# 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, 2, 3] is the Designers' Workbench [9] 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 [15], 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 speci-

fication, 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 [7] 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.

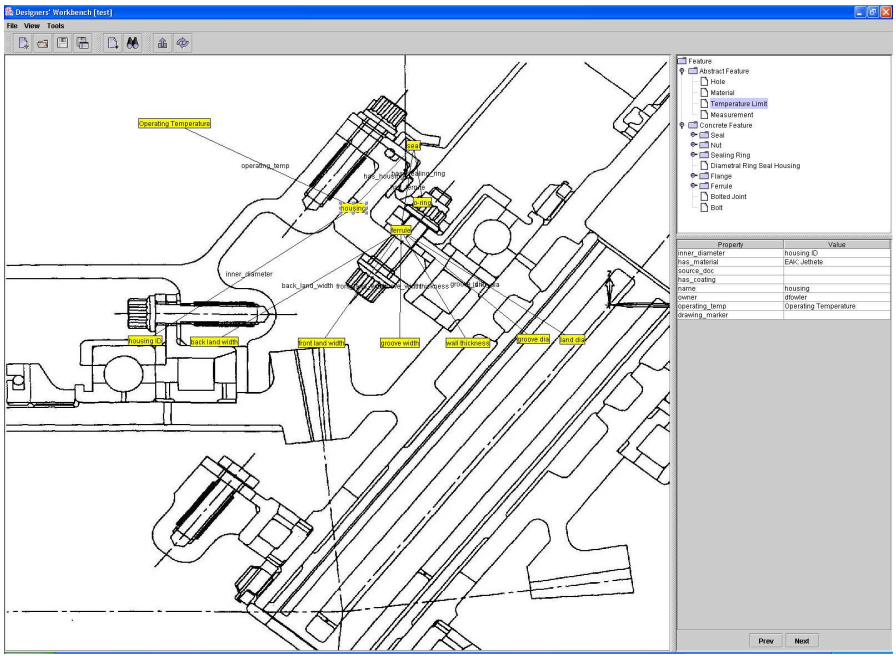


**Fig. 1.** Screenshot of the Designers' Workbench

### 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) [12], and a predicate in Sicstus Prolog.

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, 2, 3] 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 [11].

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)) >= operat-
ing_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.

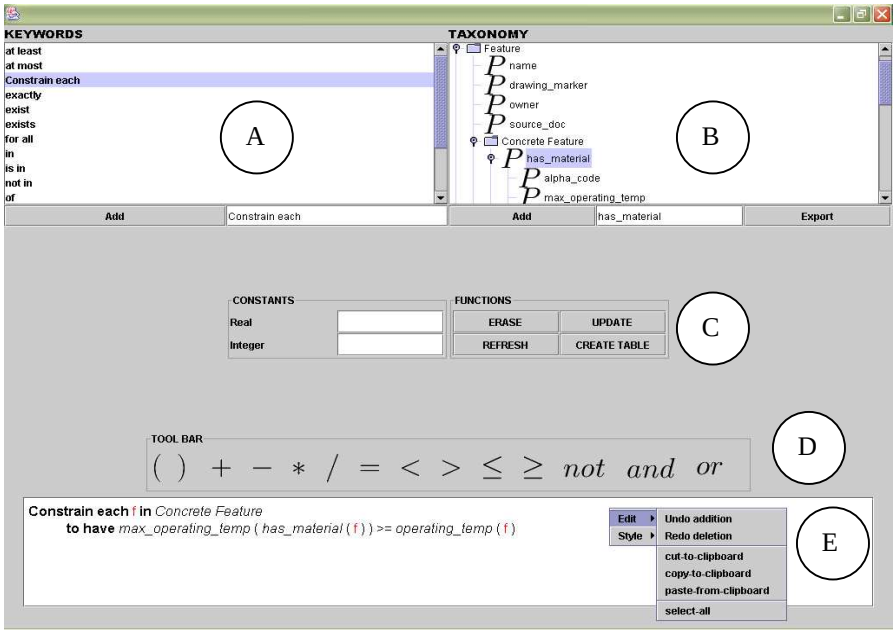


**Fig. 2.** Screenshot of ConEditor’s GUI

## 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 [13]) is read using Jena [12], and the Protégé 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) [11], making them portable. Constraint solvers or systems such as Designers' Workbench can then process these constraints.

# 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 3. 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 3 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.

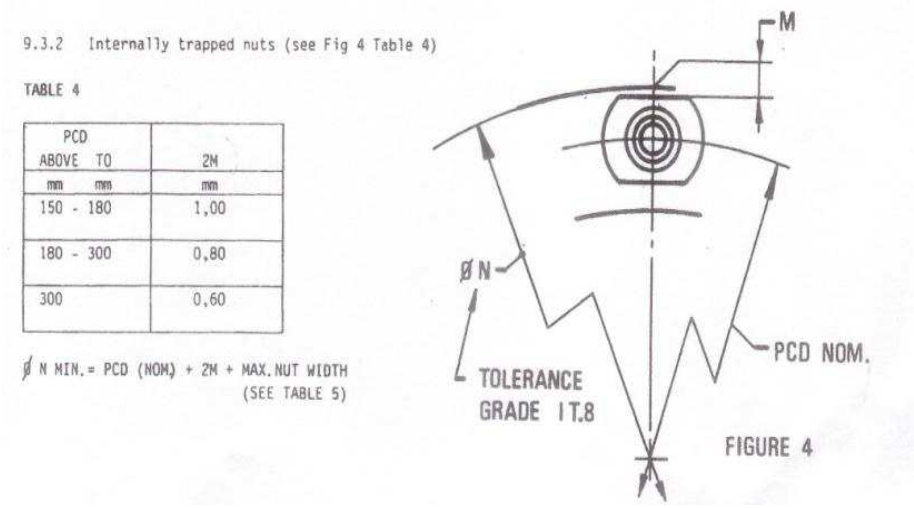


**Fig. 3.** Constraint as expressed in the rule book (s)

## 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 [5, 16]. 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 [8].

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 [6, 14] 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:

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).

## 6 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. Both the constraints and the application conditions will then be converted into a standard machine interpretable format known as CIF [11]. The system can then detect inconsistencies among the constraints and application conditions, against the domain ontology, in which case, it could notify the user and suggest the constraint(s) be removed or modified.

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

## 7 Related work

Alice [17] 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 [10] 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 [6] 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. [14] 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.

## 8 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 [12] 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.

## References


1. Ajit, S, Sleeman, D, Fowler, DW and Knott, D: ConEditor: Tool to Input and Maintain Constraints, 14th International Conference, Proceedings of EKAW 2004, Whittlebury Hall, Northampton, UK (2004) 466 - 468.
2. Ajit, S, Sleeman, D, Fowler, DW, Knott, D and Hui, K: Acquisition and Maintenance of Constraints in Engineering Design (Poster/Short Paper), Third International Conference on Knowledge Capture (KCAP 05), Banff, Canada (2005).
3. Ajit, S, Sleeman, D, Fowler, DW, Knott, D and Hui, K: Capture and Maintenance of Engineering Design Constraints (Poster), The Twenty-fifth SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence (AI 05), Cambridge, UK (2005) (to appear).
4. AKT, Advanced Knowledge Technologies, [WWW], Available from: http://www.aktors.org/akt/ [Accessed 9 August 2005]
5. Barker, VE and O'Connor, DE: 1989, Expert Systems for Configuration at Digital: XCON and Beyond, Communications of the ACM, 32, 298-318.
6. Burge, JE and Brown, DC: Reasoning with Design Rationale. In J. Gero (Ed.), Artificial Intelligence in Design '00 (Netherlands: Kluwer Academic Publishers.(2000) 611-629
7. Chandrasekaran, B, Josephson, JR and Benjamins, VR: Jan/Feb 1999, What are ontologies and why do we need them?, IEEE Intelligent Systems, 14, 20-26.
8. Coenen, FP: A Methodology for the Maintenance of Knowledge based Systems, In Niku-Lari, A. (Ed), EXPERSYS-92 (Proceedings), IITT-International, Gournay sur Marne, France (1992) 171-176.
9. Fowler, DW, 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).
10. Goonetillake, JS and Wikramanayake, GN: Management of Evolving Constraints in a Computerised Engineering Design Environment, Proceedings of the 23rd National IT Conference, Colombo, Sri Lanka (2004).
11. 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, AAAI Press, (2001) 46-53.
12. Jena, A semantic web framework for Java, [WWW], Available from: http://jena.sourceforge.net/index.html [Accessed 9 August 2005]
13. OWL, Web Ontology Language, [WWW], Available from: http://www.w3.org/TR/owl-features/ [Accessed 9 August 2005]
14. Regli, WC, Hu, X, Atwood, M and 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, vol.16, 209-235.
15. Rolls-Royce, Derby, UK, [WWW], Available from: http://www.rolls-royce.com/ [Accessed 9 August 2005]
16. 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, Seattle, USA (1987) 824-829.
17. Urban, S: ALICE: an Assertion Language for Integrity Constraint Expression, Conference on Computer Software Applications, (1989).