The role of ontologies in creating & maintaining corporate knowledge: a case study from the aero industry

Derek Sleeman[1], Suraj Ajit[1], David W. Fowler[1], & David Knott[2]
[1]*Department of Computing Science, University of Aberdeen, Scotland, UK*
Email: {sajit, sleeman, dfowler}@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). The evolving design is described against a jet engine ontology. 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.

Introduction

The context for the principal system reported here, ConEditor+ [2], is the Designers' Workbench [9] that has been developed to enable a group of designers to produce cooperatively a component that conforms to the component's overall specifications and the company's design rule book(s). One can view the design rule book(s) as an important depository of corporate knowledge, in a company whose expertise is principally in the design and maintenance of aero-engines. Moreover, we are arguing that the Designers' Workbench is an interactive environment in which this corporate knowledge is applied; further ConEditor+ allows engineers to capture and maintain (refine) these constraints (this corporate knowledge). Further, as we shall demonstrate, an ontology for describing jet engines has a central role in both these systems. Sections 1 and 2 provide an introduction to the Workbench, a description of the problem(s) faced and the need for ConEditor+. Section 3 gives a brief overview of our system ConEditor+. Section 4 then focuses on the maintenance aspects of constraints with a description of our approach and its implementation for the domain of kite design. We discuss evaluation and results in section 5. The conclusions and plans for future work follow in section 6.

The issues faced in KB maintenance within engineering were first raised by the XCON configuration system at Digital Equipment Corporation [3, 18]. 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 [7].

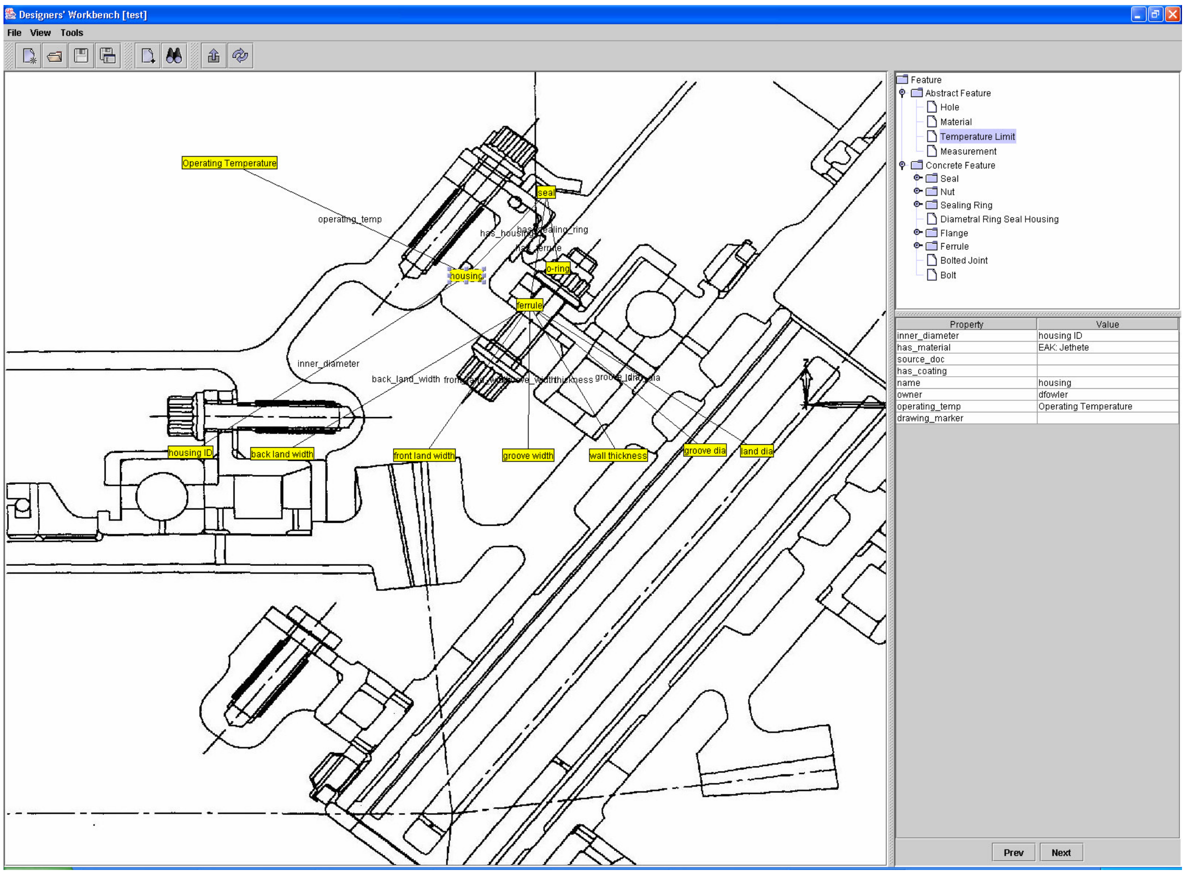
The engineering design process has an evolutionary and iterative nature as designed artifacts often develop through a series of changes before a final solution is achieved. A common problem encountered during the design process is that of constraint evolution, which may involve the identification of 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 the various maintenance problems, systems that capture and represent the rationales associated with design knowledge have been developed. Design rationales [5, 15] capture the following types of information:

- the reasons why a design decision was taken
- the design alternatives considered with reasons for acceptance or rejection
- 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 condition* 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.

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 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 we have developed the Designers' Workbench, which seeks to support these activities.



Property	Value
inner_diameter	housing ID
has_material	EAK: Jethete
source_doc	
has_coating	
name	housing
owner	dfowler
operating_temp	Operating Temperature
drawing_marker	

Figure 1: A screenshot of the Designers' Workbench

The Designers' Workbench (figure 1) uses an ontology [6] 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. When a new design is input into the system an engineering drawing is provided as a graphical backcloth, and the various parts are annotated using the domain ontology. Figure 1 shows the result of such an annotation exercise; the relevant ontology displayed in the top right hand corner can be expanded to show sub-classes, properties, and relations. 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 often resolve the violations by adjusting the property values of the affected features. On selecting a feature, a table provides the corresponding properties and their values. These property values can then be adjusted to resolve the constraint violations.

2 Capturing the knowledge in the design rule books

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 succinctly in the 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 [17].

It would be useful if a new constraint could be formulated in an intuitive way, by selecting classes and properties from the appropriate 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.

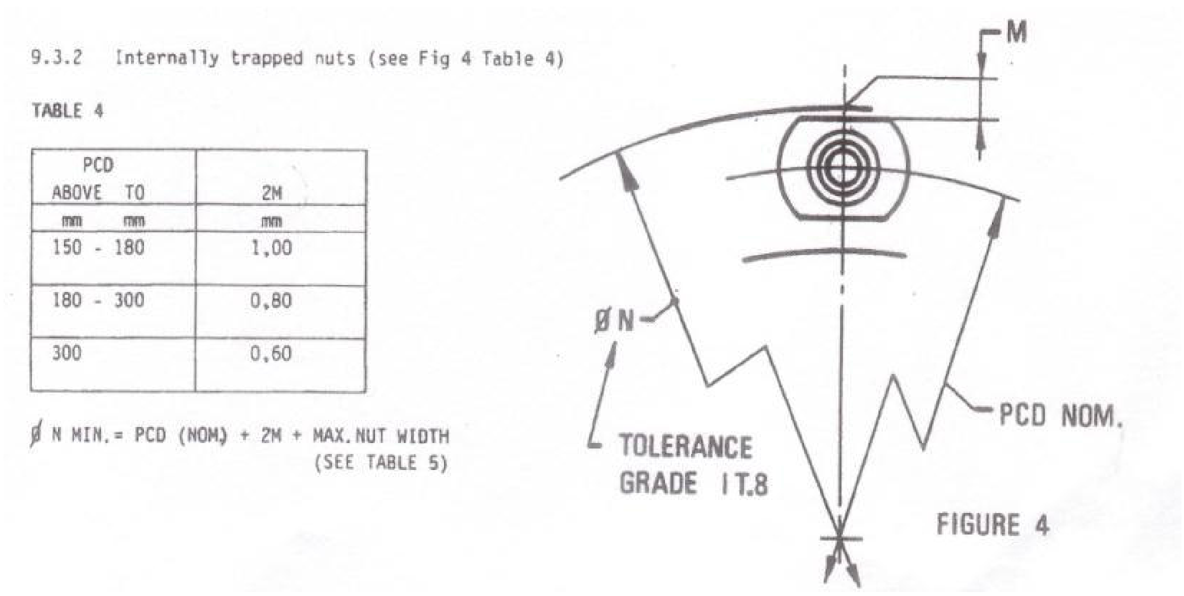

9.3.2 Internally trapped nuts (see Fig 4 Table 4)

TABLE 4

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

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

M
ØN
TOLERANCE
GRADE IT.8
PCD NOM.
FIGURE 4

Figure 2: A constraint as expressed in a rule book

3 ConEditor+

ConEditor+ is a tool to enable domain experts 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 the taxonomy (domain ontology) and combining them with a pre-defined set of keywords and operators from the high level constraint language, CoLan [4, 10]. 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) Functions Panel, (D) Taxonomy / Ontology Panel, (E) Tool Bar and (F) Result Panel (see figure 3). The user can then choose the appropriate entities by clicking the mouse and so form a constraint expression. The taxonomy in the top right hand window shows that the object under discussion is a kite, and shows the various properties and sub-properties of kites (e.g., has_bridle where the bridle has the property has_material, etc). More details about the GUI can be found in [1]. 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. Each constraint is allocated a unique identification number (ID). Hence, when a constraint is revised it is stored as a newer version. This enables the user to study how the constraint has evolved over time. The system provides facilities to retrieve constraints using keyword-based searches i.e., retrieve all constraints containing specified keyword(s) or associated with a specified ID.

4 Maintenance of constraints

Due to restricted availability of designers' time and for simplicity, we have used a kite domain for our study [8, 19, 20]. (We plan shortly to revisit the constraints underlying the Designers' Workbench, acquire the associated application conditions, and then apply a comparable analysis to that enhanced KB.) Consider the following constraint from the kite domain 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."

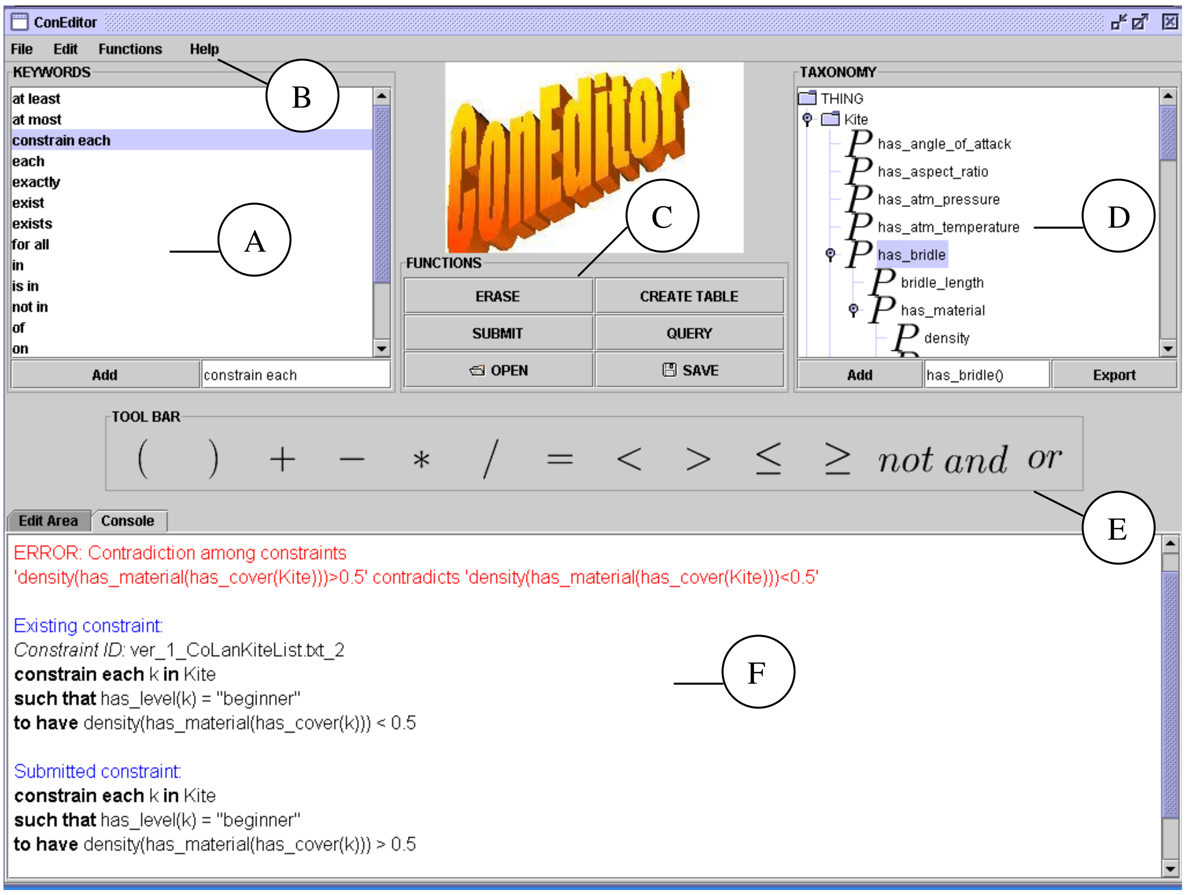
Application condition – "This is applicable to stunt kites of standard size in strong winds only."

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, our approach has a number of stages:

- 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 support verification and maintenance

[1] decaNewton, a common metric unit of force.



ERROR: Contradiction among constraints
'density(has_material(has_cover(Kite)))>0.5' contradicts 'density(has_material(has_cover(Kite)))<0.5'

Existing constraint:
Constraint ID: ver_1_CoLanKiteList.txt_2
constrain each k **in** Kite
such that has_level(k) = "beginner"
to have density(has_material(has_cover(k))) < 0.5

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

Figure 3: A screenshot of ConEditor+

We have extended ConEditor+ so that the user (the domain expert) can associate an application condition with each of the constraints. 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 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.

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) [10]. We give below a typical constraint and its application condition 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))
```

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; however, this constraint is only applicable to flat diamond-shaped kites.

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:

```
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
```

Again, 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).

2. Contradiction

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

b) 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 a) contradicts the constraint in b). The domain expert is notified of this fact and allowed to take the appropriate action (modify or delete).

3. Redundancy

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

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

Conventional_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 a) or b) is redundant. The domain expert is notified of this fact and allowed to take appropriate action to eliminate redundancy.
Similarly, ConEditor+ has a facility that enables several constraints to be fused. This will be described in Ajit's forthcoming thesis.

Implementation: ConEditor+ is implemented in the Java programming language; the domain ontology in the Web Ontology Language [14] is developed using Protégé [13] and read using Jena [11]. Any syntactic errors among constraints are detected by ConEditor+ with the help of a Daplex compiler [4]. 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) 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 refine the knowledge base by using the appropriate functions of ConEditor+ to delete, modify, or shelve constraints. Truth-tables are used to determine the equality of two expressions having the same types of quantifiers and predicates connected with different Boolean operators. Expressions are converted into conjunctive normal form (CNF) wherever necessary. However highly complex expressions would require the use of standard theorem provers [12, 16].

5 Evaluation and Results

Before adding the application condition feature, we performed an evaluation of ConEditor+. A demonstration was given to a group of 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 application condition feature, we conducted several experiments including:

Experiment: We gave demonstrations of ConEditor+'s features 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 constraints and their associated application conditions. 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 for improving the usability of ConEditor+.

6 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. To function, the designers' workbench must be provided with a set of task specific requirements, and generic (company-wide) design constraints. The latter 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.

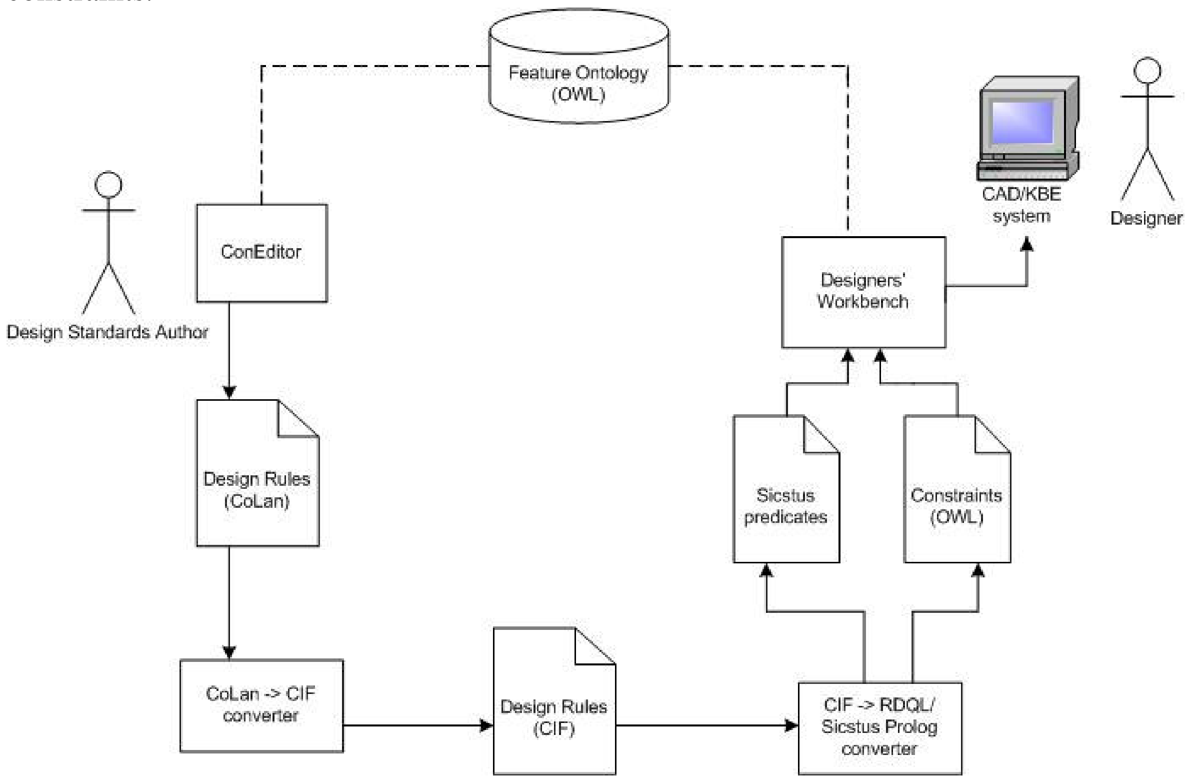


Figure 4: Proposed system architecture

We hypothesize that 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 the studies done in the domain of kite design, we believe the above hypothesis to be true, and also believe ConEditor+ is 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 (figure 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 or KBE system. Additionally, it is desirable for experienced designers to be able to indicate when current constraints or application conditions need modifying as they are inconsistent with their experience. Such modifications of the corporate knowledge need to be done consistently if such KBs are to capture the company's "cutting edge" knowledge.

7 Postscript

Within the last year we have started a further project with Rolls-Royce which is funded by the DTI. The role of the IPAS project (www.3worlds.org) is to make available to the designer in an assessable form information collected during the servicing and maintenance of engines. The huge amounts of detail collected during servicing has not been regularly reported to the design teams. Maintenance teams may be aware, for example, that bolts in certain positions in the combustion chamber deteriorate, and hence need replacing, more frequently than others. This is the sort of "routine" information which we expect to make available to the designers in the future through the so-called Knowledge Desktop. Ontologies are important components of the Knowledge Desktop as mapping between the ontologies used to represent the designers' and the service engineers' perspectives is central to this architecture. Simplifying somewhat, the designers' ontology will reflect the sort of information which the designers would like to acquire (for example about the most frequently failing parts on a particular engine, information about its deterioration mechanisms etc). The service ontology on the other hand is populated with data about service events and also links through to primary service reports. Queries posed by the designer will then initiate a transfer (a mapping) of the information between these two ontologies.

Although the work is still at an early stage we have implemented an ontology-based Web Service which enables designers to ask a narrow range of questions about some aspects of the service data; and hence this service gives some idea of the functionality to be provided by the Knowledge Desktop. Moreover, in this project we are encountering many of the problems of contemporary ontology engineering:

- ontology creation (seeking to develop ontologies systematically and to ensure that relevant aspects of trust and provenance are captured);
- 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 greater detail may be required).

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. The IPAS project is funded by Rolls-Royce and the Department of Trade and Industry under the Technology Program, DTI Reference TP/2/IC/6/I/10292.

References

[1] Suraj Ajit, Derek Sleeman, David W. Fowler and David Knott. ConEditor: Tool to Input and Maintain Constraints. In 14th International Conference on Engineering Knowledge in the Age of the Semantic Web, Proceedings of EKAW 2004, pages 466 - 468, Whittlebury Hall, Northampton, UK, 2004.

[2] Suraj Ajit, Derek Sleeman, David W. Fowler, David Knott and Kit Hui. Acquisition and Maintenance of Constraints in Engineering Design. In Proceedings of the Third International Conference on Knowledge Capture (KCAP 2005), pages 173-174, Banff, Canada, 2005.

[3] 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.

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

[5] 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.

[6] B. Chandrasekaran, J. R. Josephson and V. R. Benjamins. What are ontologies and why do we need them? IEEE Intelligent Systems, 14 (1): 20-26, 1999.

[7] 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.

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

[9] 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.

[10] 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, pages 46-53, Seattle, USA, 2001. AAAI Press.

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

[12] W. McCune and L. Wos. Otter - The CADE-13 Competition Incarnations. Journal of Automated Reasoning, 18 (2): 211-220, 1997.

[13] 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.

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

[15] 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.

[16] S. Schulz. E - A Brainiac Theorem Prover. Journal of AI communications, 15 (2/3): 111-126, 2002.

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

[18] 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.

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

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