Capture and Maintenance of Engineering Design Constraints

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
{sajit, sleeman, dfowler, khui}@csd.abdn.ac.uk
[2]Rolls Royce plc, Derby, UK
david.knott@rolls-royce.com

Abstract

The Designers' Workbench is a system, developed by the Advanced Knowledge Technologies (AKT) consortium 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. However 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 plan to make these application conditions machine interpretable and investigate how they, together with a domain ontology, can be used to support the verification and maintenance of constraints.

1. Introduction

The context for the system reported here, ConEditor [1], is the Designers' Workbench [2] that we have developed to enable a group of designers to produce cooperatively a component that conforms to the overall specifications and the company's design rule book(s). An introduction to Designers' Workbench is given in the following section.

1.1 Introduction to the Designers' Workbench

Designers in Rolls-Royce [3], 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 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 the AKT [4] consortium has developed a Designers' Workbench, which seeks to support these activities.

The Designers' Workbench (Figure 1) uses an ontology [5] to describe elements in a configuration task. The system supports human designers by checking that their configurations satisfy both physical and organisational 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 that can be clicked on, to read the source document. The designer can resolve the violations by adjusting the property values of the affected features. On selecting a feature, a table is listed with the corresponding properties and their values. These property values can then be adjusted to resolve the constraint violations.

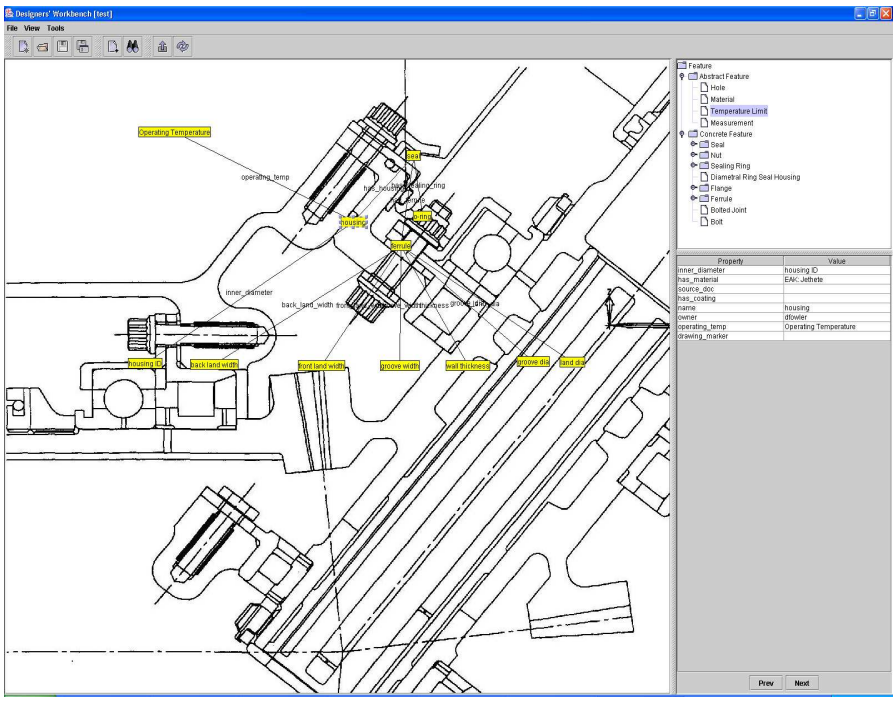

Figure 1 Screenshot of the Designers' Workbench

1.1.1 What is the problem?

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 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 the process. Adding a new constraint into Designers' Workbench's KB currently requires coding a query in RDQL (RDF Query Language) [6], and a predicate in Sicstus Prolog [7].

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. The programmer could then certainly avoid having to go through all the design rule book(s), formulate the constraints and input them into Designers' Workbench's KB. This would also enable designers to have 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.

2. ConEditor

ConEditor [1] is a tool to enable domain experts themselves to input and maintain constraints. ConEditor's graphical user interface (GUI) is shown in Figure 2. 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 [8, 9].

We shall now consider an example of a constraint and see how it can be input using ConEditor's GUI. (Figure 2). One of the constraints 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. This constraint can be expressed in CoLan as follows:

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

ConEditor's GUI essentially consists of five components, namely, keywords panel, taxonomy panel, central panel, tool bar and result panel. These panels

provide the user with the various entities required to form a constraint expression. A brief description of each panel is given below. The ordering below corresponds to the labels in Figure 2.

A. Keywords panel

This panel consists of a list of keywords from the constraint language CoLan. A keyword can be selected from this panel to form a component of a constraint expression. Clicking the "Add" button appends the selected keyword to the constraint expression being formed in the result panel. In the example considered, the keywords "Constrain each", "in" and "to have" are selected from this list of keywords.

B. Taxonomy panel

This panel displays a taxonomy of classes, subclasses and properties in a tree structure, extracted from the domain ontology (i.e. the ontology used by Designers' Workbench). A node (icon) representing the root of the tree is initially displayed on the panel. Double-clicking on this node expands the tree, displaying more nodes of the tree. Nodes represented by letter 'P' denote properties of a class. A node can be selected from this tree and clicking the "Add" button appends it to the constraint expression being formed in the result panel. In the example considered, the entities "Concrete Feature", "max_operating_temp", "has_material" and "operating_temp" are selected from this panel.

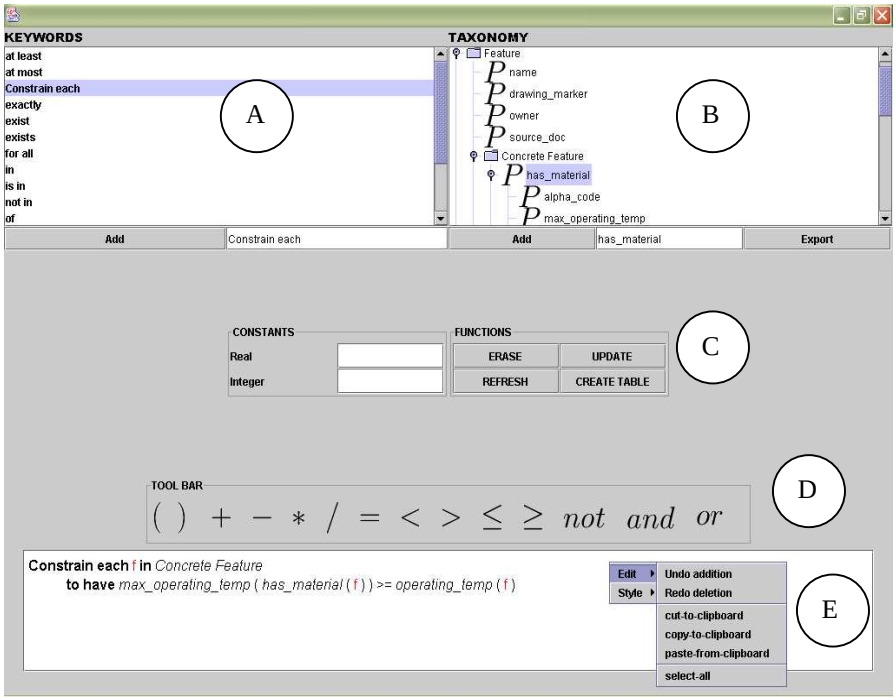

Figure 2 Screenshot of ConEditor's GUI

C. Central panel

This panel has two sections. One is the constant section, where two text fields are provided for inputting real and integer constants. Clicking the “update” button appends the constant to the constraint expression being formed. The other is the functions section, where function buttons are provided for editing the constraint expression, adding a constant, refreshing the panel and creating a table.

D. Tool bar

The Tool Bar displays the arithmetic, relational and logical operators and delimiters. Clicking on any of the operators appends the operator to the result panel. The operator ‘>=’ and the delimiters ‘(‘, ‘)’ in the example are chosen from the tool bar.

E. Result panel

This is situated at the bottom of the main screen. It contains a large text area, displaying the evolving expression specified 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 and paste text and specify the font and size of the text.

ConEditor is implemented in the Java programming language; the ontology (developed in OWL [10]) is read using Jena [3], and the Protégé [11] tool is used to develop/view the ontology. The constraints expressed in CoLan using ConEditor, can be converted into a XML encoded standard format, Constraint Interchange Format (CIF) [9], making them portable. Constraint solvers or systems such as Designers’ Workbench can then process these constraints.

2.1 Constraints in tabular form

An analysis of the Rolls-Royce’s design rule book(s) showed that a number of constraints are expressed in tables. ConEditor has a facility to input such constraints in tabular form. A table is created using the “Create Table” function button in the central panel. Clicking this button prompts the user to enter the desired number of rows and columns in the table. Further, clicking the “OK” button after each entry accepts the value. A table is then created with the specified number of rows and columns and opens up in a new window (Figure 3). The column headers of the table are set by importing entities from the taxonomy panel, using the “Export” button, located at the bottom end of the taxonomy panel. Depending on the entities in the column headers, particular columns are provided with drop-down menus in their fields, consisting of a set of values to choose from. There are facilities provided to add/delete rows and columns to/from the table, resize the columns of the table, etc. Figure 3 shows a table created using ConEditor.

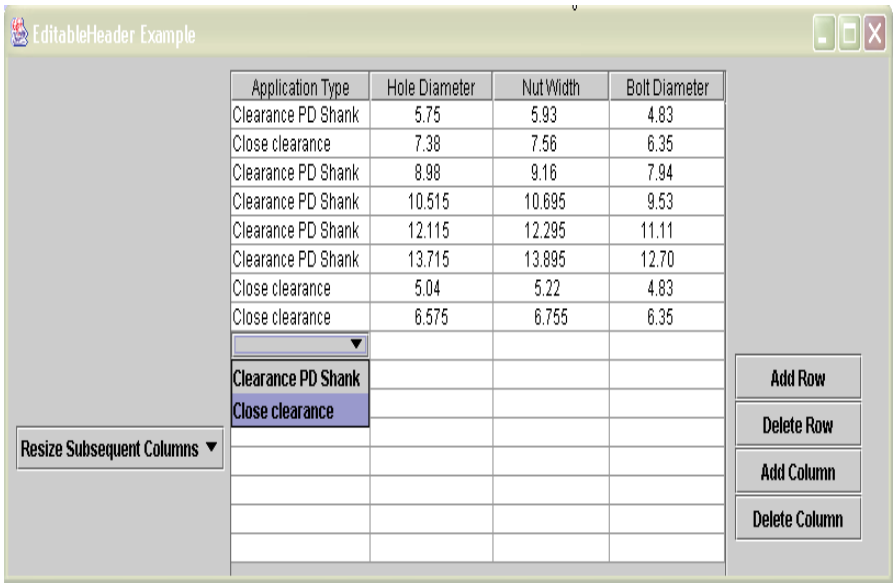

Figure 3 A screenshot showing constraints expressed in tables using ConEditor

3. Evaluation

A demonstration of the system was given to several design engineers at Rolls-Royce plc, Derby, UK. The focus of the demonstration was one of the constraints found in the design rule book(s). This demonstration involved 3 distinct phases:

Phase 1: Presenting the constraint as in the rule book(s)

The description of the constraint, as found in the rule book(s), is shown in Figure 4. The design rule book(s) gives the description of constraints, in the form of tables and figures in most cases and very little explanation is provided. Clearly, it is very hard for a non-expert to understand the context and to formulate a constraint from the rule book(s). The English rendering of a part of the constraint in Figure 4 is:

Constraint: Bolted joints must conform to the formula

N_{min} = PCD + 2*M + Max. Nut Width

where 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 can be expressed in CoLan as:

```
Constrain each j in Bolted Joint
such that has_nut(j) in Captive Nut
and dimension(pcd(has_flange(j))) > 150.0
and dimension(pcd(has_flange(j))) <= 180.0
and not is_external(has_flange(j))
to have gap(has_flange(j))= 0.5
and trap_diameter(has_flange(j))=
dimension(pcd(has_flange(j))
+ 2 * gap(has_flange(j))
+dimension(captive_nut_width(has_nut(j))
+tolerance(captive_nut_width(has_nut(j)))
```

Phase 3: Representing the expression in CoLan using ConEditor

In the final stage of the demonstration, the CoLan expression was input to ConEditor by its developer (Suraj Ajit).

3.1 Overview

- The GUI seems to be simple, user friendly and fairly intuitive to use.
- The designers were able to follow the steps where we mapped a constraint written in English to one expressed in CoLan; further they were able to understand how the CoLan expression was input to the ConEditor system. However, they felt they would need training to do either of these stages unsupported.
- Controlled Acquisition Scenario: The tool restricts the user to choose from 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 about using it as with expressing the constraint directly in English.
- They also made the general point that in this company it is the Design Standards group that has the responsibility for creating and maintaining the company-wide rule book(s), and so they would expect the standards group to input such constraints into a system like ConEditor. The designers would subsequently use the information either in the current form or in a Designers' Workbench-like environment.

It is planned to conduct more trials of ConEditor with a wide range of engineers shortly.

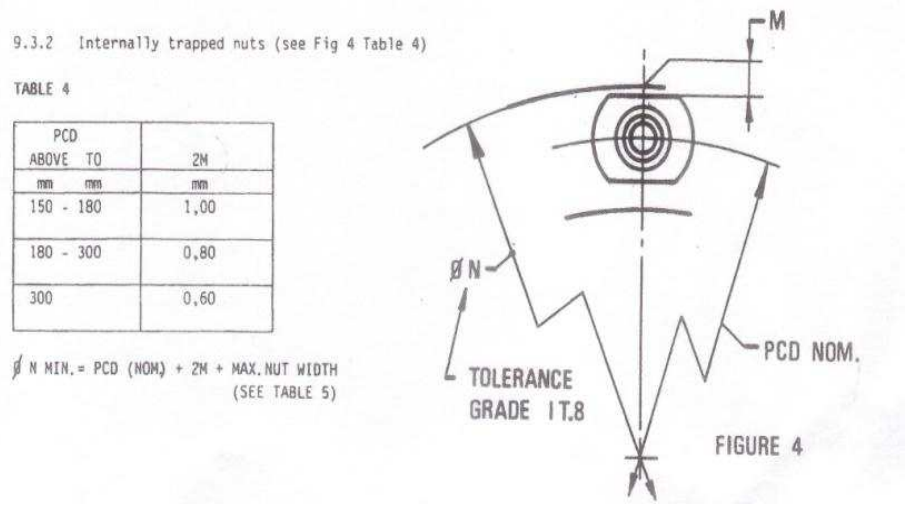

Figure 4 Constraint as expressed in the rule book (s)

4. Proposed system architecture

The proposed system architecture in Figure 5 shows how ConEditor fits into the whole framework. A Design Standards author initially inputs all the design rules (constraints) into ConEditor. The design rules are initially represented in a constraint language known as CoLan. They are then converted into a standard machine interpretable format known as Constraint Interchange Format (CIF) [9]. These are then transformed into appropriate RDQL [6] queries and Sicstus Prolog [7] predicates for use in the Designers' Workbench. More details about how they are used in Designers' Workbench can be found in [2]. As can be seen from Figure 5, it is planned to interface Designers' Workbench to a more sophisticated CAD/KBE system. The domain knowledge is represented as an ontology in OWL [10] and is used by both Designers' Workbench and ConEditor.

5. Maintenance of constraints in engineering design

The issues faced in KB maintenance were first raised by the XCON configuration system at Digital Equipment Corporation [12, 13]. 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 [14].

The engineering design process has an evolutionary and iterative nature as designed artifacts 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 and the modification or deletion of existing constraints. The reasons for such changes include changes in the technology, changes to improve performance, changes to reduce development costs, time, changes in application contexts.

In order to reduce/overcome the various maintenance issues/problems, systems that capture and represent the rationales associated with design knowledge have been developed. Design rationales [8, 17] capture the following types of information:

a) the reasons behind the design decisions taken (why a decision was taken).

b) the design alternatives considered with reasons for acceptance/rejection.

c) how certain design actions are performed.

However, design rationale systems have not been capturing information about when a particular section of the design knowledge is applicable. 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. Consider the following constraint in the domain of kite design along with its associated rationale and application condition:

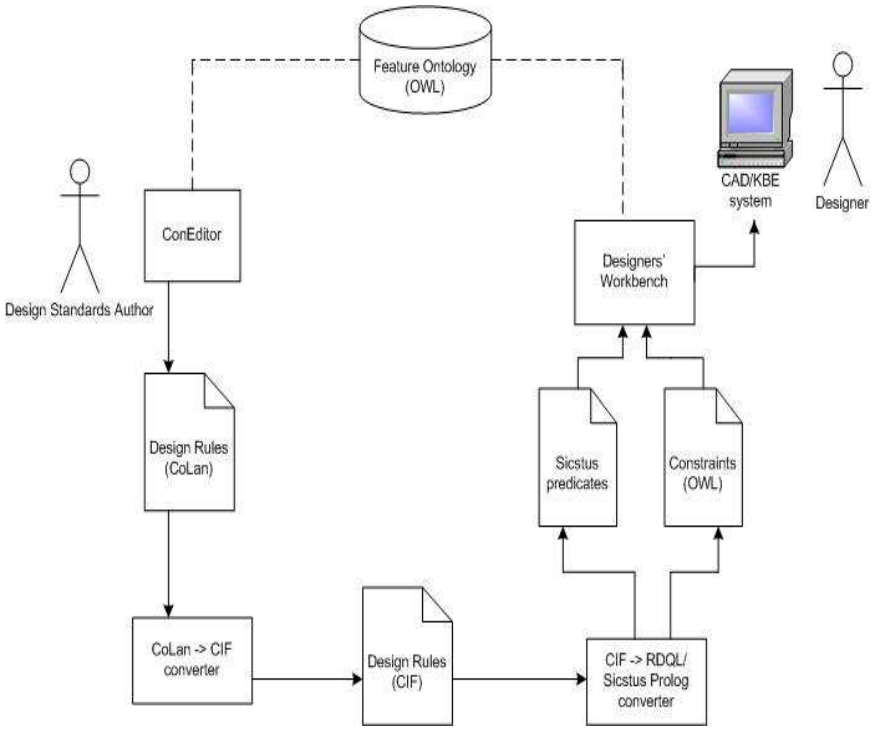

Figure 5 Proposed system architecture

Constraint – "The strength of the kite line of a kite 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 for stunt kites (of standard size) in strong winds only.

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

5.1 Our planned approach

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

- Capture the "context" of a constraint as an application condition
- Represent the application conditions in a machine interpretable form
- Use application conditions together with constraints to support maintenance

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. It is important to make the application conditions explicit so that it can be used for both maintenance and reuse. The assumptions/conditions on which a constraint is based may no longer be true/applicable; in such cases, it becomes necessary to deactivate or remove those constraints. Further an application condition may not be satisfied by a particular design task.

We plan to capture both the constraint and the application conditions in the same language, which is CoLan [6]. Both the constraints and the application conditions will then be converted into a standard machine interpretable format known as CIF [9]. The system can then detect inconsistencies among the constraints and application conditions, in which case, it could notify the user and suggest the constraint(s) be removed or modified. In the next section, we describe a case study based on a sample kite design.

5.2 Case study

Due to restricted availability of designers' time and for simplicity, we have used a kite domain for the case study. We tried to manually detect different kinds of inconsistencies among constraints and application conditions. These are explained as follows:

---

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

Representation of a sample constraint with its application condition:

```
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))
```

As shown 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. Subsumption, contradiction, redundancy are types of inconsistencies that can be detected among the constraints and application conditions using the domain ontology as background knowledge. For example, consider the following constraints:

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 a) subsumes b). The domain expert can be notified to remove or deactivate constraint b). Similarly, subsumption among application conditions occurs, when we have:

a)
```
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
```

b)
```
Constrain each s in Sled_kite
such that has_size(s) = “standard”
to have kite_line_strength(has_kite_line(s)) >= 15
```

It can be inferred that a) subsumes b) as the application conditions in b) are included in those of a). The domain expert can then be notified of this fact and be allowed to decide what action (if any) to take.

It is planned to develop a domain ontology in OWL [16] using Protégé [15] and to input both the constraints and the application conditions using an extended version of ConEditor that can detect and resolve such inconsistencies.

6. Related work

Alice [15] is another declarative language for the expression of complex logic based constraints, closely related to CoLan. In comparison with Alice, CoLan has better readability and expressive power. Goonetillake and Wikramanayake

[16] propose a framework for the management of evolving constraints in a computerized engineering design environment. But there is no mention of capturing the contexts/application conditions associated with the constraints. Enabling a domain expert to maintain his own knowledge in a knowledge-based system has long been an ideal for the knowledge engineering community. Burge and Brown [17] investigate the use of design rationales by building InfoRat, a system that makes inferences over a design's rationales in order to detect inconsistencies in the decisions made and to assess the impact of proposed changes. ConEditor's objective 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 interactions with a knowledge engineer. Regli, et al. [18] provide a survey on recent research in the area of design rationales; this paper reviews a number of recent design rationale systems, including JANUS, COMET, ADD, REMAP, HOS, PHIDIAS, DRIVE and IBIS.

7. Summary and future work

In this paper, we describe a methodology to enable domain experts to capture and maintain constraints in an engineering design environment. The context is a system known as 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. Designers' Workbench is faced with the task of accumulating all 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. We describe the tool ConEditor that has been developed to help domain experts themselves capture and maintain engineering design constraints for use in systems such as Designers' Workbench.

We hypothesize that in order to apply constraints appropriately, it is necessary to capture the context, the application conditions, associated with the constraints and that these would be beneficial for maintenance. A case study on a sample kite design domain has been performed. A number of constraints and their associated application conditions were elicited and input into ConEditor.

As part of the future work, it is planned to extend ConEditor to enable detection and resolution of inconsistencies among constraints and application conditions. A query facility using RDQL [6] to retrieve the appropriate constraints and application conditions according to specified criteria is planned. We then plan to further apply our ideas of application conditions to aspects of Rolls-Royce's design work.

8. 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

1. Ajit S., Sleeman D., Fowler D. W. and Knott D., ConEditor: Tool to Input and Maintain Constraints, Proceedings of EKAW 2004, October 5-8, Whittlebury Hall, Northampton, UK, 2004, pp. 466 - 468.

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

3. Rolls-Royce, Derby, UK, [WWW], Available from: http://www.rolls-royce.com/, [Accessed].

4. AKT, Advanced Knowledge Technologies (AKT Project), [WWW], Available from: http://www.aktors.org/akt/, [Accessed].

5. Chandrasekaran B., Josephson J. R. and Benjamins V. R. What are ontologies and why do we need them? IEEE Intelligent Systems Jan/Feb 1999; 14:20-26

6. Jena, A semantic web framework for Java, [WWW], Available from: http://jena.sourceforge.net/index.html, [Accessed].

7. Prolog S. Version 3.10.0, Swedish Institute of Computer Science 2001.

8. Bassiliades N. and Gray P. CoLan: a Functional Constraint Language and Its Implementation Data and Knowledge Engineering 1994; 14:203-249

9. Gray P., Hui K. and Preece A., An Expressive Constraint Language for Semantic Web Applications, E-Business and the intelligent Web: Papers from the IJCAI-01 Workshop, 2001, pp. 46-53.

10. OWL, OWL Web Ontology Language, [WWW], Available from: http://www.w3.org/TR/owl-features/, [Accessed].

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

12. Barker V. E. and O'Connor D. E. Expert Systems for Configuration at Digital: XCON and Beyond Communications of the ACM 1989; 32:298-318.

13. Soloway E., Bachant J. and Jensen K., Assessing the Maintainability of XCON-in-RIME: Coping with Problems of a Very Large Rule-Base, Proceedings of AAAI-87, 1987, pp. 824-829.

14. Coenen F. P., A Methodology for the Maintenance of Knowledge based Systems, In Niku-Lari, A. (Ed), EXPERSYS-92 (Proceedings), IITT-International, Gournay sur Marne, France, 1992, pp. 171-176.

15. Urban S., ALICE: an Assertion Language for Integrity Constraint Expression, Conference on Computer Software Applications, 1989.

16. Goonetillake J. S. and Wikramanayake G. N., Management of Evolving Constraints in a Computerised Engineering Design Environment, Proceedings of the 23rd National IT Conference, Colombo, Sri Lanka, 2004.

17. Burge J. E. and Brown D. C., Reasoning with Design Rationale, in Artificial Intelligence in Design '00, J. Gero, Ed. Netherlands: Kluwer Academic Publishers, 2000, pp. 611-629.

18. Regli W. C., Hu X., Atwood M. and Sun W. 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, 2000 2000; 16:209-235.