INTERNATIONAL CONFERENCE ON ENGINEERING DESIGN
ICED 01 GLASGOW, AUGUST 21-23, 2001

VERIFICATION OF REDESIGN MODELS: A CSP APPROACH

Inés Arana and Hatem Ahriz

Keywords: Design representations, product modelling, constraint-based design, knowledge representation.

1 Introduction

One of the initial stages in the manufacturing of an article is the production of a design which satisfies a set of user requirements. Much of the design work carried out in manufacturing companies can be classified as redesign, i.e. producing a new variant of a known product family which satisfies a slightly different specification. This type of redesign application is particularly well suited to computer support since: (i) the knowledge used is well-understood; (ii) solutions are highly reusable. Redesign support systems often involve the creation of a model, which is then used by the advisory system. These models rely on constraints to express a substantial amount of design knowledge, e.g. design laws, principles, company restrictions, international standards and customer requirements. Although a lot of effort has been devoted to the verification of rule-based knowledge [1], little effort appears to have been spent on constraint verification. This paper presents a methodology to verify constraints in a redesign model and argues that verification can be seen as a Constraint Satisfaction Problem (CSP).

The MAKUR methodology [2] provides a systematic approach to the construction of customised knowledge based systems which support the redesign of a family of products. MAKUR consists of three tools:

- Design Analysis Methodology (MADAM): a tool aimed at eliciting and analysing a company's expertise in the redesign of a family of products.
- Design Description Language (MADD): a formal specification language that is used to represent the knowledge acquired through MADAM. A Graphic Editor (MAGE) is used in order to support the knowledge engineer in the construction of redesign product models.
- Design Advisory System (MADAS): an interactive problem-solving tool, which provides particular design solutions for a product, given a set of requirements. It is a generic tool, and is customised by the use of a specific product model, which is described using the Design Description Language. The core of MADAS is a constraint engine, which ensures that the design is consistent at all times. As the designer makes decisions, their effects are propagated throughout the model, ensuring that all constraints are satisfied. Hence, if the designer makes an unsatisfactory decision, MADAS warns him about it. In this way, MADAS prevents the designer from making invalid choices.

The remainder of this paper describes a CSP approach to redesign knowledge verification. Section 2 presents the design description language MADD. Next, section 3 defines and explains the benefits of CSP approaches. The use of CSP techniques in redesign model verification is discussed in section 4. Finally, section 5 presents some conclusions.

2 MADD: a design description language

MADD provides a three-view representation of the design of a product family:

- *Physical model*: describes the product in terms of:

  – *Assemblies*: collections of parts, e.g. a cylinder head

  – P*arts*: non-decomposable physical elements, e.g. a pipe. *Part-types* are generic descriptions of particular kinds of parts and, thus, a part may be associated with a part-type.

  – *Design features*: characteristics of a part which give them some functionality, e.g. a pin-hole.

  Thus, an article is composed of one or more assemblies which, in turn are collections of parts. Each part may have design features, which give it some functionality.

- *Functional model:* presents the product functionality in terms of:

  – *Concepts* which it must satisfy, e.g. join two pipes.

  – *Technical solutions* which implement a concept, e.g. two pipes which are screwed together.

  – *Design features* used in a technical solution (e.g. the pipes have some threads so that they can be screwed together) link this model with the physical one.

  Thus, an article satisfies one or more concepts. Each of these concepts is implemented by means of a technical solution which may use some design features in order to provide the desired functionality.

- *Process model:* describes the

  – *Tasks* which need to be executed in order to obtain a product design, e.g. establish a joint between two pipes

  – *Methods* which can be used in order to perform each task, e.g. screw two pipes, weld them together, join them using a flange, etc. Tasks and methods are related to entities in the functional/physical model.

  Thus, an article is designed by carrying out one or more tasks. Each task can be achieved by several methods, each of which is implemented by carrying out a set of tasks.

The properties of all model entities (e.g. assemblies, parts, concepts, etc.) are described by parameters which are represented by their type and their domain. Constraints are used to specify the relationships between parameters: (i) of the same entity, e.g. two properties of a part; (ii) of various entities within the same (physical, functional or process) model, such as a parameter from a concept and a parameter from a technical solution; (iii) between entities in different models such as two parameters, one from a part and one from a concept. Both parameters and constraints are owned by an entity [3]. For example, the entity $pipe_1$ may have, among other parameters and constraints, the following:

- A parameter $pipe_1$*.diameter* which can take values between *10mm* and *2000 mm*.
- A parameter $pipe_1$*.material* which can take values in the set *[aluminum, brass, copper, carbon steel].*
- A constraint $pipe_1.diameter = pipe_2.diameter$ specifying that this pipe must have the same diameter as $pipe_2$ (another entity in the physical model).

Models which use the MADD methodology have been used to support the redesign of manufacturing articles. Before a product model can be used in an advisory system to reason about new designs, it has to go through a verification procedure, i.e. a series of checks which ensure that the knowledge used meets the specified requirements of the users [4]. Thus, a system may be measured by the following [5]:

- Did we get it right? I.e. does it meet the user requirements?
- Can we keep it right? I.e. will the system by reasonably easy to maintain?
- Can we do it again? I.e. can the same mechanism be successfully used in other projects?

Redesign models must, therefore, be tested in order to ensure their accuracy, completeness and maintainability. The techniques used in order to check the models must be general, i.e, applicable to a variety of models.

Unfortunately, although the MAKUR methodology guides the knowledge elicitation process, it does not yet provide a verification tool which ensures the correctness of the model.

3 Constraint satisfaction problems (CSPs)

A constraint satisfaction problem is a system which can be defined in terms of [6]:

- A finite set of variables $X = \{x_1, x_2, x_3, \ldots, x_n\}$
- A function which maps each of the variables to a finite domain. I.e., for each variable x_i it associates a domain D_i of possible values.
- A finite set of constraints (logical relations between variables), which restrict the values that the variables may take.

A solution to the CSP assigns each of the variables a value in its domain. This assignment satisfies all the constraints simultaneously. CSPs are, therefore, combinatorial problems which can be solved using search techniques. However, conventional search algorithms are very time-consuming and, therefore, current research focuses in reducing the expense of CSP solving. It should be noted that some CSPs are computationally intractable, i.e. NP-hard.

The use of CSPs (as opposed to mathematical programming) has the following advantages [6]:

- The description of the problem is often much closer to the real problem. For example, the variables on the CSP correspond to parameters in a redesign problem, and constraints correspond to actual restrictions in the problem at hand.

- CSP algorithms are at times, quicker than integer programming methods.

A substantial amount of effort has been devoted to the development of techniques which solve CSPs efficiently.

4 Redesign model verification

The MAKUR product model describes the various alternatives available in the design of a product family and the conditions (constraints) which must be met for each of these choices to be a suitable option. In order to verify a model, the validity of each of the options available to the designer must be checked, i.e. there must be at least one valid solution which takes that option. This verification process is not straightforward since each alternative is usually highly connected (related through constraints) to other choices in the same and other models (physical, functional and process). These other options are in turn connected to other alternatives and so on, so the constraints associated to all these choices must be satisfied simultaneously. Model verification can, therefore, be seen as a constraint satisfaction problem (CSP), where the feasibility of each design alternative is checked by ensuring that there is at least one set of parameter values (i.e. solution) which satisfies all the constraints related (in some way) to that choice. Hence, we have used CSP techniques [7] and a constraint library (Ilog Solver) [8] in order to verify redesign product models.

A simplified version of our algorithm is as follows:

Let P be the set of all paths (from the root to the leaves) in all the models (physical, functional and process).

```
Repeat

    Take a path $p \in P$, and collect all the entities involved in it in a set $E$
    Let    $Par = \varnothing$ (the set of parameters collected to date)
           $Cons = \varnothing$ (the set of constraints which have to be checked)
           $Inv = \varnothing$ (the set of parameters involved in selected constraints)

    Repeat
```

```
        Take an entity $e \in E$, $Par = Par + e$.parameters, $Cons = Cons + e$.constraints where $e$.parameters are all the parameters belonging to entity $e$ and $e$.constraints are all the constraints belonging to entity $e$

        For every constraint $c \in Cons$, $Inv = Inv + c$.parameters
        Where $c$.parameters are all the parameters involved in constraint $c$

        For every parameter $i \in Inv$ such that $i \notin Par$, take its owner (i.e. the entity it belongs to) and all its predecessors (and siblings for tasks) and add them to $E$

    Until all entities in $E$ have been considered

    Pass $Par$ and $Cons$ (i.e. a CSP) to the constraint problem solver which will check whether there is a solution for that CSP

    If the constraint problem solver detects constraint insatisfiability, then the option (path) $p$ is not consistent, and the knowledge engineer is informed of this

Until all paths in $P$ have been considered
```

Fig. 1 illustrates a simplified model of redesign. The ‘dotted arrows’ represent constraints between entities. In order to keep this example as simple as possible, we have only included a few constraints between entities and internal constraints (relating parameters in the same entity) are not represented. Assume that our algorithm selects the path *p =[Ph1, Ph2, Ph5, P10]*. For each of the entities in this path, they are included in *E*, its parameters are included in *Par* and its constraints in *Cons*. Hence, *Cons* will contain a constraint which relates *Ph5* to *Ph7* though their parameters. *Ph7* will, therefore, be included in *E*.

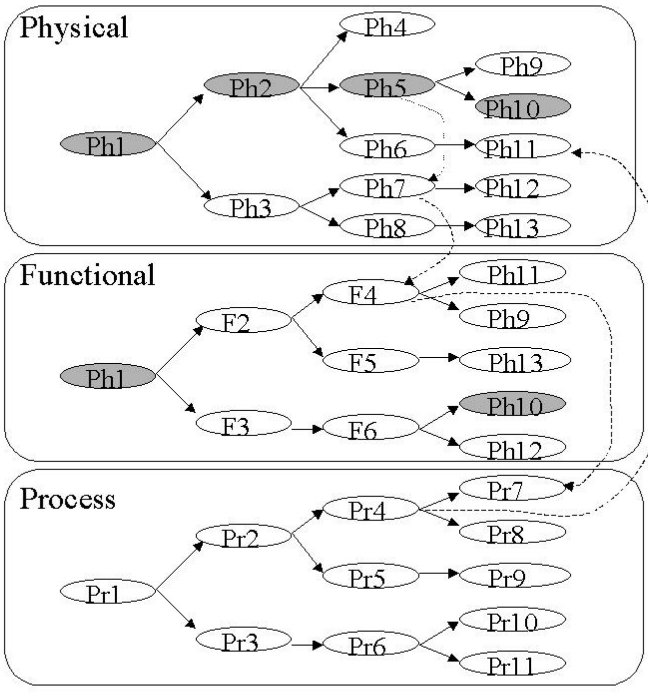

Figure 1: A path selected for checking (represented by shadowed nodes)

Since *Ph7* $\in$ *E*, all its predecessors (*Ph3* and *Ph1*) are inserted into *E*, their parameters are included in *Par* and their constraints in *Cons*. Also, because *Ph7* is related to *F4*, the latter entity will be included in *E* (and its parameters are included in *Par* and its constraints in *Cons*).

Similarly, since *F4* $\in$ *E*, all its predecessors (*F2* and *Ph1*) are inserted into *E*, their parameters are included in *Par* and their constraints in *Cons*. Also, because *F4* is related to *Pr7*, the latter entity will be included in *E* (and its parameters are included in *Par* and its constraints in *Cons*).

Since *Pr7* $\in$ *E*, all its predecessors (*Pr4*, *Pr2* and *Pr1*) and sibling tasks for each of *these (Pr 5*) are inserted into *E*, their parameters are included in *Par* and their constraints in *Cons*.

Also, because *Pr4*$\in$ *E* is related to *Ph11*, the latter entity will be included in *E* (and its parameters are included in *Par* and its constraints in *Cons*).

Since *Ph11*$\in$ *E*, all its predecessors (*Ph6*, *Ph2* and *Ph1*) are included in *E*, their parameters are included in *Par* and their constraints in *Cons*.

Through the above process, we have collected in *Cons* all the constraints related to the original path (user option) and in *Par* all the parameters which may be involved in constraints contained in *Cons*. Fig. 2 shows the entities related to this path. Thus, a constraint solver can be used to check whether the chosen user option is satisfiable, by asking it to check whether there is any solution for *Par* (assignment of a value to each parameter) which satisfies all the constraints in *Cons*. If there is such a solution, the selected user option is viable; otherwise, this option is incorrect.

The following anomalies may be found in a product model:

- *Conflict or incoherence:* there are no valid parameter values which can satisfy a single constraint. Our verification tool detects this error, and informs the knowledge engineer so that the model can be corrected. This may involve further knowledge acquisition and analysis. E.g. if the parameter *$pipe_1$.diameter* can take values between *10mm* and *2000 mm,* the constraint *$pipe_1$.diameter > 2100* is not satisfiable.

- *Partial conflict*: a whole set of the values of a parameter's domain are never valid. In this situation, our verification tool reduces that parameter's domain to exclude that set of values, thus correcting the anomaly. E.g. if the parameter *$pipe_1$.diameter* can take values between *10mm* and *2000 mm* and there is a constraint *$pipe_1$.diameter > 1500* the domain of. *$pipe_1$.diameter* is reduced to between *1501mm* and *2000 mm.*

- *Incompatible entities:* two (or more) entities must be selected together but they have incompatible constraints. Our verification tool detects this error, and informs the knowledge engineer, who is responsible for correcting the error. E.g. if the parameter *$pipe_1$.diameter* can take values between *10mm* and *2000 mm* and the parameter *$pipe_2$.diameter* can take values between *2100mm* and *3500 mm* and there is a constraint *$pipe_1$.diameter = $pipe_2$.diameter* which is active when both *$pipe_1$* and *$pipe_2$* are selected, then *$pipe_1$* and *$pipe_2$* are incompatible.

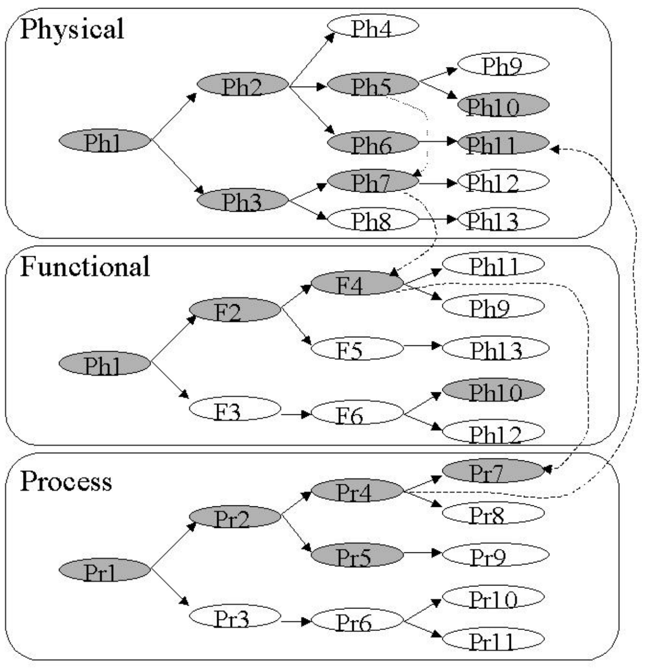

Figure 2: Entities related to [Ph1,Ph3, Ph5, Ph11] (represented by shadowed nodes)

- *Deficiency or incompleteness:* some of the values within a numeric parameter's domain, represented by its minimum and maximum value, are not valid. Although strictly speaking, this information is not invalid, it is misleading, since it implies that there are more options (parameter values) available than there really are. Our verifier does not detect this problem. It should be noted that the specification of all the valid parameter values may, overall, be more confusing for the user. E.g. if the parameter *$pipe_1$.diameter* can take values between *10mm* and *2000 mm* if the parameter *$pipe_1$.length* can take values between *30mm* and *6000 mm* and there is a constraint *$pipe_1$.length = $pipe_1$.diameter * 3* then only some of the values in the domain *$pipe_1$.length* are valid (*30, 33, 36, 39, … 6000*).

- *Redundancy ("equivalent" constraints):* two entities contain equivalent constraints. It should be noted that this is perfectly reasonable if there are solutions where only one of the entities is selected. Hence, our verifier does not detect this anomaly. Even if two constraints were identical and appeared to be always be used together in the same solutions, this might change if the product model was extended to incorporate more alternatives. It should be noted that this "anomaly" does not affect the use of the design support system. E.g. the constraints *$pipe_1$.diameter $\geq$ $pipe_2$.diameter* and *$pipe_1$.diameter $\leq$ $pipe_2$.diameter* are equivalent to *constraint $pipe_1$.diameter = $pipe_2$.diameter.*

5 Conclusion

The success of support systems often depends on the quality of the domain knowledge they use and thus, domain models should be verified. Redesign problems are well suited to the construction of support systems, since they have well-understood domains and the knowledge required to solve them is highly reusable. In this paper, we have explained a representation for redesign knowledge and presented a constraint-based approach to verifying that a redesign product model is "right”.

References

[1] Gupta, U.G., “Validating and Verifying Knowledge-based Systems”. *IEEE Press,* Los Alamitos, CA, 1990.

[2] Arana, I., Ahriz, H. and Fothergill, P. “Improving Re-Design Support.” In *Proceedings of IDPT’2000 – The Fifth International Conference on Integrated Design and Process Technology*, Texas, 8 pages, Society for Design and Process Science (SDPS), June 2000.

[3] Ahriz, H., Arana, I. and Fothergill, P. “Using Constraints in Modelling for Re-Design.” In *Proceedings of EIS2000, Second International Symposium on Engineering of Intelligent Systems*, 27-29 June 2000, Scotland, pp. 314-319. Published by: ICSC Academic Press.

[4] Preece, A. “Building the Right System Right.” In *AAAI-98 Workshop on Verification and Validation of Knowledge-Based Systems*, Technical Report WS-98-11, AAAI Press, 1998.

[5] Preece, A. “Evaluating Verification and Validation Methods in Knowledge Engineering. ” In *Industrial Knowledge Management: A Micro-level Approach*, Springer-Verlag, London, 2000, pp. 91-104.

[6] Tsang, E.P.K., “Foundations of Constraint Satisfaction. ” *Academic Press*, London and San Diego, 1993.

[7] Barták, R. “Constraint Programming: In pursuit of the Holy Grail.” In *Proceedings of WDS99 (invited lecture),* Prague, June 1999.

[8] ILOG Solver 4.4 Reference Manual, June 1999.

Corresponding Author:

Inés Arana (Dr.)
School of Computer and Mathematical Sciences
The Robert Gordon University,
St Andrew Street
Aberdeen
AB25 1HG,
U.K.

Telephone: (+44) 01224 262729
Fax: (+44) 01224 262727
E-mail: ia@scms.rgu.ac.uk