ConEditor+: Capture and Maintenance of Constraints in Engineering Design

Suraj Ajit[1], Derek Sleeman[1], David W. Fowler[1], David Knott[2] and Kit Hui[1]
[1]Department of Computing Science, University of Aberdeen, Scotland, UK
Email: {sajit, sleeman, dfowler, khui} @ csd.abdn.ac.uk
[2]Rolls-Royce plc, Derby, UK
Email: david.knott@rolls-royce.com

Abstract

The Designers' Workbench is a system, developed to support designers in large organizations, such as Rolls-Royce, by making sure that the design is consistent with the specification for the particular design as well as with the company's design rule book(s). Currently, to capture the constraint information, a domain expert (design engineer) has to work 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 knowledge base (KB). This is an error prone and time consuming task. It is highly desirable to relieve the knowledge engineer of this task, and so we have developed a tool, ConEditor+ that enables domain experts themselves to capture and maintain these constraints. The tool allows the user to combine selected entities from the domain ontology with keywords and operators of a constraint language to form a constraint expression. Further, we hypothesize that to apply constraints appropriately, it is necessary to understand the context in which each constraint is applicable. We refer to this as "application conditions". We show that an explicit representation of application conditions, in a machine interpretable format, along with the constraints and the domain ontology can be used to support the verification and maintenance of constraints.

1 Introduction

The context for the system reported here, ConEditor+ [Ajit *et al.*, 2005], is the Designers' Workbench [Fowler *et al.*, 2004] which has been developed to enable a group of designers to produce cooperatively a component which conforms to the component's overall specifications and the company's design rule book(s). Sections 1.1 and 1.1.1 provide an introduction to the Workbench, a description of the problem(s) faced and the need for ConEditor+. Section 2 gives a brief overview of our system ConEditor+. Section 3 then focuses on the maintenance aspects of constraints.

The issues faced in KB maintenance were first raised by the XCON configuration system at Digital Equipment Corporation [Barker and O'Connor, 1989; Soloway *et al.*, 1987]. 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].

The engineering design process has an evolutionary and iterative nature as designed artifacts 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 deletion of existing constraints. The reasons for such changes include development in the technology, changes to improve performance, changes to reduce development time and costs. In order to reduce/overcome the various maintenance problems, systems that capture and represent the rationales associated with design knowledge have been developed. Design rationales [Burge and Brown, 2003; Regli *et al.*, 2000] capture the following types of information:

a) the reasons why a design decision was taken
b) the design alternatives considered with reasons for acceptance/rejection
c) how certain design actions are performed

However, we are interested in capturing information about when a particular design constraint is applicable. We believe it is important to know the context in which a particular constraint or a rule can be applied. We refer to this as the *application conditions* associated with a constraint. In this paper, we present an approach that involves the explicit representation of application conditions in a machine interpretable format, along with the constraint itself. This information is used along with the appropriate domain ontology to support the verification and maintenance of constraints. Section 3 gives a description of our approach and its implementation for the domain of kite design. We discuss our evaluation and results in section 4. The conclusions and plans for future work follow in section 5.

1.1 Introduction to the Designers' Workbench

Designers in Rolls-Royce, as in many large organizations, work in teams. Thus it is important when a group of design-

ers are working on aspects of a common project, that the subcomponent designed by one engineer 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 we have developed the Designers' Workbench which seeks to support these activities.

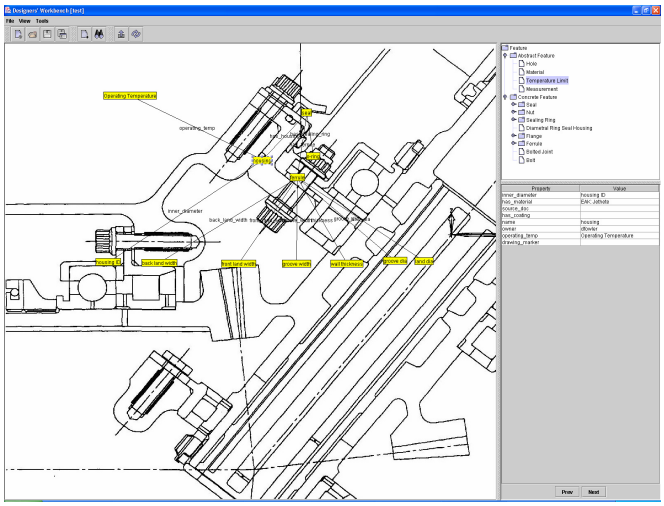

Figure 1: Screenshot of the Designers' Workbench

The Designers' Workbench (figure 1) uses an ontology [Gruber, 1995] 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. A graphical display enables the designer to easily add new features, set property values, and perform constraint checks. 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. The designer can resolve the violations by adjusting the property values of the affected features. On selecting the affected feature from the ontology tree, a table is listed with the corresponding properties and values. These property values can then be adjusted to resolve the constraint violations. More details about this system can be found in [Fowler *et al.*, 2004].

1.1.1 The problem addressed

As noted above, the Designers' Workbench needs access to the various constraints, including those inherent in the company's design rule book(s). Currently, 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 knowledge base (KB). This is an error prone and time consuming task. As constraints are explained very briefly in design rule book(s), a non-expert in the field can find 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 this process. An example of a constraint as expressed in rule book(s) is shown in figure 2. Adding a new constraint into the Designers' Workbench's KB currently requires coding a query in RDF Query Language [HP], and a predicate in Sicstus Prolog [SICStus].

It would be useful if a new constraint can 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 help engineers to input all the constraints themselves and relieve the programmer of that task. 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 constraint checks. This led to the development of a system, known as ConEditor+, which enables a domain expert to input and maintain constraints. ConEditor+ is explained further in the next section.

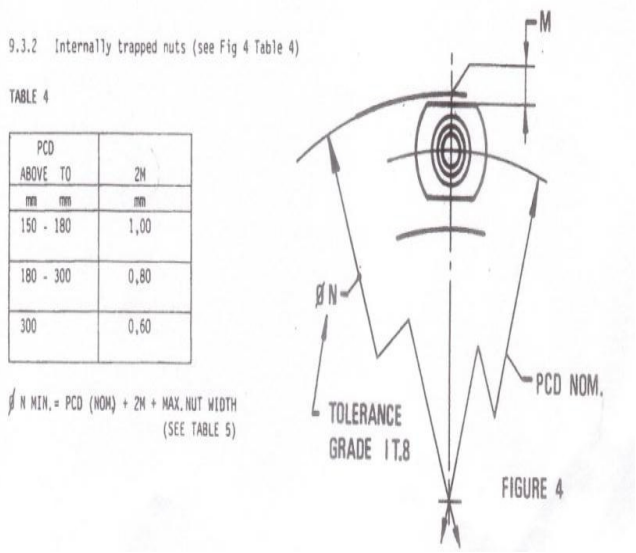

Figure 2: Constraint as expressed in rule book

2 ConEditor+

ConEditor+ is a tool to enable domain experts themselves to input and maintain constraints. ConEditor+'s graphical user interface (GUI) is shown in figure 3. A constraint expression can be created by selecting entities from a taxonomy (domain ontology) and combining them with a pre-defined set of keywords and operators from the high level constraint language, CoLan [Bassiliades and Gray, 1995; Gray *et al.*, 2001]. CoLan has features of both first-order logic and functional programming, and is intended for scientists and engineers to express constraints.

An example of a simple constraint expressed in CoLan, against a domain ontology (a jet engine ontology) used by the Designers' Workbench is as follows:

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

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

ConEditor+'s GUI essentially consists of six components, namely: (A) Keywords Panel, (B) Menu Bar, (C) Taxonomy Panel, (D) Functions Panel, (E) Tool Bar and (F) Result Panel (see figure 3). 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. More details about the GUI can be found in [Ajit *et al.*, 2004]. An analysis of the Rolls-Royce's design rule book(s) showed that a number of constraints are expressed in tables and so ConEditor+ provides a mechanism for inputting tables. ConEditor+ can store different versions of a constraint and provides facilities to retrieve constraints using keyword-based searches.

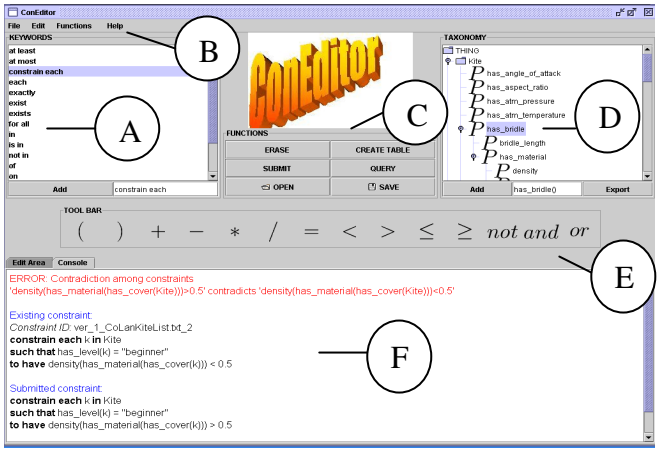

Figure 3: Screenshot of ConEditor+

3 Maintenance of constraints

Due to restricted availability of designers' time and for simplicity, we have used a kite domain [Eden, 1998; Streeter, 1980; Yolen, 1976] for our study. Consider the following constraint along with its associated rationale and application condition:

Constraint – "The strength of the kite line needs to be greater than 90 daN[1] units."

Associated rationale – "This provides the required stability for the kite to fly."

Application condition – "This is applicable to stunt kites of standard size in strong winds only."

[1] Symbol for deca Newton, a common metric unit of force.

The difference between a rationale and an application condition is evident from the example considered above; the rationale states the reason for a constraint (why), whereas the application condition states the context in which it is applicable (when).

In order to tackle the various maintenance issues/problems, our proposed solution is summarized as follows:

- Capture the "context" of a constraint, in a machine interpretable form, as an application condition
- Use the application condition together with the constraint and the appropriate domain ontology to perform the several constraint maintenance tasks described in section 3

We intend to capture the "context" of each constraint i.e. the information pertaining to when a constraint is applicable, referred to as its application conditions. Often, such information is implicit to the person who formulates the constraint. We believe that it is important to make the application conditions explicit so that it can be used for both verification and maintenance. The assumptions/conditions on which a constraint is based may no longer be true/applicable 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.

ConEditor+ captures both the constraints and the application conditions in the same language, CoLan. Both the constraints and the application conditions are then automatically converted into a standard machine interpretable format known as Constraint Interchange Format (CIF) [Gray *et al.*, 2001]. Representation of a sample constraint with its application condition in CoLan is as shown below:

```
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))
```

In the above constraint, the application condition (in italics) is introduced by the clause "such that". This constraint states that the length of a tail of a kite needs to be seven times the length of the spine of the kite; this constraint is applicable for flat, diamond shaped kites only.

There are a number of ways in which we can use the information of application conditions to enable the verification and maintenance of constraints. Some examples are described below:

1. Subsumption

```
a)constrain each s in Sled_kite
such that has_size(s) = "standard"
to have kite_line_strength(has_kite_line(s))
>= 15
```

```
b)constrain each c in Conventional_sled_kite
such that has_size(c) = "standard"
to have kite_line_strength(has_kite_line(c))
>= 15
```

Conventional_sled_kite is a subclass of *Sled_kite* in the domain ontology. It can be inferred that the constraint in a) subsumes the constraint in b). The domain expert is notified of this fact and allowed to remove, shelve or deactivate the constraint in b). Similarly, subsumption among application conditions occurs, when we have:

```
c)constrain each s in Sled_kite
such that has_size(s) = "standard" or
has_size(s) = "large"
to have kite_line_strength(has_kite_line(s))
>= 15

d)constrain each s in Sled_kite
such that has_size(s) = "standard"
to have kite_line_strength(has_kite_line(s))
>= 15
```

Again, it can be inferred that the constraint in c) subsumes the constraint in d). The domain expert is notified of this fact and allowed to remove, shelve or deactivate the constraint in d).

2. Contradiction

```
e)constrain each k in Kite
such that has_type(k) = "stunt"
to have kite_line_strength(has_kite_line(k))
> 30

f)constrain each k in Kite
such that has_type(k) = "stunt"
to have kite_line_strength(has_kite_line(k))
< 30
```

Comparing the above two constraints, it can be inferred that the constraint in e) contradicts the constraint in f). The domain expert is notified of this fact and allowed to take the appropriate action (modify/delete).

3. Redundancy

```
g)constrain each c in Conventional_sled_kite
such that has_level(c) = "beginner"
to have density(has_material(has_cover(c)))
< 0.5

h)constrain each t in Traditional_sled_kite
such that has_class(t) = "beginner"
to have density(has_material(has_cover(t)))
< 0.5
```

Coventional_sled_kite is an equivalent class to *Traditional_sled_kite* in the domain ontology. Also *has_level* is an equivalent property to *has_class*. It can be inferred that the constraint in g) or h) is redundant. The domain expert is notified of this fact and allowed to take appropriate action to eliminate redundancy.

4. Fusion

```
i)constrain each c in Conventional_sled_kite
such that has_wind_condition(c) = "moderate"
to have has_bridle_attachment_angle(c) < 40

j)constrain each m in Modern_sled_kite
such that has_wind_condition(m) = "moderate"
to have has_bridle_attachment_angle(m) < 40
```

Conventional_sled_kite and *Modern_sled_kite* are the only two subclasses of *Sled_kite* in the domain ontology. The constraints in i) and j) can be fused together and replaced by k) as follows:

```
k)constrain each s in Sled_kite
such that has_wind_condition(s) = "moderate"
to have has_bridle_attachment_angle(s) < 40
```

Also two or more application conditions or constraints can be fused together using "or" and "and" respectively. For example:

```
l)constrain each j in Japanese_kite
such that has_wind_condition(j) = "strong"
to have has_bridle_point_distance(j) > 3 *
surface_area(has_cover(j))

m)constrain each j in Japanese_kite
such that has_type(j) = "stunt"
to have has_bridle_point_distance(j) > 3 *
surface_area(has_cover(j))
```

l) and m) can be fused together and replaced by n) as follows:

```
n)constrain each j in Japanese_kite
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)
```

In this case, ConEditor+ suggests to the domain expert that several constraints be fused.

In all the examples above, we have considered universally quantified constraints involving a single variable that are common in our knowledge base. 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/CIF [Gray *et al.*, 2001] by ConEditor+.

Implementation: ConEditor+ is implemented in the Java programming language; the domain ontology in the Web Ontology Language [OWL] is developed using Protégé [Noy *et al.*, 2000] and read using Jena [HP]. Any syntactic errors among constraints are detected by ConEditor+ with the help of a Daplex compiler [Bassiliades and Gray, 1995]. The constraints are initially expressed in CoLan and then converted automatically into a standard Constraint Interchange Format (CIF) using a translator. ConEditor+ uses this machine interpretable format to detect inconsistencies (contradictions) between pairs of constraints and to suggest various ways to refine (fuse, eliminate redundancies and

subsumptions) the knowledge base, as described earlier. The domain expert could then resolve these inconsistencies and/or refine the knowledge base by using the appropriate functions of ConEditor+ to delete/modify/shelve constraints.

4 Evaluation and Results

Before implementing the maintenance features, we performed a preliminary evaluation of our system. A demonstration was given to the design engineers at Rolls-Royce. The demonstration involved the following three phases: i) Presenting the constraint as in the rule book i.e. as a mixture of textual and graphical information (figure 2) ii) Expressing the constraint in CoLan iii) Inputting the constraint using ConEditor+. The design engineers were able to follow all the three phases. They found the GUI simple, user-friendly and fairly intuitive to use. However they felt they would need some training before they could do the steps in the last two phases [(ii) and (iii)] unsupported. They also made the general point that they have a Design Standards group that has the responsibility for creating and maintaining the company-wide rule book(s). They would expect this group to use systems such as ConEditor+ to input constraints. The designers would then subsequently use the information either in the current form or in the Designers' Workbench-like environment.

After implementing the maintenance features, we conducted two experiments.

Experiment 1: We studied the domain of kite design and captured constraints along with their application conditions. We ran an experiment with ConEditor+ using: (a) set containing 15 constraints along with their application conditions, (b) set containing the same constraints **without** application conditions.

Results: For dataset in (a), ConEditor+ detected 3 subsumptions, 0 contradictions, 3 redundancies and 2 cases of fusion between pairs of constraints. For dataset in (b), ConEditor+ detected 1 subsumption, 5 contradictions, 2 redundancies and 4 cases of fusion between pairs of constraints. For dataset in (b), it is evident that the absence of application conditions caused a number of inconsistencies (5 contradictions), and also, ConEditor+ suggested a number of inappropriate refinements.

Experiment 2: We gave a demonstration of all the features of ConEditor+ to five subjects (two mechanical engineering research students, two computer science research students and one computer science research fellow). Each subject was then given the task of inputting a set of constraints in CoLan using ConEditor+. The subjects were asked to use ConEditor+ to resolve inconsistencies (contradictions) and also follow any suggestion(s) given by ConEditor+ to refine (fuse, eliminate redundancies and subsumptions) the knowledge base. A questionnaire about the usability of ConEditor+ and its maintenance features was given to the subjects, who were asked to use a 5 point rating scale (1 being poor and 5 being excellent).

Results: All the subjects found ConEditor+ fairly easy to use and helpful for the verification and maintenance of constraints. The average overall rating given by the subjects, for both the usability and maintenance features of ConEditor+ was 4. Additionally, some subjects gave helpful suggestions to improve the usability of ConEditor+.

5 Conclusions and Future Work

This paper describes a methodology to enable domain experts to capture and maintain constraints in an engineering design environment. The context is a system known as the Designers' Workbench that has been developed to automatically check if all the constraints have been satisfied and if not, enable the designers to resolve them. The Designers' Workbench is faced with the task of accumulating the constraints associated with the domain. This needs a knowledge engineer to study the design rule book(s), consult the design engineer (domain expert) and encode all the constraints into the Designers' Workbench's KB. We describe the tool ConEditor+ that has been developed to help domain experts themselves capture and maintain engineering design constraints.

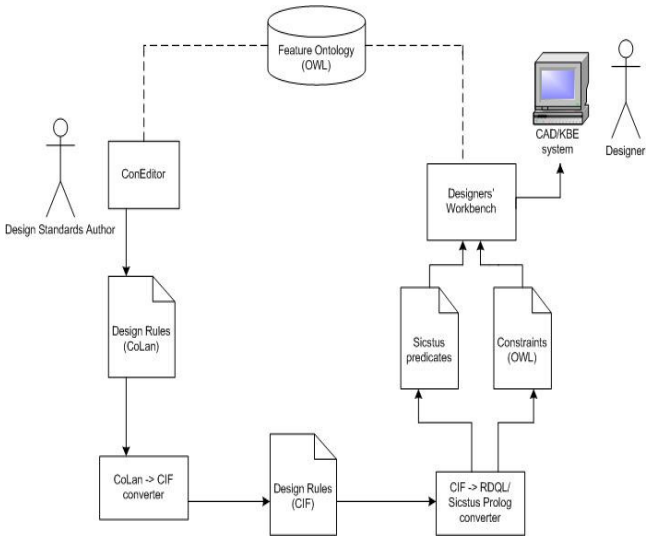

Figure 4: Proposed System Architecture

We hypothesize that in order to apply constraints appropriately, it is necessary to capture the contexts (application conditions) associated with the constraints and that these would be beneficial for verification and maintenance. On the basis of our studies and experiments done in the domain of kite design, we find the above hypothesis to be true, and also find ConEditor+ to be a useful tool for design engineers to capture and maintain constraints.

As part of the future work, it is planned to use ConEditor+ to capture the application conditions along with the constraints for a significant part of the Rolls-Royce domain and investigate how they help with verification and maintenance of this more demanding KB. We also plan to complete the implementation of the proposed architecture (fig-

ure 4) that shows how ConEditor+ fits into the whole framework. A Design Standards author initially inputs all the design rules (constraints) into ConEditor+. The design constraints are then automatically converted into a standard machine interpretable format (CIF) and processed by the Designers' Workbench. As can be seen from figure 4, it is planned to interface the Designers' Workbench to a more sophisticated CAD/KBE system.

Acknowledgements

This work is supported by the EPSRC Sponsored Advanced Knowledge Technologies project, GR/NI5764, which is an Interdisciplinary Research Collaboration involving the University of Aberdeen, the University of Edinburgh, the Open University, the University of Sheffield and the University of Southampton. We would like to acknowledge the assistance of engineers and designers in the Transmissions and Structures division of Rolls-Royce plc, Derby, UK.

References

[Ajit *et al.*, 2004] Suraj Ajit, Derek Sleeman, David W. Fowler and David Knott. 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*, 466 - 468, Whittlebury Hall, Northampton, UK, 2004.

[Ajit *et al.*, 2005] Suraj Ajit, Derek Sleeman, David W. Fowler, David Knott and Kit Hui. Acquisition and Maintenance of Constraints in Engineering Design. In *Proceedings of the 3rd International Conference on Knowledge Capture, KCAP 2005*, 173-174, Banff, Canada, 2005.

[Barker and O'Connor, 1989] V. E. Barker and D. E. O'Connor. Expert Systems for Configuration at Digital: XCON and Beyond. *Communications of the ACM*, 32 (3)**:** 298-318, 1989.

[Bassiliades and Gray, 1995] N. Bassiliades and P. Gray. CoLan: A Functional Constraint Language and Its Implementation. *Data and Knowledge Engineering*, 14 (3)**:** 203-249, 1995.

[Burge and Brown, 2003] Janet Burge and David C. Brown. Rationale Support for Maintenance of Large Scale Systems. In *Workshop on Evolution of Large-Scale Industrial Software Applications (ELISA), ICSM '03*, Amsterdam, NL, 2003.

[Coenen, 1992] F. P. Coenen. A Methodology for the Maintenance of Knowledge based Systems. In *Niku-Lari, A. (Ed), EXPERSYS-92 (Proceedings), IITT-International*, 171-176, France, 1992.

[Eden, 1998] Maxwell Eden. *The Magnificient Book of Kites: Explorations in Design, Construction, Enjoyment and Flight*. Black Dog & Levanthal Publishers, New York, 1998.

[Fowler *et al.*, 2004] D. W. Fowler, D. Sleeman, G. Wills, T. Lyon and D. Knott. Designers' Workbench. In *Proceedings of the Twenty-fourth SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence*, Cambridge, UK, 2004.

[Gray *et al.*, 2001] Peter Gray, Kit Hui and Alun Preece. An Expressive Constraint Language for Semantic Web Applications. In *E-Business and the Intelligent Web: Papers from the IJCAI-01 Workshop*, 46-53, Seattle, USA, 2001. AAAI Press.

[Gruber, 1995] T. R. Gruber. Towards Principles for the Design of Ontologies Used for for Knowledge Sharing. *International Journal of Human-Computer Studies*, 43 (5-6)**:** 907-928, 1995.

[HP] A semantic web framework for Java. [online]. Available from: http://jena.sourceforge.net/index.html [Accessed 21 June 2006].

[Noy *et al.*, 2000] N. F. Noy, R. W. Fergerson and M. A. Musen. The knowledge model of Protege-2000: Combining interoperability and flexibility. In *International Conference on Knowledge Engineering and Knowledge Management (EKAW' 2000)*, Juan-les-Pins, France, 2000.

[OWL] Web Ontology Language. [online]. Available from: http://www.w3.org/TR/owl-features/ [Accessed 21 June 2006].

[Regli *et al.*, 2000] W. C. Regli, X. Hu, M. Atwood and W. Sun. 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**:** 209-235, 2000.

[SICStus] Version 3.10.0, Swedish Institute of Computer Science. [online]. Available from: http://www.sics.se/sicstus/ [Accessed 21 June 2006].

[Soloway *et al.*, 1987] E. Soloway, J. Bachant and K. Jensen. Assessing the Maintainability of XCON-in-RIME: Coping with Problems of a Very Large Rule-Base. In *Proceedings of AAAI-87*, 824-829, Seattle, USA, 1987.

[Streeter, 1980] Tal Streeter. *The Art of the Japanese Kite*. Charles E Tuttle Company Inc, Tokyo, 1980.

[Yolen, 1976] Will Yolen. *The Complete Book of Kites and Kite Flying*. Simon and Schuster Trade, New York, 1976.