Acquisition 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, AB24 3UE, 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 a 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. 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.

Categories and Subject Descriptors

I.2.6 [Artificial Intelligence]: Learning – knowledge acquisition; I.2.4 [Artificial Intelligence]: Knowledge Representation Formalisms and Methods; J.2 Physical Sciences and Engineering- aerospace, engineering

General Terms

Design, Experimentation

Keywords

Constraints, Application conditions, ConEditor, Maintenance

INTRODUCTION

In this short paper, firstly, we briefly describe how ConEditor [1] helps design engineers in the acquisition of constraints that are then used by systems such as Designers' Workbench [2] to support the design activities. The main aim of ConEditor is to enable domain experts themselves to capture and maintain the constraints, relieving the knowledge engineer from this tedious, error prone and time consuming task. Secondly, we give a sketch of our planned approach towards the maintenance of constraints.

Acquisition

ConEditor enables acquisition of constraints in the form of a high-level constraint language known as CoLan [3]. A simple constraint expressed in CoLan is as follows:

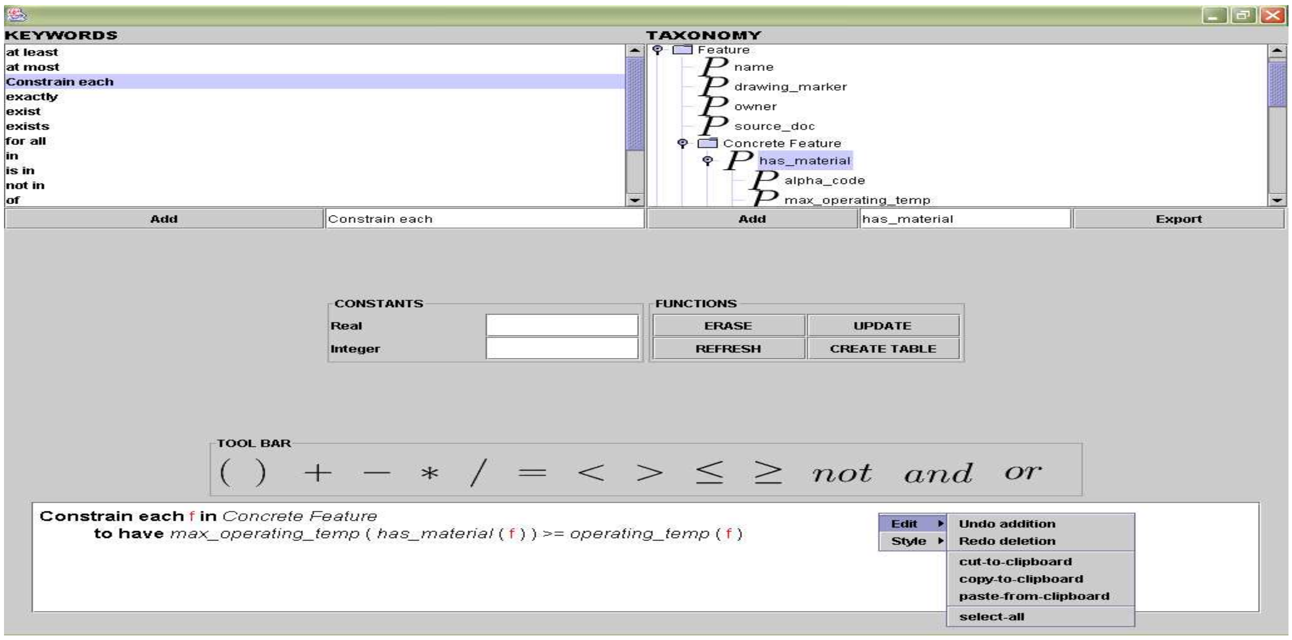
```
Constrain each f in Concrete Feature
to have max_operating_temp(has_material(f))
>= operating_temp(f)
```

This constraint states "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 (Figure 1) consists of five components, namely, keywords panel, taxonomy panel, central panel, tool bar and result panel. More details about each panel and how to express a constraint using ConEditor can be found in [1].

Maintenance

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. In order to tackle the various maintenance issues/problems, our proposed approach 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



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

Figure 1 Screenshot of ConEditor's GUI

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.

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:

Inconsistencies

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:

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
```

It can be inferred that c) subsumes d) as the application conditions in d) are included in those of c). Similarly contradiction and redundancy can be detected among constraints and their application conditions.

REFERENCES

[1] Ajit S., Sleeman D., Fowler D. W. and Knott D., ConEditor: Tool to Input and Maintain Constraints, EKAW 2004, UK, pp. 466-468.

[2] Fowler D., Sleeman D., Wills G., Lyon T. and Knott D., Designers' Workbench, AI 2004, Cambridge, UK, pp. 209-221.

[3] 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.