first order-logic sentence as:

$S \equiv \forall x_1,\ldots x_n [(C_1(x_1) \wedge \ldots\ldots \wedge C_n(x_n) \wedge P_1(x_1,\ldots x_n) \wedge \ldots\ldots \wedge P_m(x_1,\ldots x_n)) \rightarrow R(x_1,\ldots x_n)]$

where S is a sentence; $x_1,\ldots x_n$ are variables; $C_1,\ldots C_n$ are classes and $P(x_1,\ldots x_n)$, $P_m(x_1,\ldots x_n)$, $R(x_1,\ldots x_n)$ represent predicates/properties.

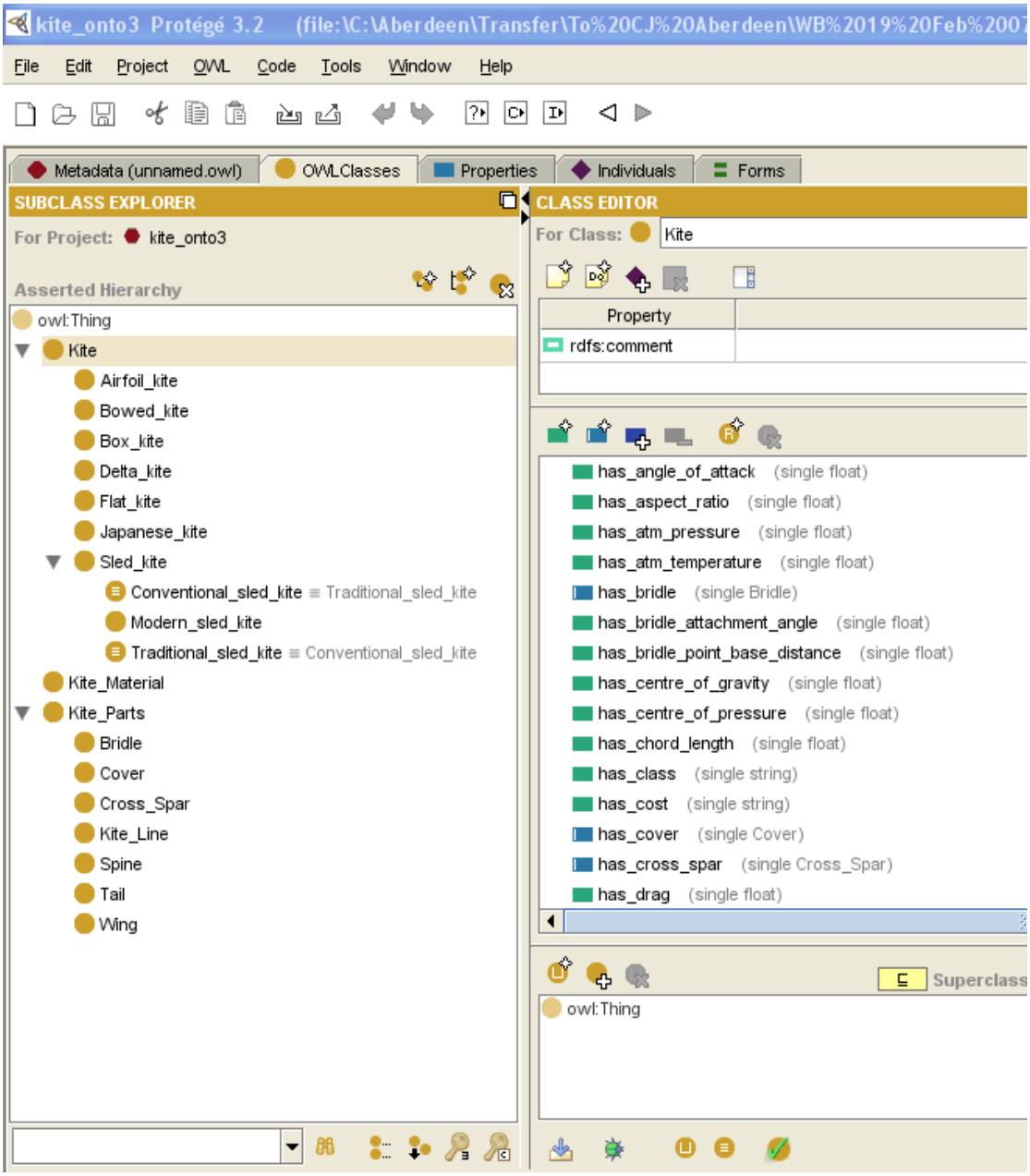


Figure 5.2: The Kite Domain ontology developed in Protégé

A domain ontology for the kite domain was developed as part of the analysis. Figure **5.2** shows the kite domain ontology developed using the Protégé editor.

5.2 Knowledge Refinement Rules

The information inherent in an application condition can be used together with the constraint and the associated domain ontology to support the maintenance of constraints. The thesis proposes four main types of knowledge refinement rules namely, redundancy, subsumption, inconsistency and fusion. The knowledge refinement rules are described below with examples from the kite domain. A formal notation in first-order logic for each knowledge refinement rule described below together with the logical proof is provided in Section 5.3.

5.2.1 Redundancy

Redundancy occurs between constraints when all the components of a constraint (antecedent, application condition and consequent) are equivalent to the corresponding components of another constraint. There are three types of redundancy and each type is described below with examples:

(a) Duplication

(i) **constrain each** c **in** ConventionalSledKite
such that has_level(c) = “beginner”
to have density(has_material(has_cover(c))) < 0.5

(ii) **constrain each** c **in** ConventionalSledKite
such that has_level(c) = “beginner”
to have density(has_material(has_cover(c))) < 0.5

By comparing the two constraints above, one can infer that the constraints (i) and (ii) are identical.

(b) Class Equivalence

(iii) **constrain each** c **in** ConventionalSledKite
such that has_level(c) = “beginner”
to have density(has_material(has_cover(c))) < 0.5

(iv) **constrain each** t **in** TraditionalSledKite
such that has_level(t) = “beginner”
to have density(has_material(has_cover(t))) < 0.5

If *CoventionalSledKite* is specified an equivalent class to *TraditionalSledKite* in the domain ontology one can infer that, the constraint (iii) is equivalent to constraint (iv).

(c) Property Equivalence

(v) **constrain each** c **in** ConventionalSledKite
such that has_level(c) = “beginner”
to have density(has_material(has_cover(c))) < 0.5

(vi) **constrain each** c **in** ConventionalSledKite
such that has_class(c) = “beginner”
to have density(has_material(has_cover(c))) < 0.5

If *has_level* is an equivalent property to *has_class* in the domain ontology one can infer that the constraint (v) is equivalent to constraint (vi).

The domain expert can be notified of all such redundancies and suggested by a system that he/she considers eliminating this redundancy.

5.2.2 Subsumption

Subsumption occurs between a pair of constraints when one constraint “covers” all the conditions of another constraint, i.e., constraint A subsumes constraint B iff constraint B is contained in constraint A. More formally, constraint A subsumes constraint B iff constraint A logically implies constraint B. An alternate way of

defining constraint subsumption (more formally) is as follows: Let us assume that we have a relation "satisfy(C, Sigma)" which holds if Sigma, a first-order substitution (of the form X/t, where X is a variable and t is a term), assigns values to each variable in C such that C holds. For instance, satisfy((20 < X < 40), {X/21}), …, satisfy((20 < X < 40), {X/39}) all hold. This relation can be used to check if a substitution satisfies the constraints and also to find all substitutions, one at a time. With this auxiliary relationship, we can define constraint subsumption as: constraint A subsumes constraint B if, and only if, for all Sigma such that satisfy(A, Sigma) holds, then satisfy(B, Sigma) also holds. There are three types of subsumption and each type is described below with examples:

(a) Subsumption via sub-class:

(vii) **constrain each** s **in** SledKite
such that has_size(s) = "standard"
to have kite_line_strength(has_kite_line(s)) >= 15

(viii) **constrain each** c **in** ConventionalSledKite
such that has_size(c) = "standard"
to have kite_line_strength(has_kite_line(c)) >= 15

If *ConventionalSledKite* is a subclass of *SledKite* in the domain ontology one can infer that the constraint (vii) subsumes constraint (viii). The domain expert can be notified of this fact and suggested by a system that he/she considers removing or deactivating constraint (viii).

(b) Subsumption via application condition

(ix) **constrain each** s **in** SledKite
such that has_size(s) = "standard" or has_size(s) = "large"
to have kite_line_strength(has_kite_line(s)) >= 15

(x) **constrain each** s **in** SledKite
such that has_size(s) = “standard”
to have kite_line_strength(has_kite_line(s)) >= 15

By comparing the two constraints above, one can infer that the constraint (ix) subsumes constraint (x) because the application condition of constraint (ix) includes the application condition of constraint (x). The domain expert can be notified of this fact and suggested by a system that he/she considers removing or deactivating constraint (x).

(c) Subsumption via conjunction

(xi) **constrain each** s **in** SledKite
such that has_size(s) = “standard”
to have kite_line_strength(has_kite_line(s)) >= 15 and
has_cord_length(s) > 21

(xii) **constrain each** s **in** SledKite
such that has_size(s) = “standard”
to have kite_line_strength(has_kite_line(s)) >= 15

Again, one can infer that the constraint (xi) subsumes constraint (xii) because the consequent of constraint (xi) includes the consequent of constraint (xii). The domain expert can be notified of this fact and suggested by a system that he/she considers removing or deactivating constraint (xii).

5.2.3 Inconsistency/Contradiction

An inconsistency/contradiction occurs between a pair of constraints when the consequent of one constraint contradicts (is inconsistent with) the consequent of another constraint while the antecedents and application conditions are equivalent i.e., constraint A contradicts constraint B or vice-versa if the consequents of constraints A and B cannot hold together.

(xiii) **constrain each** k **in** Kite
such that has_type(k) = “stunt”
to have kite_line_strength(has_kite_line(k)) > 30

(xiv) **constrain each** k **in** Kite
such that has_type(k) = “stunt”
to have kite_line_strength(has_kite_line(k)) < 25

By comparing the two constraints above, one can infer that the constraint (xiii) contradicts constraint (xiv). The domain expert can be notified of this fact and suggested by a system that he/she takes an appropriate action (modify/delete) to resolve the inconsistency.

5.2.4 Fusion

Fusion occurs between a pair of constraints when the two constraints can be combined together and replaced with another constraint, i.e. two constraints A and B can be fused together and replaced by a constraint C, iff constraint C implies constraint A and constraint C implies constraint B. There are three types of fusion and each type is described below with examples:

(a) Fusion via class

(xv) **constrain each** c **in** ConventionalSledKite
such that has_wind_condition(c) = “moderate”
to have has_bridle_attachment_angle(c) < 40

(xvi) **constrain each** m **in** ModernSledKite
such that has_wind_condition(m) = “moderate”
to have has_bridle_attachment_angle(m) < 40

If *ConventionalSledKite* and *ModernSledKite* are the only two subclasses of *SledKite* in the domain ontology and if every instance of *SledKite* is an instance of either

ConventionalSledKite or *ModernSledKite* then the constraints (xv) and (xvi) can be fused together and replaced by the constraint (xvii) as follows:

(xvii) **constrain each** s **in** SledKite
such that has_wind_condition(s) = “moderate”
to have has_bridle_attachment_angle(s) < 40

(b) Fusion via application condition

(xviii) **constrain each** j **in** JapaneseKite
such that has_wind_condition(j) = “strong”
to have has_bridle_point_distance(j) > 3 * surface_area(has_cover(j))

(xix) **constrain each** j **in** JapaneseKite
such that has_type(j) = “stunt”
to have has_bridle_point_distance(j) > 3 * surface_area(has_cover(j))

The constraints above can be fused together by using “or” between the application conditions, i.e., the constraints (xviii) and (xix) can be fused together and replaced by the constraint (xx) as follows:

(xx) **constrain each** j **in** JapaneseKite
such that has_wind_condition(j) = “strong” or has_type(j) = “stunt”
to have has_bridle_point_distance(j) > 3 * surface_area(has_cover(j))

(c) Fusion via conjunction

(xxi) **constrain each** j **in** JapaneseKite
such that has_wind_condition(j) = “strong”
to have has_bridle_point_distance(j) > 3 * surface_area(has_cover(j))

(xxii) **constrain each** j **in** JapaneseKite
such that has_wind_condition(j) = “strong”
to have kite_line_strength(has_kite_line(j)) >= 15

The constraints above can be fused together by using "and", i.e., the constraints (xxi) and (xxii) can be fused together and replaced by the constraint (xxiii) as follows:

(xxiii) **constrain each** j **in** JapaneseKite
such that has_wind_condition(j) = "strong"
to have has_bridle_point_distance(j) > 3 * surface_area(has_cover(j))
and kite_line_strength(has_kite_line(j)) >= 15

In all the above cases of fusion, the domain expert can be notified of this fact and suggested that he/she considers fusing constraints to reduce the size of the KB and possibly make it easier to maintain.

In all the examples above, universally quantified constraints involving a single variable have been considered for the sake of simplicity and also because they were common in the kite domain KB. However, more complex first-order logic expressions involving existential quantifiers or a combination of both existential and universal quantifiers can also be expressed in CoLan.

Thus, four main types of knowledge refinement rules among constraint pairs have been described with examples. All the refinement rules (except inconsistency) have sub-types: (1) Redundancy: (a) duplication (b) class equivalence (c) property equivalence (2) Subsumption: (a) via subclass (b) via application condition (c) via conjunction (3) Inconsistency (4) Fusion: (a) via class (b) via application condition (c) conjunction.

Knowledge refinement rules can be combined and applied together to a pair of constraints, which require the application of multiple refinement operators. For an example, consider the following constraints:

(E1) **constrain each** s **in** SledKite
such that has_type(s) = "stunt" or
has_wind_condition(s) = "strong"
to have kite_line_strength(has_kite_line(s)) > 30

(E2) **constrain each** c **in** ConventionalSledKite
such that has_type(c) = "stunt"
to have kite_line_strength(has_kite_line(c)) < 25

Now, if *ConventionalSledKite* is a subclass of *SledKite* in the domain ontology, then by comparing the constraints (E1) and (E2), one can infer that:

(a) The application condition of constraint (E2) is covered by the application condition of constraint (E1).

(b) The consequent of constraint (E1) contradicts the consequent of constraint (E2).

Hence, one can infer that the constraint (E1) contradicts constraint (E2) and makes the KB inconsistent. The domain expert can be notified of this fact and suggested that he/she takes an appropriate action (modify/delete). In the example above, a combination of the following knowledge refinement rules have been applied:

(a) Subsumption via subclass (b) Subsumption via application condition (c) Inconsistency. The next section (Section 5.3) provides a formal notation for all the above described knowledge refinement rules together with logical proofs.

5.3 Formal Notation and Logical Proof

Symbols:

OWL ontology classes: C_1, D_1, E_1, …..,C_n, D_n, E_n.

OWL ontology properties/predicates as:

(i) Application Conditions - P, Q, P_1, Q_1, ..., P_m, Q_m

(ii) Consequents: R, R_1,… , R_n

Variables: x_1, y_1,z_1, ..., x_n, y_n, z_n

Sentences: S, S_1 ,…. S_n

Logical Equivalence: ≡

Logical Implication: →

Logical Biconditional: ↔

Integer variables: m, n = 1, 2,…

$C_1(x_1)$, $D_1(x_1)$, $E_1(x_1)$, …..: $x_1 \in C_1$, $x_1 \in D_1$, $x_1 \in E_1$,……

Predicates: $P_1(x_1,….x_n)$, $P_m(x_1,….x_n)$, $Q(x_1,….x_n)$,…….

Refinement Rules:

5.3.1 Redundancy

(a) Duplication

If

$S_1 \equiv \forall\ x_1,\ldots x_n\ [(C_1(x_1) \wedge \ldots\ldots \wedge C_n(x_n) \wedge P_1(x_1,\ldots x_n) \wedge \ldots\ldots \wedge P_m(x_1,\ldots x_n)) \rightarrow R(x_1,\ldots x_n)]$

and

$S_2 \equiv \forall\ y_1,\ldots y_n\ [(C_1(y_1) \wedge \ldots\ldots \wedge C_n(y_n) \wedge P_1\ (y_1,\ldots y_n) \ldots\ldots \wedge P_m(y_1,\ldots y_n)) \rightarrow R(y_1,\ldots y_n)]$

Then

$S_1 \equiv S_2$.

Proof:

Given:

$S_1 \equiv \forall\ x_1,\ldots x_n\ [(C_1(x_1) \wedge \ldots\ldots \wedge C_n(x_n) \wedge P_1(x_1,\ldots x_n) \wedge \ldots\ldots \wedge P_m(x_1,\ldots x_n)) \rightarrow R(x_1,\ldots x_n)]$

and

$S_2 \equiv \forall\ y_1,\ldots y_n\ [(C_1(y_1) \wedge \ldots\ldots \wedge C_n(y_n) \wedge P_1\ (y_1,\ldots y_n) \ldots\ldots \wedge P_m(y_1,\ldots y_n)) \rightarrow R(y_1,\ldots y_n)]$

Goal: $S_1 \equiv S_2$

All the variables in a well-formed formula can be renamed consistently without altering the semantics of the formula. This means that for any well-formed formula P(x) and variable y that does not occur in P(x), we have $\forall x\ P(x) \leftrightarrow \forall y\ P(y)$.

By using the above axiom and renaming S_2, we have

$S_2 \equiv \forall\ x_1,\ldots x_n\ [(C_1(x_1) \wedge \ldots\ldots \wedge C_n(x_n) \wedge P_1(x_1,\ldots x_n) \wedge \ldots\ldots \wedge P_m(x_1,\ldots x_n)) \rightarrow R(x_1,\ldots x_n)]$

Hence we can conclude $S_1 \equiv S_2$.

(b) Class Equivalence

If

$S_1 \equiv \forall\ x_1,\ldots x_n\ [(C_1(x_1) \wedge \ldots\ldots \wedge C_n(x_n) \wedge P_1(x_1,\ldots x_n) \wedge \ldots\ldots \wedge P_m(x_1,\ldots x_n)) \rightarrow R(x_1,\ldots x_n)]$

and

$S_2 \equiv \forall\ y_1,\ldots y_n\ [(D_1(y_1) \wedge \ldots\ldots \wedge D_n(y_n) \wedge P_1(y_1,\ldots y_n) \ldots\ldots \wedge P_m(y_1,\ldots y_n)) \rightarrow R(y_1,\ldots y_n)]$

and

$[(D_i$ owl:equivalentClass $C_i)$ or $(D_i = C_i)]$, $1 \leq i \leq n$ hold

Then

$S_1 \equiv S_2$.

Proof:

Given:

$S_1 \equiv \forall\ x_1,\ldots x_n\ [(C_1(x_1) \wedge \ldots\ldots \wedge C_n(x_n) \wedge P_1\ (x_1,\ldots x_n) \wedge \ldots\ldots \wedge P_m(x_1,\ldots x_n)) \rightarrow R(x_1,\ldots x_n)]$

and

$S_2 \equiv \forall\ y_1,\ldots y_n\ [(D_1(y_1) \wedge \ldots\ldots \wedge D_n(y_n) \wedge P_1\ (y_1,\ldots y_n) \ldots\ldots \wedge P_m(y_1,\ldots y_n)) \rightarrow R(y_1,\ldots y_n)]$

and

$[(D_i$ owl:equivalentClass $C_i)$ or $(D_i = C_i)]$, $1 \leq i \leq n$ hold (1)

Goal: $S_1 \equiv S_2$

We can replace D with C in S_2 and C with D in S_1. [Using (1)]

Hence we can get S_2 from S_1 and vice-versa. [$\forall x\ P(x) \leftrightarrow \forall y\ P(y)$]

Therefore $S_1 \equiv S_2$.

(c) Property Equivalence

If

$S_1 \equiv \forall\ x_1,\ldots x_n\ [(C_1(x_1) \wedge \ldots\ldots \wedge C_n(x_n) \wedge P_1(x_1,\ldots x_n) \wedge \ldots\ldots \wedge P_m(x_1,\ldots x_n)) \rightarrow R_1(x_1,\ldots x_n)]$

and

$S_2 \equiv \forall\ y_1,\ldots y_n\ [(C_1(y_1) \wedge \ldots.. \wedge C_n(y_n) \wedge Q_1(y_1,\ldots y_n) \wedge \ldots.. \wedge Q_m(y_1,\ldots y_n)) \rightarrow R_2(y_1,\ldots y_n)]$

and

$[(P_i$ owl:equivalentProperty $Q_i)$ or $(P_i = Q_i)]$, $1 \leq i \leq m$ hold

and

$[(R_1$ owl:equivalentProperty $R_2)$ or $(R_1 = R_2)]$ hold

Then

$S_1 \equiv S_2$.

Proof:

Given:

$S_1 \equiv \forall\ x_1,\ldots x_n\ [(C_1(x_1) \wedge \ldots.. \wedge C_n(x_n) \wedge P_1(x_1,\ldots x_n) \wedge \ldots.. \wedge P_m(x_1,\ldots x_n)) \rightarrow R_1(x_1,\ldots x_n)]$

and

$S_2 \equiv \forall\ y_1,\ldots y_n\ [(C_1(y_1) \wedge \ldots.. \wedge C_n(y_n) \wedge Q_1(y_1,\ldots y_n) \wedge \ldots.. \wedge Q_m(y_1,\ldots y_n)) \rightarrow R_2(y_1,\ldots y_n)]$

and

$[(P_i$ owl:equivalentProperty $Q_i)$ or $(P_i = Q_i)]$, $1 \leq i \leq m$ hold (2)

and

$[(R_1$ owl:equivalentProperty $R_2)$ or $(R_1 = R_2)]$ hold (3)

Goal: $S_1 \equiv S_2$

We can replace Q with P in S_2 and P with Q in S_1. [Using (2)]

Similarly, we can replace R_2 with R_1 in S_2 and R_1 with R_2 in S_1. [Using (3)]

Hence we can get S_2 from S_1 and vice-versa. $[\forall x\ P(x) \leftrightarrow \forall y\ P(y)]$

Therefore $S_1 \equiv S_2$.

5.3.2 Subsumption

(a) Subsumption via sub-class

If

$S_1 \equiv \forall\ x_1,\ldots x_n\ [(C_1(x_1) \wedge \ldots.. \wedge C_n(x_n) \wedge P_1(x_1,\ldots x_n) \wedge \ldots.. \wedge P_m(x_1,\ldots x_n)) \rightarrow R(x_1,\ldots x_n)]$

and

$S_2 \equiv \forall\ y_1,\ldots y_n\ [(D_1(y_1) \wedge \ldots.. \wedge D_n(y_n) \wedge P_1(y_1,\ldots y_n) \wedge \ldots.. \wedge P_m(y_1,\ldots y_n)) \rightarrow R(y_1,\ldots y_n)]$

and

$[(D_i$ owl:subClassOf $C_i)$ or $(D_i = C_i)]$, $1 \leq i \leq n$ hold

Then

S_1 subsumes S_2 (i.e. S_1 logically implies S_2).

Proof:

Given:

$S_1 \equiv \forall\ x_1,\ldots x_n\ [(C_1(x_1) \wedge \ldots.. \wedge C_n(x_n) \wedge P_1(x_1, \ldots x_n) \wedge \ldots.. \wedge P_m(x_1,\ldots x_n)) \rightarrow R(x_1,\ldots x_n)]$

and

$S_2 \equiv \forall\ y_1,\ldots y_n\ [(D_1(y_1) \wedge \ldots.. \wedge D_n(y_n) \wedge P_1(y_1,\ldots y_n) \wedge \ldots.. \wedge P_m(y_1,\ldots y_n)) \rightarrow R(y_1,\ldots y_n)]$

and

$[(D_i$ owl:subClassOf $C_i)$ or $(D_i = C_i)]$, $1 \leq i \leq n$ hold

Goal: S_1 subsumes S_2 (i.e. S_1 logically implies S_2).

Converting S_1 and S_2 into clausal form by using Universal Elimination, $(P \rightarrow Q) \equiv (\neg P \vee Q)$ and DeMorgan's law, we have

$S_1 \equiv \neg C_1(x_1) \vee \ldots.. \vee \neg C_n(x_n) \vee \neg P_1(x_1,\ldots x_n) \vee \ldots.. \vee \neg P_m\ (x_1,\ldots x_n) \vee R\ (x_1,\ldots x_n)$

and

$S_2 \equiv \neg D_1(y_1) \vee \ldots.. \vee \neg D_n(y_n) \vee \neg P_1(y_1,\ldots y_n) \vee \ldots.. \vee \neg P_m\ (y_1,\ldots y_n) \vee R\ (y_1,\ldots y_n)$

where x_i and y_i are any values $(1 \leq i \leq n)$.

To prove S_1 subsumes S_2 , S_1 holds.

Now $[(D_i$ owl:subClassOf $C_i)$ or $(D_i = C_i)]$, $1 \leq i \leq n$ hold implies that all instances of D_i are also instances of C_i. Hence, C_i subsumes D_i where $1 \leq i \leq n$ hold (i.e. whenever the constraints in C_i hold the same constraints must also hold in D_i).

Therefore, S_1 subsumes S_2 (i.e. S_1 logically implies S_2).

This can be explained further by considering the following example:

S_1: The total number of wheels in Vehicles must be between 2 and 8 (inclusive).

If Car is a subclass of Vehicle in the ontology then whenever the constraints in S_1 for Vehicle hold then the same constraints must also hold for Car. Therefore the

constraint stating that the total number of wheels must be between 2 and 8 (inclusive) must hold for Car.

(b) Subsumption via application condition

If

$S_1 \equiv \forall\ x_1,\ldots.x_n\ [(C_1(x_1) \wedge \ldots.. \wedge C_n(x_n) \wedge ((P_1(x_1,\ldots.x_n) \wedge \ldots.. \wedge P_m(x_1,\ldots.x_n)) \vee Q(x_1,\ldots.x_n))) \rightarrow R(x_1,\ldots.x_n)]$

and

$S_2 \equiv \forall\ y_1,\ldots.y_n\ [(C_1(y_1) \wedge \ldots.. \wedge C_n(y_n) \wedge P_1(y_1,\ldots.y_n) \wedge \ldots.. \wedge P_m(y_1,\ldots.y_n)) \rightarrow R(y_1,\ldots.y_n)]$

Then

S_1 subsumes S_2 (i.e. S_1 logically implies S_2).

Proof:

Given:

$S_1 \equiv \forall\ x_1,\ldots.x_n\ [(C_1(x_1) \wedge \ldots.. \wedge C_n(x_n) \wedge ((P_1(x_1,\ldots.x_n) \wedge \ldots.. \wedge P_m(x_1,\ldots.x_n)) \vee Q(x_1,\ldots.x_n))) \rightarrow R(x_1,\ldots.x_n)]$

and

$S_2 \equiv \forall\ y_1,\ldots.y_n\ [(C_1(y_1) \wedge \ldots.. \wedge C_n(y_n) \wedge P_1(y_1,\ldots.y_n) \wedge \ldots.. \wedge P_m(y_1,\ldots.y_n)) \rightarrow R(y_1,\ldots.y_n)]$

Goal: S_1 subsumes S_2 (i.e. S_1 logically implies S_2).

Converting S_1 and S_2 into clausal form by using Universal Elimination, $(P \rightarrow Q) \equiv (\neg P \vee Q)$, Distributive law $[A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C)]$ and DeMorgan's law,

we have

$S_1 \equiv \neg C_1(x_1) \vee \ldots.. \vee \neg C_n(x_n) \vee (\neg P_1(x_1,\ldots.x_n) \wedge \neg Q(x_1,\ldots.x_n)) \vee \ldots.. \vee (\neg P_m(x_1,\ldots.x_n) \wedge \neg Q(x_1,\ldots.x_n)) \vee R(x_1,\ldots.x_n)$

and

$S_2 \equiv \neg C_1(y_1) \vee \ldots.. \vee \neg C_n(y_n) \vee \neg P_1(y_1,\ldots.y_n) \vee \ldots.. \vee \neg P_m(y_1,\ldots.y_n) \vee R(y_1,\ldots.y_n)$

where x_i and y_i are any values $(1 \leq i \leq n)$.

To prove S_1 subsumes S_2 , let us S_1 holds.

Now $(P \wedge Q) \rightarrow P$.

Therefore, S_1 logically implies S_2 (i.e. S_1subsumes S_2).

This can be explained further by considering the following example:

S_1: If a Vehicle is either a Car or a Truck then the total number of wheels in it must be between 2 and 8 (inclusive).

A disjunction (Car or Truck) is used in S_1. Hence, whenever the consequent (i.e. "total number of wheels must be between 2 and 8 (inclusive)") in S_1 for vehicle holds then the constraint stating that the total number of wheels must be between 2 and 8 (inclusive) must also hold for both Car and Truck.

(c) Subsumption via conjunction

If

$S_1 \equiv \forall\ x_1,\ldots.x_n\ [(C_1(x_1) \wedge \ldots.. \wedge C_n(x_n) \wedge P_1(x_1,\ldots.x_n) \wedge \ldots.. \wedge P_m(x_1,\ldots.x_n)) \rightarrow (R_1(x_1,\ldots.x_n) \wedge R_2(x_1,\ldots.x_n))]$

and

$S_2 \equiv \forall\ y_1,\ldots.y_n\ [(C_1(y_1) \wedge \ldots.. \wedge C_n(y_n) \wedge P_1(y_1,\ldots.y_n) \wedge \ldots.. \wedge P_m(y_1,\ldots.y_n)) \rightarrow R_1(y_1,\ldots.y_n)]$

Then

S_1 subsumes S_2 (i.e. S_1 logically implies S_2).

Proof:

Given:

$S_1 \equiv \forall\ x_1,\ldots.x_n\ [(C_1(x_1) \wedge \ldots.. \wedge C_n(x_n) \wedge P_1(x_1,\ldots.x_n) \wedge \ldots.. \wedge P_m(x_1,\ldots.x_n)) \rightarrow (R_1(x_1,\ldots.x_n) \wedge R_2(x_1,\ldots.x_n))]$

and

$S_2 \equiv \forall\ y_1,\ldots.y_n\ [(C_1(y_1) \wedge \ldots.. \wedge C_n(y_n) \wedge P_1(y_1,\ldots.y_n) \wedge \ldots.. \wedge P_m(y_1,\ldots.y_n)) \rightarrow R_1(y_1,\ldots.y_n)]$

Goal: S_1 subsumes S_2 (i.e. S_1 logically implies S_2).

Converting S_1 and S_2 into clausal form by using Universal Elimination,

$(P \rightarrow Q) \equiv (\neg P \vee Q)$ and DeMorgan's law, we have

$S_1 \equiv \neg C_1(x_1) \vee \ldots.. \vee \neg C_n(x_n) \vee \neg P_1\ (x_1,\ldots.x_n) \vee \ldots.. \vee \neg P_m\ (x_1,\ldots.x_n) \vee (R_1(x_1,\ldots.x_n) \wedge R_2(x_1,\ldots.x_n))$

and

$S_2 \equiv \neg C_1(y_1) \vee \ldots\ldots \vee \neg C_n(y_n) \vee \neg P_1(y_1,\ldots y_n) \vee \ldots\ldots \vee \neg P_m(y_1,\ldots y_n) \vee R_1(y_1,\ldots y_n)$

where x_i and y_i are any values ($1 \leq i \leq n$).

To prove S_1subsumes S_2 , let us assume S_1 holds.

Now, $(P \wedge Q) \rightarrow P$.

Therefore, S_1 logically implies S_2 (i.e. S_1subsumes S_2).

This can be explained further by considering the following example:

S_1: All Cars must have 6 wheels and 4 doors.

A conjunction (6 wheels and 4 doors) is used in S_1. Hence, whenever the consequent (i.e. "must have 6 wheels and 4 doors") for Cars in S_1 hold then the consequent of 6 wheels must hold in Car and the consequent of 4 doors must also hold in Car.

5.3.3 Inconsistency

If

$S_1 \equiv \forall\, x_1,\ldots x_n\, [(C_1(x_1) \wedge \ldots\ldots \wedge C_n(x_n) \wedge P_1(x_1,\ldots x_n) \wedge \ldots\ldots \wedge P_m(x_1,\ldots x_n)) \rightarrow R_1(x_1,\ldots x_n)]$

and

$S_2 \equiv \forall\, y_1,\ldots y_n\, [(C_1(y_1) \wedge \ldots\ldots \wedge C_n(y_n) \wedge P_1(y_1,\ldots y_n) \wedge \ldots\ldots \wedge P_m(y_1,\ldots y_n)) \rightarrow R_2(y_1,\ldots y_n)]$

and

$\forall\, x_1, y_1, \ldots, x_n, y_n$ $R_1(x_1,\ldots x_n)$ and $R_2(y_1,\ldots y_n)$ are inconsistent (i.e. the conditions of both R_1 and R_2 cannot hold together).

Then

S_1 and S_2 are inconsistent (i.e. the conditions specified in the consequents of both S_1 and S_2 cannot hold together).

Proof:

Given:

$S_1 \equiv \forall\, x_1,\ldots x_n\, [(C_1(x_1) \wedge \ldots\ldots \wedge C_n(x_n) \wedge P_1(x_1,\ldots x_n) \wedge \ldots\ldots \wedge P_m(x_1,\ldots x_n)) \rightarrow R_1(x_1,\ldots x_n)]$

and

$S_2 \equiv \forall\ y_1,\ldots y_n\ [(C_1(y_1) \wedge \ldots.. \wedge C_n(y_n) \wedge P_1(y_1,\ldots y_n) \wedge \ldots.. \wedge P_m(y_1,\ldots y_n)) \rightarrow R_2(y_1,\ldots y_n)]$

and

$\forall\ x_1, y_1, \ldots, x_n, y_n\ R_1\ (x_1,\ldots x_n)$ and $R_2\ (y_1,\ldots y_n)$ are inconsistent (i.e. the conditions of both R_1 and R_2 cannot hold together).

Goal: S_1 and S_2 are inconsistent (i.e. the conditions specified in the consequents of both S_1 and S_2 cannot hold together).

Converting S_1 and S_2 into clausal form by using Universal Elimination, $(P \rightarrow Q) \equiv (\neg P \vee Q)$ and DeMorgan's law, we have

$S_1 \equiv \neg C_1(x_1) \vee \ldots.. \vee \neg C_n(x_n) \vee \neg P_1(x_1,\ldots x_n) \vee \ldots.. \vee \neg P_m\ (x_1,\ldots x_n) \vee R_1(x_1,\ldots x_n)$

and

$S_2 \equiv \neg C_1(y_1) \vee \ldots.. \vee \neg C_n(y_n) \vee \neg P_1(y_1,\ldots y_n) \vee \ldots.. \vee \neg P_m\ (y_1,\ldots y_n) \vee R_2\ (y_1,\ldots y_n)$,

where x_i and y_i are any values $(1 \leq i \leq n)$.

Now $R_1\ (x_1,\ldots x_n)$ and $R_2\ (y_1,\ldots y_n)$ are inconsistent (i.e. the conditions of both R_1 and R_2 cannot hold together), where x_i and y_i are any values $(1 \leq i \leq n)$.

Hence we can conclude that S_1 and S_2 are inconsistent (i.e. the conditions specified in the consequents of both S_1 and S_2 cannot hold together).

5.3.4 Fusion

(a) Fusion via class

If

$S_1 \equiv \forall\ x_1,\ldots x_n\ [(C_1(x_1) \wedge \ldots.. \wedge C_n(x_n) \wedge P_1(x_1,\ldots x_n) \wedge \ldots.. \wedge P_m(x_1,\ldots x_n)) \rightarrow R(x_1,\ldots x_n)]$

and

$S_2 \equiv \forall\ y_1,\ldots y_n\ [(D_1(y_1) \wedge \ldots.. \wedge D_n(y_n) \wedge P_1(y_1,\ldots y_n) \wedge \ldots.. \wedge P_m(y_1,\ldots y_n)) \rightarrow R(y_1,\ldots y_n)]$

and

$C_i \cup D_i = E_i$, $1 \leq i \leq n$ hold [i.e. C_i and D_i are the only two subclasses of E_i in the domain ontology and every instance of E_i is an instance of either C_i or D_i]

Then

S_1 and S_2 can be replaced by S_3 as follows:

$S_3 \equiv \forall\ z_1, z_2, \ldots z_n\ [(E_1\ (z_1) \wedge \ldots\ldots \wedge E_n(z_n) \wedge P_1(z_1,\ldots z_n) \wedge \ldots\ldots \wedge P_m(z_1,\ldots z_n)) \rightarrow R(z_1,\ldots z_n)]$

Proof:

Given:

$S_1 \equiv \forall\ x_1,\ldots x_n\ [(C_1(x_1) \wedge \ldots\ldots \wedge C_n(x_n) \wedge P_1(x_1,\ldots x_n) \wedge \ldots\ldots \wedge P_m(x_1,\ldots x_n)) \rightarrow R(x_1,\ldots x_n)]$

and

$S_2 \equiv \forall\ y_1,\ldots y_n\ [(D_1(y_1) \wedge \ldots\ldots \wedge D_n(y_n) \wedge P_1(y_1,\ldots y_n) \wedge \ldots\ldots \wedge P_m(y_1,\ldots y_n)) \rightarrow R(y_1,\ldots y_n)]$

and

$C_i \cup D_i = E_i$, $1 \leq i \leq n$ hold [i.e. C_i and D_i are the only two subclasses of E_i in the domain ontology and every instance of E_i is an instance of either C_i or D_i]

Goal: S_1 and S_2 can be replaced by S_3 as follows:

$S_3 \equiv \forall\ z_1, z_2, \ldots z_n\ [(E_1\ (z_1) \wedge \ldots\ldots \wedge E_n(z_n) \wedge P_1(z_1,\ldots z_n) \wedge \ldots\ldots \wedge P_m(z_1,\ldots z_n)) \rightarrow R(z_1,\ldots z_n)]$

From the proof of 5.3.2(a), we can infer that S_3 subsumes S_1 (i.e. S_3 logically implies S_1) and S_3 subsumes S_2 (i.e. S_3 logically implies S_2). Hence we can conclude that S_1 and S_2 can be replaced by S_3.

(b) Fusion via application condition

If

$S_1 \equiv \forall\ x_1,\ldots x_n\ [(C_1(x_1) \wedge \ldots\ldots \wedge C_n(x_n) \wedge P_1(x_1,\ldots x_n) \wedge \ldots\ldots \wedge P_m(x_1,\ldots x_n)) \rightarrow R(x_1,\ldots x_n)]$

and

$S_2 \equiv \forall\ y_1,\ldots y_n\ [(C_1(y_1) \wedge \ldots\ldots \wedge C_n(y_n) \wedge Q(y_1,\ldots y_n)) \rightarrow R(y_1,\ldots y_n)]$

Then

S_1 and S_2 can be replaced by S_3 as follows:

$S_3 \equiv \forall\ z_1,\ldots z_n\ [(C_1(z_1) \wedge \ldots\ldots \wedge C_n(x_n) \wedge ((P_1(z_1,\ldots z_n) \wedge \ldots\ldots \wedge P_m(z_1,\ldots z_n)) \vee Q(z_1,\ldots z_n))) \rightarrow R(z_1,\ldots z_n)]$

Proof:

Given:

$S_1 \equiv \forall\ x_1,\ldots x_n\ [(C_1(x_1) \wedge \ldots\ldots \wedge C_n(x_n) \wedge P_1(x_1,\ldots x_n) \wedge \ldots\ldots \wedge P_m(x_1,\ldots x_n)) \rightarrow R(x_1,\ldots x_n)]$

and

$S_2 \equiv \forall\ y_1,\ldots y_n\ [(C_1(y_1) \wedge \ldots\ldots \wedge C_n(y_n) \wedge Q(y_1,\ldots y_n)) \rightarrow R(y_1,\ldots y_n)]$

Goal: S_1 and S_2 can be replaced by S_3 as follows:

$S_3 \equiv \forall\ z_1,\ldots z_n\ [(C_1(z_1) \wedge \ldots\ldots \wedge C_n(x_n) \wedge ((P_1(z_1,\ldots z_n) \wedge \ldots\ldots \wedge P_m(z_1,\ldots z_n)) \vee Q(z_1,\ldots z_n))) \rightarrow R(z_1,\ldots z_n)]$

From the proof of 5.3.2(b), we can infer that S_3 subsumes S_1 (i.e. S_3 logically implies S_1) and S_3 subsumes S_2 (i.e. S_3 logically implies S_2). Hence we can conclude that S_1 and S_2 can be replaced by S_3.

(c) Fusion via conjunction

If

$S_1 \equiv \forall\ x_1,\ldots x_n\ [(C_1(x_1) \wedge \ldots\ldots \wedge C_n(x_n) \wedge P_1(x_1,\ldots x_n) \wedge \ldots\ldots \wedge P_m(x_1,\ldots x_n)) \rightarrow R_1(x_1,\ldots x_n)]$

and

$S_2 \equiv \forall\ y_1,\ldots y_n\ [(C_1(y_1) \wedge \ldots\ldots \wedge C_n(y_n) \wedge P_1(y_1,\ldots y_n) \wedge \ldots\ldots \wedge P_m(y_1,\ldots y_n)) \rightarrow R_2(y_1,\ldots y_n)]$

Then

S_1 and S_2 can be replaced by S_3 as follows:

$S_3 \equiv \forall\ z_1,\ldots z_n\ [(C_1\ (z_1) \wedge \ldots\ldots \wedge C_n(z_n \wedge P_1(z_1,\ldots z_n) \wedge \ldots\ldots \wedge P_m(z_1,\ldots z_n)) \rightarrow (R_1(z_1,\ldots z_n) \wedge R_2(z_1,\ldots z_n))]$

Proof:

Given:

$S_1 \equiv \forall\ x_1,\ldots x_n\ [(C_1(x_1) \wedge \ldots\ldots \wedge C_n(x_n) \wedge P_1(x_1,\ldots x_n) \wedge \ldots\ldots \wedge P_m(x_1,\ldots x_n)) \rightarrow R_1(x_1,\ldots x_n)]$

and

$S_2 \equiv \forall\ y_1,\ldots y_n\ [(C_1(y_1) \wedge \ldots\ldots \wedge C_n(y_n) \wedge P_1(y_1,\ldots y_n) \wedge \ldots\ldots \wedge P_m(y_1,\ldots y_n)) \rightarrow R_2(y_1,\ldots y_n)]$

Goal: S_1 and S_2 can be replaced by S_3 as follows:

$S_3 \equiv \forall\ z_1,\ldots.z_n\ [(C_1\ (z_1)\ \hat{}\ldots..\hat{}\ C_n(z_n\hat{}\ P_1(z_1,\ldots.z_n)\ \hat{}\ \ldots..\hat{}\ P_m(z_1,\ldots.z_n)) \rightarrow (R_1(z_1,\ldots.z_n)\ \hat{}\ R_2(z_1,\ldots.z_n))]$

From the proof of 5.3.2(c), we can infer that S_3 subsumes S_1 (i.e. S_3 logically implies S_1) and S_3 subsumes S_2. (i.e. S_3 logically implies S_2). Hence we can conclude that S_1 and S_2 can be replaced by S_3.

A formal notation for all the proposed refinement rules has been provided in this section together with the logical proofs. Descriptions of these refinements with corresponding examples of constraints were provided in the previous section. It has to be noted that the units of all the numerical values have to be taken into account while comparing constraints. For example, consider the following two constraints:

(i) **constrain each** c **in** ModernSledKite
such that has_level(c) = “expert”
to have density(has_material(has_cover(c))) > 0.5

(ii) **constrain each** t **in** ModernSledKite
such that has_level(t) = “expert”
to have density(has_material(has_cover(t))) < 0.5

One can infer that the constraints (i) and (ii) above contradict each other only after considering the units of the numerical values in both constraints. If 0.5 specified in constraint (i) refers to 0.5 kilogram per square metre and 0.5 specified in constraint (ii) refers to 0.5 gram per square centimetre, then the constraints (i) and (ii) do not contradict each other. Hence, it is important to consider the units in such cases before making any inferences. Throughout this thesis, all the inferences made between constraint pairs are based on the assumption that the numerical values in both the constraints are specified using the same units. Organisations such as Rolls-Royce adopt a uniform set of units as part of their design standards to specify all the measurements. Moreover, in the methodology described in this thesis, the concepts and properties used in the constraints are taken from the domain ontology. Hence, the units used for all the measurements are to defined in the domain ontology, instead of explicitly specifying them in each constraint. As part of the future work, the author

plans to integrate the domain ontology with the engineering mathematics ontology developed by Gruber & Olsen (1994) to incorporate physical dimensions, units of measure, etc. and enhance the ability to ensure that there is consistency between the units inherent in the constraints.

5.4 Summary

The chapter provides an analysis of the kite domain and introduces the concept of an application condition associated with a constraint. Four main types of knowledge refinement rules have been proposed in this chapter. The rules have been described with examples from the kite domain. The rules detect redundancy, subsumption, inconsistency/contradiction and fusion by comparing constraints together with the corresponding application conditions and the domain ontology. With the help of an example, a description of how knowledge refinement rules can be combined together and applied to a pair of constraints has also been provided. A kite domain ontology has been developed using Protégé. Constraints and application conditions have been expressed, over the kite domain ontology in a high-level constraint language, namely, CoLan. Further, all the proposed refinement rules have been expressed in predicate calculus and proven to be logically sound. The importance of units associated with these constraints has also been stressed. The next chapter describes the implementation of a system that incorporates these refinement rules to support the maintenance of constraints.

Chapter 6

ConEditor+

> 'Knowledge is of no value unless you can put it into practice.'
> **- Anton Chekhov**

This chapter describes the design, implementation and functionality of ConEditor+. ConEditor+ is an extended version of ConEditor. ConEditor was initially developed mainly to facilitate domain experts themselves to capture constraints. ConEditor also provided basic maintenance facilities to detect syntax errors between constraints, and allows reading constraints from text files, editing the constraints and then writing them to the same or new file. Following encouraging results from the preliminary evaluation (described in Section 7.1, Chapter 7) of ConEditor, some changes were made to the GUI and the system was extended to provide additional support for maintenance by detecting inconsistencies, subsumption, redundancy, fusion between pairs of constraints and suggesting appropriate refinements. The extended system became known as ConEditor+.

The chapter is structured as follows: Section 6.1 highlights the main changes made when extending ConEditor to ConEditor+. Section 6.2 describes the design of ConEditor+'s GUI. Section 6.3 describes the implementation and functionality of ConEditor+. Section 6.4 outlines the algorithm implemented by ConEditor+ to verify and suggest refinements of constraints in CIF. Section 6.5 describes the interpretation of CIF constraints by ConEditor+. Section 6.6 provides a summary of the chapter.

6.1 Evolution from ConEditor to ConEditor+

ConEditor+ is an extended version of ConEditor. The main changes made from ConEditor to ConEditor+ can be summarised as follows:

(i) Extended support for maintenance: ConEditor only provided basic support for maintenance that included detecting syntax errors between constraints,

reading constraints from text files, editing the constraints and then writing them to the same or new file. In addition, ConEditor+ detects inconsistencies, subsumption, redundancy, fusion between pairs of constraints and suggests appropriate refinements to support maintenance. ConEditor+ uses the CIF representation of constraints to interpret and suggest the above refinements.

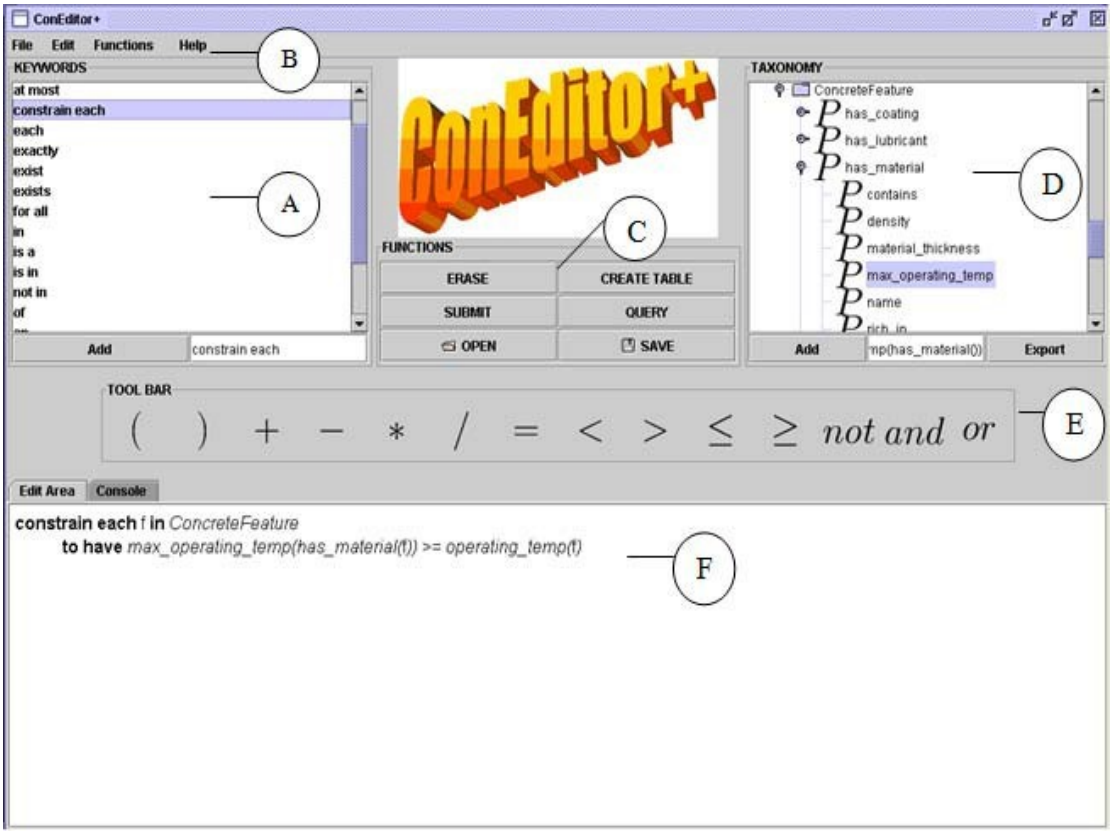


Figure 6.1: A screenshot of ConEditor+'s GUI

(ii) GUI modification (Figure 6.1): A few changes were made to the ConEditor's GUI. The layout was modified to enlarge the result panel. This was done to improve the readability of the output given by ConEditor+. An additional tab named "console" was created in the result panel to display the output messages from ConEditor+. Other changes include the addition of new components such as menu bar and a "Query" function button to enable keyword-based search of constraints. The keywords panel and taxonomy panel were added with new facilities to

improve the usability. Additionally, two modes, namely, “semi-auto” and “auto” were introduced. All the components of ConEditor+ are explained below.

The following section provides a detailed description of ConEditor+’s GUI.

6.2 ConEditor+’s GUI

ConEditor+’s GUI (Figure 6.1) essentially consists of six components, namely: (A) Keywords Panel, (B) Menu Bar, (C) Functions Panel, (D) Taxonomy Panel, (E) Tool Bar and (F) Result Panel. These components provide the user with entities required to form a constraint expression. The user can then choose the appropriate entities by clicking the mouse and so form a constraint expression. Appendix C presents an annotated walkthrough of constraint capture using screenshots of ConEditor+. The process of formulating a constraint using ConEditor+ is explained further by considering the same example, reported earlier in Chapter 4 for ConEditor. It is assumed that the consideration of the same example would make it easier for the reader to recognise the changes made to ConEditor’s GUI. This example is stated below:

```
constrain each f in ConcreteFeature
to have max_operating_temp(has_material(f)) >= operating_temp(f)
```

The above constraint states that for every instance of the class ConcreteFeature, the value of the maximum operating temperature of its material must be greater than or equal to the environmental operating temperature. ConEditor+’s six components are described below together with this example.

(A) Keywords Panel: The keywords panel consists of a list of keywords from the CoLan language. In the example considered, the keywords *constrain each*, *in*, *to have* can be expressed by selecting them from this panel. A single mouse click on an entity appends it to the text area in the result panel. Alternatively, clicking the “Add” button in the panel also appends it to the text area in the result panel.

(B) Menu Bar: The menu bar contains a list of menus and submenus with operations for loading, editing, deleting, searching and saving constraints, performing syntax checks, creating tables and so on. It also contains an option to choose one of the two modes, semi-auto and auto. ConEditor+ is set to auto mode by default. The differences between the two modes are described later in this section.

(C) Functions Panel: The functions panel consists of six buttons ('Erase', 'Create Table', 'Submit', 'Query', 'Open', 'Save') that can be clicked to perform some of the frequently used operations from the menu bar. This is provided for easier and quicker access as compared to the menu bar.

(D) Taxonomy Panel: The taxonomy panel lists all the top level classes (i.e. classes having its parent as "Thing" in OWL ontology) in the domain ontology together with their subclasses, properties (both object and datatype), and properties of the range classes of object properties and so on, as a taxonomy. Each class or object property can be expanded by a double mouse click to list all the subclasses and properties below it in the taxonomy. Clicking the "Add" button in the panel appends the selected entity (class or property) to the text area in the result panel. In Figure 6.2, one can see the class *ConcreteFeature* together with its properties *has_coating*, *has_lubricant*, *has_material* and so on. Now each object property has a range class associated with it. For each object property, all the properties of its range class are listed below it.

For example, in Figure 6.2, *has_material* is an object property having a range class *Material*. A double mouse click on *has_material* displays all the properties of *Material* class, i.e., *contains*, *density*, *material_thickness*, *max_operating_temp*, *name* and so on. For the example constraint considered, the entities *ConcreteFeature*, *max_operating_temp*, *has_material*, *operating_temp* can be selected from this panel and added to the result panel by clicking the "Add" button.

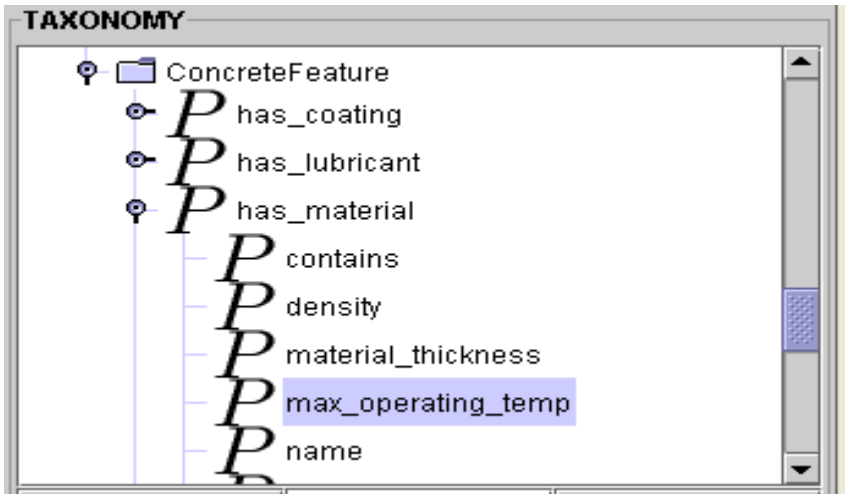


Figure 6.2: Taxonomy Panel of ConEditor+

(E) Tool Bar: The tool bar displays the operators (arithmetic, relational and logical) and delimiters. In the example considered, the operator '>=' and the delimiters '(', ')' can be selected from the tool bar. Again, a single mouse click on the selected operator will append it to the text area in the result panel.

(F) Result Panel: The result panel consists of a text area, displaying the constraint expression formulated by the user and any output messages (e.g., syntax error message) from ConEditor+. This panel consists of two tabs: namely, the "Edit Area" and the "Console" that displays the constraint expression formulated by the user and the output messages from the system respectively.

There are two modes in ConEditor+, namely, auto and semi-auto. ConEditor+ is set to auto mode by default. The differences between the two modes, auto and semi-auto are now described with respect to Figure 6.2:

Auto Mode: When using the auto (default) mode, selecting the property *max_operating_temp* and clicking the "Add" button appends *max_operating_temp(has_material())* to the text area in the result panel. This means that when a property is chosen, the properties in the levels above it in the taxonomy (*has_material* is chosen together with *max_operating_temp*) are automatically

appended to it. In addition, when using the auto mode, a combo-box listing all the alphabets (‘a’ – ‘z’) appears automatically in each position where a variable needs to be entered. In the example constraint considered above, combo-boxes appear in positions of all the occurrences of variable ‘f’ in the constraint. The positions where variable ‘f’ occurs in the example constraint are listed as follows:
(i) Between ‘*constrain each’* and ‘*in*’
(ii) Between ‘*max_operating_temp(has_material(’* and *‘))’*
(iii) Between ‘*operating_temp*(’ and ‘)’.
The user can then choose the appropriate variable required (‘f’ in the example constraint) instead of typing it using the keyboard.

Semi-auto mode: When using the semi-auto mode, selecting the property and clicking the “Add” button will append only that particular property to the text area in the result panel. The properties above it in the taxonomy are not automatically appended. Therefore, in the example considered, while using the semi-auto mode, selecting the property *max_operating_temp* and clicking the “Add” button will append only *max_operating_temp* to the result panel and not *max_operating_temp(has_material(),* as in the case of auto mode. In addition, when using a semi-auto mode, all the variables required in the constraint (‘f’ in the example constraint) need to be typed manually using the keyboard. Combo-boxes do not appear automatically at the positions where a variable needs to be entered.

The following section describes the functionality (including implementation) of ConEditor+.

6.3 Functionality of ConEditor+

The framework of ConEditor+ and Designers’ Workbench is as shown in Figure 6.3. The domain expert captures constraints in CoLan using ConEditor+. CoLan is converted into a standard semantic web enabled XML Constraint Interchange Format (CIF) using a translator. ConEditor+ processes the constraints in CIF to detect inconsistencies, subsumption, redundancy and fusion and suggest appropriate refinements between pairs of constraints. The processing of CIF by ConEditor+ to detect inconsistencies, subsumption, redundancy and fusion and suggest appropriate

refinements between pairs of constraints is explained in Section 6.5. Interpretation of CIF to support maintenance is the main difference between the frameworks of ConEditor+ and ConEditor, as indicated by an arrow from CIF to ConEditor+ in Figure 6.3. The constraints in CIF are converted into Sicstus predicates and RDQL queries for processing by the Designers' Workbench. Both ConEditor+ and the Designers' Workbench make use of the domain ontology represented in OWL. ConEditor+ converts the domain ontology in OWL into a Daplex schema that is used by both the Daplex compiler and the CoLan to CIF translator to process constraints in CoLan.

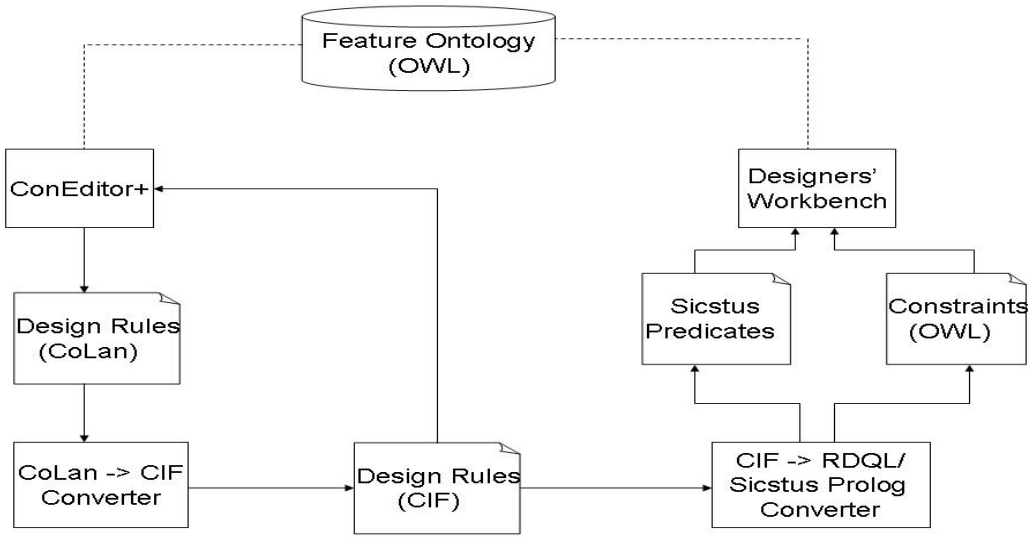


Figure 6.3: Framework of ConEditor+ and Designers' Workbench

When a constraint is modified and saved, ConEditor+ stores the modified constraint as a new version together with the original (before modification) constraint. The rationale for storing different versions of a constraint is to enable designers to study the constraint evolution (Goonetillake & Wikramanayake, 2004). Each constraint is allocated a unique identification number (ID) that also denotes its version number. For example, the first constraint stored is allocated "ver_1_CoLanKiteList.txt_1", the second "ver_1_CoLanKiteList.txt_2", etc. By default, ConEditor+ uses the latest version of each constraint. The system provides

facilities to retrieve constraints using keyword-based searches e.g., search and retrieve all the constraints containing the specified keyword(s) or the constraint associated with a specified ID. Hence, an old version of a constraint can be retrieved through the 'search' mechanism.

6.4 Algorithm

The algorithm used by ConEditor+ to determine the order in which refinement rules are applied, is outlined below. Consider a pair of constraints A and B. Let the antecedents be represented by AN_a and AN_b, application conditions by AC_a and AC_b, consequents by C_a and C_b for constraints A and B respectively.

Step 1: Check for redundancy (whether A is identical to B):

If AN_a not equal/equivalent to AN_b then go to step 2a.

If AC_a not equal/equivalent to AC_b then go to step 2a.

If C_a equal/equivalent to C_b then conclude redundancy, notify user (domain expert), suggest refinement action(s) and exit.

Step 2a: Check for subsumption (whether A subsumes B):

If AN_a not equal/equivalent/subsumes AN_b then go to step 2b.

If AC_a not equal/equivalent/subsumes AC_b then go to step 2b.

If C_a equal/equivalent/subsumes C_b then conclude subsumption, notify user (domain expert), suggest refinement action(s) and exit.

Step 2b: Check for subsumption (whether B subsumes A):

If AN_b not equal/equivalent/subsumes AN_a then go to step 3a.

If AC_b not equal/equivalent/subsumes AC_a then go to step 3a.

If C_b equal/equivalent/subsumes C_a then conclude subsumption, notify user (domain expert), suggest refinement action(s) and exit.

Step 3a: Check for inconsistency (whether A contradicts B):

If AN_a not equal/equivalent/subsumes AN_b then go to step 3b.

If AC_a not equal/equivalent/subsumes AC_b then go to step 3b.

If C_a contradicts C_b then conclude inconsistency, notify user (domain

expert), suggest refinement action(s) and exit.

Step 3b: Check for inconsistency (continued):

If AN_b not equal/equivalent/subsumes AN_a then go to step 4a.

If AC_b not equal/equivalent/subsumes AC_a then go to step 4a.

If C_a contradicts C_b then conclude inconsistency, notify user (domain expert), suggest refinement action(s) and exit.

Step 4a: Check for fusion (whether A and B can be fused):

If AN_a not equal/equivalent to AN_b then go to step 4c.

If AC_a not equal/equivalent to AC_b then go to step 4b.

Conclude that fusion is possible, notify user (domain expert), suggest refinement action(s) and exit.

Step 4b: Check for fusion (continued):

If C_a not equal/equivalent to C_b then exit.

Conclude that fusion is possible, notify user (domain expert), suggest refinement action(s) and exit.

Step 4c: Check for fusion (continued):

If AC_a not equal/equivalent to AC_b then exit.

If C_a not equal/equivalent to C_b then exit.

If AN_a can be fused with AN_b [using Rule 4 (a)] then conclude that fusion is possible, notify user (domain expert), suggest refinement action(s) and exit.

ConEditor+ has also been developed in the Java programming language. The domain ontology is represented in OWL and has been developed using Protégé. ConEditor+ captures constraints in the CoLan language that is based on the syntax of the Daplex language (Shipman, 1981; Bassiliades & Gray, 1995). ConEditor+ uses a translator developed by Gray *et al*. (2001) to convert CoLan to CIF. ConEditor+ also makes use of a Daplex compiler to verify the syntax of the constraint in CoLan and report any syntactic errors. The Daplex schema is used by both the Daplex compiler and the CoLan to CIF translator. ConEditor+ interprets the constraints in CIF and applies the algorithm outlined above to detect inconsistencies (contradictions) and to suggest

various ways to refine (fuse constraints, eliminate redundancies and subsumptions) pairs of constraints. ConEditor+'s interpretation of constraints in CIF is described further in Section 6.5.

When a new constraint is input (or submitted) into ConEditor+, it is first checked for any syntax errors. If there are no syntax errors, the submitted constraint is converted from CoLan into CIF. The CIF constraint is compared with every other CIF constraint in the KB. ConEditor+ reports any inconsistency, redundancy, subsumption or fusion found between the pairs of constraints. This results in a time complexity of O(n). ConEditor+ can also read a KB (i.e., a list of constraints) and perform comparison of all possible pairs of constraint expressions within the KB to detect inconsistencies and suggest refinements. Comparison of all possible pairs of constraints results in the time complexity of $O(n^2)$.

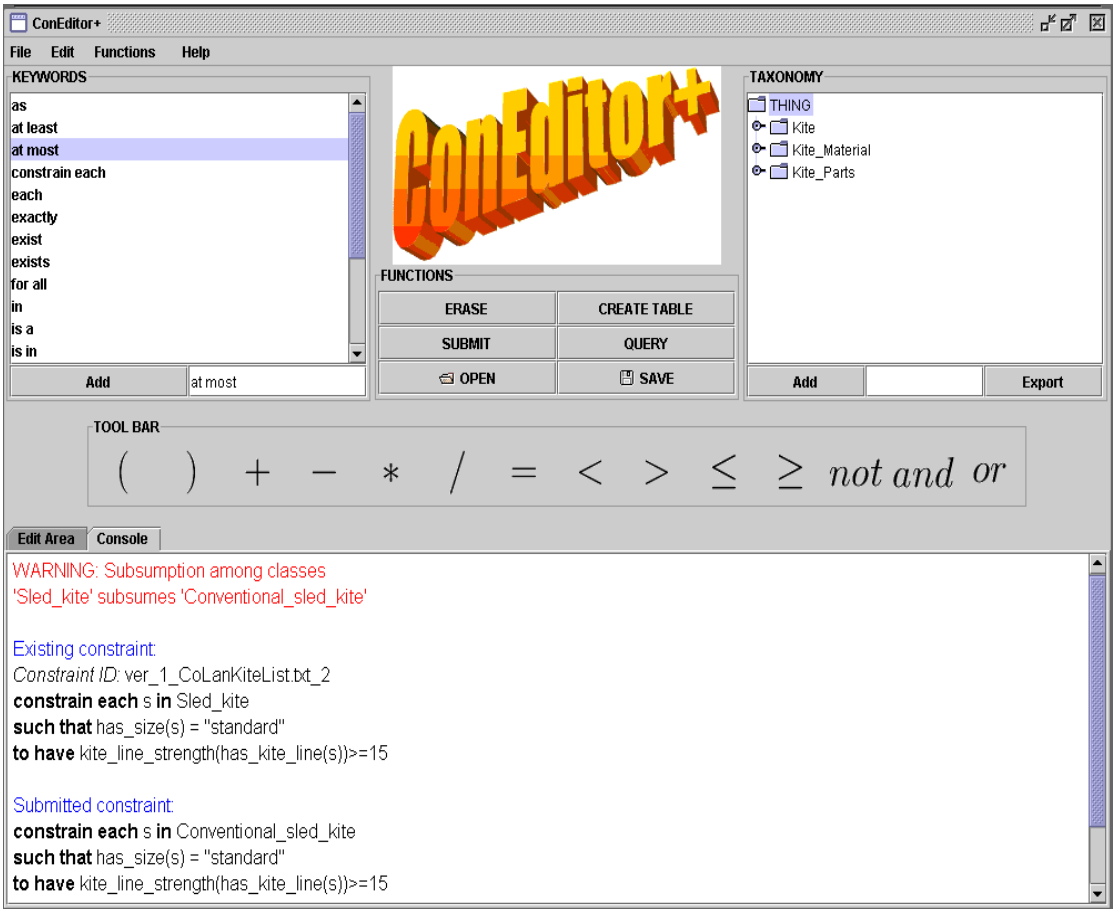


Figure 6.4: A screenshot of ConEditor+ showing subsumption between a pair of constraints

ConEditor+ compares constraints at the syntactical level, rather than comparing the solution sets. So ConEditor+ is comparing pairs of constraints of the form e.g., P(x1, x2) & Q(x1,x3,a) and P(x1, x2) & Q(x1,x3,b). By looking at the values of the constants (a, b), the structure of the predicates (P, Q), and inferring the relationship between the corresponding classes and properties in the domain ontology, ConEditor+ determines whether there is an inconsistency, subsumption, redundancy or fusion. Further, in each comparison, all the terms in one constraint are compared with all the corresponding terms in another constraint. Hence, the complexity of each comparison is $O(n^2)$. Figure 6.4 shows a screenshot of ConEditor+ with a message to notify the domain expert about subsumption between a pair of constraints.

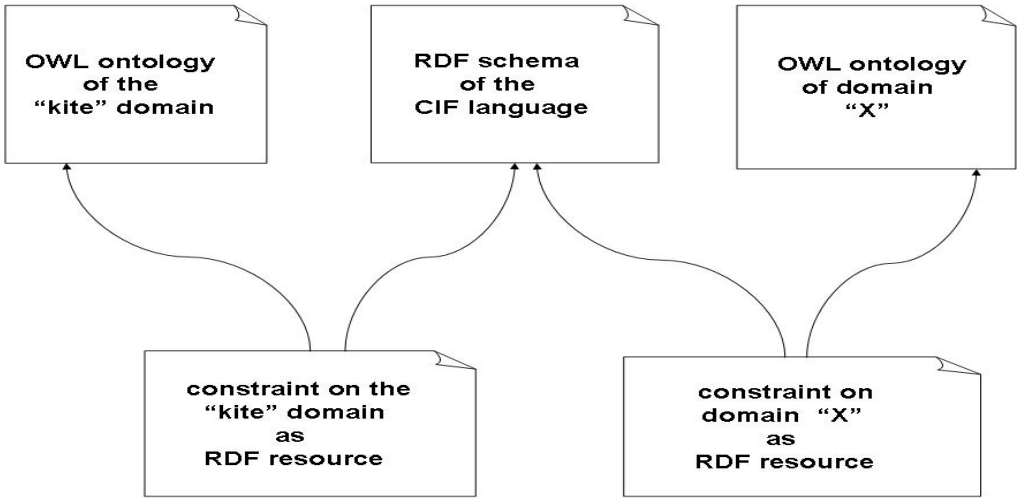


Figure 6.5: Constraints in RDF make references to the CIF language definition and the domain ontology in OWL

6.5 CIF Interpretation by ConEditor+

Section 4.5 in Chapter 4 provided an overview of CIF together with the principles involved in the conversion of CoLan to CIF. This section describes how ConEditor+ interprets the constraints in CIF in order to apply the algorithm (in Section 6.4) and suggest appropriate KB refinements. CIF constraints are encoded in RDF by defining

a RDF schema for the CIF language that is layered cleanly on top of RDF, serving as a metaschema (Gray *et al*., 2001). The RDF schema provides knowledge on the class hierarchy and class properties with type information. A constraint encoded in RDF makes explicit references to classes defined in the domain model (OWL ontology) as well as the CIF language definition in RDF Schema as shown in Figure 6.5. The CIF constraint class has three subclasses, namely, impliesconstr, unquantified_constraint and existsconstr. The impliesconstr class is used to represent fully quantified constraint expressions while existsconstr is used to represent existentially quantified constraint expressions. Further, each class has properties (object and data type properties) defined in the RDF schema. An ontology model is created with the classes and properties defined in the RDF Schema using Jena. The code snippet in Jena to do this is as shown below:

```
//create an ontology model
OntModel ontologyModel = ModelFactory.createOntologyModel();
// create classes in the ontology model
OntClass c_impliesconstr = ontologyModel
.createClass("http://www.csd.abdn.ac.uk/~khui/akt/cif/cifv2#impliesconstr");
OntClass c_unquantifiedconstr = ontologyModel
.createClass("http://www.csd.abdn.ac.uk/~khui/akt/cif/cifv2#unquantified_cons
traint");
OntClass  c_enset = ontologyModel
.createClass("http://www.csd.abdn.ac.uk/~khui/akt/cif/cifv2#entset");
OntClass c_enmet = ontologyModel
.createClass("http://www.csd.abdn.ac.uk/~khui/akt/cif/cifv2#entmet");
.
.
.
//create object properties in the ontology model
ObjectProperty p_qvar = ontologyModel
.createObjectProperty("http://www.csd.abdn.ac.uk/~khui/akt/cif/cifv2#impliesc
onstr_qvar");
ObjectProperty p_set = ontologyModel
```

```
.createObjectProperty("http://www.csd.abdn.ac.uk/~khui/akt/cif/cifv2#setmem
_set");
ObjectProperty p_prop = ontologyModel
.createObjectProperty("http://www.csd.abdn.ac.uk/~khui/akt/cif/cifv2#mvfncall
_prop");
.
.
//create data type properties in the ontology model
DatatypeProperty p_entmet_rdfname = ontologyModel
.createDatatypeProperty("http://www.csd.abdn.ac.uk/~khui/akt/cif/cifv2#entme
t_rdfname");
DatatypeProperty p_propmet_rdfname = ontologyModel
.createDatatypeProperty("http://www.csd.abdn.ac.uk/~khui/akt/cif/cifv2#propm
et_rdfname");
DatatypeProperty p_operator = ontologyModel
.createDatatypeProperty("http://www.csd.abdn.ac.uk/~khui/akt/cif/cifv2#comp
arison_operator");
.
.
.
```

Consider the following constraint in CoLan:

constrain each k **in** Kite
such that has_level(k) = "beginner"
to have density(has_material(has_cover(k))) > 0.5

The above constraint in English is as follows:
"Every kite that is of beginner level should have a cover material density greater than 0.5 units." The constraint can be divided into three parts as follows:
Antecedent: constrain each k in Kite
Application condition: such that has_level(k) = "beginner"
Consequent: to have density(has_material(has_cover(k))) > 0.5

Each CIF constraint is stored in a XML file (e.g., kite_cif_1.xml). The XML file is read (parsed) by the ontology model using Jena (e.g., ontologyModel.read(new

FileInputStream("kite_cif_1.xml"), null) parses the CIF constraint stored in 'kite_cif_1.xml'). A description of ConEditor+'s interpretation of the CIF constraint is given below by considering specific fragments of CIF representation.

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:cif="http://www.csd.abdn.ac.uk/~khui/akt/cif/cifv2#"
xml:base="http://www.csd.abdn.ac.uk/~sajit/constraint1#">

<cif:impliesconstr rdf:ID="test_constraint_1">
        <cif:impliesconstr_qvar>
                <cif:setmem>
                <cif:setmem_set>
                <cif:entset>
                <cif:entset_entclass>
                <cif:entmet rdf:ID="generic_4">
                <cif:entmet_rdfname>
                file:///D/exp2/kit/RR_Onto1.owl#Kite
                </cif:entmet_rdfname>
                </cif:entmet>
                </cif:entset_entclass>
                </cif:entset>
                </cif:setmem_set>
                <cif:setmem_var>
                <cif:variable rdf:ID="generic_2">
                <cif:variable_varname>
                uevar1
                </cif:variable_varname>
                </cif:variable>
                </cif:setmem_var>
                </cif:setmem>
        </cif:impliesconstr_qvar>
```

The RDF fragment above defines the namespaces and provides explicit references to the CIF language definition and the domain ontology in OWL. The fragment also defines instances, namely, 'test_constraint_1', 'generic_4' and 'generic_2' of the 'implies_constr class', 'entmet class' and 'variable' class respectively. The 'entmet' class has a property 'entmet_rdfname' that has a value of 'file:///D/exp2/kit/RR_Onto1.owl#Kite' representing the 'Kite' class. The 'variable' class has a property 'variable_varname' that has a value of 'uevar1' representing the variable 'k' in the CoLan constraint. The variable 'uevar1' is restricted to be an instance of the entity class 'Kite' which is defined by the domain ontology in 'file:///D/exp2/kit/RR_Onto1.owl#Kite'. The value of the 'cif:entmet_rdfname'

property provides a reference to the kite domain ontology in OWL . Thus, the above fragment defines the antecedent: constrain each k in Kite.

```
<cif:impliesconstr>
    <cif:impliesconstr_qvar>
        <cif:setmem>
        <cif:setmem_set>
        <cif:mvfncall>
        <cif:mvfncall_prop>
        <cif:propmet>
        <cif:propmet_fname>
        has_level
        </cif:propmet_fname>
        <cif:propmet_resulttype>
        <cif:entmet>
        <cif:entmet_rdfname>
        file:///D/exp2/kit/RR_Onto1.owl#string
        </cif:entmet_rdfname>
        </cif:entmet>
        </cif:propmet_resulttype>
        <cif:propmet_rdfname>
        file:///D/exp2/kit/RR_Onto1.owl#has_level
        </cif:propmet_rdfname>
        <cif:propmet_firstargtype>
        <cif:entmet rdf:about="#generic_4"/>
        </cif:propmet_firstargtype>
        </cif:propmet>
        </cif:mvfncall_prop>
        <cif:mvfncall_arg>
        <cif:variable rdf:about="#generic_2"/>
        </cif:mvfncall_arg>
        </cif:mvfncall>
        </cif:setmem_set>
        <cif:setmem_var>
        <cif:variable rdf:ID="generic_9">
        <cif:variable_varname>
        evar2
        </cif:variable_varname>
        </cif:variable>
        </cif:setmem_var>
        </cif:setmem>
    </cif:impliesconstr_qvar>
```

The above fragment defines a multi-valued function ‘has_level’ with rdfname ‘file:///D/exp2/kit/RR_Onto1.owl#has_level’. The function contains an instance of class Kite as its argument. The result of the function is stored as an instance ‘generic_9’ of the variable class that has a property ‘variable_varname’

with value ‘evar2’. ‘evar2’ is of type string, which is defined in the kite domain ontology in OWL.

```
<cif:impliesconstr_if>
        <cif:comparison>
        <cif:comparison_operator>
         =
        </cif:comparison_operator>
     <cif:comparison_op2>
        <cif:stringconst>
        <cif:stringconst_value>
        beginner
        </cif:stringconst_value>
        </cif:stringconst>
        </cif:comparison_op2>
        <cif:comparison_op1>
        <cif:variable rdf:about="#generic_9"/>
        </cif:comparison_op1>
        </cif:comparison>
</cif:impliesconstr_if>
```

The above fragment defines the property ‘impliesconstr_if’ that represents the application condition of the constraint. The value of ‘impliesconstr_if’ contains an instance of the comparison class with property values of ‘=’, ‘beginner’ and ‘generic_9’. From the previous fragment, it can be inferred that ‘generic_9’ represents the value of the multi-valued function ‘has_level(k)’, where k is an instance of class ‘Kite’. Thus the above fragment represents the application condition: such that has_level(k) = "beginner".

```
<cif:impliesconstr>
        <cif:impliesconstr_qvar>
                <cif:setmem>
                <cif:setmem_set>
                <cif:mvfncall>
                <cif:mvfncall_prop>
                <cif:propmet>
                <cif:propmet_fname>
                has_cover
                </cif:propmet_fname>
                <cif:propmet_rdfname>
                file:///D/exp2/kit/RR_Onto1.owl#has_cover
                </cif:propmet_rdfname>
                <cif:propmet_firstargtype>
                <cif:entmet rdf:about="#generic_4"/>
                </cif:propmet_firstargtype>
```

```
                <cif:propmet_resulttype>
                <cif:entmet rdf:ID="generic_5">
                <cif:entmet_rdfname>
                file:///D/exp2/kit/RR_Onto1.owl#Cover
                </cif:entmet_rdfname>
                </cif:entmet>
                </cif:propmet_resulttype>
                </cif:propmet>
                </cif:mvfncall_prop>
                <cif:mvfncall_arg>
                <cif:variable rdf:about="#generic_2"/>
                </cif:mvfncall_arg>
                </cif:mvfncall>
                </cif:setmem_set>
                <cif:setmem_var>
                <cif:variable rdf:ID="generic_8">
                <cif:variable_varname>
                evar3
                </cif:variable_varname>
                </cif:variable>
                </cif:setmem_var>
                </cif:setmem>
        </cif:impliesconstr_qvar>
```

The above fragment defines a multi-valued function ‘has_cover’ with rdfname ‘file:///D/exp2/kit/RR_Onto1.owl#has_cover’ that contains an instance of class ‘Kite’ as its argument. The result of the function is stored as an instance ‘generic_8’ of the ‘variable’ class that has an object property ‘variable_varname’ with value ‘evar3’. The result type is an instance ‘generic_5’ of class ‘Cover’ with rdfname ‘file:///D/exp2/kit/RR_Onto1.owl#Cover’. All the RDF names provide references to the kite domain ontology in OWL

```
<cif:impliesconstr>
        <cif:impliesconstr_qvar>
                <cif:setmem>
                <cif:setmem_set>
                <cif:mvfncall>
                <cif:mvfncall_prop>
                <cif:propmet>
                <cif:propmet_fname>
                has_material
                </cif:propmet_fname>
                <cif:propmet_rdfname>
                file:///D/exp2/kit/RR_Onto1.owl#has_material
                </cif:propmet_rdfname>
                <cif:propmet_resulttype>
```

```
                <cif:entmet rdf:ID="generic_3">
                <cif:entmet_rdfname>
                file:///D/exp2/kit/RR_Onto1.owl#Kite_Material
                </cif:entmet_rdfname>
                </cif:entmet>
                </cif:propmet_resulttype>
                <cif:propmet_firstargtype>
                <cif:entmet rdf:about="#generic_5"/>
                </cif:propmet_firstargtype>
                </cif:propmet>
                </cif:mvfncall_prop>
                <cif:mvfncall_arg>
                <cif:variable rdf:about="#generic_8"/>
                </cif:mvfncall_arg>
                </cif:mvfncall>
                </cif:setmem_set>
                <cif:setmem_var>
                <cif:variable rdf:ID="generic_7">
                <cif:variable_varname>
                evar4
                </cif:variable_varname>
                </cif:variable>
                </cif:setmem_var>
                </cif:setmem>
        </cif:impliesconstr_qvar>
```

The above fragment defines a multi-valued function ‘has_material’ with rdfname ‘file:///D/exp2/kit/RR_Onto1.owl#has_material’ that contains an instance ‘generic_8’ of class ‘Cover’ (‘generic_5’) as its argument. The result of the function is stored as an instance ‘generic_7’ of the ‘variable’ class that has an object property ‘variable_varname’ with value ‘evar4’. The result type is an instance ‘generic_3’ of the class ‘Kite_Material’ with rdfname ‘file:///D/exp2/kit/RR_Onto1.owl#Kite_Material’.

```
<cif:impliesconstr>
        <cif:impliesconstr_qvar>
                <cif:setmem>
                <cif:setmem_set>
                <cif:mvfncall>
                <cif:mvfncall_prop>
                <cif:propmet>
                <cif:propmet_fname>
                density
                </cif:propmet_fname>
                <cif:propmet_resulttype>
                <cif:entmet>
```

```
            <cif:entmet_rdfname>
            file:///D/exp2/kit/RR_Onto1.owl#float
            </cif:entmet_rdfname>
            </cif:entmet>
            </cif:propmet_resulttype>
            <cif:propmet_rdfname>
            file:///D/exp2/kit/RR_Onto1.owl#density
            </cif:propmet_rdfname>
            <cif:propmet_firstargtype>
            <cif:entmet rdf:about="#generic_3"/>
            </cif:propmet_firstargtype>
            </cif:propmet>
            </cif:mvfncall_prop>
            <cif:mvfncall_arg>
            <cif:variable rdf:about="#generic_7"/>
            </cif:mvfncall_arg>
            </cif:mvfncall>
            </cif:setmem_set>
            <cif:setmem_var>
            <cif:variable rdf:ID="generic_6">
            <cif:variable_varname>
            evar5
            </cif:variable_varname>
            </cif:variable>
            </cif:setmem_var>
            </cif:setmem>
    </cif:impliesconstr_qvar>
```

The above fragment defines a multi-valued function ‘density’ with rdfname ‘file:///D/exp2/kit/RR_Onto1.owl#density’ that contains an instance (‘generic_7’) of class ‘Kite_Material’ (‘generic_3’) as its argument. The result of the function is stored as an instance ‘generic_6’ of the ‘variable’ class that has an object property ‘variable_varname’ with value ‘evar5’. The function has a result type of float.

```
 <cif:unquantified_constraint>
    <cif:unquantified_constraint_body>
            <cif:comparison>
            <cif:comparison_operator>
            &gt;
            </cif:comparison_operator>
            <cif:comparison_op2>
            <cif:floatconst>
            <cif:floatconst_value>
            0.5
            </cif:floatconst_value>
```

```
        </cif:floatconst>
        </cif:comparison_op2>
        <cif:comparison_op1>
        <cif:variable rdf:about="#generic_6"/>
        </cif:comparison_op1>
        </cif:comparison>
</cif:unquantified_constraint_body>
  </cif:unquantified_constraint>
```

The above fragment defines the class unquantified_constraint that represents the consequent of the constraint. The class has a property 'unquantified_constraint_body' that contains an instance of the 'comparison' class as its value. The instance of the 'comparison' class has property values of '>', '0.5' and 'generic_6'. From the previous fragment, it can be inferred that 'generic_6' represents the value of the multi-valued function 'density (has_material(has_cover(k)))', where 'k' is an instance of class 'Kite'. Thus the above fragment represents the consequent: to have density(has_material(has_cover(k))) > 0.5.

ConEditor+ retrieves the antecedent, application condition and consequent of each constraint by interpreting CIF as described above. Subsequently ConEditor+ applies the algorithm outlined in Section 6.4 to compare pairs of constraints, detect inconsistencies, subsumption, redundancy, fusion and suggest appropriate refinements between them to support maintenance.

6.6 Summary

This chapter describes the design, implementation and functionality of ConEditor+, together with a description of its framework consisting of the Designers' Workbench. ConEditor+ is an extended version of ConEditor. ConEditor was developed with the focus being mainly to facilitate domain experts in capturing constraints. The system provides basic maintenance facilities to check the constraints for syntax errors and allows reading constraints from text files, editing the constraints and then writing them to the same or new file. Following the implementation of ConEditor, the system has been extended to interpret constraints in CIF and provide further support to the maintenance of constraints. In addition, some changes were made to ConEditor's GUI. The extended system became known as ConEditor+. Further support to maintenance of constraints is provided in ConEditor+ by implementing the proposed

knowledge refinement rules described in Chapter 5. The knowledge refinement rules use constraints together with the associated application conditions and domain ontology to support maintenance. The algorithm used to determine the order in which ConEditor+ applies the refinement rules is outlined in this chapter. The chapter also describes the interpretation of CIF constraints by ConEditor+ with an example. The following chapter describes the evaluation of the research work reported in this thesis.

Chapter 7

Evaluation

> 'The true worth of an experimenter consists in his pursuing not only what he seeks in his experiment, but also what he did not seek.'
>
> **- Claude Bernard**

This chapter describes how the research work has been evaluated. The chapter is divided into three main sections. Section 7.1 describes a preliminary evaluation performed using ConEditor at Rolls-Royce, Derby, UK. The aim of this evaluation was to determine whether the design engineers would consider using a system such as ConEditor to capture and maintain design rules. Following the implementation of ConEditor+, three experiments were performed in the kite (design) domain. Experiment 1 was carried out to evaluate parts (IIb, IIc) of Research Question II, i.e., determine whether an explicit representation of application conditions together with the constraints and domain ontology can be used to: i) reduce the number of spurious inconsistencies and ii) prevent the identification of inappropriate refinements of redundancy, subsumption and fusion between pairs of constraints. Experiment 2 (Usability Studies) was carried out to evaluate parts (Ia, Ib, Ic) of Research Question I, i.e., examine whether ConEditor+ can facilitate (domain) experts in capturing and maintaining constraints in engineering design. Experiment 3 (Scalability Studies) was carried out to evaluate Research Question Id, i.e., determine whether the time taken by ConEditor+ to process constraints and detect inconsistencies/refinements on realistic tasks is viable for domain experts to use. Section 7.2 provides a description of each experiment and discusses the results obtained. After successful application of the proposed refinement rules to the kite domain, a part of the more complex Rolls-Royce domain was analysed, to determine whether the proposed system/approach could be used to capture and maintain constraints in a more complex KB (described in Section 7.3). An analysis of a part of the Rolls-Royce domain together with Sections 5.2 (kite

domain) and Section 5.3 (logical proofs) in Chapter 5 evaluate part (IIa) of Research Question II, i.e., an explicit representation of application conditions together with the corresponding constraints and the domain ontology can be used to detect inconsistencies, redundancy, subsumption and fusion between pairs of constraints. Further, an experiment was carried out using the KB with and without application conditions. Section 7.4 provides a summary of the chapter.

7.1 Preliminary Evaluation

This section describes a preliminary evaluation performed using ConEditor at Rolls-Royce, Derby. The main aim of this evaluation was to determine whether the design engineers at Rolls-Royce would consider using ConEditor to capture and maintain design rules as constraints. A demonstration was given by the experimenter (author) to a group of five design engineers at Rolls-Royce. The focus of the demonstration was one of the constraints from the design rule book. The demonstration involved the following three phases:

Phase 1: Presenting the constraint as in the rule book
The description of the constraint, as found in the rule book, is shown in Figure 7.1. After consulting the design engineers, it became clear that $\varnothing N_{min}$ denotes the trap diameter of the bolted joint containing internally trapped nuts, PCD denotes the pitch circle diameter and M denotes the gap in the flange. The English rendering of the constraint considered is:
Bolted joints must conform to the formula for internally trapped nuts:
$\varnothing N_{min}$ = PCD + 2*M + Maximum Nut Width
where $\varnothing N_{min}$ = trap diameter of the flange, PCD = pitch circle diameter of flange and 150.0<PCD<=180.0, M = gap in the flange = 0.5.

Phase 2: Expressing the constraint in CoLan
This constraint was expressed in CoLan by the author and discussed with the design engineers. In particular, the translation of the rule in English into a CoLan constraint comprising of antecedent, application condition and consequent was explained to the design engineers. The constraint in CoLan is as follows:

9.3.2 Internally trapped nuts (see Fig 4 Table 4)

TABLE 4

PCD ABOVE mm	TO mm	2M mm
150	180	1,00
180	300	0,80
300		0,60

ø N MIN. = PCD (NOM) + 2M + MAX. NUT WIDTH (SEE TABLE 5)

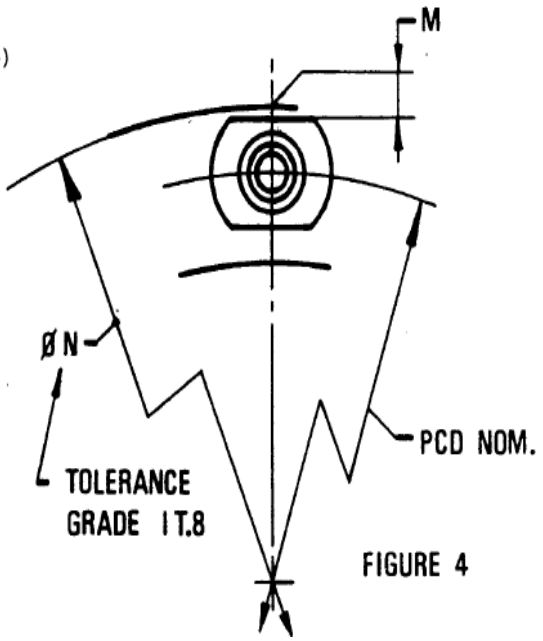


FIGURE 4

TABLE 5

THREAD SIZE	MAX. NUT WIDTH mm
0.1900-32	10,13
0.2500-28	11,56
0.3125-24	12,33
0.3750-24	15,37
0.4375-20	17,53
0.5000-20	19,05
0.5625-18	20,78
0.6250-18	23,02

Figure 7.1: Constraint as expressed in the design rule book

```
constrain each j in BoltedJoint
such that has_nut_type(j) = "Captive Nut"
and dimension(pitch_circle_diameter(has_flange(j))) >= 150.0
and dimension(pitch_circle_diameter(has_flange(j))) < 180.0
and  is_internal(has_flange(j))
to have gap(has_flange(j)) = 0.5
and dimension(trap_diameter(has_flange(j))) =
dimension(pitch_circle_diameter(has_flange(j)))
+ 2*gap(has_flange(j)) + dimension(nut_width(has_nut(j))) +
tolerance(nut_width(has_nut(j)))
```

Phase 3: Formulating the constraint using ConEditor

In the final phase of the demonstration, the CoLan expression was input to ConEditor by the experimenter, together with a running commentary of the steps taken while using ConEditor's GUI. As an example, the commentary included description such as: "I am selecting keywords "constrain" and "each" from the keywords panel using a single mouse click. On selecting the keywords, they appear in the result panel."

After the demonstration, the design engineers were interviewed by the experimenter. The aim of the interview was to get feedback from design engineers on their views about ConEditor's usability and whether they would consider using such a system for capturing and maintaining design rules. An overview of the results is presented below.

7.1.1 Overview of Results

The following is a summary of the feedback obtained from this interview with Rolls-Royce design engineers:

- The design engineers found the GUI simple, user friendly and intuitive.
- The design engineers were able to follow the steps where a constraint written in English was mapped to one expressed in CoLan; further they were able to understand how the CoLan expression was formulated using ConEditor. However, they felt they would need training to do either of these phases unsupported.

- Controlled Acquisition Scenario: The tool restricts the user's choice to a limited number of pre-defined keywords of the constraint language CoLan. Even though the constraint language is expressive and user-friendly, the engineers said they were not as comfortable using CoLan as with expressing the constraint directly in English.
- They also made the general point that in the company, they have a Design Standards group that has the responsibility for creating and maintaining the company-wide rule books, and so they would expect the standards group to formulate such constraints using ConEditor. The designers would subsequently use the information either in the current form or in a Designers' Workbench-like environment.

Overall, the evaluation results were encouraging and indicated that the domain experts would consider using the system to capture and maintain design rules. Following encouraging results from the evaluation of ConEditor, the system was extended with modifications to the GUI and addition of new features to provide additional support to the maintenance of constraints. The extended system became known as ConEditor+. An evaluation of parts (Ia, Ib, Ic) of Research Question I (usability studies) was carried out using the latest version of ConEditor, i.e. ConEditor+. Experiments performed using ConEditor+ are described in the next section.

7.2 Experiments using ConEditor+

The three experiments that were conducted using ConEditor+ in the kite design domain are described below:

Experiment 1: The aim of this experiment was to address the following parts of Research Question II:

- Can an explicit representation of application conditions together with the corresponding constraints and the domain ontology be used to:
  a) Reduce the number of spurious inconsistencies and,
  b) Prevent the identification of inappropriate refinements of redundancy, subsumption and fusion between pairs of constraints?

The kite domain was studied, and subsequently constraints were captured together with their application conditions. An experiment was run with ConEditor+ using: (I) KB_1 containing 15 constraints together with their application conditions, (II) KB_2 containing the same constraints without any application conditions.

Results: For KB_1, ConEditor+ detected 3 subsumptions, 0 inconsistencies, 3 redundancies and 2 cases of fusion between pairs of constraints. For KB_2, ConEditor+ detected 2 subsumptions, 5 inconsistencies, 3 redundancies and 4 cases of fusion between pairs of constraints. The investigator confirmed that the inconsistencies and some of the refinements (subsumption, redundancy, fusion) reported for KB_2 were spurious, and concluded that the absence of application conditions have caused these to be reported by ConEditor+. This is explained further below with examples. Consider two KBs, namely, KB_A and KB_B containing the following constraints:

KB_A (with application conditions):

(i) **constrain each** k **in** Kite
such that has_level(k) = “beginner”
to have density(has_material(has_cover(k))) < 0.5

(ii) **constrain each** k **in** Kite
such that has_level(k) = “advanced”
to have density(has_material(has_cover(k))) > 1.0

KB_B (without application conditions):

(iii) **constrain each** k **in** Kite
to have density(has_material(has_cover(k))) < 0.5

(iv) **constrain each** k **in** Kite
to have density(has_material(has_cover(k))) > 1.0

As shown above, the KB_A contains two constraints [(i) and (ii)] with the corresponding application conditions. The KB_B contains the same pair of constraints

[(iii) and (iv)] without the corresponding application conditions. For KB_A, ConEditor+ does not detect any inconsistency (or contradiction). For KB_B, ConEditor+ detects an inconsistency between the two constraints [(iii) and (iv)]. Hence, it can be concluded that the absence of application conditions can cause a number of spurious inconsistencies between constraints. In addition, this can cause ConEditor+ to suggest inappropriate refinements of redundancy, subsumption and fusion between pairs of constraints, as described below with examples.

a) Redundancy

Consider two KBs, namely, KB_C and KB_D containing the following constraints:

KB_C (with application conditions):

(v) **constrain each** j **in** JapaneseKite
such that has_wind_condition(j) = “strong”
to have has_bridle_point_distance(j) > 3 * surface_area(has_cover(j))

(vi) **constrain each** j **in** JapaneseKite
such that has_type(j) = “stunt”
to have has_bridle_point_distance(j) > 3 * surface_area(has_cover(j))

KB_D (without application conditions):

(vii) **constrain each** j **in** JapaneseKite
to have has_bridle_point_distance(j) > 3 * surface_area(has_cover(j))

(viii) **constrain each** j **in** JapaneseKite
to have has_bridle_point_distance(j) > 3 * surface_area(has_cover(j))

For KB_C, ConEditor+ suggests that the two constraints [(v) and (vi)] be fused and replaced by the constraint (ix):

(ix) **constrain each** j **in** JapaneseKite
such that has_wind_condition(j) = “strong” or has_type(j) = “stunt”
to have has_bridle_point_distance(j) > 3 * surface_area(has_cover(j))

For KB_D, ConEditor+ inappropriately suggests that either constraint (vii) or constraint (viii) be deleted because they are redundant.

b) Subsumption

Consider two KBs, namely, KB_E and KB_F containing the following constraints:

KB_E (with application conditions):

(x) **constrain each** s **in** SledKite
such that has_size(s) = “standard”
to have kite_line_strength(has_kite_line(s)) >= 15

(xi) **constrain each** s **in** ConventionalSledKite
such that has_size(s) = “small”
to have kite_line_strength(has_kite_line(s)) >= 15

KB_F (without application conditions):

(xii) **constrain each** s **in** SledKite
to have kite_line_strength(has_kite_line(s)) >= 15

(xiii) **constrain each** s **in** ConventionalSledKite
to have kite_line_strength(has_kite_line(s)) >= 15

ConventionalSledKite is a subclass of SledKite in the domain ontology. For KB_E, ConEditor+ does not suggest any refinements. For KB_F, ConEditor+ inappropriately suggests that constraint (xiii) be removed or deactivated because constraint (xii) subsumes constraint (xiii).

c) Fusion

Consider two KBs, namely, KB_G and KB_H containing the following constraints:

KB_G (with application conditions):

(xiv) **constrain each** d **in** Delta_kite
such that has_level(d) = “beginner”
to have bridle_length(has_bridle(d)) > 3 * has_height(d)

(xv) **constrain each** d **in** Delta_kite
such that has_wind_condition(d) = “strong”
to have kite_line_strength(has_kite_line(d)) > 90

KB_H (without application conditions):

(xvi) **constrain each** d **in** Delta_kite
to have bridle_length(has_bridle(d)) > 3 * has_height(d)

(xvii) **constrain each** d **in** Delta_kite
to have kite_line_strength(has_kite_line(d)) > 90

Again, two KBs have been considered: KB_G and KB_H, with and without application conditions respectively. For KB_G, ConEditor+ does not suggest any refinements. For KB_H, ConEditor+ inappropriately suggests that the two constraints [(xvi) and (xvii)] be fused and replaced by the constraint (xviii):

(xviii) **constrain each** d **in** Delta_kite
to have bridle_length(has_bridle(d)) > 3 * has_height(d) and
kite_line_strength(has_kite_line(d)) > 90

Thus, the results of experiment 1 demonstrate that an explicit representation of the application conditions together with the corresponding constraints and the domain ontology can be used to: (i) reduce the number of spurious inconsistencies and, (ii)

prevent the identification of inappropriate refinements of redundancy, subsumption and fusion between pairs of constraints.

Experiment 2: The aim of this experiment was to determine the usability of ConEditor+ and address the following parts of Research Question I:

- Examine whether it is possible to design and construct a system to facilitate (domain) experts in capturing and maintaining constraints in engineering design. In particular, the aim was to seek answers for the following main questions:

    a) Can the subjects successfully perform the allocated tasks within acceptable time limits?
    b) Did the subjects perform the tasks accurately? What kind of mistakes did the subjects make? Can the GUI be modified to eliminate or minimize these errors?
    c) How easy and intuitive did the subjects find the system to use?

The above research questions (Ia, Ib, Ic) are common in the field of usability testing and research (Rubin, 1994; Jordan, 1998; Dumas & Redish, 1999; Barnum, 2002). The evaluation method used was a combination of expert evaluation and survey evaluation methods. Expert evaluation is a diagnostic method lying between the theoretical approach taken in analytic evaluation and more empirical methods such as observational and experimental evaluation (Preece, 1993). In expert evaluation, 'experts' (usually people experienced in interface design or human factors research or both) assume the role of less experienced users and describe the potential problems they foresee arising for users of the system. This method has certain appeal because it is efficient and provides prescriptive feedback. In particular, a small number of experts can usually identify a whole range of potential problems for users during a single session with an interface. Survey evaluation is a method used to address users' subjective opinions following use of the system through either interviews or questionnaires. In a survey evaluation, potential or actual users (domain experts in the case of this experiment) take the role of subjects. Survey evaluation has certain appeal because the subjects are actual users (or potential users) and they can identify

problems that others (interface experts) cannot identify. The types of subjects recruited for this experiment are discussed below.

Recruiting subjects: For reasons relating to timeliness and cost, the recruiting method adopted was purposive/judgement sampling (Levy & Lemeshow, 1991; Kothari, 2005; Tongco, 2007). More discussion about purposive/judgement sampling can be found later in this section. Two computer science research (PhD) students who have had considerable experience in designing/developing user interfaces were recruited evaluation experts. Two post graduate engineering students (MEng and PhD) were recruited to constitute the subjects in the survey evaluation. A computer science research fellow who had neither prior experience in developing user interfaces nor any domain expertise was chosen as a neutral subject. The neutral subject was chosen such that he could identify problems, which neither domain experts nor user interface design experts identified. The experimental procedure carried out is described below.

Experimental procedure: A pilot experiment was conducted before the actual experiment using a computer science research student as the subject and that helped in detecting some elementary errors in the experiment's script and GUI. The pilot experiment also helped in estimating the average time taken for each task. The estimated time was used to allocate an acceptable time limit (benchmark) for each task. In the actual experiment, a demonstration was given by the experimenter (developer of ConEditor+) to each of the five subjects individually. The demonstration was given by following instructions from a script to maintain consistency and consisted of the following main tasks: description of the features of ConEditor+; a walkthrough of the process of converting a sample constraint in English to CoLan, inputting the CoLan constraint using ConEditor+, eliminating syntactic errors and performing appropriate refinements (redundancy, subsumption, contradiction, fusion). Each subject was then asked to perform the following tasks:

Task 1: The following constraint was presented in English and CoLan.
English: "Every standard sized or stunt type Sled Kite must have a kite line with strength greater than or equal to 15 units"

CoLan:

```
constrain each s in SledKite
such that has_size(c) = “standard” or has_type(s) = “stunt”
to have kite_line_strength(has_kite_line(c)) >= 15
```

The subject was asked to input the above constraint in CoLan using ConEditor+.

Task 2: ConEditor+’s KB already consisted of a constraint (shown below) that was subsumed by the constraint, the subject input in task 1. After successfully inputting the constraint in task 1, ConEditor+ detects subsumption and suggests the user considers deleting the following constraint:

```
constrain each c in ConventionalSledKite
such that has_size(c) = “standard”
to have kite_line_strength(has_kite_line(c)) >= 15
```

Each subject was asked to follow ConEditor+’s suggestion and delete the above constraint.

Task 3: Each subject was asked to answer a questionnaire and also provide oral feedback on the usability of ConEditor+ to the experimenter. The questionnaire contained various questions regarding the usability and usefulness of various features of ConEditor+. The methodology used to develop this questionnaire is discussed below. The subjects were asked to use a 5-point rating scale (1 being poor and 5 being excellent). The questionnaire used for this experiment is listed in Appendix B. The experimenter also observed all the actions performed by each subject and took notes.

Methodology of Developing Questionnaires: There is an extensive literature (Payne, 1951; Sudman & Bradburn, 1982; Rubin, 1994; Dumas & Redish, 1999; Bradburn, 2000; Barnum, 2002) available on how to develop effective questionnaires. Some of the principles that have been adopted from the literature to develop the questionnaire used in this experiment are discussed below. According to Dumas & Redish (1999), there are two reasons for having written questionnaires: (1) so that one asks every participant the same question and (2) so that one does not forget to ask questions. However, the main purpose of the written post-test questionnaire is to gather

preference information from the subjects in order to clarify and deepen the understanding of the product's strengths and weaknesses (Rubin, 1994). The questions created have to be unambiguous, unbiased, non-threatening and should prompt users to respond in a consistent way. The consistency of response is frequently verified by framing the same type of question in two different places or by asking for a certain kind of information in two different ways (Barnum, 2002). Questionnaires should be pilot-tested so that any problems can be addressed and corrected before using them in the actual test. Questions should be structured into formats where the respondents can provide ratings for their answers. A rating scale has to be used wherever possible instead of creating questions with just yes/no responses because yes/no responses force respondents into making a specific choice. The advantage of using a 5-point scale is that this allows respondents to choose the neutral central point. O'Muircheartaigh *et al*. (1999) concluded that offering a middle alternative in rating scales reduces the amount of random measurement error and does not affect validity. Some researchers find that the number of items on a scale is not as significant as the fact that there should be an odd number, thus providing a neutral point, which an even numbered scale (e.g., 4-point scale) would not provide. However, the error of central tendency, which is the tendency to avoid the extremes for the middle, comes into play with the larger scale. For instance, participants using a 7-point scale are less likely to choose either 1 or 7, avoiding the extremes of the scale (Barnum, 2002).

The advice offered by Sudman & Bradburn (1982) to those starting to write attitude questions is to plagiarize good-quality questions because most of the bugs will have been ironed out. For many years, following the classic book by Payne (1951), there has been no question in practitioner's minds that questionnaire design was still an art and not a science (Bradburn, 2000). "Although there is a lot of literature that is accumulating now to lay a foundation for a science of asking questions, it will always involve an element of art" (Schaeffer & Presser, 2003).

Results: All the subjects completed the allocated tasks accurately within the acceptable time limits (benchmarks). Tasks 1 and 2 were allocated a time limit (benchmark) of 5 and 3 minutes each respectively. The subjects were not aware of these predefined time limits (benchmarks). The errors committed by subjects can be summarized as follows: Two subjects double clicked on the keywords panel instead of single clicking. This resulted in the selected keyword being appended twice to the

constraint expression. The GUI has now been changed to support a double mouse click instead of a single click. Two subjects mentioned that they would like to see the console tab in the display panel activated automatically after inputting a constraint rather than having to do it manually. The GUI has been modified to support this feature. Two subjects also suggested that they would like a search facility in the taxonomy panel to help locate entities in a large taxonomy. This feature is planned to be implemented as part of the future work. All the subjects reported that they found ConEditor+ easy to use and helpful in both the capture and maintenance of constraints. Appendix D lists the scanned versions of the questionnaires that were answered by the subjects during this evaluation. The average overall rating given by the subjects, for the usability (including capture and maintenance facilities) of ConEditor+ was 3.8 on a 5-point rating scale (1 being poor and 5 being excellent) (Figure 7.2). On the basis of purposive/judgemental sampling the results of experiment 2 indicate that ConEditor+ is easy to use and facilitates domain experts in capturing and maintaining constraints in engineering design.

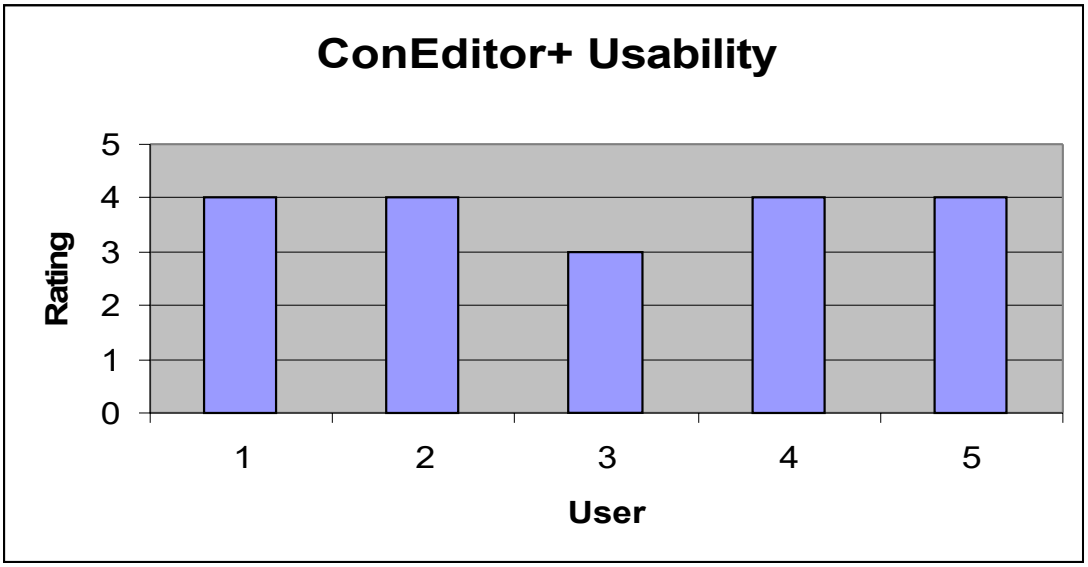


Figure 7.2: Graph showing results of an experiment to evaluate usability of ConEditor+ (Rating scale: 1-poor and 5-excellent)

Discussion (Critical Analysis): It is important to critically discuss the adopted method. The method (sampling or sample survey) used here may be defined as a study

involving a subset (or sample) of individuals selected from a larger population. Variables or characteristics of interest are observed or measured on each of the sampled individuals. These measurements are then aggregated over all individuals in the sample to obtain summary statistics for the sample. It is from these summary statistics that extrapolations can be made concerning the entire population. Sampling is adopted when it is infeasible to conduct a census due to various reasons including timeliness, cost, limited access or inaccessibility of some of the population. In addition, within sampling, it is often not feasible to sample the elementary units directly. This is because lists of elementary units from which the sample can be taken are often not readily available, and can be constructed only at considerable cost (Levy & Lemeshow, 1991). The method adopted in the above experiment is a type of non- probability sampling that is called purposive or judgemental sampling. In this type of sampling, individuals are selected (by the researcher) who are considered to be most representative of the population as a whole. "Non-probability samples are used quite frequently because probability sampling is often a time-consuming and expensive procedure, and in fact, may not be feasible in many situations" (Levy & Lemeshow, 1991). "Purpose sampling is a practical and efficient tool when used properly, and can be just as effective as, and even more efficient than random sampling" (Tongco, 2007). The advantages and disadvantages of non-probability based purposive or judgement sampling, used in the above experiment are as follows:

Advantages:

- Time and Costs are relatively lower when compared to other methods.
- Non-probability based approaches to sampling can be used when the objective is to conduct an exploratory or descriptive study on an issue or process that has not been studied in detail. In the above experiment, the objective was to explore whether the proposed approach/system (not studied/developed earlier) is feasible for use by domain experts. "Judgement or Purposive sampling is useful in qualitative studies when a researchable hypothesis needs to be explored, where other sampling techniques prove difficult to apply" (Okolo, 1990).
- "Provides a dynamic picture of the data and serves as the basis for process improvement" (Lloyd, 2004). In the above experiment, the investigator was able to diagnose problems/shortcomings and thus improve the usability of

ConEditor+. In particular, the feedback given by the subjects has been useful in exploring what kind of problems/shortcomings they could experience when using such a system.

- Non-probability based purposive sampling can be more efficient than probabilistic random sampling. Missing data can render random samples invalid for traditional probabilistic inference (Godambe, 1982). This can occur because not everybody is willing to participate, and possibly not be around during sampling. In addition, some respondents may be disinterested and hence not answer all items in questionnaires and provide proper feedback. These problems can be avoided in purposive sampling because it is the researcher who selects the samples after doing a background study.

Disadvantages:

- "The disadvantage of judgemental sampling is that no insight can be obtained mathematically concerning the reliability of the resulting estimates"(Levy & Lemeshow, 1991). Sampling error cannot be calculated. Sampling error comprises the differences between the sample and the population that are solely due to the particular units that happen to have been selected.
- Unlike random sampling, non-probability methods such as purposive sampling are not free from bias. Sampling bias is a tendency to favour the selection of units that have particular characteristics. However, it is argued that the inherent bias of the method contributes to its efficiency, and the method stays robust even when tested against random probability sampling (Tongco, 2007). A sample is expected to mirror the population from which it comes, however there is no guarantee that any sample will be precisely representative of the population from which it comes (due to potential subjectivity of researcher).

Another experiment performed to determine the time taken by ConEditor+ to process constraints is described below.

Experiment 3: The aim of this experiment was to determine the time taken by ConEditor+ to process constraints (including application conditions) and detect syntax errors, inconsistencies, redundancy, subsumption and fusion. These results were then used to address the following part of Research Question I:

- Is the speed of the system on realistic tasks viable for (domain) experts to use?

The constraints and application conditions acquired from the kite domain were used for this experiment. Four KBs containing 30, 60, 90 and 120 constraints, together with their application conditions were used in the experiment. The KBs of larger size were constructed by repeatedly using the same set of constraints, i.e., the same set of constraints was duplicated to make it a larger KB. For each KB, twelve tasks were performed. All the twelve tasks performed are listed and described below. The time taken by ConEditor+ in each task (tasks 2 to 11) that involves only one comparison between a pair of constraints is referred to as the 'best case' time. The time taken by ConEditor+ in each task that involves 'n' comparisons between pairs of constraints (where n = 30, 60, 90, 120 for the four KBs) is referred to as the 'worst case' time. The time taken by ConEditor+ in each task is shown in Table 7.1. The best-case time is indicated by (B) and worst-case time by (W) respectively in the table. Figure 7.3 shows a graph comparing the average refinement time taken by ConEditor+ versus number of constraints in KB.

Task 1: A constraint (including application condition) that would not cause any error was submitted. The time taken by ConEditor+ to process the constraints and check for any syntactic errors, inconsistencies and refinements (subsumption, redundancy and fusion) was recorded. The time taken by ConEditor+ was calculated programmatically by subtracting the time taken at the start of program and subtracting it from the time taken at the end of the program. The Java statement "System.currentTimeMillis()" was used in appropriate places in the code to get the time in milliseconds. The submitted constraint was as follows:

```
constrain each d in Delta_kite
such that has_level(d) = "beginner"
to have bridle_length(has_bridle(d)) > 3 * has_height(d)
```

Task 2: A constraint (including application condition) that caused an inconsistency when compared with the first constraint in the KB (i.e. ver_1_CoLanKiteList.txt_1) was submitted. The time taken by ConEditor+ to process the constraints (involving one comparison between the pair of constraints) and report the error was recorded

programmatically. The first constraint in the KB (i.e. ver_1_CoLanKiteList.txt_1) was as follows:

```
constrain each c in ConventionalSledKite
such that has_level(c) = "beginner"
to have kite_line_strength(has_kite_line(c)) >= 15
```

The submitted constraint was as follows:

```
constrain each c in ConventionalSledKite
such that has_level(c) = "beginner”
to have kite_line_strength(has_kite_line(c)) < 15
```

Task 3: A constraint (including application condition) that caused a redundancy (by duplication) when compared with the first constraint in the KB (i.e. ver_1_CoLanKiteList.txt_1) was submitted. The time taken by ConEditor+ to process the constraints (involving one comparison between the pair of constraints) and suggest refinement was recorded programmatically. The submitted constraint was as follows:

```
constrain each c in ConventionalSledKite
such that has_level(c) = "beginner"
to have kite_line_strength(has_kite_line(c)) >= 15
```

Task 4: A constraint (including application condition) that caused a redundancy (by class equivalence) when compared with the first constraint in the KB (i.e. ver_1_CoLanKiteList.txt_1) was submitted. The time taken by ConEditor+ to process the constraints (involving one comparison between the pair of constraints) and suggest refinement was recorded programmatically. The submitted constraint was as follows:

```
constrain each t in TraditionalSledKite
such that has_level(t) = "beginner""
to have kite_line_strength(has_kite_line(t)) >= 15
```

And ‘ConventionalSledKite’ is an equivalent class to ‘TraditionalSledKite’ in the domain ontology.

Task 5: A constraint (including application condition) that caused a redundancy (by property equivalence) when compared with the first constraint in the KB (i.e. ver_1_CoLanKiteList.txt_1) was submitted. The time taken by ConEditor+ to process the constraints (involving one comparison between the pair of constraints) and suggest refinement was recorded programmatically. The submitted constraint was as follows:

```
constrain each c in ConventionalSledKite
such that has_class(c) = "beginner""
to have kite_line_strength(has_kite_line(c)) >= 15
```

And ‘has_level’ is an equivalent class to ‘has_class’ in the domain ontology.

Task 6: A constraint (including application condition) that caused a subsumption (via subclass) when compared with the first constraint in the KB (i.e. ver_1_CoLanKiteList.txt_1) was submitted. The time taken by ConEditor+ to process the constraints (involving one comparison between the pair of constraints) and suggest refinement was recorded programmatically. The submitted constraint was as follows:

```
constrain each s in SledKite
such that has_level(s) = "beginner""
to have kite_line_strength(has_kite_line(s)) >= 15
```

And ‘ConventionalSledKite’ is a subclass of ‘SledKite’ in the domain ontology.

Task 7: A constraint (including application condition) that caused a subsumption (via application condition) when compared with the first constraint in the KB (i.e. ver_1_CoLanKiteList.txt_1) was submitted. The time taken by ConEditor+ to process the constraints (involving one comparison between the pair of constraints) and suggest refinement was recorded programmatically. The submitted constraint was as follows:

```
constrain each c in ConventionalSledKite
such that has_level(c) = "beginner" or has_size(c) = "standard"
to have kite_line_strength(has_kite_line(c)) >= 15
```

Task 8: A constraint (including application condition) that caused a subsumption (via conjunction) when compared with the first constraint in the KB (i.e. ver_1_CoLanKiteList.txt_1) was submitted. The time taken by ConEditor+ to process

the constraints (involving one comparison between the pair of constraints) and suggest refinement was recorded programmatically. The submitted constraint was as follows:

```
constrain each c in ConventionalSledKite
such that has_level(c) = "beginner"
to have kite_line_strength(has_kite_line(c)) >= 15
and bridle_length(has_bridle(c)) > 25
```

Task 9: A constraint (including application condition) that caused a fusion (via class) when compared with the first constraint in the KB (i.e. ver_1_CoLanKiteList.txt_1) was submitted. The time taken by ConEditor+ to process the constraints (involving one comparison between the pair of constraints) and suggest refinement was recorded programmatically. The submitted constraint was as follows:

```
constrain each m in ModernSledKite
such that has_level(m) = "beginner""
to have kite_line_strength(has_kite_line(m)) >= 15
```

'ConventionalSledKite' and 'ModernSledKite' are the only two subclasses of 'SledKite' in the domain ontology.

Task 10: A constraint (including application condition) that caused a fusion (via application condition) when compared with the first constraint in the KB (i.e. ver_1_CoLanKiteList.txt_1) was submitted. The time taken by ConEditor+ to process the constraints and suggest refinement was recorded programmatically. The submitted constraint was as follows:

```
constrain each c in ConventionalSledKite
such that has_size(c) = "standard"
to have kite_line_strength(has_kite_line(c)) >= 15
```

Task No.	Type of Error/ Refinement	Time in milliseconds (B- best case; W- worst case)			
		Number of constraints in ConEditor+'s KB			
		30	60	90	120
1	No Error	41734	76125	112891	144704
2	Inconsistency	375 (B) 37543 (W)	418 (B) 72735 (W)	438 (B) 108578 (W)	456 (B) 140078 (W)
3	Redundancy (via duplication)	437 (B) 38110(W)	453 (B) 73328 (W)	469 (B) 109922 (W)	489 (B) 141015 (W)
4	Redundancy (via class equivalence)	469 (B) 38125 (W)	546 (B) 73188	612 (B) 109156 (W)	683 (B) 141109 (W)
5	Redundancy (via property equivalence)	475 (B) 37250 (W)	547 (B) 72672 (W)	618 (B) 108157 (W)	687 (B) 140750 (W)
6	Subsumption (via subclass)	687 (B) 38282 (W)	750 (B) 73297 (W)	814 (B) 109312 (W)	874 (B) 141203 (W)
7	Subsumption (via application condition)	546 (B) 32640 (W)	547 (B) 67985 (W)	578 (B) 103031 (W)	638 (B) 135422 (W)
8	Subsumption (via conjunction)	609 (B) 34484 (W)	703 (B) 69953 (W)	786 (B) 105735 (W)	847 (B) 137219 (W)
9	Fusion (via class)	2656 (B) 39609 (W)	2891 (B) 74704 (W)	3134 (B) 110859 (W)	3380 (B) 142219 (W)
10	Fusion (via application condition)	875 (B) 37609 (W)	906 (B) 72610 (W)	943 (B) 108579 (W)	980 (B) 140157 (W)
11	Fusion (via conjunction)	2453 (B) 38875 (W)	2593 (B) 73110 (W)	2729 (B) 109653 (W)	2863 (B) 141406 (W)

Table 7.1: Time Taken by ConEditor+ to detect inconsistencies and refinements for various KB sizes

Task 11: A constraint (including application condition) that caused a fusion (via conjunction) when compared with the first constraint in the KB (i.e. ver_1_CoLanKiteList.txt_1) was submitted. The time taken by ConEditor+ to process the constraints (involving one comparison between the pair of constraints) and suggest refinement was recorded programmatically. The submitted constraint was as follows:

```
constrain each c in ConventionalSledKite
such that has_level(c) = "beginner"
to have bridle_length(has_bridle(c)) > 25
```

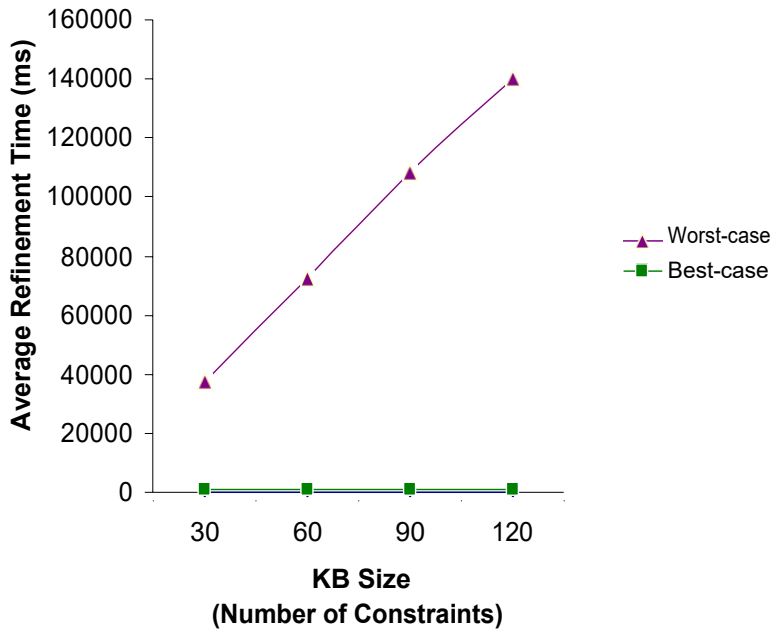


Figure 7.3: Graph showing average refinement time taken by ConEditor+ versus number of constraints in KB

Task 12: The first constraint in the KB was swapped with the last constraint in the KB. Tasks 2 to 11 were repeated by submitting a constraint (including application condition) that caused an inconsistency/refinement when compared with the last constraint in the KB (i.e. ver_1_CoLanKiteList.txt_n where n = 30, 60, 90, 120 for the four KBs respectively). It has to be noted here that the constraints in the KB were chosen so that an inconsistency/refinement is detected only when comparing the

submitted constraint against the last constraint in the KB (i.e. comparisons with other constraints in the KB result in no syntax errors, inconsistencies, redundancy, subsumption or fusion). The time taken by ConEditor+ to process the constraints (involving 'n' comparisons between pairs of constraints where n =30, 60, 90, 120 for the four KBs) and detect inconsistency/refinement in each task was recorded programmatically.

The experiment was run on a computer with the following configuration: AMD Athlon 64-bit processor, clock frequency of 2.21 GHz, 960 MB of RAM, operating system: Windows XP, JDK (Java Development Kit) 1.4.2 and Jena 2.1. The time taken by ConEditor+ to report a syntax error in the submitted constraint was recorded programmatically and it was equal to 500 milliseconds. Also, the time taken to submit a constraint to a KB with no constraints in it was recorded programmatically and it was equal to 484 milliseconds.

Results: It can be observed from Table 7.1 and Figure 7.3 that the average worst-case time taken by ConEditor+ for refinements essentially increases linearly as the KB size increases while the average best-case time taken is almost a constant. ConEditor+ uses Jena to parse the domain ontology, constraints and application conditions in CIF. Currently a file system (text files) is used to store the constraints. The increase in average worst-case refinement time might become non-linear for larger KBs that involve manipulation of information that cannot all be held in main memory. Semantic web technologies such as Jena face scalability issues, and work is being carried out by semantic web researchers to tackle them. For large KBs containing thousands of constraints, the author plans to use 3-store (Harris & Gibbins, 2003) which is a RDF bulk storage and query engine developed to enable the efficient handling of large RDF KBs. Moreover, although the total number of design constraints formulated by Rolls-Royce is in the order of thousands, it is expected that only a small subset (say in the order of hundreds) will be needed for any particular design. For a small subset, the above results suggest that speed should not be an issue. The following section describes the application of ConEditor+ to capture and maintain constraints in a more complex KB.

7.3 Extension/Evaluation of Jet Engine Ontology and Maintenance of a more complex set of Constraints

After successful application and evaluation of ConEditor+ in the domain of kite design, a part of the considerably more demanding Rolls-Royce domain was analysed. The aim of this analysis was to demonstrate that the proposed system/approach could be used in capturing and maintaining constraints in a more complex and extensive KB containing real-world constraints.

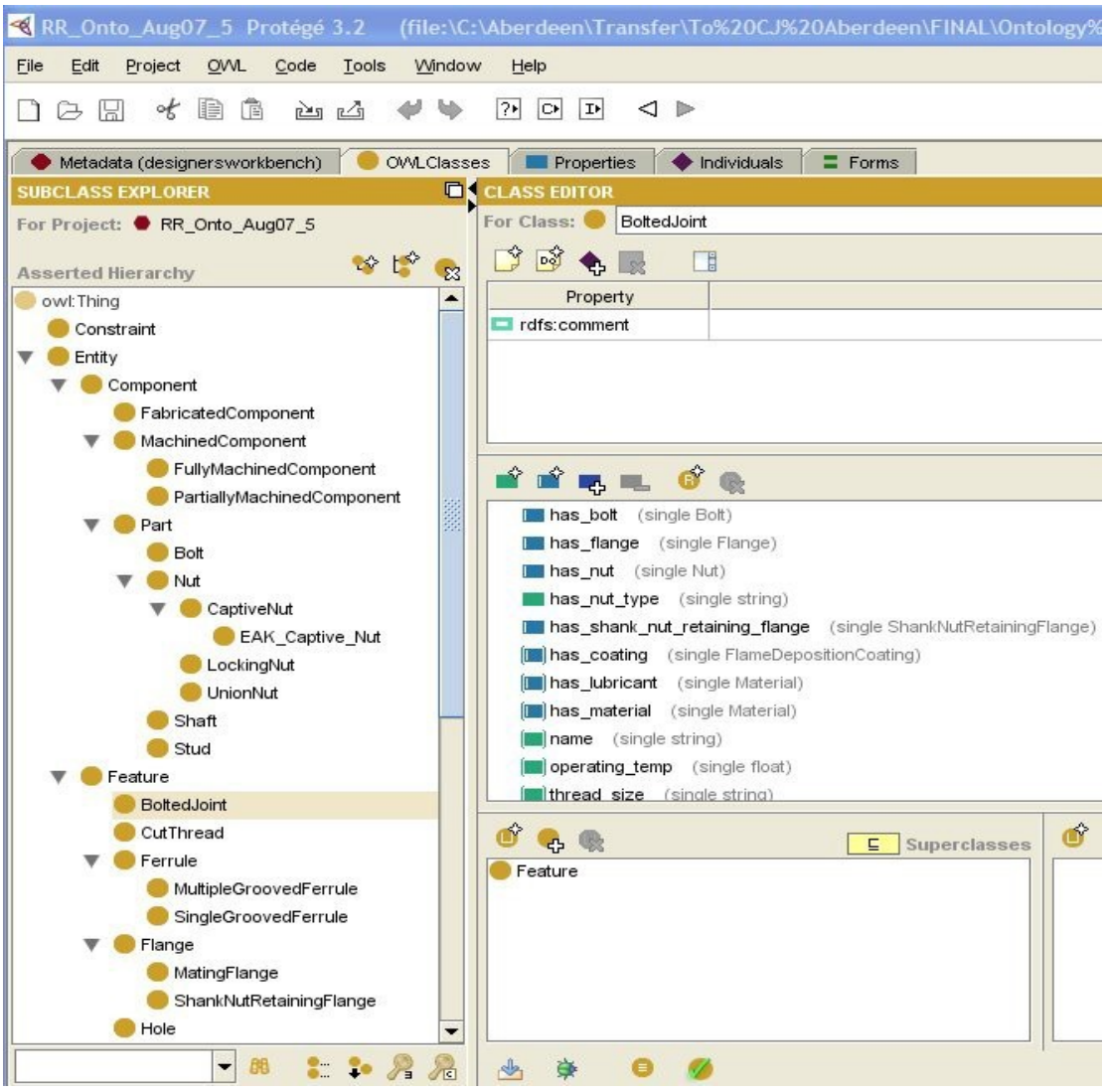


Figure 7.4: Extended/Evaluated Jet Engine Ontology of part of the Rolls-Royce domain in Protégé

The analysis aims to address the following part of Research Question II:

- Can an explicit representation of application conditions together with the corresponding constraints and the domain ontology be used to detect inconsistencies, redundancy, subsumption and fusion between pairs of constraints?

An analysis of a considerable number of Rolls-Royce design standard documents (72), which contain rules/standards for the design of various parts and processes involved in civil aero-engines was carried out. Interviews were held with a design engineer at Rolls-Royce, Derby to clarify the succinctly described rules; additionally, the engineer was asked to describe the conditions under which they thought the rule (constraint) was applicable. The investigator then formulated this as an application condition for the constraint. The ontology used to support Designers' Workbench did not contain all the concepts and properties that were needed to express the design rules obtained from this analysis. As a result, the jet engine ontology was extended (e.g., additional classes and properties) to incorporate the additional information obtained from this analysis. The jet engine ontology was then evaluated by a further independent assessor in Rolls-Royce. Following several discussions with the assessor and modifications to the ontology, the ontology was approved by the assessor. Figure 7.4 shows a screenshot of the extended jet engine ontology developed using Protégé editor (Noy *et al.*, 2000).

A confidential technical report (Ajit *et al*., 2008b) describes the list of all the constraints and application conditions obtained from this analysis, together with their corresponding representations in CoLan. Appendix E describes sample refinements from the Rolls-Royce domain and demonstrates how the proposed approach can be applied to support the maintenance of this more demanding KB, containing a series of real-world constraints. Section 5.2 (kite domain), Section 5.3 (logical proofs) of Chapter 5 and the sample refinements in Appendix E demonstrate that an explicit representation of application conditions together with the corresponding constraints and the domain ontology can be used to detect inconsistencies, redundancy, subsumption and fusion between pairs of constraints. Further, an experiment was carried out using a selected list of constraints and application conditions from the Rolls-Royce domain. The details of the experiment are given below:

Experiment 4: The aim of this experiment was to further demonstrate in a more complex domain than the kite domain that an explicit representation of application conditions together with the corresponding constraints and the domain ontology can be used to: i) reduce the number of spurious inconsistencies and ii) prevent the identification of inappropriate refinements (for example, fusion) between pairs of constraints.

The experiment was run with ConEditor+ using: (I) KB_3 containing 63 constraints together with their application conditions, (II) KB_4 containing the same constraints without any application conditions.

Results: For KB_3, ConEditor+ detected 0 subsumptions, 0 inconsistencies, 0 redundancies and 8 cases of fusion between pairs of constraints. For KB_4, ConEditor+ detected 0 subsumptions, 54 inconsistencies, 0 redundancies and 128 cases of fusion between pairs of constraints. The investigator confirmed that the 54 inconsistencies and 120 (out of 128) cases of fusion for KB_4 were spurious, and concluded that the absence of application conditions have caused these to be reported by ConEditor+. The list of constraints and application conditions used for this experiment are part of a confidential technical report (Ajit *et al*., 2008b). The spurious inconsistencies and inappropriate refinement (fusion) reported by ConEditor+ can be demonstrated using the following examples:

(xxiv)
```
constrain each f in Forging
such that not type(has_forging_material(f)) = "light alloy"
to have has_external_draw_angle(f) >= 5
and has_internal_draw_angle(f) >= 7
```

(xxv)
```
constrain each f in Forging
such that type(has_forging_material(f)) = "light alloy"
to have has_external_draw_angle(f) >= 3
and has_internal_draw_angle(f) >= 5
```

For constraints (xxiv) and (xxv), the absence of application conditions would cause ConEditor+ to report a spurious inconsistency.

(xxvi) **constrain each** s in FaceRingSeal
such that has_ring(s) is a ElastometricToroidalORing
and not name(has_material(has_ring(s))) = "perfluorocarbon"
and pressure_type_hou_mat_flange(s) = "internal"
to have min_face_groove_dia(s) =
max_face_groove_dia(s) - 0.25

(xxvii) **constrain each** s in FaceRingSeal
such that has_ring(s) is a ElastometricToroidalORing
and not name(has_material(has_ring(s))) = "perfluorocarbon"
and pressure_type_hou_mat_flange(s) = "external"
to have min_face_groove_dia(s) =
mean_inside_diameter(has_ring(s))
- (tolerance(face_groove_dia(s))/2)

For constraints (xxvi) and (xxvii), the absence of application conditions would cause ConEditor+ to suggest inappropriately that the constraints (xxvi) and (xxvii) be fused. Hence, one can infer that an explicit representation of application conditions together with the corresponding constraints and the domain ontology can be used to: (i) reduce the number of spurious inconsistencies, and (ii) prevent identification of inappropriate refinements (fusion in this case).

7.4 Summary

This chapter describes the evaluations performed during the research work. A preliminary evaluation of ConEditor was conducted at Rolls-Royce, Derby. The design engineers at Rolls-Royce were given a demonstration of ConEditor and asked to provide feedback on the system. The design engineers found the GUI simple, user friendly and intuitive. They were able to understand the various phases in the demonstration but they felt that they would need training to do them unsupported. ConEditor was then extended with modifications to the GUI and addition of new features, to provide support for the maintenance of constraints. The extended system became known as ConEditor+. Three experiments were carried out using ConEditor+ in the kite design domain. Experiment 1 involved applying ConEditor+ to a kite

domain KB with application conditions and to the same kite domain KB without application conditions. For the KB with application conditions, ConEditor+ did not detect any inconsistencies but suggested appropriate refinements. For the KB without any application conditions, ConEditor+ detected a number of spurious inconsistencies and also suggested inappropriate refinements of redundancy, subsumption and fusion. Hence, one can conclude that an explicit representation of application conditions together with the corresponding constraints and the domain ontology can be used to i) reduce the number of spurious inconsistencies and ii) prevent identification of inappropriate refinements of redundancy, subsumption and fusion.

Experiment 2 was performed using five subjects to determine the usability of ConEditor+ and examine whether it can facilitate (domain) experts in capturing and maintaining constraints in engineering design. The evaluation method was a combination of expert evaluation and survey evaluation methods. The subjects were recruited using purposive/judgement sampling. After providing an initial demonstration, the subjects were asked to capture and refine a constraint using ConEditor+. The subjects were then asked to answer a questionnaire and provide feedback on the usability of ConEditor+. The average overall rating given by the subjects for the usability of ConEditor+ was 3.8 on a 5-point scale. The subjects also suggested some modifications to the GUI. Based on purposive/judgemental sampling, the results indicate that ConEditor+ can facilitate domain experts in capturing and maintaining constraints. The advantages and disadvanatages of this type of evaluation have been discussed. The advantages include relatively lower time and costs, more efficiency, usefulness to conduct an exploratory study and diagnose problems/shortcomings. However, the disadvantages include no mathematical insight concerning reliability of resulting estimates, possibility of sampling bias and inability to precisely represent the population. (i.e actual users of ConEditor+) due to potential subjectivity of researcher.

Experiment 3 was performed to determine the time taken by ConEditor+ to process constraints (including application conditions) and detect syntax errors, inconsistencies, redundancy, subsumption and fusion. Four KBs containing 30, 60, 90 and 120 constraints from the kite domain together with application conditions were used. The best and worst time taken by ConEditor+ to detect inconsistency and refinements in each KB were recorded programmatically. The results of Experiment 3

have shown that the time taken by ConEditor+ to process constraints, detect inconsistencies, and suggest refinements should not be an issue for realistic tasks, considering that only a small subset (say in the order of hundreds) will be needed for any particular design.

In addition, an evaluation was performed by analysing a part of the Rolls-Royce domain and applying ConEditor+ to this more demanding KB. Design rules were elicited from 72 Rolls-Royce design standard documents. The domain ontology was extended to incorporate the additional information obtained from these analyses. The ontology was evaluated by an assessor at Rolls-Royce. Sample refinements of this KB (provided in Appendix E) indicate that the proposed approach/system is viable in the case of a more complex KB. The refinements discussed in Section 5.2 (kite domain), Section 5.3 (logical proofs) of Chapter 5 and sample refinements provided in Appendix E demonstrate that an explicit representation of application conditions together with the corresponding constraints and the domain ontology can be used to detect inconsistencies, redundancy, subsumption and fusion between pairs of constraints. Further, an experiment was carried out in the Rolls-Royce domain by applying ConEditor+ to KBs with and without application consitions. The results of this experiment were similar to the results of Experiment 1 (kite domain) and demonstrated that an explicit representation of application conditions together with the corresponding constraints and the domain ontology can be used to: (i) reduce the number of spurious inconsistencies, and (ii) prevent identification of inappropriate refinements (fusion).

Chapter 8

Conclusions and Future Work

> 'There is nothing like a dream to create the future.'
>
> **- Victor Hugo**

The thesis has focused on knowledge management with engineering design as an application domain. Within engineering design, the thesis aims to tackle issues/problems in the capture and maintenance of constraints. This final chapter highlights the main research contributions of the thesis, discusses some limitations of the work and provides some possible directions for future work. The chapter is structured as follows: Section 8.1 highlights the research contributions of the work reported in this thesis; Section 8.2 discusses some limitations of the work; Section 8.3 discusses some possible directions for future work.

8.1 Research Contributions

The thesis identifies a situation where it is highly desirable to eliminate the knowledge engineer from the tedious, error-prone and time-consuming task of capturing and maintaining constraints for systems such as the Designers' Workbench. In order to relieve the knowledge engineer from the above task, the thesis proposes a novel approach to facilitate domain experts in capturing and maintaining constraints. The thesis further reports the design and construction of a system that embodies the proposed approach. The feedback obtained from a preliminary evaluation performed at Rolls-Royce was encouraging and indicated that the domain experts would consider using such a system for capturing and maintaining constraints in engineering design.

The thesis identifies potential problems faced during maintenance of constraints. In order to reduce/overcome the maintenance problems, the thesis reports that it is important to capture the underlying assumptions and context in which each

constraint is applicable. These assumptions and contexts are referred to as application conditions. The thesis proposes an approach to capture and use the application conditions in a machine-interpretable format together with the corresponding constraints and the domain ontology to support the maintenance of constraints. The thesis hypothesises that an explicit representation of application conditions together with the corresponding constraints and the domain ontology can be used to: a) detect inconsistencies, subsumption, redundancy and fusion, b) reduce the number of spurious inconsistencies, and c) prevent the identification of inappropriate refinements of subsumption, redundancy and fusion between pairs of constraints. The thesis proposes four types of refinement rules to detect inconsistencies, subsumption, redundancy and fusion between pairs of constraints using the associated application conditions and domain ontology.

The thesis extends the system (ConEditor) with an ability to implement the refinement rules and support the maintenance of constraints. The thesis reports on experiments, usability and scalability studies that apply the extended system (ConEditor+) to support the capture and maintenance of constraints from a kite design KB. The usability studies demonstrate that ConEditor+ can facilitate domain experts in capturing and maintaining constraints in engineering design. The scalability studies demonstrate that the speed of ConEditor+ on realistic tasks is viable for domain experts to use. Further, the thesis investigates part of the more complex Rolls-Royce domain and demonstrates that the proposed approach/system can be used to support the capture and maintenance of a more complex KB consisting of real world design constraints. The logical proofs of refinement rules together with the results of experiments in the kite domain and part of the Rolls-Royce domain demonstrate that an explicit representation (machine-interpretable format) of application conditions together with the corresponding constraints and the domain ontology can be used in: i) detecting inconsistencies, subsumption, redundancy and fusion, ii) reducing the number of spurious inconsistencies, and iii) preventing the identification of inappropriate refinements of subsumption, redundancy and fusion between pairs of constraints.

Finally, the overall research work reported in this thesis demonstrates the use of ontologies/semantic web technologies for knowledge management in an organisation. Inferencing using ontologies was done to detect subsumption, redundancy, inconsistency and fusion between pairs of constraints. The key point emphasised by the

thesis is that by adding some additional information at the knowledge acquisition stage, one can greatly enhance the maintainability of a KB.

In summary, the main research contributions[8] can be listed as:

- Proposed a novel approach, designed and constructed system(s) to facilitate domain experts in capturing and maintaining constraints in engineering design. [Chapter 3 and ConEditor/ConEditor+ in Chapters 4 and 6].

- Demonstrated the effectiveness of the above system to facilitate (domain) experts in capturing and maintaining constraints in engineering design. [Chapter 7, Experiment 2].

- Demonstrated that the speed of such a system on realistic tasks is viable for domain experts to use. [Chapter 7, Experiment 3].

- Analysed engineering design domains (kite domain and a part of Rolls-Royce domain) and demonstrated various types of contexts and underlying assumptions associated with constraints in these domains with the help of examples. These contexts and underlying assumptions are referred to as "application conditions". [Chapter 5, Section 5.2 and Chapter 7, Section 7.3].

- Proposed four main types of refinement rules to detect inconsistencies, redundancy, subsumption and fusion between pairs of constraints using the associated application conditions and domain ontology. Proved that the refinement rules are logically sound. [Chapter 5, Section 5.3].

- Demonstrated that an explicit representation of application conditions together with the corresponding constraints and the domain ontology can be used to:

  i) Detect inconsistencies, redundancy, subsumption and fusion between pairs of constraints, [Chapter 5, Sections 5.2 and 5.3; Chapter 7, Section 7.3].

  ii) Reduce the number of spurious inconsistencies, and

[8] Pointers in square brackets tie the contributions back to the original source in the thesis.

iii) Prevent identification of inappropriate refinements of redundancy, subsumption and fusion between pairs of constraints. [(ii) and (iii) in Chapter 7, Experiment 1].

- Demonstrated that the proposed approach/system can be used to support the capture and maintenance of a more complex KB, consisting of real world design constraints. [Chapter 7, Section 7.3].

- Demonstrated the use of ontologies/semantic web technologies for knowledge management in an organisation. [Use of OWL, Protégé, Jena, RDQL, CIF in Chapters 4 and 6; Inferencing done using ontology to detect subsumption, redundancy, inconsistency and fusion between pairs of constraints (Chapter 5, Sections 5.2 and 5.3)].

The following section describes some possible limitations of the research work reported in this thesis.

8.2 Limitations

ConEditor+ has been implemented to detect subsumption, inconsistency (contradiction), redundancy, fusion and suggest appropriate refinements between pairs of constraints. Comparison of pairs of constraints is sufficient for detecting all kinds of (i) Redundancy and (ii) Subsumption, but not for detecting all kinds of (iii) Inconsistency and (iv) Fusion. This is explained as follows:

(i) Redundancy

Consider 'n' constraints, namely, S_1, S_2, ……., S_n. Let us assume $S_1 \equiv S_2 \equiv \ldots\ldots. \equiv S_n$, i.e., redundancy exists between all the n constraints. By comparing all possible pairs of constraints, ConEditor+ detects the following ${}^nC_2$ cases: $S_1 \equiv S_2$, $S_1 \equiv S_3$, ……, $S_1 \equiv S_n$, $S_2 \equiv S_3$, …… $S_2 \equiv S_n$,….,$S_{n-1} \equiv S_n$. One can infer from the above ${}^nC_2$ cases that redundancy exists between all 'n' constraints. Moreover, when the domain expert eliminates redundancy in each of the ${}^nC_2$ cases, redundancy between all the 'n' constraints are eliminated.

(ii) Subsumption

The principles described above in (i) apply here too. Consider 'n' constraints, namely, S_1, S_2, ……., S_n. Let us assume S_1 subsumes {S_2, S_3,…, S_n}, i.e., one constraint subsumes all the other (n-1) constraints. By comparing all possible pairs of constraints, ConEditor+ detects the following (n-1) cases: S_1 subsumes S_2, S_1 subsumes S_3, ……, S_1 subsumes S_n. One can infer from the above (n-1) cases that S_1 subsumes {S_2, S_3,…, S_n}. Moreover, when the domain expert eliminates subsumption in each of the (n-1) cases, all cases of subsumption are eliminated.

However, comparison of all possible pairs of constraints is insufficient (or incomplete) for detecting (iv) Inconsistency and (v) Fusion. This is explained as follows:

(iii) Inconsistency

Consider 'n' constraints, namely, S_1, S_2, ……., S_n. Let us assume $\forall x$ {S_1: $P(x) < Q(x)$, S_2: $Q(x) < R(x)$,..…, S_n: $R(x) < P(x)$}, where $x \in C$, C is a class in the domain ontology, Q and R are properties in the domain ontology. By comparing S_1, S_2 and S_n, one can infer that there exists an inconsistency between them. This kind of inconsistency cannot be detected by comparing all pairs of constraints.

(iv) Fusion

Consider 'n' constraints, namely, S_1, S_2, ……., S_n. Let us assume S_1, S_2, ……., S_n, can be fused into a single constraint S by applying the rule of fusion via class to 'n' constraints. This kind of fusion cannot be detected by comparing all pairs of constraints.

The reasons for comparing only pairs of constraints in ConEditor+ are as follows:

(a) Comparison of all combinations of constraints is more complex and substantially increases the complexity of the algorithm, especially, when one considers an arbitrary number of first-order logic expressions. It is planned to investigate this issue as part of future work.

(b) Moreover, the main aim of the research work is to demonstrate the usefulness of an explicit representation (machine-interpretable format) of application conditions together with the domain ontology in supporting the maintenance of constraints. This has been demonstrated by the results of experiments 1 and 4 that apply ConEditor+ to two KBs, one KB with application conditions and the other KB without application conditions.

The following section describes some possible directions for future work.

8.3 Future Work

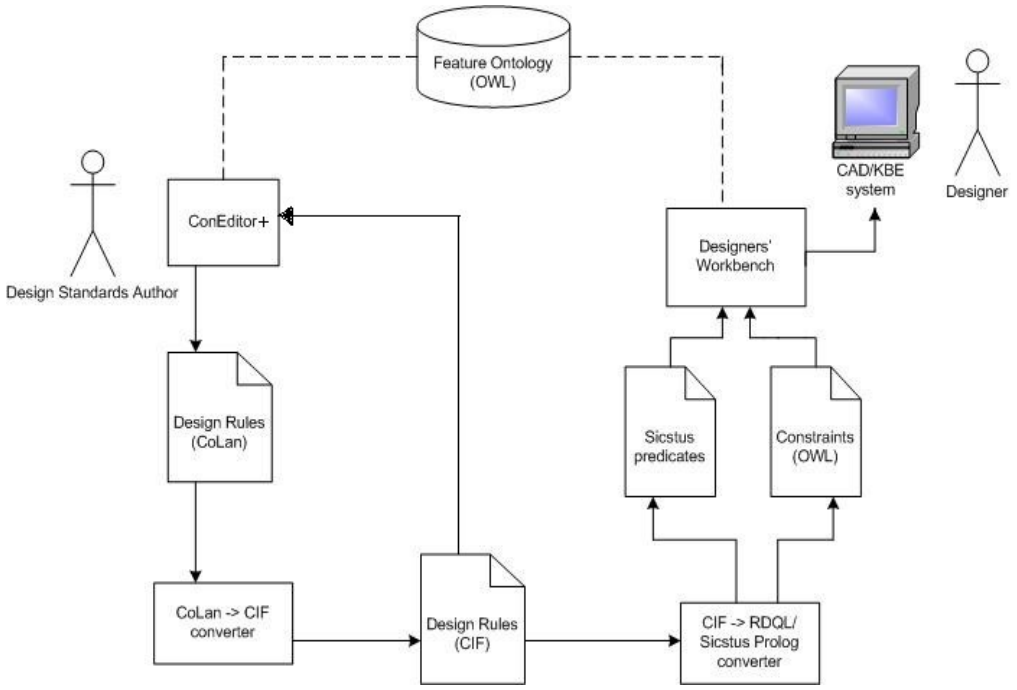


Figure 8.1: Proposed System Architecture

Figure 8.1 shows how ConEditor+ fits into a wider framework of the sophiscated KBE system. A Design Standards author captures and maintains all the design rules (constraints) using ConEditor+. The constraints are converted by ConEditor+ into the standard constraint interchange format (CIF); subsequently the constraints in CIF

format are converted into a predicate in prolog and a query in RDQL, and the latter are then processed by the Designers' Workbench. The domain ontology is represented in OWL and used by both ConEditor+ and the Designers' Workbench to express constraints. As part of the future work, it is planned to interface the Designers' Workbench to a sophisticated knowledge-based engineering (KBE) system. The Designers' Workbench will be called from the KBE system, effectively as a sub- process to check the consistency of a design, or part of a design.

Acquisition and Maintenance of design knowledge from additional knowledge repositories: In fact, Figure 8.1 only represents some aspects (rule books/design constraints and ontologies) of the knowledge that is both generated and used by a contemporary of the knowledge-based engineering firm that is involved in the design, manufacturing and maintenance of artefacts. For example, there are additional knowledge repositories needed by today's KBE systems, including:

- Design templates (and conditions under which they should be used, i.e. *application conditions*)
- Libraries of designs for components and their rationales
- Requirements and constraints of the various manufacturing environments
- Best practices as collected by several parts of the organization (including designers)
- Requirements and constraints mandated by the several organizations which service the engines
- Feedback from the servicing and maintenance organizations that indicate which problems actually arise in the field, some analysis of their possible causes, and suggested remedies.

Acquisition and maintenance of some of the above mentioned knowledge repositories (in particular, the feedback from servicing and maintenance organisations to the designers) is the focus of IPAS (2005), a DTI / Rolls-Royce funded project. Further information on work done in this area can be found in Wong *et al*. (2008).

Ontology Creation and Maintenance: It can be observed that ontologies have played an important role in both the systems discussed in this thesis, namely, the

Designers' Workbench and ConEditor+. The design rules have been expressed against the appropriate domain ontology. The domain ontology has been used along with the constraints and application conditions to support the maintenance of constraints. Ontologies play an important role in the IPAS project too. There are vast amounts of data and information available from a variety of sources, and to make this information inter-operational, there is potentially a major role for ontologies as many of the data / information sources use different terminologies. Creation and maintenance of these domain ontologies involves various issues/problems. In fact, in both the projects undertaken with Rolls-Royce, i.e. AKT (2000) and IPAS (2005), there are many problems of contemporary ontology engineering, namely:

- ontology creation (seeking to develop ontologies systematically and to ensure that relevant aspects of trust and provenance are captured; deciding whether or not domain ontologies should be developed from high-level ontologies);
- ontology evolution (an ontology developed for one engine may need to be modified so that it is applicable to a future engine) and,
- ontology modularization (for some services a sparse description of, say, the combustion chamber may be sufficient, but for other services much detail may be required).

Further information on work done in this area can be found in Sleeman *et al*. (2008). Some other directions for future work include:

Constraint Evolution History: Engineering design constraints are evolutionary in nature (Goonetillake *et al*., 2002; Goonetillake & Wikramanayake, 2004). ConEditor+ assigns each constraint a unique identification number. When a constraint is modified and saved, the number is incremented and a new identification number is assigned. ConEditor+ uses the latest versions of constraints by default. A previous version of the constraint can be retrieved by using a keyword-based search. It would be useful to provide users with a facility to view the way in which a particular constraint has evolved. ConEditor+ could provide a feature to view the evolution of constraints together with the corresponding application conditions.

Protégé Plug-in: Protégé (Noy *et al.*, 2000) is a popular tool for developing ontologies. It would be useful to provide ConEditor+ as a Protégé plug-in to help users capture and maintain constraints over ontologies. ConEditor+ currently captures the constraints in the CoLan language and converts them into CIF. Recently, an extension to the SWRL language (Horrocks *et al.*, 2004) to express fully quantified constraints has been developed and is known as CIF/SWRL (McKenzie *et al*., 2004). Developing a transformation program to enable ConEditor+ to produce constraints in CIF/SWRL would also be useful.

Design rationales: The research work reported in this thesis has concentrated on capturing the assumptions and context associated with a constraint as an application condition. Application conditions constitute one type of design rationales that refer to "when" a constraint is applicable. There are other rationales that refer to "why", "who", "how", etc. It would be interesting to investigate the capture of these additional rationales that could be associated with a constraint. Applying the design principles of ConEditor+ to enable the capture and use of these other types of rationales (some of which are captured by DRed) for maintenance could be investigated as part of future work.

Bibliography

Abdalla, H. S. (1997). A constraint knowledge based system for supporting a mechatronics design environment. In *Proceedings of the 14th International Conference on Production Research*, pages 842-845, Osaka, Japan.

Abdalla, H. S. (1998). A Concurrent Engineering Constraint-based System. *Computers in Industry,* 35(3-4):459-462.

Aiken, A., & Sleeman, D. (2003). Refiner++: A Knowledge Acquisition and Refinement Tool. In *Proceedings of the Workshop on Capturing Knowledge from Domain Experts: Progress & Prospects, KCAP 2003*, Sanibel Island, Florida.

Ajit, S., Sleeman, D., Fowler, D. W., & Knott, D. (2004). ConEditor: Tool to Input and Maintain Constraints. In *Proceedings of the 14th International Conference on Engineering Knowledge in the Age of the Semantic Web, EKAW 2004*, pages 466 - 468, Whittlebury Hall, Northampton, UK.

Ajit, S., Sleeman, D., Fowler, D. W., & Knott, D. (2008a). Constraint capture and maintenance in engineering design. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing (Special Issue on Design Rationale),* 22(4):325-343.

Ajit, S., Sleeman, D., Fowler, D. W., Knott, D., & Hui, K. (2005a). Acquisition and Maintenance of Constraints in Engineering Design. In *Proceedings of the 3rd International Conference on Knowledge Capture, KCAP 2005*, pages 173-174, Banff, Canada.

Ajit, S., Sleeman, D., Fowler, D. W., Knott, D., & Hui, K. (2005b). Capture and Maintenance of Engineering Design Constraints. In *Proceedings of the Twenty-fifth SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence as Poster (CD Proceedings), AI 2005*, Cambridge, UK.

Ajit, S., Sleeman, D., Fowler, D. W., Knott, D., & Hui, K. (2005c). Capture and Maintenance of Engineering Design Constraints. In N. Shadbolt & Y. Kalfoglou (Eds.), *Advanced Knowledge technologies: Selected Papers 2005* (pp. 309-322).

Ajit, S., Sleeman, D., Fowler, D. W., Knott, D., & Hui, K. (2006). Capture and Maintenance of Engineering Design Constraints. In *Proceedings of the 2nd AKT Doctoral Symposium 2006*, pages 4-13, Aberdeen, UK.

Ajit, S., Sleeman, D., & Knott, D. (2008b). *Analysis of Design Rule Books of part of the Rolls-Royce domain*. Aberdeen: Department of Computing Science, University of Aberdeen.

AKA. (2006). *American Kite Association*. Retrieved 28 June, 2006, from http://www.aka.org.au/kites_in_the_classroom/index.htm

Anumba, C. J., Ugwu, O., Newnham, L., & Thorpe, A. (2001). A multi-agent system for distributed collaborative design. *Logistics Information Management,* 14:355-366.

Aurisicchio, M., Bracewell, R., & Wallace, K. M. (2006). Evaluation of DRed: a way of capturing and structuring engineering processes. In *Proceedings of the NordDesign 2006*, pages 169-178, Reykjavik, Iceland.

Bahler, D., & Bowen, J. (1992). Design Rationale Management in Concurrent Engineering. In *Proceedings of the Workshop on Design Rationale Capture and Use,10th Natl. Conf. on Artificial Intelligence (AAAI-92)*, San Jose, USA.

Bahler, D., Dupont, C., & Bowen, J. (1994). An axiomatic approach that supports negotiated resolution of design conflicts in Concurrent Engineering. *Artificial Intelligence in Design*:363-379.

Barker, V. E., & O'Connor, D. E. (1989). Expert Systems for Configuration at Digital: XCON and Beyond. *Communications of the ACM,* 32(3):298-318.

Barnum, C. M. (2002). *Usability Testing and Research*: The Allyn and Bacon series in Technical Communication.

Bassiliades, N., & Gray, P. (1995). CoLan: A Functional Constraint Language and Its Implementation. *Data and Knowledge Engineering,* 14(3):203-249.

Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The Semantic Web. *Scientific American (May 2001 issue)*.

Berry, D. (1987). The problem of implicit knowledge. *Expert Systems,* 4(3):144-151.

Bhansali, S., Kramer, G. A., & Hoar, T. J. (1996). A principled approach towards symbolic geometric constraint satisfaction. *Journal of Artificial Intelligence Research,* 4:419-443.

Blythe, J., Kim, J., Ramachandran, S., & Gil, Y. (2001). An Integrated Environment for Knowledge Acquisition. In *Proceedings of the International Conference on Intelligent User Interfaces*, pages 13-20.

Blythe, J., & Ramachandran, S. (1999). Knowledge acquisition using an english- based method editor. In *Proceedings of the Twelfth Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Alberta.

Boose, J. H., & Bradshaw, J. M. (1999). Expertise transfer and complex problems: Using AQUINAS as a knowledge acquisition workbench for knowledge-based systems. *International Journal of Human-Computer Studies,* 51:453-478.

Borning, A., Maher, M., Martindale, A., & Wilson, M. (1989). Constraint Hierarchies and Logic Programming. In *Proceedings of the International Conference on Logic Programming (ICLP)*, pages 149-164, Lisbon, Portugal.

Borst, W. N. (1997). *Construction of Engineering Ontologies for Knowledge sharing and Reuse.* PhD thesis, University of Twente, Enschede, The Netherlands.

Borst, W. N., Akkermans, J. M., & Top, J. L. (1997). Engineering Ontologies. *International Journal of Human-Computer Studies,* 46(2-3):365-406.

Bowen, J. (1997). Using dependency records to generate design coordination device in a constraint-based approach to Concurrent Engineering. *Computers in Industry,* 33(2-3):191-199.

Bowen, J. (2001). Constraint-Based Co-operative Problem-Solving: A case study from Concurrent Engineering. In *Proceedings of the Workshop on Cooperative Solvers in Constraint Programming, CP 2001*, Cyprus.

Bowen, J., & Bahler, D. (1992). Frames, quantification, perspectives and negotiation in constraint networks in life-cycle engineering. *Artificial Intelligence in Engineering,* 7(4):199-226.

Bracewell, R. H., & Wallace, K. M. (2003). A Tool for Capturing Design Rationale. In *Proceedings of the International Conference on Engineering Design (ICED 03)*, Stockholm.

Bradburn, N. M. (2000). Questionnaire Design: from Art into Science. In *Proceedings of the Fifth International Conference on Social Science Methodology*, Cologne, Germany.

Brazier, F. M., Langen, P. H. G. V., & Treur, J. (1997). A compositional approach to modelling design rationale. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing,* 11:125-139.

Brimble, R., & Sellini, F. (2000). The MOKA Modelling Language. In *Proceedings of the EKAW 2000 conference*, pages 49-56, Dagstuhl, Germany.

Bromby, M., Macmillan, M., & McKellar., P. (2003). A CommonKADS Representation for a Knowledge-based System to Evaluate Eyewitness Identification. *International Review of Law Computers,* 17(1):99-108.

Brown, D. C. (2006). Assumptions in Design and Design Rationale. In *Proceedings of the Design Rationale Workshop (DCC'06)*, Eindhoven, Netherlands.

Brown, D. C., & Chandrasekaran, B. (1985). Expert systems for a class of mechanical design activity: Knowledge engineering. In J. Gero (Ed.), *Computer-Aided Design* (pp. 259-283).

Brown, D. C., & Chandrasekaran, B. (1989). *Design Problem Solving: Knowledge Structures and Control Strategies*. London: Pitman.

Buchanan, B. G., Barstow, D., Bechtel, R., Bennet, J., Clancey, W., Kulikowski, C., *et al*. (1983). Constructing an Expert System. In F. Hayes-Roth, D. A. Waterman & D. Lenat (Eds.), *Building Expert Systems*.

Bultman, A., Kuipers, J., & Harmelen, F. V. (2000a). Maintenance of KBS's by Domain Experts: The Holy Grail in Practice. In *Proceedings of the Thirteenth International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems IEA/AIE'00*.

Bultman, A., Kuipers, J., & Harmelen, F. V. (2000b). Maintenance of KBS's by Domain Experts: The Holy Grail in Practice. In *Proceedings of the Thirteenth International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems IEA/AIE'00*, pages 139-148, New Orleans, USA.

Burge, J. (1998). *Knowledge Elicitation for Design Task Sequencing Knowledge.* MSc thesis, Worcester Polytechnic Institute, Worcester, USA.

Burge, J., & Brown, D. C. (2003). Rationale Support for Maintenance of Large Scale Systems. In *Proceedings of the Workshop on Evolution of Large-Scale Industrial Software Applications (ELISA), ICSM '03*, Amsterdam, NL.

Burge, J. E., & Brown, D. C. (2000). Reasoning with Design Rationale. In J. Gero (Ed.), *Artificial Intelligence in Design '00* (pp. 611-629). Netherlands: Kluwer Academic Publishers.

Bylander, T., & Chandrasekaran, B. (1987). Generic Tasks for Knowledge-Based Reasoning: The "Right" Level of Abstraction for Knowledge Acquisition. *International Journal of Man-Machine Studies,* 26(2):231-243.

Callot, M., Kneebone, S., Oldham, K., Murton, A., & Brimble, R. (1999). MOKA - A Methodology for developing Knowledge Based Engineering Applications. In *Proceedings of the 8th European Conference on Product Data Technology*, pages 361-366, Stavanger, Norway.

Carbonara, L., & Sleeman, D. (1999). Effective and Efficient Knowledge Base Refinement. *Machine Learning,* 37:143-181.

Carnduff, T. W., & Goonetillake, J. S. (2004). Configuration management in evolutionary engineering design using versioning and integrity constraints. *Advances in Engineering Software,* 35:161-177.

CEKS. (2006). *Cutting Edge Kite Shop*. Retrieved 28 June, 2006, from http://www.cuttingedgekites.com/faq.htm

Chandrasekaran, B. (1986). Generic Tasks in Knowledge-based Reasoning: High- level Building Blocks for Expert System Design. *IEEE Expert,* 1(3):23-30.

Chandrasekaran, B., Johnson, T. R., & Smith, J. W. (1992). Task Structure Analysis for Knowledge Modeling. *Communications of the ACM,* 35(9):124-137.

Cheung, W. M., Bramall, D. G., Maropoulos, P. G., Gao, J. X., & Aziz, H. (2006). Organizational knowledge encapsulation and re-use in collaborative product development. *International Journal of Computer Integrated Manufacturing,* 19(7):736-750.

Ciocoiu, M., Gruninger, M., & Nau, D. S. (2001). Ontologies for integrating engineering applications. *Journal of Computers, Information Science and Engineering,* 1:12-22.

Clancey, W. J. (1985). Heuristic Classification. *Artificial Intelligence,* 27:289-350.

Clarkson, P. J., & Hamilton, J. R. (2000). 'Signposting', A Parameter-driven Task-based Model of the Design Process. *Research in Engineering Design,* 12:18-38.

Cleland, D. I. (2004). *Field Guide to Project Management*: Wiley.

Coenen, F. P. (1992). A Methodology for the Maintenance of Knowledge based Systems. In *Proceedings of the Niku-Lari, A. (Ed), EXPERSYS-92, IITT-International*, pages 171-176, France.

Conklin, J., & Begeman, M. L. (1988). gIBIS: A hypertext tool for exploratory policy discussion. *ACM Transactions Office Information Systems,* 4:303-331.

Conklin, J., & Burgess, Y. (1991). A Process-Oriented Approach to Design Rationale. *Human Computer Interaction,* 6(3-4):357-391.

Corbridge, C., Rugg, G., Major, N. P., Shadbolt, N. R., & Burton, A. M. (1994). Laddering: technique and tool use in knowledge acquisition. *Knowledge Acquisition,* 6(3):315-341.

Cordingley, E. S. (1989). *Knowledge Elicitation Techniques for knowledge based systems (D. Diaper, Eds.)*. Chichester, England: Ellis Horwood Ltd.

Craw, S., & Sleeman, D. (1990). Automating the Refinement of Knowledge-Based Systems. In *Proceedings of the ECAI 1990*, pages 167-172, Stockholm, Sweden.

Craw, S., & Sleeman, D. (1995). *Knowledge-based Refinement of Knowledge Based Systems* (Technical Report No. 95/2). Aberdeen, UK: School of Computer and Mathematical Sciences, The Robert Gordon University.

Crowder, R., Bracewell, R., Hughes, G., Kerr, M., Knott, D., & Moss, M. (2003). A Future Vision for the Engineering Design Environment: A Future SocioTechnical Scenario. In *Proceedings of the International Conference on Engineering Design (ICED 03)*, Stockholm.

Cutkosky, M. R., Engelmore, R. S., Fikes, R. E., Geneserth, M. R., Gruber, T. R., Mark, W. S., *et al*. (1993). PACT: an experiment in integrating concurrent engineering systems. *IEEE Computer,* 26(1):28-37.

Darlington, M. J., & Culley, S. J. (2008). Investigating ontology development for engineering design support. *Advanced Engineering Informatics,* 22(1):112- 134.

Davis, R. (1979). Interactive transfer of expertise: Acquisition of new inference rules. *Artificial Intelligence,* 12(2):409-427.

Debowski, S. (2006). *Knowledge Management*. Australia: John Wiley & Sons.

Demian, P., & Fruchter, R. (2005). Measuring Relevance in support of design reuse from archives of building product models. *ASCE Journal of Computing in Civil Engineering,* 29(2):119-136.

Diaper, D. (1989). *Knowledge Elicitation: Principle, Techniques and Applications*. England, UK.

Dieng, R., & Corby, O. (2000). Knowledge Acquisition, Modeling and Management. In *Proceedings of the EKAW 2000*, Juan-les-Pins, France.

Dieng, R., Corby, O., Giboin, A., & Ribiere, M. (1999). Methods and tools for corporate knowledge management. *International Journal of Human-Computer Studies,* 51(3):567-598.

Dumas, J. S., & Redish, J. C. (1999). *A Practical guide to Usability Testing*: Intellect Books.

Eden, M. (1998). *The Magnificient Book of Kites: Explorations in Design, Construction, Enjoyment and Flight*: Black Dog & Levanthal Publishers, New York.

Embury, S., & Gray, P. (1995). The Declarative Expression of Semantic Integrity in a Database of Protein Structure. In *Proceedings of the Data Management Systems: Proceedings of the Basque International Workshop on Information Technology (BIWIT 95)*, pages 216-224, San Sebastian, Spain.

Ericsson, K. A., & Simon, H. A. (1984). *Protocol Analysis: Verbal reports as data (revised edition)*. Cambridge, MA.

Eriksson, H., & Musen, M. (1993). Conceptual Models for Automatic Generation of Knowledge-Acquisition Tools. *Knowledge Engineering Review,* 8:27-47.

Eriksson, H., Puerta, A., Gennari, J., Rothenfluh, T., Tu, S., & Musen, M. (1995a). Custom-tailored development tools for knowledge-based systems. In *Proceedings of the Ninth Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada.

Eriksson, H., Puerta, A., & Musen, M. (1994). Generation of knowledge-acquisition tools from domain ontologies. *International Journal of Human-Computer Studies,* 41:425-453.

Eriksson, H., Shahar, Y., Tu, S. W., Puerta, A. R., & Musen, M. A. (1995b). Task Modeling with Reusable Problem-Solving Methods. *Artificial Intelligence,* 79(2):293-326.

Eshelman, L., Ehret, D., McDermott, J., & Tan, M. (1988). MOLE: A Tenacious Knowledge Acquisition Tool. *Knowledge Based Systems,* 1:95-108.

Feigenbaum, E. A. (1977). The Art of Artificial Intelligence: Themes and Case Studies in Knowledge Engineering. In *Proceedings of the IJCAI-77*, pages 1014-1029.

Felfernig, A., Friedrich, G., Jannach, D., & Stumptner, M. (2004). Consistency-based diagnosis of configuration knowledge bases. *Artificial Intelligence,* 152:213-234.

Fensel, D. (2004). *Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce* (2nd ed.): Springer-Verlag Berlin and Heidelberg GmbH & Co.

Fensel, D., Angele, J., & Studer, R. (1998). The Knowledge Acquisition and Representation Language, KARL. *IEEE Transactions on Knowledge and Data Engineering,* 10(4):527-550.

Fensel, D., & Poeck, K. (1994). A Comparison of Two Approaches to Model-based Knowledge Acquisition. In *Proceedings of the EKAW 94*, pages 46-62, Belgium.

Fischer, G., Lemke, A., McCall, R., & Morch, A. (1995). Making Argumentation Serve Design. In T. M. a. J. Carroll (Ed.), *Design Rationale Concepts, techniques, and Use* (pp. 267-294).

Fleischanderl, G., Friedrich, G., Haselbock, A., & Schreiner, H. (1998). Configuring large systems using generative constraint satisfaction. *IEEE Intelligent Systems and their applications,* 13(4):59-68.

Fletcher, D., & Gu, P. (2005). Adaptable Design for Design Reuse. In *Proceedings of the Second CDEN International Conference on design Education, Innovation, and Practice*, Canada.

Fowler, D. W., Sleeman, D., Wills, G., Lyon, T., & Knott, D. (2004). Designers' Workbench. In *Proceedings of the Twenty-fourth SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence*, pages 209-221, Cambridge, UK.

Frayman, F., & Mittal, S. (1987). COSSACK: A constraints-based expert system for configuration tasks. In D. Sriram & R. A. Adey (Eds.), *Knowledge based Expert systems in Engineering: Planning and Design* (pp. 143-166). Southampton, UK: Computational Mechanics Publications.

French, T. E., Vierck, C. J., & Foster, R. J. (1993). *Engineering Drawing and Graphics Technology*: McGraw-Hill Inc.

Fruchter, R., & Demian, P. (2002). CoMem: designing an interaction experience for reuse of rich contextual information from a corporate memory. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing,* 16(3):127-147.

Fruchter, R., Saxena, K., Breidenthal, M., & Demian, P. (2007). Collaborative Design Exploration in an Interactive Workspace. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing,* 21(3):279-293.

Gao, X. S., & Chou, S. C. (1998a). Solving geometric constraint systems I: A global propogation approach. *Computer-Aided Design,* 30(1):47-54.

Gao, X. S., & Chou, S. C. (1998b). Solving geometric constraint systems II: A symbolic approach and decision of Rc-constructability. *Computer-Aided Design,* 30(2):115-122.

Garcia, A. C. B., & Howard, H. C. (1992). Acquiring design knowledge through design decision justification. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing,* 6(1):59-71.

Gennari, J. H., Altman, R. B., & Musen, M. A. (1995). Reuse with PROTEGE-II: From Elevators to Ribosomes. In *Proceedings of the Symposium on Software Reuse*, Seattle.

Ginsberg, A. (1988). Knowledge-Base Reduction: A New Approach to Checking Knowledge Bases for Inconsistency and Redundancy. In *Proceedings of the AAAI 88*, pages 585-589, Saint Paul, Minnesota.

Godambe, V. P. (1982). Estimation in survey sampling: robustness and optimality. *Journal of the American Statistical Association,* 77:393-403.

Goodall. (1996). The PC PACK Knowledge Analysis Tool. *AI Watch,* 5:1-9.

Goonetillake, J. S., Carnduff, T. W., & Gray, W. A. (2002). An integrity constraint management framework in engineering design. *Computers in Industry,* 48(1):29-44.

Goonetillake, J. S., & Wikramanayake, G. N. (2004). Management of Evolving Constraints in a Computerised Engineering Design Environment. In *Proceedings of the 23rd National IT Conference*, pages 43-54, Colombo, Sri Lanka.

Gray, P., Embury, S., Hui, K., & Kemp, G. (1999a). The evolving role of constraints in the functional data model. *Journal of Intelligent Information Systems,* 12:113-137.

Gray, P., Hui, K., & Preece, A. (1999b). Finding and moving constraints in cyberspace. *Intelligent Systems in cyberspace*:121-127.

Gray, P., Hui, K., & Preece, A. (2001). An Expressive Constraint Language for Semantic Web Applications. In *Proceedings of the E-Business and the Intelligent Web: Papers from the IJCAI-01 Workshop*, pages 46-53, Seattle, USA.

Gross, M. D., Ervin, S. M., Anderson, J. A., & Fleisher, A. (1988). Constraints: Knowledge representation in design. *Design Studies,* 9(3):133-143.

Grosso, W., Eriksson, H., Fergerson, R., Gennari, J., Tu, S., & Musen, M. (1999). Knowledge Modeling at the Millenium (The Design and Evolution of Protege-2000). In *Proceedings of the Twelfth Banff Workshop on Knowledge Acquisition, Modeling and Management*, Banff, Alberta.

Gruber, T. (1993). A translation approach to portable ontology specification. *Knowledge Acquisition,* 5(2):199-221.

Gruber, T. R., & Olsen, G. R. (1994). An Ontology for Engineering Mathematics. In *Proceedings of the Fourth International Conference on Principles of Knowledge Representation and Reasoning*, pages 258-269, Bonn, Germany.

Gruber, T. R., & Russell, D. M. (1991). *Design Knowledge and Design Rationale: A Framework for Representation, Capture and Use* (Technical Report KSL 90-45). California: Computer Science Department, Stanford University.

Grudin, J. (1996). Evaluating opportunities for design rationale capture. In T. P. Moran & J. M. Carroll (Eds.), *Design rationale: Concepts, techniques, and use.* (pp. 453-470). Mahwah, NJ, USA: Lawrence Erlbaum Associates, Inc.

Guihua, L., Bracewell, R., & Wallace, K. (2002). *A Literature Survey Technical Report*. Cambridge: Cambridge Engineering Design Centre.

Harary, F. (1962). A Graph Theoretic Approach to Matrix Inversion by Partitioning. In *Numerische Mathematik* (Vol. 4, pp. 128-135).

Haroud, D., Boulanger, S., Gelle, E., & Smith, I. (1995). Management of conflict for preliminary engineering design tasks. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing,* 9:313-323.

Harris, S., & Gibbins, N. (2003). 3store:Efficient Bulk RDF Storage. In *Proceedings of the 1st International Workshop on Practical and Scalable Semantic Systems (PSSS'03), International Semantic Web Conference*, Sanibel Island, Florida.

Haselbock, A., & Stumptner, M. (1993). An integrated approach for modelling complex configuration domains. In *Proceedings of the 13th International Conference on Artificial Intelligence*, pages 625-634, Avignon, France.

Hayes-Roth, F., Waterman, D. A., & Lenat, D. B. (1983). *Building Expert Systems* (Vol. 1): Addison-Wesley.

Hengl, T. (2004). Protege, Ontology and Knowledge Acquisition: Knowledge Representation, the Foundation of Intelligent Systems. *PCAI magazine,* vol. 18.3*,* pages 37-48.

Hicks, R. C. (2003). Knowledge base management systems-tools for creating verified intelligent systems. *Knowledge-Based Systems,* 16(3):165-171.

Hinkle, D. N. (1965). *The change of personal constructs from the viewpoint of a theory of implications.* PhD thesis, Ohio State University, Ohio, USA.

Hooey, B. L., & Foyle, D. C. (2007). Requirements for a Design Rationale Capture Tool to Support NASA's Complex Systems. In *Proceedings of the International Workshop on Managing Knowledge for Space Missions,* , Pasadena, USA.

Horrocks, I., Patel-Schneider, P. F., Boley, H., Tabet, S., Grosof, B., & Dean, M. (2004). *SWRL: A Semantic Web Rule Language: Combining OWL and*

RuleML. Retrieved 11 June 2007, from http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/

HP. (2004). *Jena-a semantic web framework for java. Developed at Helwett Packard Labs*. Retrieved 28 June, 2006, from http://jena.sourceforge.net/index.html

Hu, X., Pang, J., Pang, Y., Atwood, M., Sun, W., & Regli, W. C. (2000). A Survey on Design Rationale: Representation, Capture and Retrieval. In *Proceedings of the ASME Design Engineering Technical Conference*, Baltimore, Maryland.

IPAS. (2005). *Integrated Products and Services*. Retrieved 9 May, 2008, from http://www.3worlds.org/

Johanson, B., Fox, A., & Winograd, T. (2002). The Interactive Workspaces Project: experiences with ubiquitous computing rooms. *IEEE Pervasive Computing,* 1(2):67-75.

Jordan, P. W. (1998). *An Introduction to Usability*. UK: Taylor & Francis.

Junker, U., & Mailharro, D. (2003). The logic of ilog(j) configurator: Combining constraint programming with description logic. In *Proceedings of IJCAI'03 Workshop on Configuration*.

Kahn, G., Nowlan, S., & McDermott, J. (1985). MORE: An Intelligent Knowledge Acquisition Tool. In *Proceedings of the Ninth Joint Conference on Artificial Intelligence*, pages 581-584, Los Angeles, CA.

Karensty, L. (1996). An Empirical Evaluation of Design Rationale Documents. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 150-156, Canada.

Kelly, G. A. (1955). *The Psychology of Personal Constructs*. New York, USA: W. W. Norton & Company Inc.

Kim, J., & Gil, Y. (1999). Deriving expectations to guide knowledge base creation. In *Proceedings of the AAAI/IAAI*, pages 235-241.

Kingston, J. K. C. (1998). Designing knowledge based systems: the CommonKADS design model. *Knowledge-Based Systems,* 11:311-319.

Klein, R. (2000). Knowledge Modeling in Design - the MOKA framework. In *Proceedings of the International Conference on AI in Design*, pages 77-102, Worcester, MA.

Knublauch, H., Fergerson, R. W., Noy, N. F., & Musen, M. A. (2004). The Protege OWL Plugin: An Open Development Environment for Semantic Web Applications. In *Proceedings of the International Semantic Web Conference*, pages 229-243, Hiroshima, Japan.

Koo, D. Y., Han, S. H., & Lee, J. W. (1998). An object-oriented configuration design method for paper feeding mechanisms. *Expert Systems with Applications,* 14(3):283-289.

Kothari, C. R. (2005). *Research Methodology: Methods and Techniques (Second Edition)*. New Delhi: New Age Publishers.

Laburthe, F. (2003). Constraints over ontologies. In *Proceedings of the CP 2003*, pages 878-882.

Lam, S., Sleeman, D., Pan, J., & Vasconcelos, W. (2008). A Fine-Grained Approach to Resolving Unsatisfiable Ontologies. *Journal on Data Semantics,* 10:62-95.

Lam, S., Sleeman, D., & Vasconcelos, W. (2005). ReTAX++: a Tool for Browsing and Revising Ontologies. In *Proceedings of the Posters and Demonstrations of ISWC-05*, Galway, Ireland.

Landauer, T. K., & Dumais, S. T. (1995). A solution to Plato's problem: the latent semantic analysis theory of acquisition, induction and representation of knowledge. *Psychological Review,* 104(2):211-240.

Landes, D. (1994). An Approach to the Design of Knowledge-Based Systems. In *Software Quality Management II Vol 2 Building Quality into Software* (pp. 707-722).

Lee, J. (1997). Design Rationale Systems: Understanding the Issues. *IEEE Expert,* 12(3):78-85.

Lee, J., Chae, H., Kim, C.-H., & Kim, K. (2009). Design of product ontology architecture for collaborative enterprises. *Expert Systems with Applications (Available online 28 December 2007),* 36(2):2300-2309.

Leigh, D. (2006). *Dan Leigh Delta Kite Designs*. Retrieved 28 June, 2006, from http://www.deltas.freeserve.co.uk/home.html

Leo, P. (1995). *S-SALT: A Problem Solver plus Knowledge Acquisition Tool which additionally can refine its knowledge base.* MSc thesis, University of Aberdeen, Aberdeen.

Levy, P. S., & Lemeshow, S. (1991). *Sampling of Populations: Methods and Applications (Second Edition)*. New York: Wiley-Interscience.

Lin, H. K., & Harding, J. A. (2003). An ontology driven manufacturing system engineering moderator for global virtual enterprise teams. In *Proceedings of the 1st International Conference on Manufacturing Research*, pages 365-370, Glasgow.

Lin, L., & Chen, L. C. (2002). Constraints modelling in product design. *Journal of Engineering Design,* 13(3):205-214.

Lloyd, R. C. (2004). *Quality Health Care: A Guide to Developing and Using Indicators*. Sudbury, USA: Jones & Bartlett.

Lords, D. (2006). *Kite, Kite Buggy and Land Yacht Page*. Retrieved 28 June, 2006, from http://users.techline.com/lord/index.html

Lottaz, C., Smith, I. F. C., Robert-Nicoud, Y., & Faltings, B. V. (2000). Constraint-based support for negotiation in collaborative design. *Artificial Intelligence in Engineering,* 14:261-280.

Lottaz, C., Stalker, R., & Smith, I. (1998). Constraint solving and preference activation for interactive design. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing,* 12:13-27.

Lu, S. C.-Y., & Cai, J. (2001). A collaborative design process model in the socio technical engineering design framework. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing,* 15:3-20.

Maher, M. L. (1988). HI-RISE: An expert system for preliminary structural design. In M. Rychener (Ed.), *Expert Systems for Engineering Design* (pp. 37-52).

Marcus, S. (1988). *Automating Knowledge Acquisition for Expert Systems*. Boston: Kluwer Academic Publisher.

Marcus, S., & McDermott, J. (1989). SALT: A knowledge acquisition language for propose-and-revise systems. *Artificial Intelligence,* 39(1):1-37.

Marcus, S., Stout, J., & McDermott, J. (1992). VT: An Expert Elevator Designer that uses knowledge-based backtracking. In C. Tong & D. Sriram (Eds.), *Artificial Intelligence in Engineering Design* (Vol. 1, pp. 317-356).

McDermott, J. (1982). R1-A rule-based configurer of computer systems. *Artificial Intelligence,* 19(1):39-88.

McDermott, J. (1988). Preliminary steps towards a taxonomy of problem-solving methods. In S. Marcus (Ed.), *Automating Knowledge Acquisition for Expert Systems* (pp. 225-256): Kluwer Academic.

McDermott, J. (1993). R1("XCON") at age 12: lessons from an elementary school achiever. *Artificial Intelligence,* 59:241-247.

McGuinness, D. L., & Harmelen, F. v. (2004). *OWL Web Ontology Language Overview, W3C Recommendation 10 February 2004*. Retrieved 29 August, 2006, from http://www.w3.org/TR/owl-features/

McGuinness, D. L., & Wright, J. R. (1998a). Conceptual Modelling for Configuration: A description logic based approach. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing,* 12:333-344.

McGuinness, D. L., & Wright, J. R. (1998b). An Industrial-Strength Description Logic Based Configurator Platform. *IEEE Intelligent Systems and their applications,* 13(4):69-77.

McKenzie, C., Gray, P., & Preece, A. (2004). Extending SWRL to Express Fully-Quantified Constraints. In *Proceedings of the Workshop on Rules and Rule Markup Languages for the Semantic Web (RuleML 2004), International Semantic Web Conference*, pages 139-154, Hiroshima, Japan.

McMahon, C., Lowe, A., & Culley, S. (2004). Knowledge management in engineering design: personalization and codification. *Journal of Engineering Design,* 15(4):307-325.

Menzies, T. (1999). Knowledge Maintenance: the State of the Art. *The Knowledge Engineering Review,* 14(1):1-61.

Meseguer, P., & Preece, A. D. (1995). Verification and Validation of Knowledge-Based Systems with Formal Specifications. *Knowledge Engineering Review,* 10:331-343.

Milton, N., Shadbolt, N., Cottam, H., & Hammersley, M. (1999). Towards a knowledge technology for knowledge management. *International Journal of Human-Computer Studies,* 51:615-641.

Milton, N. R. (2007). *Knowledge Acquisition in Practice*: Springer.

Milton, N. R. (2008). *Knowledge Technologies*: Springer.

Mitchell, T. M., Steinberg, L. I., & Shulman, J. S. (1985). A knowledge based approach to design. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 7(5):502-510.

Mittal, S., & Falkenhainer, B. (1990). Dynamic constraint satisfaction problems. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI 90)*, pages 25-32, Boston, USA.

Moore, J., Stader, J., Chung, P., Jarvis, P., & Macintosh, A. (1999). Ontologies to support the management of new product development in the chemical process industries. In *Proceedings of the 12th International Conference on Engineering Design (ICED 99)*, pages 159-164, Munich.

Musen, M., Tu, S., Eriksson, H., Gennari, J., & Puerta, A. (1993). PROTEGE-II: An Environment for Reusable Problem-Solving Methods and Domain Ontologies. In *Proceedings of the International Joint Conference on Artificial Intelligence*, Chambery, France.

Musen, M. A., Combs, D. M., Shortliffe, E. H., & Fagan, L. M. (1988). OPAL: Towards the computer aided design of oncology advice systems. *Selected Topics in Medical Artificial Intelligence*:166-180.

Myers, K., Zumel, N., & Garcia, P. (2000). Acquiring Design Rationale Automatically. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing,* 14(2):115-135.

Newnes, L. B., Mileham, A. R., Cheung, W. M., Marsh, R., Lanham, J. D., Saravi, M. E., *et al*. (2008). Predicting the whole-life cost of a product at the conceptual design stage. *Journal of Engineering Design,* 19(2):99-112.

Nguyen, T. A., Perkins, W. A., Laffey, T. J., & Pecora, D. (1985). Checking an Expert Systems Knowledge Base for Consistency and Completeness. In *Proceedings of the IJCAI '85*, pages 375-378, Los Angeles, USA.

Noy, N. F., Fergerson, R. W., & Musen, M. A. (2000). The knowledge model of Protege-2000: Combining interoperability and flexibility. In *Proceedings of the International Conference on Knowledge Engineering and Knowledge Management (EKAW' 2000)*, pages 17-32, Juan-les-Pins, France.

Noy, N. F., Sintek, M., Decker, S., Crubezy, M., Fergerson, R. W., & Musen, M. A. (2001). Creating Semantic Web Contents with Protege 2000. *IEEE Intelligent Systems,* 16(2):60-71.

Okolo, E. N. (1990). *Health Research Design and Methodology*. USA: CRC Press.

Olson, J. R., & Reuter, H. H. (1987). Extracting expertise from experts: methods for knowledge acquisition. *Expert Systems,* 4:152-168.

O'Muircheartaigh, C. A., Krosnick, J. A., & Helic, A. (1999). Middle alternatives, acquiescence, and the quality of questionnaire data. In *Proceedings of the American Association for Public Opinion Research Annual Meeting*, St. Petersburg, Florida.

O'Sullivan, B. (2002a). Constraint-based product structuring for configuration. In *Proceedings of the ECAI 2002 Workshop on Configuration*.

O'Sullivan, B. (2002b). Interactive constraint-aided conceptual design. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing,* 16:303- 328.

Pahl, G., & Beitz, W. (1995). *Engineering Design: A systematic approach*. London: Springer.

Payne, S. L. (1951). *The Art of Asking Questions*. New Jersey: Princeton University Press.

Poeck, K., & Gappa, U. (1993). Making Role-Limiting Shells more Flexible. In *Proceedings of the EKAW 93*, pages 103-122, Toulouse, France.

Preece, A. (1999). COVERAGE: Verifying Multiple-Agent Knowledge-Based Systems. *Knowledge-Based Systems,* 12:37-44.

Preece, A., Flett, A., Sleeman, D., Curry, D., Meaney, N., & Perry, P. (2001). Better Knowledge Management through Knowledge Engineering. *IEEE Intelligent Systems,* 14:26-36.

Preece, A. D., Shinghal, R., & Batarekh, A. (1992). Verifying Expert Systems: A Logical Framework and a Practical Tool. *Expert Systems with Applications,* 5(3/4):421-436.

Preece, J. (1993). *A Guide to Usability: Human Factors in Computing*: Addison Wesley.

Puerta, A. R., Egar, J. W., Tu, S. W., & Musen, M. A. (1992). A Multiple-Method Knowledge Acquisition Shell for the Automatic Generation of Knowledge Acquisition Tools. *Knowledge Acquisition,* 4(2):171-196.

Puerta, A. R., & Eriksson, H. (1996). Knowledge Engineering Environments and Knowledge Sharing: From Reusable Knowledge Components to the Internet (Tutorial Material). In *Proceedings of the ECAI 1996*, Budapest, Hungary.

Qian, Y., Zheng, M., Li, X., & Lin, L. (2005). Implementation of knowledge maintenance modules in an expert system for fault diagnosis of chemical process operation. *Expert Systems with Applications,* 28:249-257.

Regli, W. C., Hu, X., Atwood, M., & Sun, W. (2000). A Survey of Design Rationale Systems: Approaches, Representation, Capture and Retrieval. *Engineering with Computers: An Int'l Journal for Simulation-Based Engineering, special*

issue on Computer Aided Engineering in Honor of Professor Steven J. Fenves, 16(3-4):209-235.

Reichgelt, H., & Shadbolt, N. (1992). ProtoKEW: A knowledge-based system for knowledge acquisition. In D. Sleeman & N. O. Bernsen (Eds.), *Artificial Intelligence, Research Directions in Cognitive Science (Series)*. Hove, UK: L. Erlbaum Associates.

Richter, H. A., & Abowd, G. D. (1999). *Automating the capture of design knowledge: a preliminary study* (Technical Report No. GVU-99-45). Atlanta, USA: Georgia Institute of Technology.

Rittel, H. W. J. (1972). Second generation design methods. *The DMG 5th Anniversary Report: DMG Occasional paper No.1 (reprinted in Cross, N. (ed.) Developments in Design Methodology, 317-327)*.

Roche, C. (2000). Corporate ontologies and concurrent engineering. *Journal of Materials Process Technology,* 107:187-193.

Rothenfluh, T., Gennari, J., Eriksson, H., Puerta, A., Tu, S., & Musen, M. (1994). Reusable Ontologies, Knowledge-Acquisition Tools, and Performance Systems: PROTEGE-II Solutions to Sisyphus-2. In *Proceedings of the Eighth Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Alberta, Canada.

Rothenfluh, T. E., Gennari, J. H., Eriksson, H., Puerta, A. R., Samson, W. T., & Musen, M. A. (1996). Reusable ontologies, knowledge-acquisition tools, and performance systems: PROTEGE-II solutions to Sisyphus-2. *International Journal of Human-Computer Studies,* 44(3-4):303-332.

Rousset, M. C. (1988). On the Consistency of Knowledge Bases: the COVADIS System. *Computational Intelligence,* 4(2):166-170.

Rubin, J. (1994). *Handbook of Usability Testing*: Wiley Technical Communication Library.

Rychtyckyj, N., & Reynolds, R. G. (2000). Long-Term Maintainability of Deployed Knowledge Representation Systems. In *Proceedings of the 7th International Conference on the Principles of Knowledge Representation and Reasoning*, pages 494-504, Breckenridge.

Sabin, D., & Weigel, R. (1998). Product Configuration frameworks-a survey. *IEEE Intelligent Systems and their applications,* 13(4):42-49.

Salonen, M., Holtta-Otto, K., & Otto, K. (2008). Effecting product reliability and life cycle costs with early design phase product architecture decisions. *International Journal of Product Development,* 5(1/2):109-124.

Sanghee, K., Bracewell, R., & Wallace, K. (2008). Some reflections on ontologies in engineering domain. In *Proceedings of the 8th International Symposium on Tools and Methods of Competitive Engineering (TMCE 2008)*, Kusadasi, Turkey.

Sanghee, K., Bracewell, R. H., & Wallace, K. M. (2007). Improving design reuse using context. In *Proceedings of the International Conference on Engineering Design*, Paris, France.

Sapuan, S. M., Osman, M. R., & Nukman, Y. (2006). State of the art of the concurrent engineering technique in the automotive industry. *Journal of Engineering Design,* 17(2):143-157.

Schaeffer, N. C., & Presser, S. (2003). The Science of Asking Questions. *Annual Review of Sociology,* 29(65-88).

Schreiber, G., Akkermans, H., Anjewierden, A., Hoog, R. d., Shadbolt, N., VandeVelde, W., *et al*. (2000). *Knowledge Engineering and Management: the CommonKADS Methodology*: MIT Press.

Seaborne, A. (2004). *RDQL - A Query Language for RDF, W3C Member Submission 9 January 2004, HP Labs, Bristol*. Retrieved 29 August 2006, from http://www.w3.org/Submission/RDQL/

Selpi. (2004). *An FDM Prototype for Pathway and Protein Interaction Data.* International Master's programme in Bioinformatics thesis, Chalmers University of Technology, Goteborg, Sweden.

Serrano, D., & Gossard, D. (1992). Tools and Techniques for Conceptual Design. In C. Tong & D. Sriram (Eds.), *Artificial Intelligence in Engineering Design* (Vol. 1, pp. 71-116). San Diego, CA, USA: Academic Press Professional, Inc.

Shadbolt, N., Berners-Lee, T., & Hall, W. (2006). The Semantic Web Revisited. *IEEE Intelligent Systems,* 21(3):96-101.

Shimizu, S., & Numao, M. (1997). Constraint-based design for 3D shapes. *Artificial Intelligence,* 91:51-69.

Shin, H. Y., & Lee, J. W. (1998). Expert system for pneumatic design. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing,* 12(1):3-11.

Shipman, D. W. (1981). The Functional Data Model and the Data Language DAPLEX. *ACM Transactions Database Systems,* 6(1):140-173.

Shortliffe, E. H. (1981). *Computer-Based Medical Consultations:MYCIN*. New York: Elsevier.

Shortliffe, E. H., Scott, A. C., Bischoff, M. B., Melle, W. V., & Jacobs, C. D. (1981). ONCOCIN: An expert system for oncology protocol management. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pages 876-881, Vancouver, British Columbia.

Shum, S. B. (1991). *A Cognitive Analysis of Design Rationale Representation.* [Awarded the 1992 Kathleen Stott Proze for Best Doctoral Thesis] PhD thesis, University of York, York, UK.

Shum, S. B., & Hammond, N. (1994). Argumentation-Based Design Rationale: What Use at What Cost? *International Journal of Human-Computer Studies,* 40(4):603-652.

Sleeman, D., Ajit, S., Fowler, D. W., & Knott, D. (2008). The role of ontologies in creating and maintaining corporate knowledge: a case study from the aero industry. *Journal of Applied Ontology,* 3(3):151-172.

Sleeman, D., & Mitchell, F. (1996). Towards Painless Knowledge Acquisition. In *Proceedings of the Advances in Knowledge Acquisition*, pages 262-277, Berlin.

Smithers, T. (1998). KLDE - a knowledge level theories of design process. In J. Gero & F. Sudweeks (Eds.), *Artificial Intelligence in Design '98*. Dordrecht: Kluwer.

Soloway, E., Bachant, J., & Jensen, K. (1987). Assessing the Maintainability of XCON-in-RIME: Coping with Problems of a Very Large Rule-Base. In *Proceedings of the AAAI-87*, pages 824-829, Seattle, USA.

Sriram, D. (1997). *Intelligent systems for engineering: A knowledge based approach*.

Stokes, M. (Ed.). (2001). *Managing Engineering Knowledge. MOKA: Methodology for Knowledge Based Engineering Applications*: Professional Engineering Publishing.

Streeter, T. (1980). *The Art of the Japanese Kite*. Tokyo: Charles E Tuttle Company Inc.

Studer, R., Benjamins, V. R., & Fensel, D. (1998). Knowledge Engineering: Principles and Methods. *Data and Knowledge Engineering,* 25(1-2):161-197.

Sudman, S., & Bradburn, N. (1982). *Asking Questions: A Practical Guide to Questionnaire Design*. London: Jossey-Bass Inc.

Suwa, M., Scott, A. C., & Shortliffe, E. H. (1982). An Approach to Verifying Completeness and Consistency in a Rule-based System. *AI Magazine,* 3(4):16-21.

Thorton, A. C. (1996). A support tool for constraint processes in embodiment design. In *Proceedings of the ASME Design Theory and Methodology Conference*, pages 231-239, Minneapolis, USA.

Tian, L., Chen, J., Wang, Q., Hao, W., & Tong, B. (2007). CoDesign Space: a collaborative design support system in a network environment. *International Journal of Computer Integrated Manufacturing,* 20(2-3):265-279.

Tommelein, I. D., Leit, R. E., Hayes-Roth, B., & Confrey, T. (1991). Sight-Plan experiments: Alternate Strategies for site layout design. *ASCE Journal of Computing in Civil Engineering,* 5(1):42-63.

Tongco, M. D. C. (2007). Purposive sampling as a tool for informant selection. *Ethnobotany Research & Applications: A Journal for Plants, People and Applied Research,* 5:147-158.

Tsai, W.-T., Vishnuvajjala, R., & Zhang, D. (1999). Verification and Validation of Knowledge-Based Systems. *IEEE Transactions on Knowledge and Data Engineering,* 11(1):202-212.

Ullman, D. G. (2003). *The Mechanical Design Process*. New York: McGraw-Hill.

Valarakos, A., Paliouras, G., Karkaletsis, V., & Vouros, G. (2004). Enhancing ontological knowledge through ontology population and enrichment. In *Proceedings of the EKAW 2004*, pages 144-156, Northampton, UK.

Veeke, H. P. M., Lodewijks, G., & Ottjes, J. A. (2006). Conceptual design of industrial systems: an approach to support collaboration. *Research in Engineering Design,* 17:85-101.

Wallace, K. M., & Ahmed, S. (2003). How engineering designers obtain information. *Human behaviour in design.Individuals, teams, tools*:184-194.

Wardley, A. (2006). *Basics of Stunt Kite Design*. Retrieved 28 June, 2006, from http://www.kfs.org/~abw/kite/rec.kites/skdesign1.html

White, S. (2000). *Enhancing knowledge Acquisition with Constraint Technology.* PhD thesis, University of Aberdeen, Aberdeen.

White, S., & Sleeman, D. (2001). A Grammar-Driven Knowledge Acquisition Tool that incorporates Constraint Propagation. In *Proceedings of the KCAP-01 Conference*, pages 187-193, Canada.

Wielinga, B., & Schreiber, G. (1997). Configuration-Design Problem Solving. *IEEE Expert,* 12(2):49-57.

Winner, R. I., Pennel, J. P., Bertrand, E. H., & Slusarczuk, M. (1988). *The Role of Concurrent Engineering in Weapons System Acquisition*: VA: Institute for Defense Analyses.

Winter, M., Sleeman, D., & Parsons, T. (1998). Inventory Management using constraint satisfaction and knowledge refinement techniques. *Knowledge Based Systems*(11):293-300.

Wong, S. C., Crowder, R. M., Wills, G. B., & Shadbolt, N. R. (2008). Knowledge Transfer: From Maintenance to Engine Design. *Journal of Computing and Information Science in Engineering (Transactions of the ASME),* 8(1).

Wu, S., Ghenniwa, H., Zhang, Y., & Shen, W. (2006). Personal assistant agents for collaborative design environments. *Computers in Industry,* 57:732-739.
Yao, Z. (1996). *Constraint Management for Engineering Design.* PhD thesis, Cambridge University, Cambridge.
Yolen, W. (1976). *The Complete Book of Kites and Kite Flying*. New York: Simon and Schuster Trade.
Zhang, S., Shen, W., & Ghenniwa, H. (2004). A review of Internet-based product information sharing and visualisation. *Computers in Industry,* 54:1-15.

Appendix A

Equations, Constraints and Application Conditions in Kite Design

1. Like an aircraft, kites are **heavier than air** and rely on aerodynamic forces to fly. The relevant **aerodynamic equations** for calculating lift coefficient (C_l) and drag coefficients (Cd) are:

$$C_l = \frac{L}{0.5A\rho_{air}V^2}$$

$$C_d = \frac{D}{0.5A\rho_{air}V^2}$$

where L = lift in Newton (N), D = drag in Newton (N), A = projected surface area in square metres (m^2), ρ_{air} = air density in kilograms per cubic metre (kg/m^3), V = wind velocity over the surface in metre per second (m/s).

Notes: Now, the same set of equations **cannot usually be used** for flying objects such as **gas balloons and bubbles**, which are lighter than air and rely on **buoyancy forces** to fly, unless they are designed to have lift characteristics to improve control and performance. In these circumstances, buoyancy equations have to be used. The equation is also not applicable to things like bricks, i.e. things that do not have significant lifting surfaces.

Source: http://www.grc.nasa.gov/WWW/K-12/airplane/kite1.html

2. The **lift coefficient** used by aerodynamicists to model the complex dependencies of shape, inclination, and some flow conditions on lift is given by:

$$C_l = \frac{L}{0.5A\rho_{air}V^2}$$

where C_l = lift coefficient, L = lift in Newton (N), A = projected surface area in square metres (m^2), ρ_{air} = air density, V = wind velocity over the surface in metre per second (m/s).

Notes: This is applicable **for very low speeds (<321.87 Km^{-hr})**, when the compressibility effects of air are negligible. However, **at higher speeds it is incorrect** to use this equation. [The compressibility of air will alter the physics]. It has to be noted here that the speed of 321.87 Km^{-hr} is low in the field of aerodynamics but is high for the kite flying domain. The above equation and notes are stated mainly to illustrate another type of application condition.

Source: http://www.grc.nasa.gov/WWW/K-12/airplane/liftco.html

3. The **weight of the kite**, **W**, is equal to the weight of the surfaces in kilograms (kg), $\mathbf{W_s}$, plus the weight of the frame in kilograms (kg), $\mathbf{W_f}$.

$$W = W_s + W_f$$

Notes: It is assumed that **all the materials are subjected to the same gravitational acceleration** and **so weight is used**. Otherwise, mass has to be used in aerodynamic equations.

Source: http://www.grc.nasa.gov/WWW/K-12/airplane/kitewt.html

4. The **weight of the surface of kite in kilograms (kg), $\mathbf{W_s}$**, is the product of the surface material density in kilograms per square metre (kg/m^2), $\mathbf{d_s}$, and the surface area in square metres (m^2), $\mathbf{A_s}$.

$$W_s = d_s A_s$$

Notes: Area is being used here because **one is dealing with very thin coverings that are nearly the same for all materials**. Otherwise, **volume** has to be used.

Source: http://www.grc.nasa.gov/WWW/K-12/airplane/kitewt.html

5. The **weight of the frame, W_f,** is the product of the frame material density in kilograms per metre (kg/m), **d_f**, and the length of the frame in metre (m), **L_f**.

$$W_f = d_f L_f$$

Notes: The frame material is characterised by "density", a weight per unit length of the material. Notice that this definition of "density" is different from the standard mass divided by volume and from the surface definition of weight divided by area. This definition is used because it is **dealing with long thin sticks in the frame and the material is uniform along the length of the sticks.**

Source: http://www.grc.nasa.gov/WWW/K-12/airplane/kitewt.html

6. **The drag D** in Newton (N) is the product of a drag coefficient **C_d,** the projected surface area **A** in square metres (m^2)**,** the air density ρ_{air}= 1.229 kg/m^3, and one-half the square of the wind velocity over the surface **V** in metre per second (m/s).

$$C_l = \frac{1.229 C_d A V^2}{2}$$

Note 1: The drag depends on two properties of the air; the density and velocity. In general, the density depends on your location on the earth. The higher the elevation, the lower the density. The standard value for air density ρ_{air} used above (i.e., 1.229 kg/m^3) is **at sea level conditions**. The air velocity is the relative speed between the kite and the air. **When the kite is held fixed by the control line**, the air velocity is the wind speed. **If the line breaks, or if you let out line**, the velocity is something less than the wind speed; **if you pull on the control line** the velocity is the wind speed plus the speed of your pull.

Note 2: It has to be noted that the area (A) used in this equation is a **reference area. It is important to clearly specify the area used**. The computed value of the drag coefficient will vary according to the type of area used. For example, if one chooses the wing area, rather than the cross-sectional area, the computed coefficient will have a different value.

Source: http://www.grc.nasa.gov/WWW/K-12/airplane/kitedrag.html

7. Kite designs have constraints concerning the **requirement of tails**.

Notes: Tails are required **only for flat kites**, but not for example, for bowed kites. It is important to make this context explicit.

Source: http://www.aka.org.au/kites_in_the_classroom/chap3.htm

8. A constraint requires **the length of the tail of the kite needs to be 7 times the length of the spine**.

Notes: Again, this is applicable **only** to **flat diamond kites**.

Source: http://www.aka.org.au/kites_in_the_classroom/chap3.htm

9. A constraint requires the **density of sail material of the kite to be greater than 21.9 kilograms per square metre**.

Notes: This constraint is applicable only when there is a requirement to produce **low cost kites for beginners**. [Kites for experts have lighter materials, which are of higher quality and hence costlier]

Source: http://www.cuttingedgekites.com/faq.htm

10. **Aspect ratio is the wing span divided by the chord length**.

Notes: Applicable **only** to kites of **simple shapes** with **rectangular wings**.

Source: http://users.techline.com/lord/musing.html

11. A constraint requires the kites to be made from **lightweight non-porous** spinnaker fabric.

Notes: Applicable only to **light wind delta kites**.

Source: http://www.deltas.freeserve.co.uk/whatsadelta.html

12. The two wing halves of kites should be exactly **symmetrical**.

Notes: This need for strict symmetry is applicable **only to ordinary delta kites**.

Source: http://www.deltas.freeserve.co.uk/whatsadelta.html

13. The **ratio** between the **height and width of a kite should be five**.

Notes: This is applicable **only** to **basic 2-stick flat kites**.

Source: “The complete book of kites and kite flying” by Will Yolen.

14. The **bridle attachment angle** for a kite has to be close to **100 degrees** (for good performance).

Notes: This is applicable **only** to **airfoil kites**.

Source: “The complete book of kites and kite flying” by Will Yolen.

15. For a **wind speed** of 8 to 10 miles per hour, the kite’s **lifting surface area** has to be about 120 square feet.

Notes: This is applicable **only** to **Hargrave weather kites**.

Source: “The complete book of kites and kite flying” by Will Yolen.

16. The **edges of cloth kites need reinforcing** to prevent rips or unravelling.

Notes: **New Nylon Ripstop** cloth kites are an **exception**.

Source: “The complete book of kites and kite flying” by Will Yolen.

17. A constraint requires the **spine** and **cross-spar** of a kite to be **tapered** such that the weight is gradually decreased from the top descending to the bottom edge.

Notes: Applicable **only** to **Japanese kites**.

Source: “The art of the Japanese kite” by Tal Streeter.

18. The **bridling point** has to be **positioned one-third** of the way down from the top of the kite.

Notes: Applicable **only** to **Japanese kites**.

Source: “The art of the Japanese kite” by Tal Streeter.

19. The **strength** of the **kite line** should be **three times**, in pounds pull, the square feet of the **kite’s surface**.

Notes: Applicable **only** to **Japanese kites**.

Source: “The art of the Japanese kite” by Tal Streeter.

20. The **faces** and **wings** of a box kite should be the **same distance** apart from top to bottom.

Notes: Applicable **only** to **Japanese kites**.

Source: “The art of the Japanese kite” by Tal Streeter.

21. For the kite’s bridle, the **dynamic line has to be as long as the leading edge and** a **static line has to be 50-60% of the leading edge's length**.

Notes: This is applicable **only at the start of flying a kite**.

Source: http://www.idemployee.id.tue.nl/p.j.f.peters/kites/index.html

22. A kite must have a **tight sail, curved leading edge, outboard, low set bridle and outboard stand-offs.**

Notes: Applicable **only** to **radical trick kites**.

Source: http://www.kfs.org/~abw/kite/

Appendix B

Evaluation of ConEditor+ - Questionnaire

Please reply to as many of the questions below as possible using the scale **[1(poor)-5(excellent)]** wherever applicable:

Q) Do you find the GUI of the system intuitive to use? Please rate it.
What, if anything, would you like to have changed?

Q) How easy was it for you to input constraints? Please rate it.
What, if anything, would you like to have changed?

Q) How easy was it for you to input application conditions together with the constraints?
Please rate it. What, if anything, would you like to have changed?

Q) Did the system provide you with helpful suggestions to resolve/remove inconsistencies? What, if anything, would you like to have changed?

Q) Which mode would you prefer to use (1) semi-auto mode (2) auto (default) mode?

Q) How well does the system help you in the maintenance of constraints?
Please rate it.

Q) Would you like to see any additional features added?
Please specify details.

Q) Would you like to see any existing features removed/changed?
Please specify details.

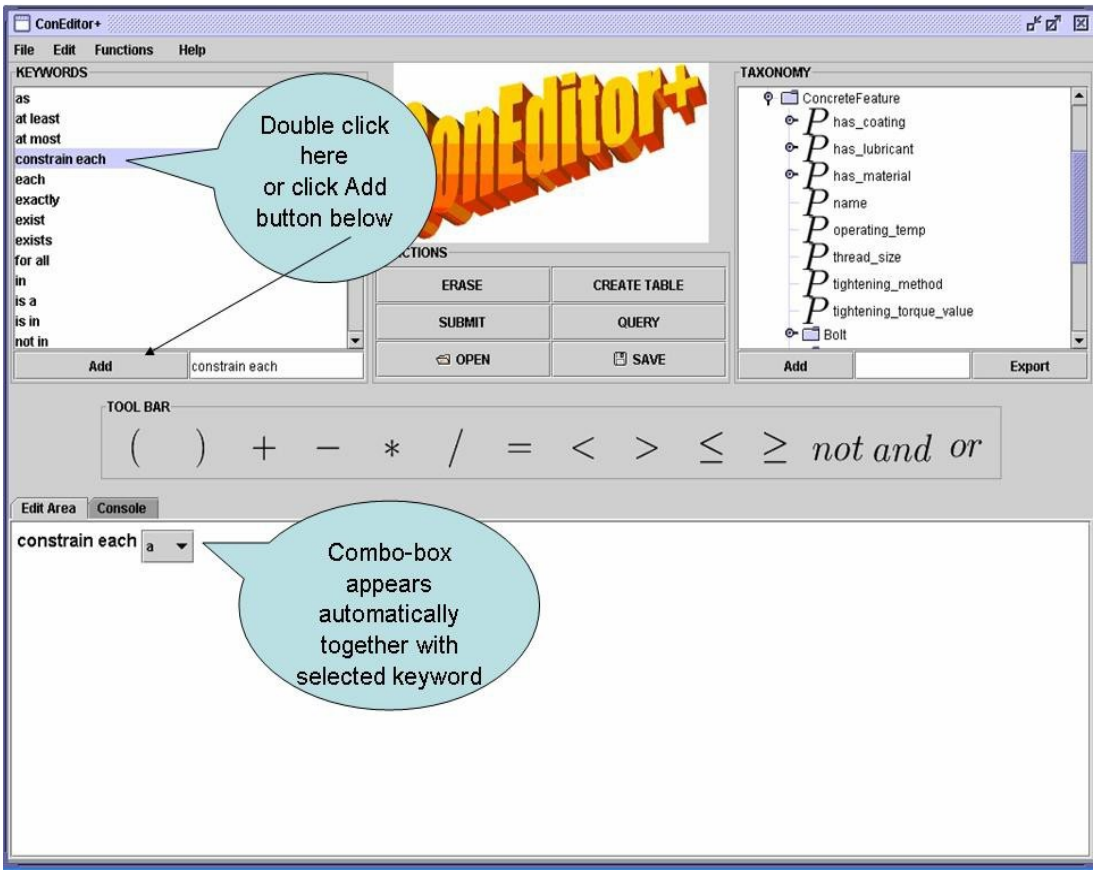
Q) Does the system provide you with new functionality? If so, which ones?
If not please specify which of your current system(s) provides this.

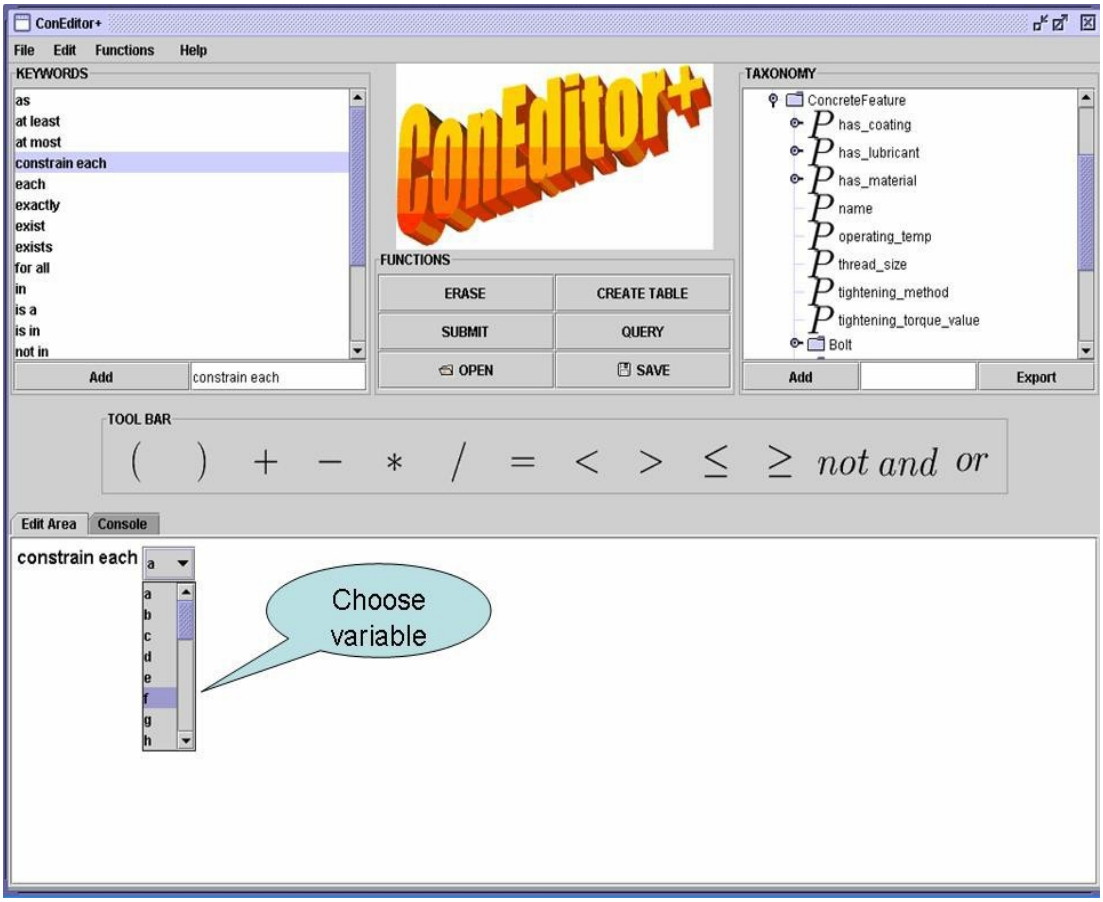
Q) Considering all the factors, could you give an overall rating of the system (ConEditor+: Acquisition and Maintenance of Constraints in Engineering Design)?

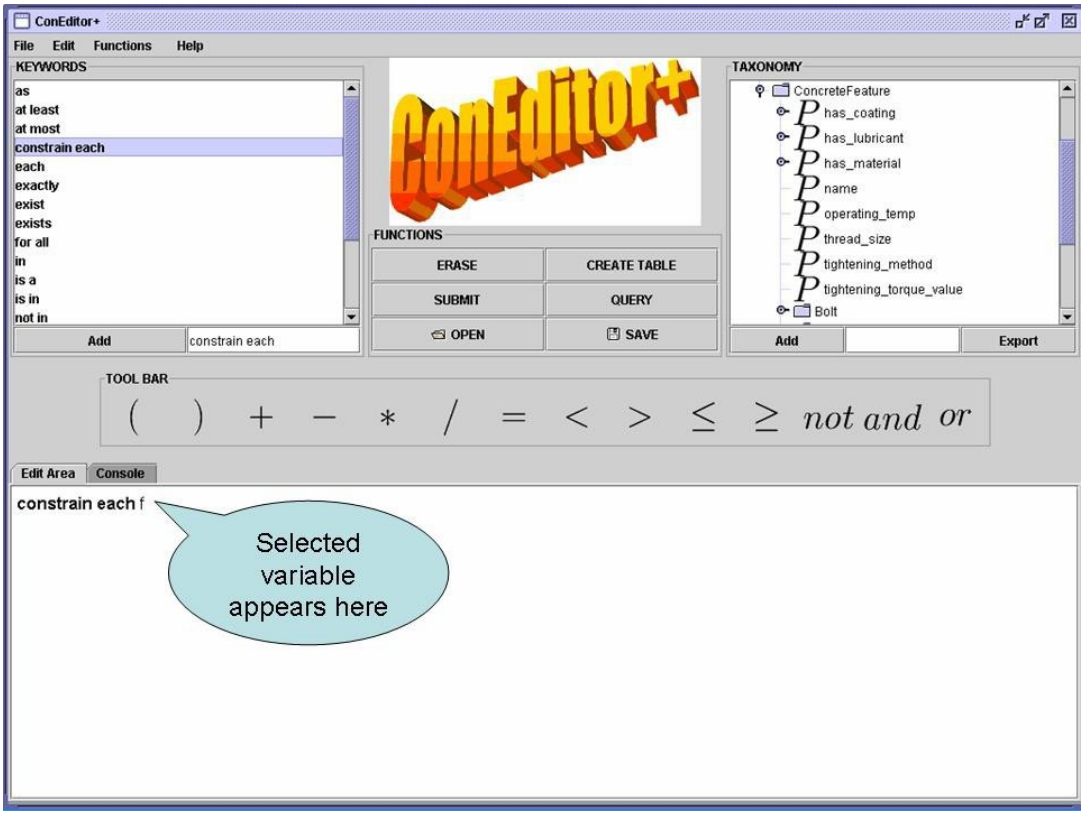
Q) Any other additional comments:

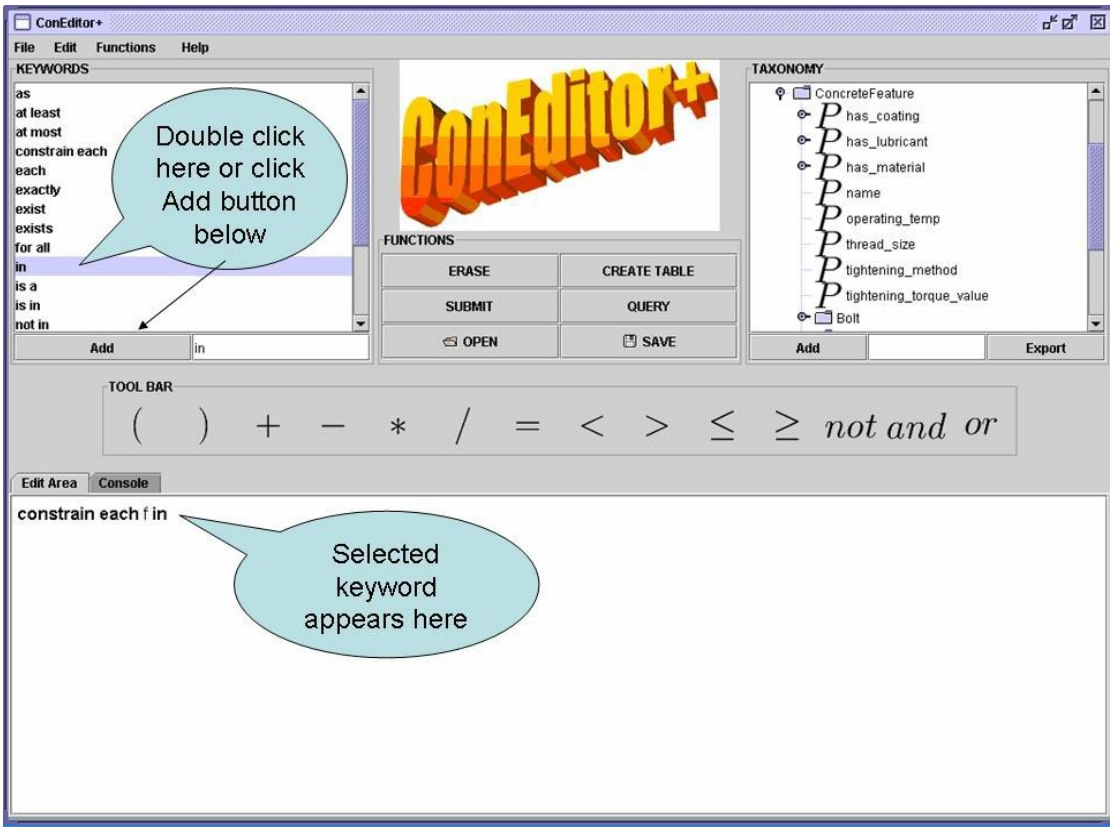
Appendix C

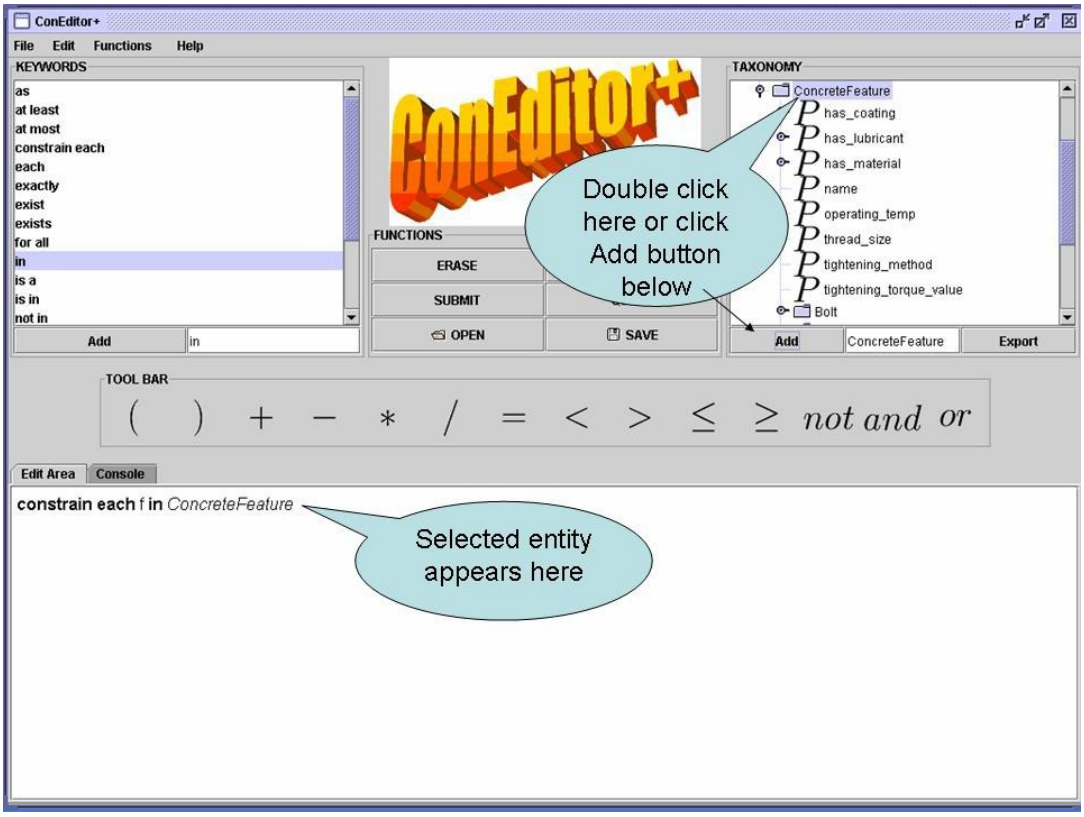
Annotated Walkthrough of capturing a constraint using ConEditor+

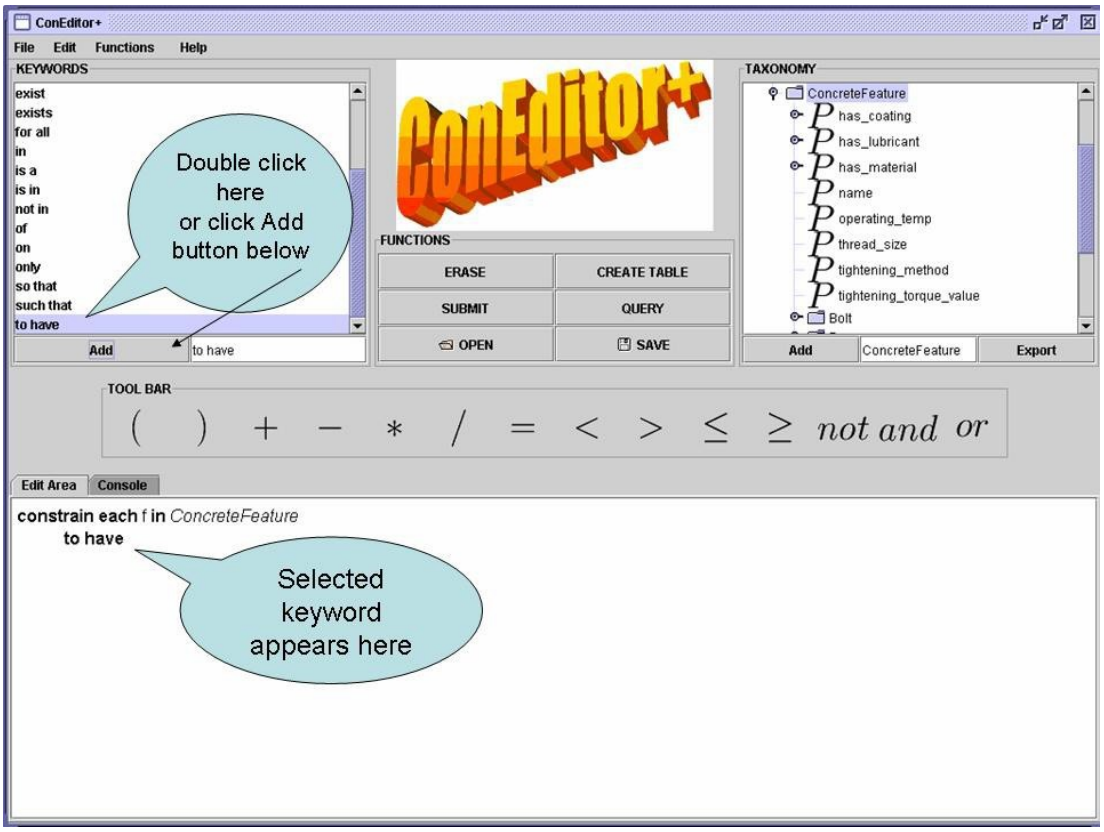


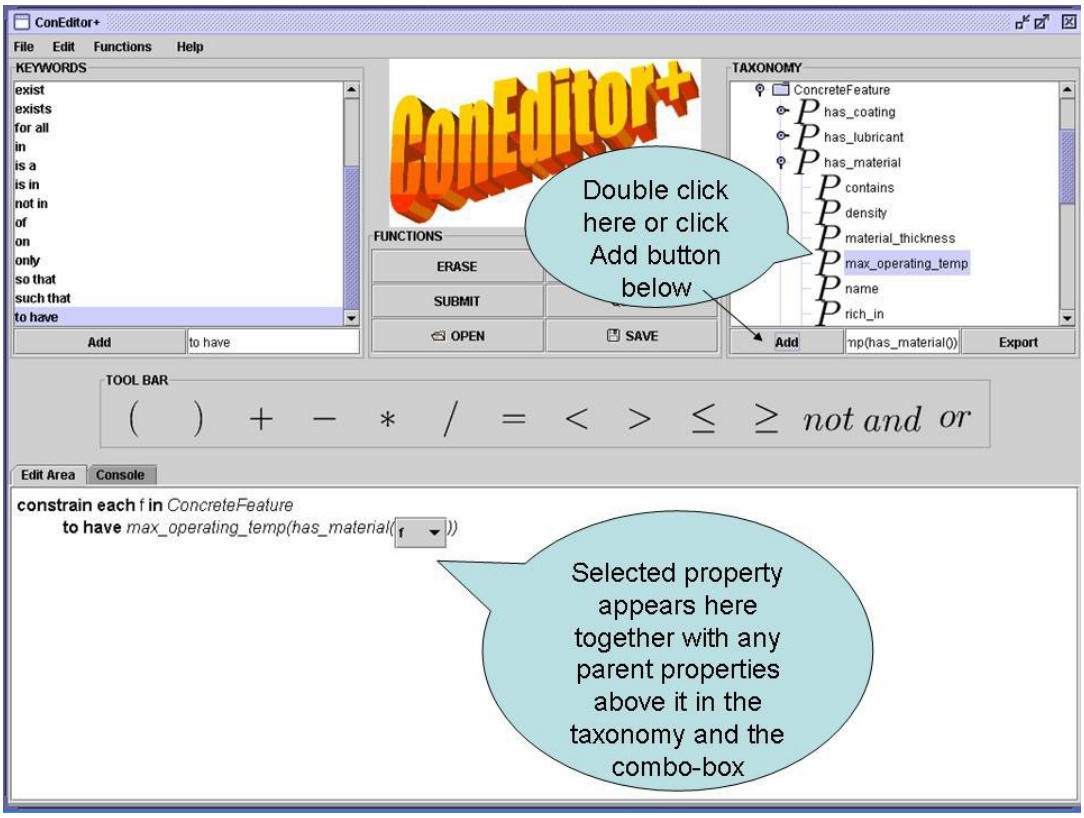

ConEditor+
File Edit Functions Help
KEYWORDS
as
at least
at most
constrain each
each
exactly
exist
exists
for all
in
is a
is in
not in
Add
constrain each
FUNCTIONS
ERASE
CREATE TABLE
SUBMIT
QUERY
OPEN
SAVE
TAXONOMY
ConcreteFeature
has_coating
has_lubricant
has_material
name
operating_temp
thread_size
tightening_method
tightening_torque_value
Bolt
Add
Export
TOOL BAR
( ) + − * / = < > ≤ ≥ not and or
Edit Area
Console
constrain each
a
b
c
d
e
f
g
h
Choose variable

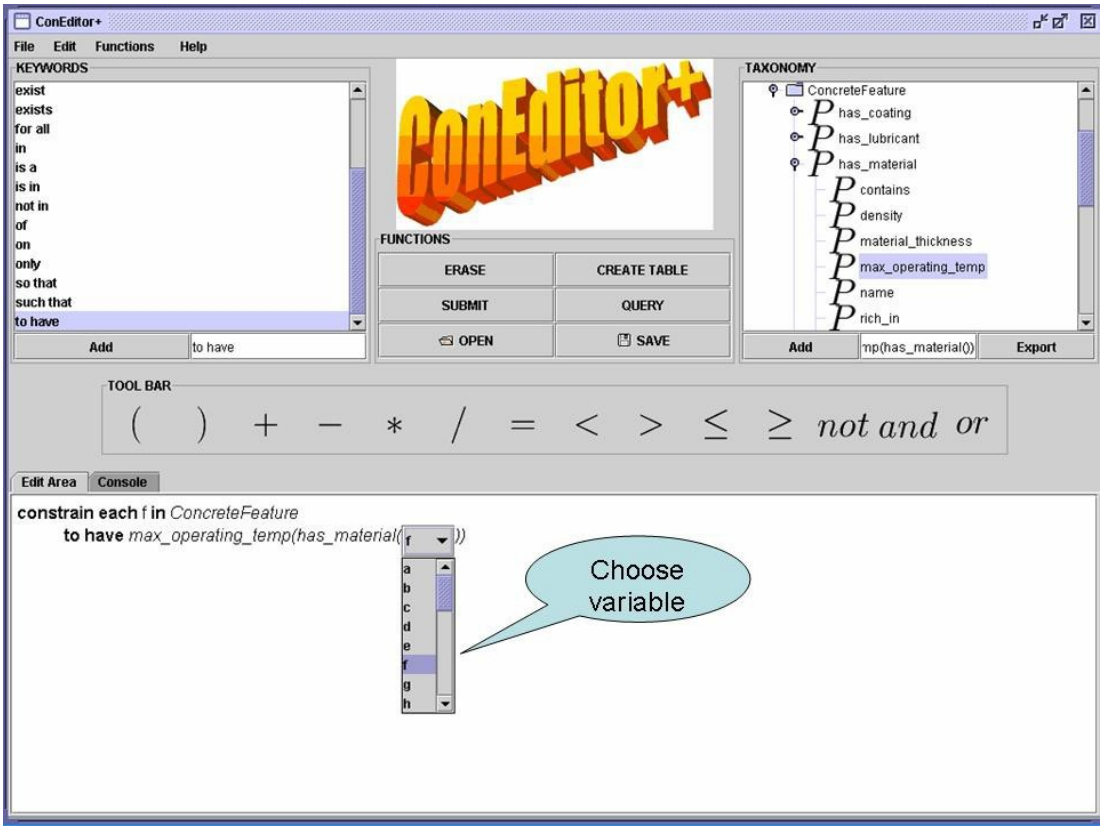

ConEditor+
File Edit Functions Help
KEYWORDS
as
at least
at most
constrain each
each
exactly
exist
exists
for all
in
is a
is in
not in
Add
constrain each
FUNCTIONS
ERASE
CREATE TABLE
SUBMIT
QUERY
OPEN
SAVE
TAXONOMY
ConcreteFeature
has_coating
has_lubricant
has_material
name
operating_temp
thread_size
tightening_method
tightening_torque_value
Bolt
Add
Export
TOOL BAR
( ) + − * / = < > ≤ ≥ not and or
Edit Area
Console
constrain each f
Selected variable appears here

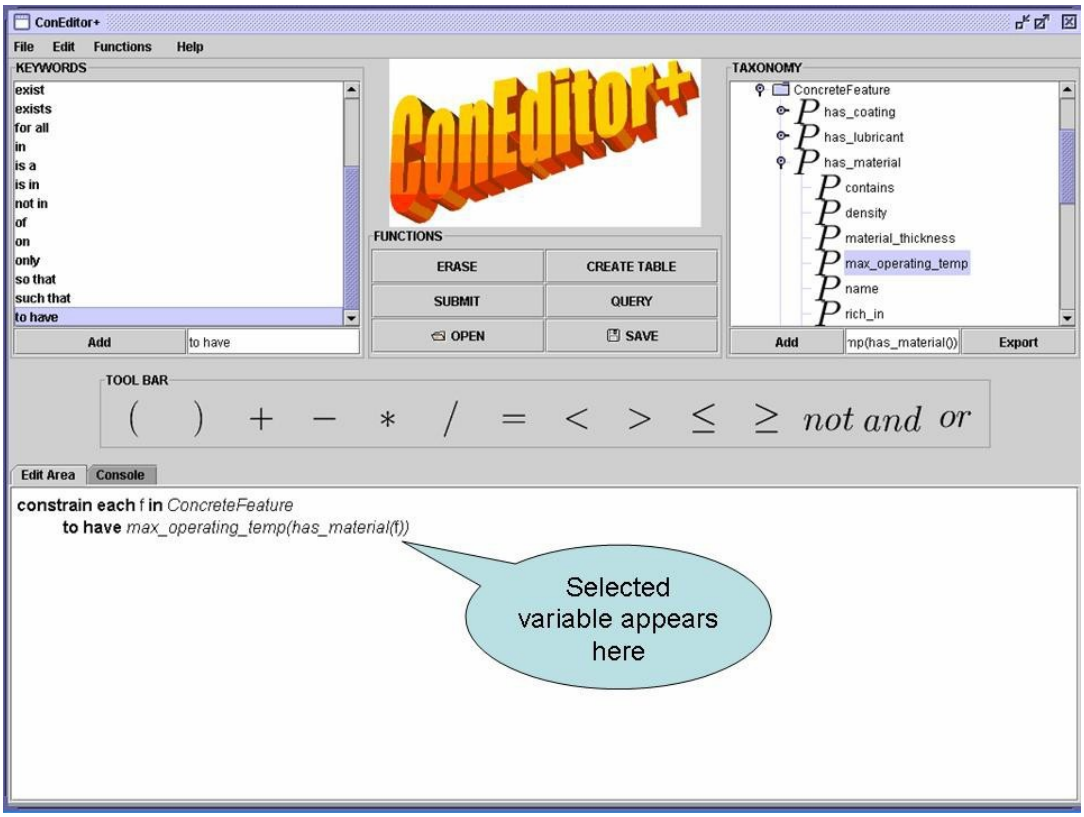

ConEditor+
File Edit Functions Help
KEYWORDS
as
at least
at most
constrain each
each
exactly
exist
exists
for all
in
is a
is in
not in
Add
in
Double click here or click Add button below
FUNCTIONS
ERASE
CREATE TABLE
SUBMIT
QUERY
OPEN
SAVE
TAXONOMY
ConcreteFeature
has_coating
has_lubricant
has_material
name
operating_temp
thread_size
tightening_method
tightening_torque_value
Bolt
Add
Export
TOOL BAR
( ) + − * / = < > ≤ ≥ not and or
Edit Area
Console
constrain each f in
Selected keyword appears here

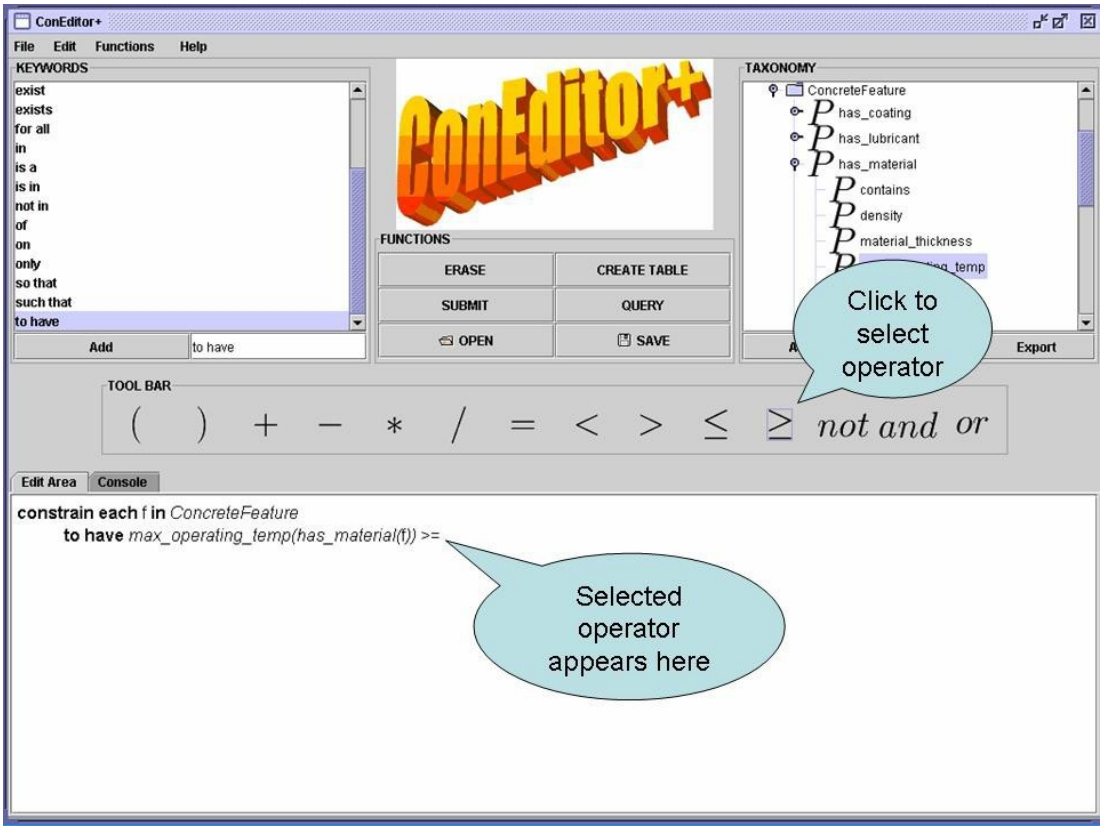

ConEditor+
File Edit Functions Help
KEYWORDS
as
at least
at most
constrain each
each
exactly
exist
exists
for all
in
is a
is in
not in
Add
in
FUNCTIONS
ERASE
SUBMIT
OPEN
SAVE
Double click here or click Add button below
TAXONOMY
ConcreteFeature
has_coating
has_lubricant
has_material
name
operating_temp
thread_size
tightening_method
tightening_torque_value
Bolt
Add
ConcreteFeature
Export
TOOL BAR
( ) + − * / = < > ≤ ≥ not and or
Edit Area
Console
constrain each f in ConcreteFeature
Selected entity appears here

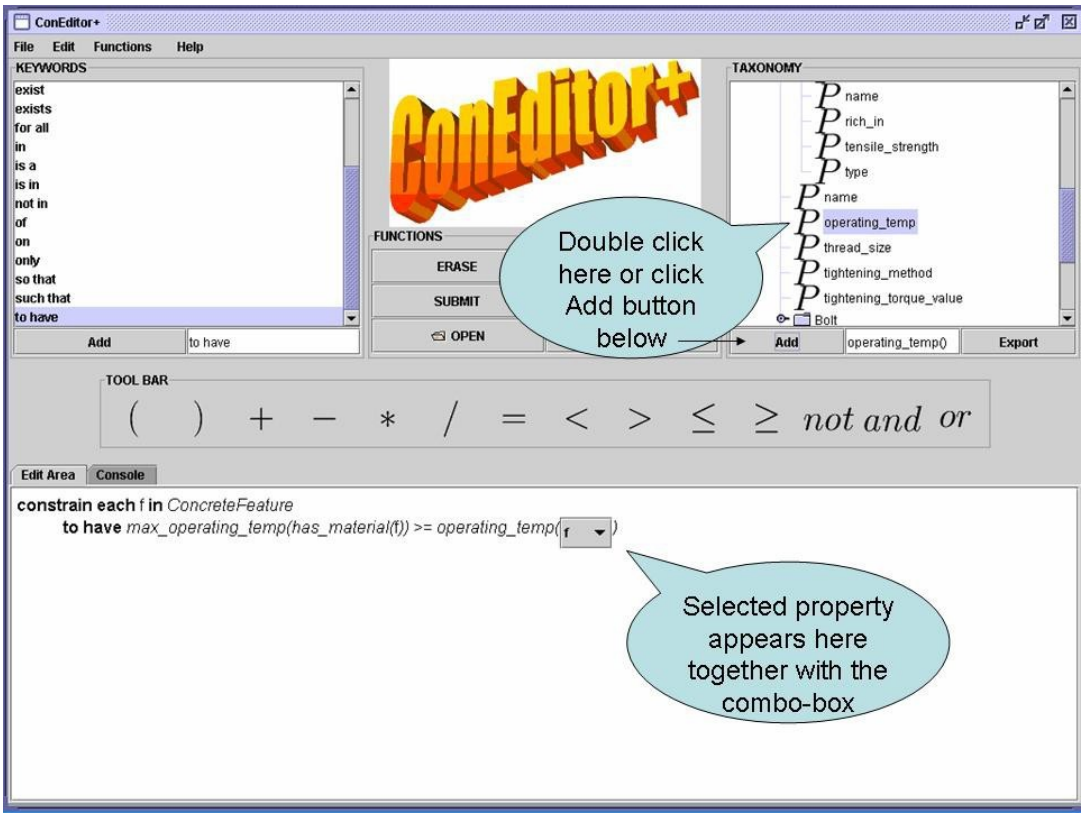

ConEditor+
File Edit Functions Help
KEYWORDS
exist
exists
for all
in
is a
is in
not in
of
on
only
so that
such that
to have
Add
to have
Double click here or click Add button below
FUNCTIONS
ERASE
CREATE TABLE
SUBMIT
QUERY
OPEN
SAVE
TAXONOMY
ConcreteFeature
has_coating
has_lubricant
has_material
name
operating_temp
thread_size
tightening_method
tightening_torque_value
Bolt
Add
ConcreteFeature
Export
TOOL BAR
( ) + − * / = < > ≤ ≥ not and or
Edit Area
Console
constrain each f in ConcreteFeature
to have
Selected keyword appears here

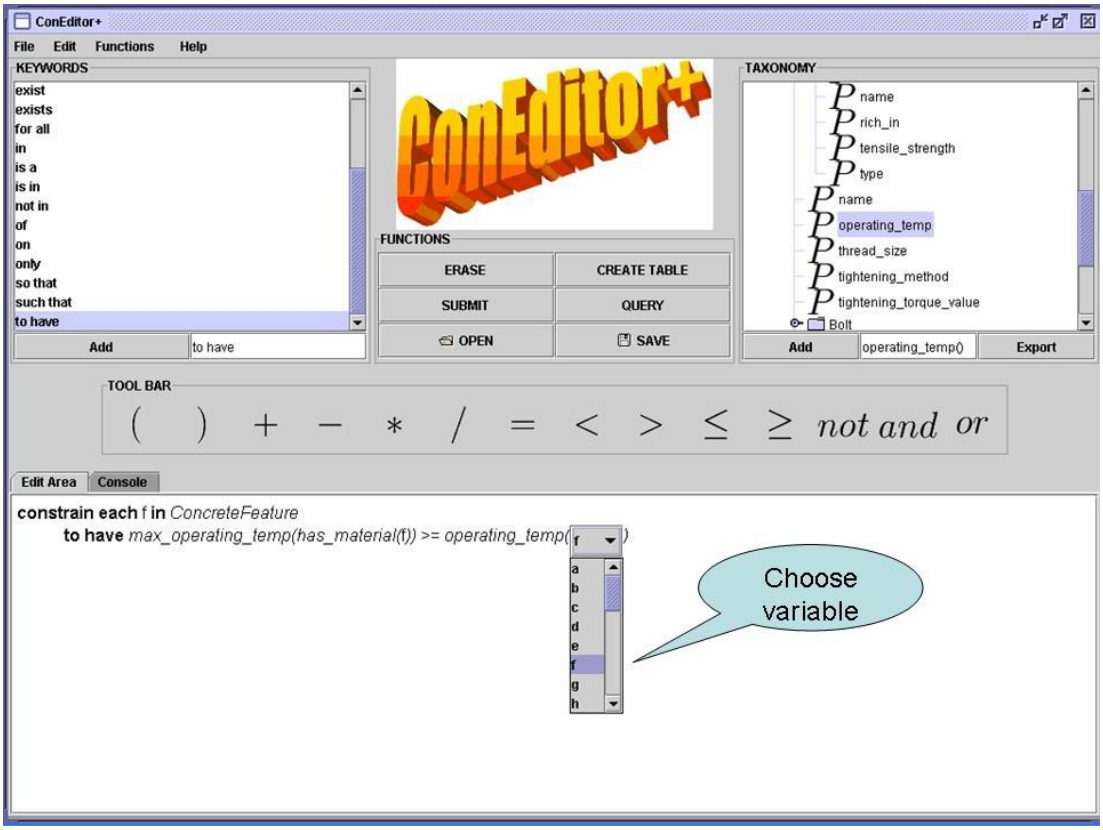

ConEditor+
File Edit Functions Help
KEYWORDS
exist
exists
for all
in
is a
is in
not in
of
on
only
so that
such that
to have
Add
to have
FUNCTIONS
ERASE
SUBMIT
OPEN
SAVE
TAXONOMY
ConcreteFeature
has_coating
has_lubricant
has_material
contains
density
material_thickness
max_operating_temp
name
rich_in
Add
mp(has_material())
Export
Double click here or click Add button below
TOOL BAR
( ) + − * / = < > ≤ ≥ not and or
Edit Area
Console
constrain each f in ConcreteFeature
to have max_operating_temp(has_material(f))
Selected property appears here together with any parent properties above it in the taxonomy and the combo-box

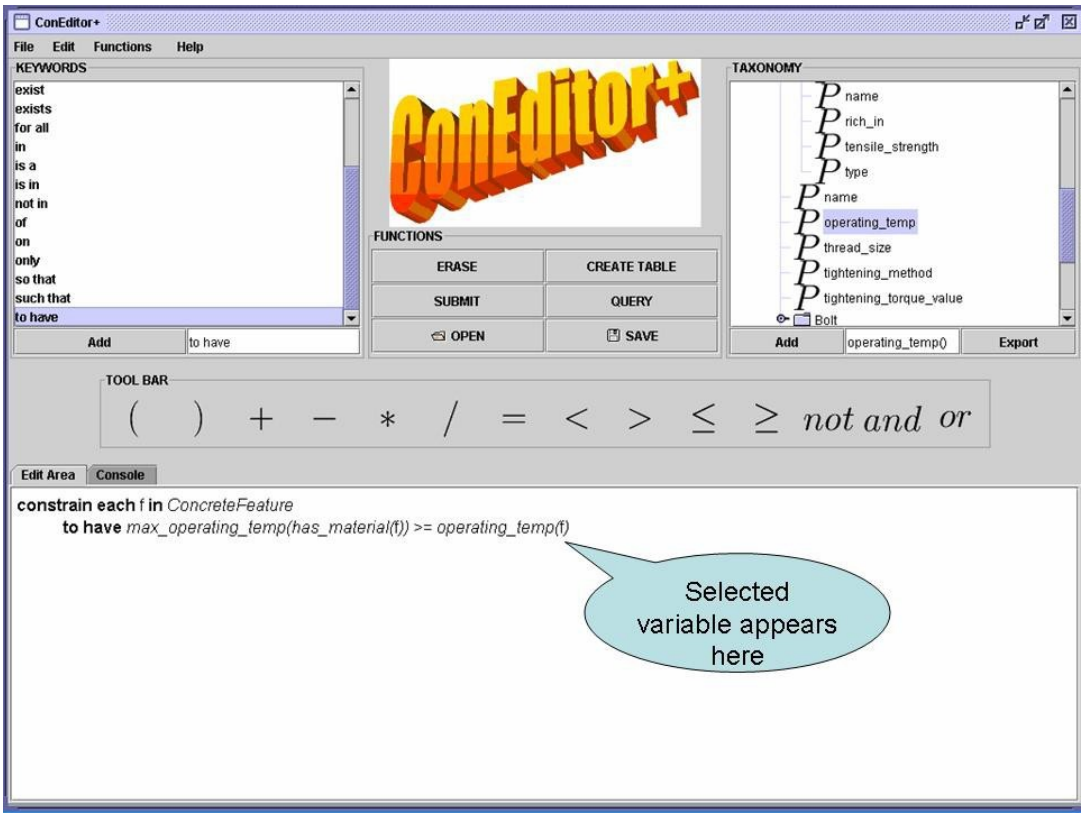

ConEditor+
File Edit Functions Help
KEYWORDS
exist
exists
for all
in
is a
is in
not in
of
on
only
so that
such that
to have
Add
to have
FUNCTIONS
ERASE
CREATE TABLE
SUBMIT
QUERY
OPEN
SAVE
TAXONOMY
ConcreteFeature
has_coating
has_lubricant
has_material
contains
density
material_thickness
max_operating_temp
name
rich_in
Add
mp(has_material())
Export
TOOL BAR
( ) + − * / = < > ≤ ≥ not and or
Edit Area
Console
constrain each f in ConcreteFeature
to have max_operating_temp(has_material(f))
a
b
c
d
e
f
g
h
Choose variable

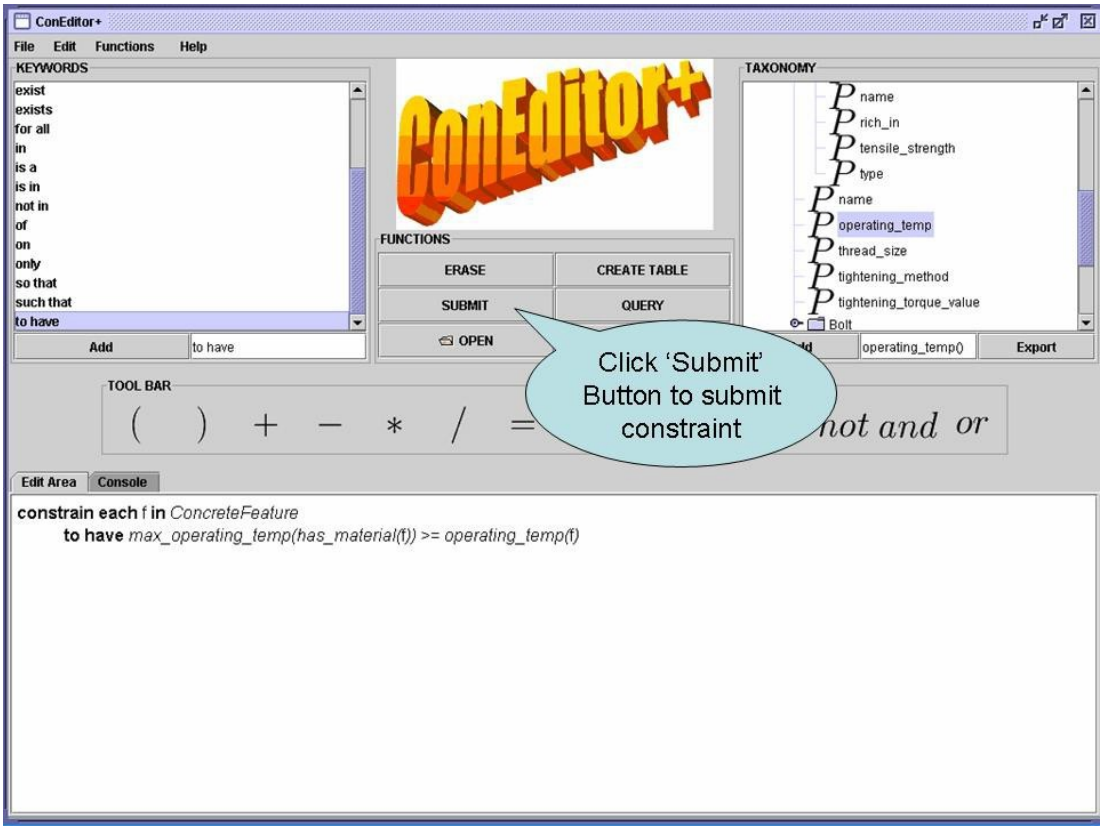

ConEditor+
File Edit Functions Help
KEYWORDS
exist
exists
for all
in
is a
is in
not in
of
on
only
so that
such that
to have
Add
to have
FUNCTIONS
ERASE
CREATE TABLE
SUBMIT
QUERY
OPEN
SAVE
TAXONOMY
ConcreteFeature
has_coating
has_lubricant
has_material
contains
density
material_thickness
max_operating_temp
name
rich_in
Add
mp(has_material())
Export
TOOL BAR
( ) + − * / = < > ≤ ≥ not and or
Edit Area
Console
constrain each f in ConcreteFeature
to have max_operating_temp(has_material(f))
Selected variable appears here

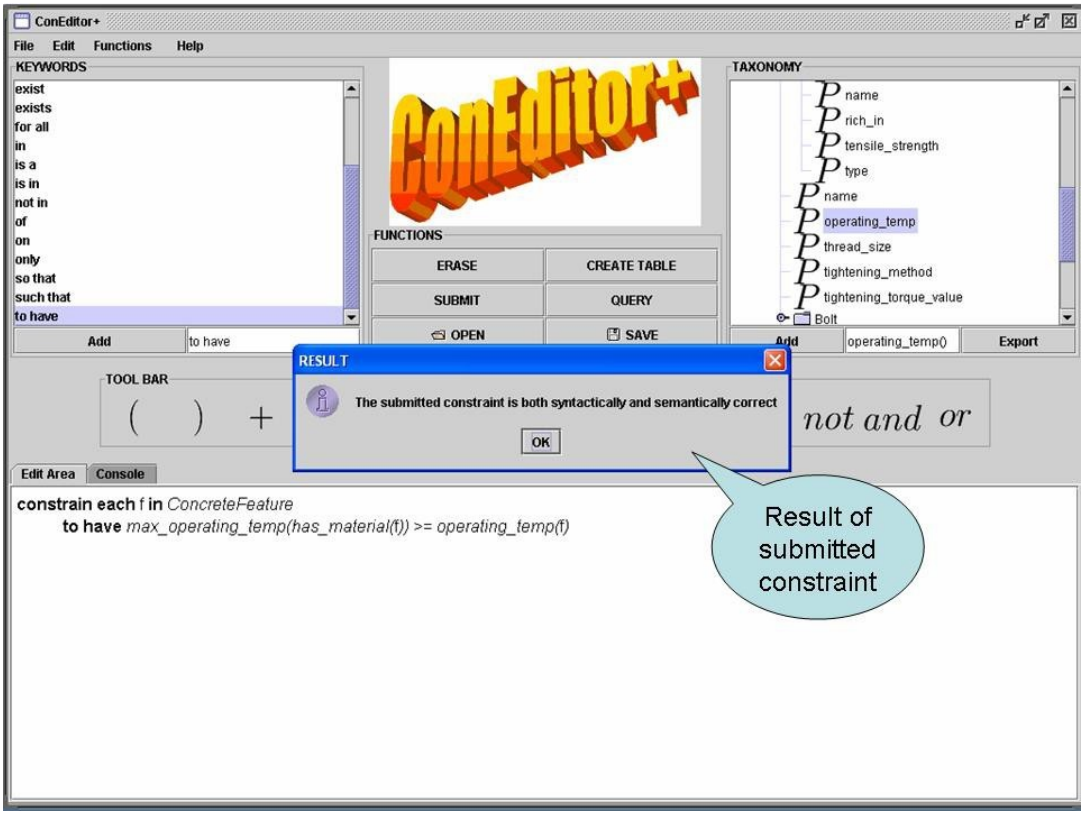

ConEditor+
File Edit Functions Help
KEYWORDS
exist
exists
for all
in
is a
is in
not in
of
on
only
so that
such that
to have
Add
to have
FUNCTIONS
ERASE
CREATE TABLE
SUBMIT
QUERY
OPEN
SAVE
TAXONOMY
ConcreteFeature
has_coating
has_lubricant
has_material
contains
density
material_thickness
Export
Click to select operator
TOOL BAR
( ) + − * / = < > ≤ ≥ not and or
Edit Area
Console
constrain each f in ConcreteFeature
to have max_operating_temp(has_material(f)) >=
Selected operator appears here

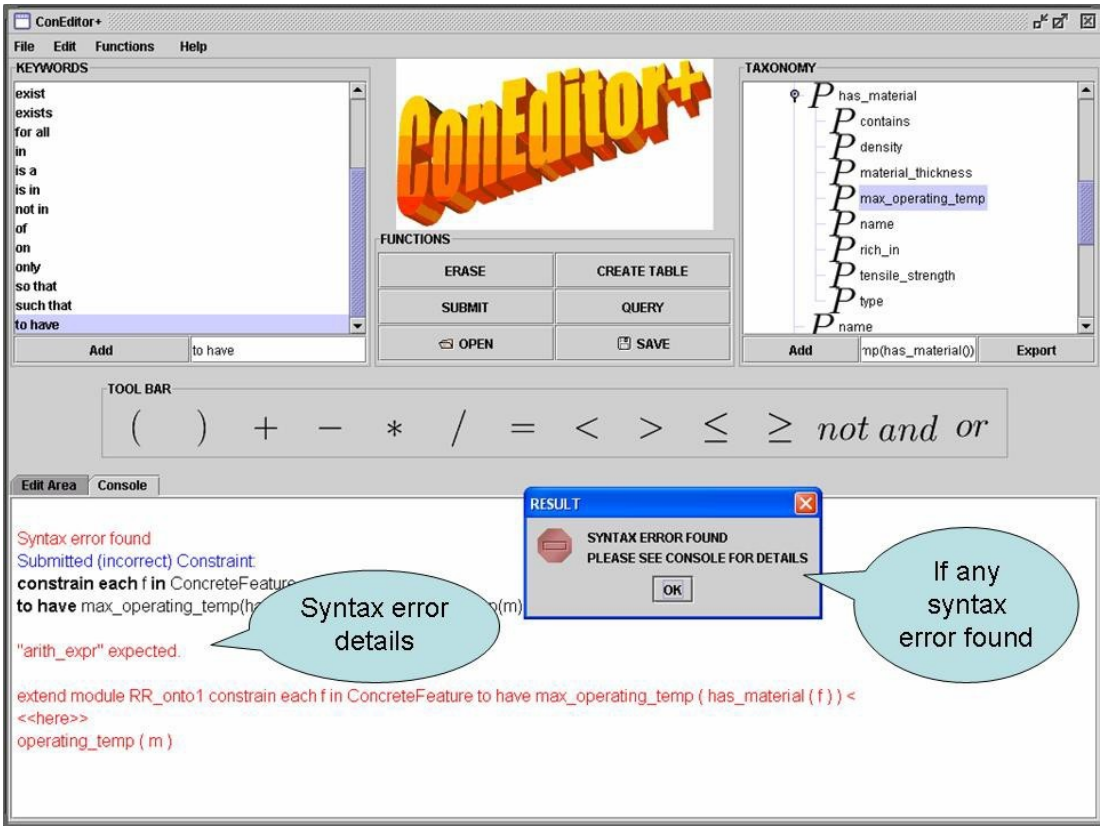

ConEditor+
File Edit Functions Help
KEYWORDS
exist
exists
for all
in
is a
is in
not in
of
on
only
so that
such that
to have
Add
to have
FUNCTIONS
ERASE
SUBMIT
OPEN
TAXONOMY
name
rich_in
tensile_strength
type
name
operating_temp
thread_size
tightening_method
tightening_torque_value
Bolt
Add
operating_temp()
Export
Double click here or click Add button below
TOOL BAR
( ) + − * / = < > ≤ ≥ not and or
Edit Area
Console
constrain each f in ConcreteFeature
to have max_operating_temp(has_material(f)) >= operating_temp(f)
Selected property appears here together with the combo-box

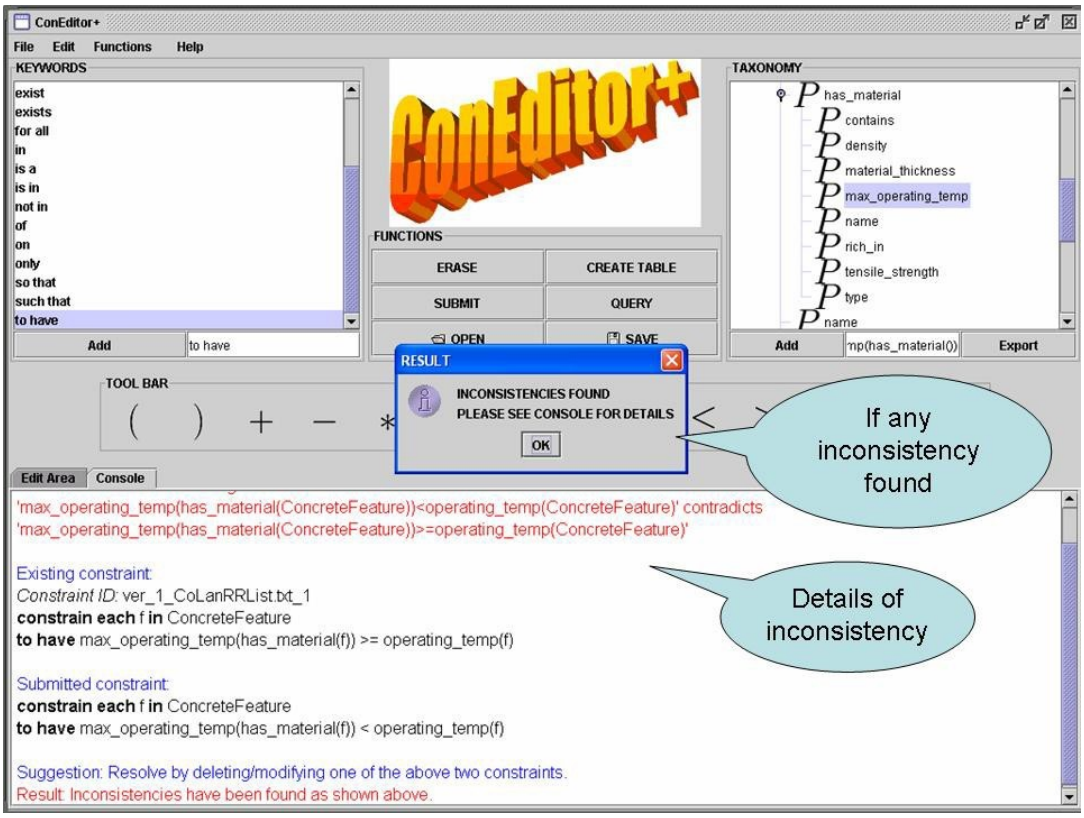

ConEditor+
File Edit Functions Help
KEYWORDS
exist
exists
for all
in
is a
is in
not in
of
on
only
so that
such that
to have
Add
to have
FUNCTIONS
ERASE
CREATE TABLE
SUBMIT
QUERY
OPEN
SAVE
TAXONOMY
name
rich_in
tensile_strength
type
name
operating_temp
thread_size
tightening_method
tightening_torque_value
Bolt
Add
operating_temp()
Export
TOOL BAR
( ) + − * / = < > ≤ ≥ not and or
Edit Area
Console
constrain each f in ConcreteFeature
to have max_operating_temp(has_material(f)) >= operating_temp(f)
a b c d e f g h
Choose variable


ConEditor+
File Edit Functions Help
KEYWORDS
exist
exists
for all
in
is a
is in
not in
of
on
only
so that
such that
to have
Add
to have
FUNCTIONS
ERASE
CREATE TABLE
SUBMIT
QUERY
OPEN
SAVE
TAXONOMY
name
rich_in
tensile_strength
type
name
operating_temp
thread_size
tightening_method
tightening_torque_value
Bolt
Add
operating_temp()
Export
TOOL BAR
( ) + − * / = < > ≤ ≥ not and or
Edit Area
Console
constrain each f in ConcreteFeature
to have max_operating_temp(has_material(f)) >= operating_temp(f)
Selected variable appears here


ConEditor+
File Edit Functions Help
KEYWORDS
exist
exists
for all
in
is a
is in
not in
of
on
only
so that
such that
to have
Add
to have
FUNCTIONS
ERASE
CREATE TABLE
SUBMIT
QUERY
OPEN
TAXONOMY
name
rich_in
tensile_strength
type
name
operating_temp
thread_size
tightening_method
tightening_torque_value
Bolt
operating_temp()
Export
Click 'Submit' Button to submit constraint
TOOL BAR
( ) + − * / = not and or
Edit Area
Console
constrain each f in ConcreteFeature
to have max_operating_temp(has_material(f)) >= operating_temp(f)


ConEditor+
File Edit Functions Help
KEYWORDS
exist
exists
for all
in
is a
is in
not in
of
on
only
so that
such that
to have
Add
to have
FUNCTIONS
ERASE
CREATE TABLE
SUBMIT
QUERY
OPEN
SAVE
TAXONOMY
name
rich_in
tensile_strength
type
name
operating_temp
thread_size
tightening_method
tightening_torque_value
Bolt
operating_temp()
Export
RESULT
The submitted constraint is both syntactically and semantically correct
OK
TOOL BAR
not and or
Edit Area
Console
constrain each f in ConcreteFeature
to have max_operating_temp(has_material(f)) >= operating_temp(f)
Result of submitted constraint


ConEditor+
File Edit Functions Help
KEYWORDS
exist
exists
for all
in
is a
is in
not in
of
on
only
so that
such that
to have
Add
to have
FUNCTIONS
ERASE
CREATE TABLE
SUBMIT
QUERY
OPEN
SAVE
TAXONOMY
has_material
contains
density
material_thickness
max_operating_temp
name
rich_in
tensile_strength
type
name
Add
mp(has_material())
Export
TOOL BAR
( ) + − * / = < > ≤ ≥ not and or
Edit Area
Console
RESULT
SYNTAX ERROR FOUND
PLEASE SEE CONSOLE FOR DETAILS
OK
If any syntax error found
Syntax error found
Submitted (incorrect) Constraint:
constrain each f in ConcreteFeature
to have max_operating_temp(h
(m)
"arith_expr" expected.
Syntax error details
extend module RR_onto1 constrain each f in ConcreteFeature to have max_operating_temp (has_material (f)) <
<<here>>
operating_temp (m)


ConEditor+
File Edit Functions Help
KEYWORDS
exist
exists
for all
in
is a
is in
not in
of
on
only
so that
such that
to have
Add
to have
FUNCTIONS
ERASE
CREATE TABLE
SUBMIT
QUERY
OPEN
SAVE
TAXONOMY
has_material
contains
density
material_thickness
max_operating_temp
name
rich_in
tensile_strength
type
name
Add
mp(has_material())
Export
TOOL BAR
RESULT
INCONSISTENCIES FOUND
PLEASE SEE CONSOLE FOR DETAILS
OK
If any inconsistency found
Edit Area
Console
'max_operating_temp(has_material(ConcreteFeature))<operating_temp(ConcreteFeature)' contradicts
'max_operating_temp(has_material(ConcreteFeature))>=operating_temp(ConcreteFeature)'
Existing constraint:
Constraint ID: ver_1_CoLanRRList.txt_1
constrain each f in ConcreteFeature
to have max_operating_temp(has_material(f)) >= operating_temp(f)
Details of inconsistency
Submitted constraint:
constrain each f in ConcreteFeature
to have max_operating_temp(has_material(f)) < operating_temp(f)
Suggestion: Resolve by deleting/modifying one of the above two constraints.
Result: Inconsistencies have been found as shown above.

Appendix D

Scanned copies of the Questionnaires answered by subjects during the Evaluation of ConEditor+

NAME : HIDDEN
POSITION: PhD STUDENT
DEPARTMENT: MECHANICAL ENGINEERING

Evaluation of ConEditor+

Please reply to as many of the questions below as possible using the scale **[1(poor)-5(excellent)]** wherever applicable:

Do you find the GUI of the system intuitive to use? Please rate it. What, if anything, would you like to have changed? 5

How easy was it for you to input constraints? Please rate it. What, if anything, would you like to have changed? 4

How easy was it for you to input application conditions along with the constraints? Please rate it. What, if anything, would you like to have changed? 4

Did the system provide you with helpful suggestions to resolve/remove inconsistencies? What, if anything, would you like to have changed?
Yes. 5

Which mode would you prefer to use (1) manual mode (2) automatic (default) mode?
(2) automatic

How well does the system help you in the maintenance of constraints? Please rate it.
4

Would you like to see any additional features added? Please specify details.
Command Search tool.

Would you like to see any existing features removed/changed? Please specify details.
No.

Does the system provide you with new functionality? If so, which ones? If not please specify which of your current system(s) provides this.

Yes, its a new functionality.

Considering all the factors, could you give an overall rating of the system (ConEditor: Acquisition and Maintenance of Constraints in Engineering Design)?

4.

Any other additional comments:

(mEng)

NAME: HIDDEN
POSITION: M.Eng STUDENT
DEPARTMENT: MECHANICAL ENGINEERING

Evaluation of ConEditor+

Please reply to as many of the questions below as possible using the scale **[1(poor)-5(excellent)]** wherever applicable:

Do you find the GUI of the system intuitive to use? Please rate it.
What, if anything, would you like to have changed?

How easy was it for you to input constraints? Please rate it. 4
What, if anything, would you like to have changed?

How easy was it for you to input application conditions along with the constraints? 3
Please rate it. What, if anything, would you like to have changed?

Did the system provide you with helpful suggestions to resolve/remove 4
inconsistencies? What, if anything, would you like to have changed?

Which mode would you prefer to use (1) manual mode (2) automatic (default) mode? (2)

How well does the system help you in the maintenance of constraints? Please rate it. (5)

Would you like to see any additional features added? Please specify details.
a user friendly interface, allowing someone who has no knowledge to be guided by using its constraints

Would you like to see any existing features removed/changed? Please specify details.

Does the system provide you with new functionality? If so, which ones? If not please specify which of your current system(s) provides this.

Considering all the factors, could you give an overall rating of the system (ConEditor: Acquisition and Maintenance of Constraints in Engineering Design)?

3.

Any other additional comments:

To learn a new language would be disadvantageous. for prospective coments.

A user friendly interface which guides a person through "Basic" Automatic reasoning driven constraints and an "Advanced" manual input constraints.

NAME : HIDDEN
POSITION : RESEARCH FELLOW
DEPARTMENT: COMPUTING SCIENCE

Evaluation of ConEditor+

Please reply to as many of the questions below as possible using the scale **[1(poor)-5(excellent)]** wherever applicable:

Do you find the GUI of the system intuitive to use? Please rate it.
What, if anything, would you like to have changed?
(4)

How easy was it for you to input constraints? Please rate it.
What, if anything, would you like to have changed?
(4) Quite easy. I would suggest a "double-click" on keywords to "Add" them to the constraint as well.

How easy was it for you to input application conditions along with the constraints?
Please rate it. What, if anything, would you like to have changed?
(3) I guess with a big taxonomy it will be a pain to find the appropriate components

Did the system provide you with helpful suggestions to resolve/remove inconsistencies? What, if anything, would you like to have changed?
(3) I would like the description of inconsistencies to be more detailed (e.g. highlighting where a subsumption occurs etc.)
Which mode would you prefer to use (1) manual mode (2) automatic (default) mode?
2. automatic

How well does the system help you in the maintenance of constraints? Please rate it.
(4) I think the consistency checking works OK

Would you like to see any additional features added? Please specify details.

—

Would you like to see any existing features removed/changed? Please specify details.

—

Does the system provide you with new functionality? If so, which ones? If not please specify which of your current system(s) provides this.

—

Considering all the factors, could you give an overall rating of the system (ConEditor: Acquisition and Maintenance of Constraints in Engineering Design)?

I've never used any such system in practice.

Any other additional comments:

NAME: HIDDEN
POSITION: PhD STUDENT
DEPARTMENT: COMPUTING SCIENCE

Evaluation of ConEditor+

Please reply to as many of the questions below as possible using the scale **[1(poor)-5(excellent)]** wherever applicable:

Do you find the GUI of the system intuitive to use? Please rate it. 3
What, if anything, would you like to have changed?
I'd like to see the constraints already in the ontologies.
Filter/Search on the classes & Properties of the ontology

How easy was it for you to input constraints? Please rate it. 4
What, if anything, would you like to have changed?

How easy was it for you to input application conditions along with the constraints? 4
Please rate it. What, if anything, would you like to have changed?

Did the system provide you with helpful suggestions to resolve/remove inconsistencies? What, if anything, would you like to have changed? 5
Maybe the problem in the pop-up or automatically jumps to the console.

Which mode would you prefer to use (1) manual mode (2) automatic (default) mode? (2)

How well does the system help you in the maintenance of constraints? Please rate it. 3
Works well, but I would like to see the existing constraints too.

Would you like to see any additional features added? Please specify details.

Would you like to see any existing features removed/changed? Please specify details.

Does the system provide you with new functionality? If so, which ones? If not please specify which of your current system(s) provides this.

Considering all the factors, could you give an overall rating of the system (ConEditor: Acquisition and Maintenance of Constraints in Engineering Design)?

Any other additional comments:

NAME: HIDDEN
POSITION: PhD STUDENT
DEPARTMENT: COMPUTING SCIENCE

Evaluation of ConEditor+

Please reply to as many of the questions below as possible using the scale **[1(poor)-5(excellent)]** wherever applicable:

Do you find the GUI of the system intuitive to use? Please rate it. 4
What, if anything, would you like to have changed?
I FOUND MYSELF DOUBLE CLICKING ON PROPERTIES WHERE SINGLE CLICK WAS CORRECT

How easy was it for you to input constraints? Please rate it. 5
What, if anything, would you like to have changed?
SEEMED INTUITIVE

How easy was it for you to input application conditions along with the constraints? Please rate it. What, if anything, would you like to have changed? 4

Did the system provide you with helpful suggestions to resolve/remove inconsistencies? What, if anything, would you like to have changed? 4
THERE WERE HELPFUL MESSAGES ON THE CONSOLE TAB.

Which mode would you prefer to use (1) manual mode (2) automatic (default) mode?
AUTOMATIC WOULD BE MY PREFERENCE.

How well does the system help you in the maintenance of constraints? Please rate it. 4

Would you like to see any additional features added? Please specify details.
—

Would you like to see any existing features removed/changed? Please specify details.
—

Does the system provide you with new functionality? If so, which ones? If not please specify which of your current system(s) provides this.

THE CONSTRAINT VALIDITY CHECK APPEARS VERY USEFUL.

Considering all the factors, could you give an overall rating of the system (ConEditor: Acquisition and Maintenance of Constraints in Engineering Design)?

4/5

Any other additional comments:

I WOULD LIKE TO DO FURTHER WORK WITH CONEDITOR LOOKING AT VT DOMAIN CONSTRAINTS

Appendix E

Sample Refinements of Constraints and Application Conditions by ConEditor+ in the Rolls-Royce domain

Redundancy:

(a) Duplication

(i) **constrain each** c **in** FlameDepositionCoating
such that has_fabricated_component(c)
and has_mask_location_level(c) = "difficult"
to have has_max_overspray_thickness(c) = 4.0

(ii) **constrain each** c **in** FlameDepositionCoating
such that has_fabricated_component(c)
and has_mask_location_level(c) = "difficult"
to have has_max_overspray_thickness(c) = 4.0

By comparing the two constraints above, one can infer that the constraint (i) is identical to (ii).

(b) Class Equivalence

(iii) **constrain each** c **in** DepositionCoating
such that has_fabricated_component(c)
and has_mask_location_level(c) = "difficult"
to have has_max_overspray_thickness(c) = 4.0

(iv) **constrain each** c **in** FlameDepositionCoating
such that has_fabricated_component(c)
and has_mask_location_level(c) = "difficult"
to have has_max_overspray_thickness(c) = 4.0

As *DepositionCoating* is an equivalent class to *FlameDepositionCoating* in the domain ontology one can infer that the constraint (iii) is equivalent to constraint (iv).

(c) Property Equivalence

(v) **constrain each** c **in** FlameDepositionCoating
such that has_fabricated_component(c)
and has_mask_location_level(c) = "difficult"
to have has_max_overspray_thickness(c) = 4.0

(vi) **constrain each** c **in** FlameDepositionCoating
such that has_fabricated_part(c)
and has_mask_location_level(c) = "difficult"
to have has_max_overspray_thickness(c) = 4.0

As *has_fabricated_component* is an equivalent property to *has_fabricated_part* in the domain ontology one can infer that the constraint v) is equivalent to constraint vi). ConEditor+ notifies the user (domain expert) of all occurrences of redundancy and suggests that the user takes appropriate action(s) to eliminate redundancy.

Subsumption:

(a) Subsumption via sub-class:

(vii) **constrain each** s **in** RingSeal
such that has_elastometric_toroidal_oring(s)
and name(has_material(has_sealing_ring(s))) <> "perfluorocarbon"
and pressure_type_hou_mat_flange(s) = "internal"
to have min_face_groove_dia(s) = max_face_groove_dia(s) - 0.25

(viii) **constrain each** s **in** FaceRingSeal
such that has_elastometric_toroidal_oring(s)
and name(has_material(has_sealing_ring(s))) <> "perfluorocarbon"
and pressure_type_hou_mat_flange(s) = "internal"
to have min_face_groove_dia(s) = max_face_groove_dia(s) - 0.25

As *FaceRingSeal* is a subclass of *RingSeal* in the domain ontology one can infer that the constraint (vii) subsumes constraint (viii). ConEditor+ notifies the user (domain expert) of this fact and suggests that the user considers removing or deactivating constraint (viii).

(b) Subsumption via application condition

(ix) **constrain each** s **in** Stud
such that has_stud_type(s) = "standard"
or has_stud_type(s) = "large"
to have min_stand_out(s) = min_length(s) - (max_cbore(s) +
max_pull_in(s))

(x) **constrain each** s **in** Stud
such that has_stud_type(s) = "standard"
to have min_stand_out(s) = min_length(s) - (max_cbore(s) +
max_pull_in(s))

By comparing the two constraints above, one can infer that the constraint (ix) subsumes constraint (x). ConEditor+ notifies the user (domain expert) of this fact and suggests that the user considers removing or deactivating constraint (x).

(c) Subsumption via conjunction

(xi) **constrain each** f **in** Forging
such that type(has_forging_material(f)) <> "light alloy"
to have has_external_fillet_radii(f) >= 10
and has_internal_fillet_radii(f) >= 10

(xii) **constrain each** f **in** Forging
such that type(has_forging_material(f)) <> "light alloy"
to have has_external_fillet_radii(f) >= 10

Again, one can infer that constraint (xi) subsumes constraint (xii). ConEditor+ notifies the user (domain expert) of this fact and suggests that the user considers removing or deactivating constraint (xii).

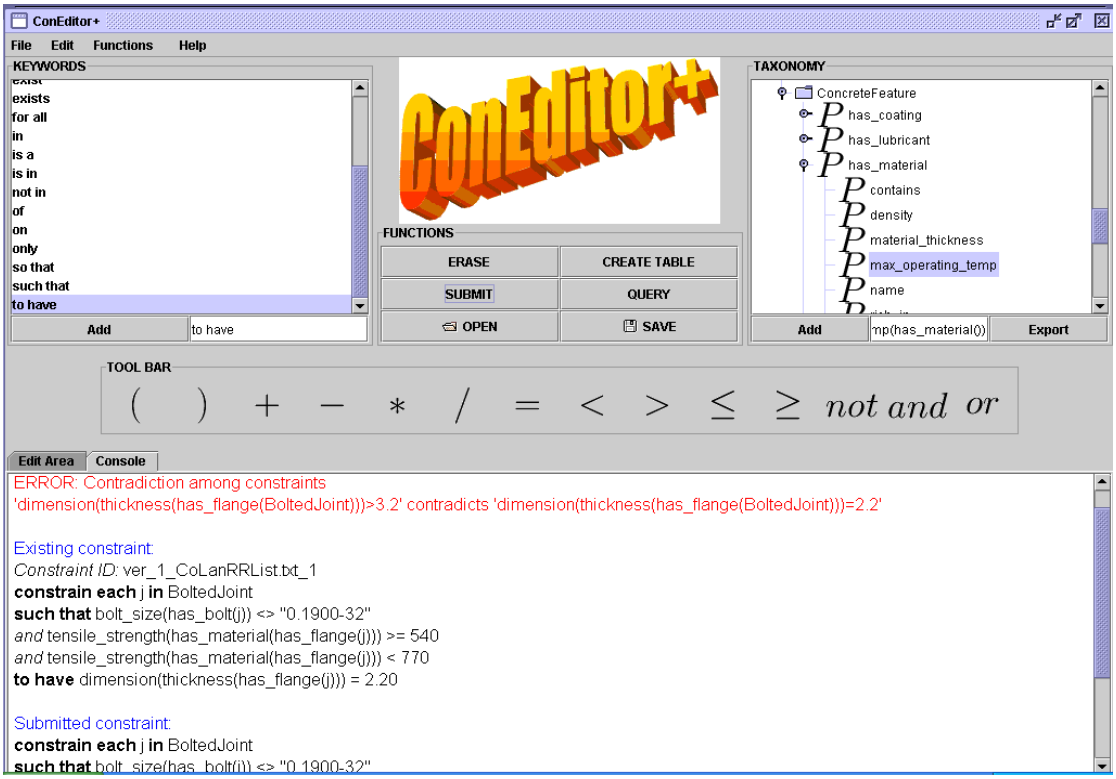


Figure 9.1: A screenshot of ConEditor+ showing inconsistency between a pair of constraints

Inconsistency:

(xiii) **constrain each** c **in** Component
such that name(component_coating(c)) = "silver"
and name(component_material(c)) = "steel"
to have tensile_strength(component_material(c)) < 1390

(xiv) **constrain each** c **in** Component
such that name(component_coating(c)) = "silver"
and name(component_material(c)) = "steel"
to have tensile_strength(component_material(c)) > 1590

By comparing the two constraints above, one can infer that the constraint (xiii) contradicts constraint (xiv). ConEditor+ notifies the user (domain expert) of this fact and suggests that the user takes an appropriate action (modify/delete) to resolve the inconsistency. An example of such an inconsistency (contradiction) detected by ConEditor+ is shown in Figure 9.1.

Fusion:

(a) Fusion via class

(xv) **constrain each** c **in** LowVelocityPlasmaCoating
such that contains(component_material(has_component(c))) <>
"magnesium"
to have material_thickness(has_coating_material(c)) >= 0.05
(xvi) **constrain each** c **in** HighVelocityPlasmaCoating
such that contains(component_material(has_component(c))) <>
"magnesium"
to have material_thickness(has_coating_material(c)) >= 0.05

If *LowVelocityPlasmaCoating* and *HighVelocityPlasmaCoating* are the only two subclasses of *PlasmaCoating* in the domain ontology and if every instance of *PlasmaCoating* is an instance of either *LowVelocityPlasmaCoating* or *HighVelocityPlasmaCoating* then the constraints (xv) and (xvi) can be fused together and replaced by the constraint (xvii) as follows:

(xvii) **constrain each** c **in** PlasmaCoating
such that contains(component_material(has_component(c))) <>
"magnesium"
to have material_thickness(has_coating_material(c)) >= 0.05

(b) Fusion via application condition

(xviii) **constrain each** c **in** PlasmaCoating
such that contains(component_material(has_component(c))) <>
"magnesium"
to have material_thickness(has_coating_material(c)) < 0.20

(xix) **constrain each** c **in** PlasmaCoating
such that contains(component_material(has_component(c))) <>
"copper"
to have material_thickness(has_coating_material(c)) < 0.20

The constraints above can be fused together by using "or" between the application conditions, i.e., the constraints (xviii) and (xix) can be fused together and replaced by the constraint (xx) as follows:

(xx) **constrain each** c **in** PlasmaCoating
such that contains(component_material(has_component(c))) <>
"magnesium" or contains(component_material(has_component(c))) <>
"copper"
to have material_thickness(has_coating_material(c)) < 0.20

(c) Fusion via conjunction

(xxi) **constrain each** c **in** PlasmaCoating
such that contains(component_material(has_component(c))) <>
"magnesium" or contains(component_material(has_component(c))) <>
"copper"
to have material_thickness(has_coating_material(c)) >= 0.05

(xxii) **constrain each** c **in** PlasmaCoating
such that contains(component_material(has_component(c))) <>
"magnesium" or contains(component_material(has_component(c))) <>
"copper"
to have material_thickness(has_coating_material(c)) < 0.20

The constraints above can be fused together by using “and”, i.e., the constraints (xxi) and (xxii) can be fused together and replaced by the constraint (xxiii) as follows:

(xxiii) **constrain each** c **in** PlasmaCoating
such that contains(component_material(has_component(c))) <>
"magnesium"
or contains(component_material(has_component(c))) <> "copper"
to have material_thickness(has_coating_material(c)) >= 0.05
and material_thickness(has_coating_material(c)) < 0.20

In the above cases of fusion, ConEditor+ notifies the user (domain expert) and suggests the user considers fusing the appropriate pairs of constraints. It is then left to the user to take appropriate action.