Customer-Based Design with Constraint Reasoning

Gunilla Sivard, Lars Lindberg, Erik Agerman (2)

Received on January 15, 1993

This paper identifies some principles for computer-aided customer-based design. In this type of design, technical prerequisites and the range of customer requirements are typically the same over time. From time to time there are larger changes in manufacturing methods and customer needs, though. Thus the degree of automation in a design support system can be fairly large, but it still needs to be interactive and flexible for changes. A case study based on an industrial application, implemented using a commercial CAD system and a CLP (Constraint Logic Programming) system, is described. Special emphasis is put on interactivity - the collaboration between designer and automated design system.

Keywords: Interactive design systems, Automated design, Constraint Logic Programming

1 Background

The design of products which are highly customized can be divided into two phases: the development phase and the order phase [Lindberg 91]. The order phase typically consists of activities such as functional design, NC programming, and process preparation. For some design problems, these activities are today automated by specialized computer programs. These programs are usually efficient for their specific application, but are narrow and hard to maintain and modify. Thus these systems are mainly used for established products with high volumes and a fairly low degree of customization, eg. power transformers and special ballbearings. For smaller series, the design should be done at a lower total cost. The total cost includes system development and maintenance as well as runtime costs.

In this paper we describe a testcase based on an industrial application. For this application, a preliminary design aid was implemented using a commercial CAD system and a CLP (Constraint Logic Programming) system. The development of this design aid has resulted in some principles for computer-aided customer-based design. It has been found that in order to generate a system that is cost efficient over the whole life cycle of an order based product, it is important that the system is flexible (easy to update), interactive, benefitting from the capabilities of both designer and computer, and modular, using commercially available shells as far as possible.

2 Approach

A design problem in Swedish industry was studied and a test system implemented. The experience from this work has formed the basis for specifications of a general order design aid and revealed some general characteristics of computer systems for order design. Our plan is to go forward and test these characteristics on other design problems for verification and revision until a solid set of general specifications is formed.

2.1 Pilot project

As a pilot project, a design problem from ABB Transformers in Ludvika was studied; design of a pipe connection between the transformer tank and an oil conservator. The design task is in principle to connect two pipes in space using a set of different kinds of pipes under space and inclination constraints. A design aid was implemented for this task using Intergraph's CAD system, and SICSTUS Prolog with PCS (Prolog Constraint Solver).

ABB has had an automated system (APDS, ASE Parametric Design System) for their medium sized transformers for a long time. This system is in principle a simple selectable-table program written in FORTRAN. The structure of the product is known in advance, but the specific components and some of their attributes can be chosen according to the customer specification. This system is still in use, but when changes have to be made to the preconditions or results, the only way is to go back to paper and pencil, or to an ordinary CAD system. The order system once was fully automatic, but, partly due to the devastating consequences of a product failure [Agerman 91], it had to be broken down into smaller pieces that could be evaluated, verified, and modified by a human. Contrary to the medium sized transformers, large transformers are mostly designed at the drawing board (or CAD system) since they vary too much to divide into a set of predefined variants. Still, large parts of the design principles, as well as specific solutions, are known in advance and could serve as a basis for a computerized design aid.

Thus ABB has an interest in a design system which can automate the parts that are well known, but still be open enough to permit (and facilitate) special design cases, additions and changes, in a well structured manner.

2.2 Design description

The design of the pipe connection between the transformer tank and the oil conservator of large transformers is an example that was recommended by the responsible designer, since it is a limited problem that still reveals many of the problems in order design. Some of these problems are:

- trading off between many requirements in a small design space
- checking that all constraints are satisfied, both for the initial design and after changes. Update dependent variables when changes are made.
- remembering all the different steps which have to be performed
- remembering which components should be present, and their alternatives
- generating detail drawings from 3D model

The designer also pointed out the desire of having access to information about old solutions, both drawings and experience. This is a type of "indirect" design activity [Peters 90] that a final order design system should probably support. Unfortunately, this issue falls out of the scope of this paper.

Figure 2-1 Sub assembly of pipe connection

The transformer tank and the oil conservator is to be connected using around seven components. The components are pipe-angles with variable lengths and angles, and geometrically static components (such as gaugedetectors and valves) which come in different variants. There are different constraints: the pipes cannot be longer than 1 meter, the inclination up to the gaugedetector must be over 3 degrees, the distance between the components and highvoltage parts on the transformer must be larger than a certain value, etc. There are different ways of configuring the components, but usually one of two configurations is used. The main task is to place the components in the electrically safe distance to highvoltage parts while keeping all the other constraints satisfied, a task which is difficult when the permitted space is small.

2.3 Description of implemented system

An interactive system for the pipe connection problem was implemented in the fall of 1992.
This system generates an initial design suggestion based on the requirements and default values on components. This provides information to the designer about the incoming components and gives a first idea of how the connection should be configured. The designer then modifies the geometry of the solution or change details in the specification. He can for example move the gas detector so that it is in a safe position, and specify that the pipes should be shorter than 870mm. The system then modifies the design according to the changes, and generates a solution which is consistent with all the detailed constraints.

Architecture of the system:
1. Geometrical model and interface to the designer - Intergraph’s CAD system EMS
2. Design module - SICSTUS prolog and PCS (Pico Constraint Solver)
   a. In the constraint system the design model and constraints on the component variables are defined. Values on unknown variables are calculated. The connections between components and constraints on the connections are also represented as constraints.
   b. The configuration of components, generation of values based on default values, and a "precompilation" of dependent requirements are done using Prolog predicates. An example of a dependent requirement: "If a component is located between the transformer bank and the gas detector it should have an inclination over 3 degrees"

2.4 Constraint Logic Programming
A Constraint Logic Programming System (CLP) is a constraint reasoning system integrated in Prolog [Van Honkheirck 89]. In constraint programming, knowledge is represented as variables and relations between these variables, constraining their possible values. Variables and constraints build up constraint networks in which the variable values are kept consistent by the constraint propagation mechanism. In addition to this consistency mechanism, a mechanism for generating values can be added. If all variables are not known, the value of some variable is instantiated and propagated through the network. If the network is still consistent, this value is accepted and another variable is instantiated. If not, the value will be extracted and another value will be tried out. This process goes on until all variables are known, or until it is found that no consistent value combination exists. In principle the generator chooses a variable to instantiate, chooses a value for the variable, and chooses what to do if a value isn’t consistent. Together, the generating and consistency maintaining mechanisms are used for generating a set of consistent values on unknown variables. Large parts of the design knowledge can be naturally represented as constraints, and the synthesis process, where values are assumed and tested, is a natural and efficient way of representing part of the design process. The rules of logic are needed for the more verbose and complex representation [Genesereth&Nilsson 87]. Thus CLP is a natural choice for representing design knowledge as well as the design verification and synthesis mechanisms.

3 General Design aid for Order based design

3.1 Characteristics of design tasks
The order design process comprises many design tasks with different degrees of complexity. There is a span from pure routine design with few, well defined choices, to a more creative design where there is no previous detailed solution. The characteristics of the computer system supporting these different activities depend on the task, the flexibility needed, the space of possible solutions and the probability that this space will change. Another factor to consider is the type of reasoning that is needed in order to reach a solution (analytic reasoning, simulation, decision making based on different results)

We define the tasks in the order design process according to three parameters: uncertainty level, complexity, and frequency of changes. By uncertainty level, we mean the extent to which there are unknowns in the process of generating values on the design variables based on the requirements. Complexity in this context means the complexity of the types of reasoning that has to be performed, and change frequency is of course a measure of how often the preconditions (component libraries and context) are changing. At this point we divide the design tasks into three types: "Track" design, "Road" design and "Offroad" design.

- Track design.
  This is a pure variant and/or parametric design where the problem is very well known, there is only one solution to any input, and no search is needed. In addition, the preconditions change seldom so that flexibility is of less importance. Efficiency and the availability of suitable programs could be prioritized, and the problem can be solved by a specialized program with no generality.

- Road design.
  This is a more complex type of design, well known but requiring a more powerful reasoning. Preconditions change from time to time. Here a system is needed where advanced reasoning mechanisms can be integrated (search and specialized analysis tools e.g. FEM) and where generality is easier to encode.

- Offroad design.
  Here we have a creative type of design. It is still fairly well known, but it involves reasoning that is not totally known in advance (or which is hard to elicit) and requirements that are hard to satisfy and trade off. An interactive system that can make use of the designer’s capabilities is required.

Figure 3-1 Characterization of order design tasks
The task of selecting a type of system for the design task at hand is cost related. The total cost of a series of order products can be described as [Lindberg 91]:

\[
\text{Initial development cost} + n \times \text{runcost} + \text{maintenance cost}
\]

Initial development cost is the cost of developing a system based on the product template that was created during the development phase. Runcost is the cost of running the system at each order, and a is the total number of orders for one product. In maintenance cost is included the continuous maintenance of the initial code as well as the cost of updating the system to changes in prerequisites. Development cost and runcost is often higher for a more general system than for a specialized system. On the other hand, the maintenance cost for a general system should be lower, since this type of system covers more cases and thus does not have to be changed if the new problem is within the problem space. A change should also be easier to perform since the representation is on a higher level. Apart from the generality issue, we have the issue of how powerful the system should be - the more powerful, the more costly to develop, but the less
costly to run since no (in theory) designer is needed. Thus there is a trade off between having a system that is advanced enough to take over most of the designer's work, or a mixed system with a less complex design system supporting a designer.

In this project, we have concentrated on the creative 'Offroad' design problem. In particular the scope of interest is on the issues of flexibility and interactivity.

3.2 Flexibility

A flexible system, in this context, is a system in which new situations can easily be handled. These new situations were not intended (but possibly foreseen) at system development time.

Flexibility can be reached through:

- **Generality.**
  The program is from the beginning general enough to handle new input and variable values. This means that the changes, if not the exact changes, at least the structure of the changes, were foreseen from the beginning. A general system is often slower and more time consuming to develop.

- **Object programming** [Koenen 89] is one type of generalized programming. Data about components are stored in objects, and the characteristics of a whole class of components can be defined at once. A specific component is easily generated as a specialized instance of a class, inheriting the class attributes.

- **Changeable programs.**
  If the program is easy to overview and understand, changes are facilitated. This can be reached by having programs which are well structured and using a transparent programming language.

  - Declarative programming. In declarative programming the information content of the code is separated from the control, from the mechanism that manages the data. Databases are an example of examples, where data is stored in a database and is fetched and used by routines. Rule-based systems is another type, where data and rules are stored in a knowledge-base that is operated by an inference engine.

  - In design it is natural to deal with objects, and an object-oriented paradigm is preferable. The equations defining the behavior and geometrical characteristics of a component can be represented as constraints using a constraint reasoning system.

  Relations between components, such as how they are connected, are nicely represented using rules. If a is connected to b with a flange connection then the angle in a = the angle in b.

  - Apart from the programming language, there is a need for a model language for mechanical design, where system components, and their characteristics can be expressed on a high level. For a system which will permit a building of a model by just connecting components from a library, a "no function in structure" representation is needed. That is, the behavior of the components is described irrespectively of their context.

- **Dependency mechanisms.**
  A change presumes that it is understood what has to be changed, how it should be changed, and how this change will effect the rest of the system. Thus the whole picture has to be understood to know what other, dependent changes that have to be made. A dependency managing mechanism which could keep track of dependencies, and in some cases effectuate the whole change, would be of great help.

Some examples:

- Linked documents, where in each document there is an indication of which the other related documents are.

- A dependency mechanism where the designer can express dependencies between components or requirements on a high, qualitative, level. One way of implementing this is by using a qualitative representation [Weld & de Kleer 90].

3.3 Interactivity

The pilot project revealed the importance of interactive systems. In an interactive system, the system can facilitate the management of details and make modifications and verifications, and still provide the designer with the possibility of making more advanced decisions. The examples in this section are related to the pipe connection design problem.

The designer's forte is to shape a general layout that "looks good" and to trade off between the modifications that are needed when the requirements are not met. A typical trade off could for example be between adding a supporting rod since the pipes become too long, or moving the connection on the transformer.

The computer system, on the other hand, is efficient at verifying, and in some cases making modifications so, that all the detailed requirements are met. Examples of requirements are: that the pipe lengths are <= limits, that the pipes are rotated in steps of 45 degrees, and that there are no bends in the flangeconnections, that the inclination is over 3 degrees, that the fluepipe isn't twisted, etc.

A system that fully performs all the design tasks would require that the knowledge of the designer be elicited and would also need to have a good representation of this knowledge. It would also require an efficient routing method with the type of reasoning required for pipeline layout in a 3D space with obstacles.

We believe that it is more cost-as well as time efficient to create an interactive design aid that utilizes designer and computer at what is each their specific strength. In some cases it may be more efficient to let the designer solve a conflict, specifically if the solution is involved. In other cases the conflict resolution is so straightforward that the system should do it automatically.

3.4 Integration of existing systems

For a mechanical designer, the most natural way of interacting with a model is through the geometry. Existing CAD systems, which have been developed through many years, provide a geometrical Interface to the designer. The verifying and redesigning tasks require another type of programming, and we strive to utilize existing Prolog and constraint satisfaction systems as far as possible.

3.5 Specifications on the design aid

We will now present an initial cut at defining specifications for a general order-design aid. The contents is based on the result from the pilot project.

3.5.1 Architecture

In this section the different types of knowledge that an order design system should be able to handle is stated, as well as what type of control the system should perform. A first cut at representation methods is given as well.

- The system should be able to reason about different view models [Schmoker 92] of the artifact and its requirements, using contextual rules and restrictions as well as design knowledge like rules of thumb and the order of design steps.

  - Design descriptions
    The models are built using components and subsystems (aggregation of components). The component descriptions include geometrical characteristics and/or other attributes and definitions fundamental to the purpose of the model, such as those pertaining to behavior, manufacturing, etc. The subsystem description consists of ingoing components and the way these are connected. In addition, system characteristics not clearly described by the component composition could be defined on the system level.

  - Requirements on a specific design
    - Geometrical constraints from the surrounding subsystems. These are "derived" constraints [Ullman 88], depending on decisions made during design of surrounding components/systems. These constraints can thus change frequently during the design.
    - Customer requirements
      - These are "introduced" constraints, specific to each order. They change for every order, but do not change (normally) during the design of one order.
    - Context constraints
      These are "given" constraints from manufacturing, company standards, physical laws etc. Constraint constraints are typically the same for all orders but they may change from time to time, e.g. when new equipment is added.

  - Analytical and experience based design rules
    Background knowledge that is needed for the design. The analytical rules typically come from handbooks and the experience based heuristics range from company rules of thumb to the designer's own experience.

  - Design plan
    Activities: Laying out specific components. Assuming values (based on stated default values) on component parameters. Order: The order in which components should be laid out. Alternatives: Alternative configurations and parameter values.

- Control
  All knowledge in the system can be used either as Reminding, Verifying, or Controlling knowledge. The control type depends on the level of automation desired

  - Reminding
    In an interactive system, reminding might be the most efficient use of the information if the activity or information that is reminded about is too difficult to encode or too time consuming to use by an automated procedure.

  - Verifying
    For cases when it is enough to verify the consistency of a result, e.g. that a design suggestion meet the requirements. Verification of many details is often difficult for the designer and doesn't require a design mechanism of the system.

  - Controlling
    In this case the constraints and rules are used for automatically generating a design suggestion, or part thereof.
The system should be able to facilitate the designer by visualizing information, pointing, and verifying about the design task, generating a new set of consistent variable values, it will normally just flag that the values are inconsistent. When the designer makes a change, he would like to get a consistent solution that is close to what he intended. Thus the system needs to be able to generate a new set of values based on a set of inconsistent ones. This requires a routine that changes values based on what variables could change and how much, in what order, etc. We have some ideas of how to solve this problem, but a deeper analysis needs to be done.

- How is the requirement that is causing a constraint violation flagged?

5 Conclusion

Order-based design is a mixture of design tasks with different complexity. A computer system for order-based design needs several mechanisms, automating a larger or smaller part of the process. In order to create a system that is both time and cost efficient for large as well as small order series, it is imperative to take advantage of the strengths of both human designer and computerized system.

6 References


4 Issues

During this research it has become evident that we need a method for defining the characteristics of a design problem with respect to the type of design systems that should be used. We will also need a method for relating the task division between system and designer to the total cost of an order system. An issue that we have only touched is the need for a design language for describing components and requirements.

During the implementation of the pilot design aid, issues of using constraint satisfaction systems for mechanical design rose, such as:

- How do general and efficient value generators look like?

  General routines without any heuristics are very inefficient and the execution time grows exponentially with the number of variables and values. A designer uses a lot of knowledge when assuming a value. This knowledge needs to be formalized and consistently represented.

- How to generate a modified design based on a given configuration?

  If the constraint system is given a set of inconsistent variable values, it will normally just flag that the values are inconsistent. When the designer makes a change, he would like to get a consistent solution that is close to what he intended. Thus the system needs to be able to generate a new set of values based on a set of inconsistent ones. This requires a routine that changes values based on what variables could change and how much, in what order, etc. We have some ideas of how to solve this problem, but a deeper analysis needs to be done.

- How is the requirement that is causing a constraint violation flagged?

  How should non-linear equations be handled and over-constrained systems be managed?