1991

Research on constraint-based design systems

Mark Sapossnek
Carnegie Mellon University

Carnegie Mellon University.Engineering Design Research Center.

Research on Constraint-Based Design Systems

by

Mark Sapossnek

EDRC 18-24-91

RESEARCH ON CONSTRAINT-BASED DESIGN SYSTEMS

Mark Sapossnek

Engineering Design Research Center
Carnegie Mellon University
Pittsburgh, PA 15213
U.S.A.

ABSTRACT Design can be viewed as a constraint satisfaction problem: given constraints on functionality, structure, and manufacturability, produce a detailed structural description of an artifact. We present the notion of a *constraint-based design system* as one that explicitly represents and operates upon these constrairus. A key feature of constraint-based systems is the separation of problem statement from problem solution techniques. Constraint systems are distinguished from *parametric design systems,* which do not separate problem statement and solution. Current constraint-based systems arc examined, and their strengths and weaknesses discussed. We describe DOC *(Design Objects and Constraints),* an object-oriented constraint-based design system, and WoRM *(Window Regulator Mechanism design),* a mechanical design system built upon DOC. Issues requiring further research are discussed, and a description of our own research directions is presented.

1. INTRODUCTION

Design can be viewed as a constraint satisfaction problem [Simon81], One of the goals of a design process is to produce a detailed structural description of an artifact, within explicit and implicit constraints on functionality, structure and manufacturability. Mechanical design has a number of characteristics that complicate the design process, e.g. complex geometries, nonlinearitics, consideration of material properties, tight coupling between structure and function, and a variety of manufacturing techniques.

We define a *constraint-based design system* as a system capable of explicitly representing and operating upon the relationships (explicit and implicit, given, derived and assumed) between the aspects (abstract and concrete) of an artifact relating to its life-cycle concerns (including functionality, structure, manufacturability and scrvicability) for the purpose of maintaining the truth values of the relationships. Throughout this paper we will use the terms *constraint-based design system* and *constraint system* synonymously. A *constraint language* is a language used to state constraint relationships. The term *constraint satisfaction system* refers to that part of a constraint system that tries to maintain relationships.

The purpose of this paper is to review the state of current constraint-based (particularly mechanical and geometric) design systems, to report on the preliminary work of a new research effort in this area at the Engineering Design Research Center (EDRC) at Carnegie Mellon University, and to suggest future research directions for this work.

The remainder of this paper is organized as follows: **the distinction between constraint-based and** parametric design systems is made in section 2. Section 3 **reviews related work. Section 4** describes the preliminary work in this area done as **part of the CASE *(Computer Aided Simultaneous Engineering)*** project at the EDRC. In section 5 a list of research issues is presented, and section 6 discusses our plans to address some **of these issues. A summary is** given in section 7.

2. CONSTRAINT-BASED vs PARAMETRIC DESIGN SYSTEMS

Parametric and constraint-based design systems both consist **of design *models* composed of** elemental relations (usually numerical equations) between design *variables* (usually **real-**valued quantities).

A parametric system requires that the relations be written and connected in **a** *causal* **ordering** suitable for solution, i.e. they must be arranged in a *directed acyclic* ***graph.*** **This is an** essentially *procedural* representation; thus parametric systems **intermix the statement of a** problem with methods for its solution. Because of this, **the onus of determining how the** relations are solved is on the model builder, not the system.

Constraint-based systems, however, are essentially *declarative.* **Constraint-based models do not** require a causal ordering. The relations in a constraint-based **model are arranged in an** undirected graph. Constraint systems maintain a separation of problem statement **and** solution.

Problem solving with a constraint system requires that some of the variables in the *constraint network* be fixed. The system then uses its knowledge of the relations together with its problem solving knowledge (e.g. numerical solution techniques) to compute **the** values of **the** other variables in the model. This approach permits the sets of **fixed and computed variables to be** easily changed. In contrast, the directed nature of a parametric system implies **that the** sets of fixed and computed variables are explicitly identified in the model. It is possible to effectively re-direct the causality of a parametric model by enclosing **the model** in **an iteration loop;** this can be inefficient, and is not done in many parametric systems.

Parametric systems do not provide any support for handling the systems of simultaneous equations that frequently arise in engineering design problems. Such problems simply **cannot be** expressed in a directed acyclic graph. These problems, **however, can be handled by a** constraint-based system that can identify and solve sets of design relations **that must be solved** simultaneously.

Parametric design systems can be very useful in the design process, and **are** certainly more useful than completely static systems (i.e. those that do **not store any relations, e.g. most mechanical CAD** systems). As part of a design system, **parametric techniques are useful for** problems with established design procedures that do not change frequently. **However, these** techniques provide little support for structurally modifying the design procedure; to do so it is

necessary to partially or fully re-specify the design model.

Because parametric models arc essentially procedural **programs, there** is **no inherent support for explanation** facilities. The declarative nature of constraint systems, **however, easily supports explanation** facilities. Such facilities can be useful when constructing, modifying **and** using models.

Consider the problems involved in designing a backhoe. **In the course of developing a design,** it is necessary to determine the path of the bucket given the **lengths of the hydraulic cylinders. It** is **also** desired to determine the lengths of the hydraulic **cylinders given a certain bucket path. A** parametric design system could not readily handle this **because: 1) two different models would** be required (one for each problem), and 2) **it** is **necessary to solve a system of equations** for the inverse problem. However a constraint-based system **can easily handle this.** This example is taken from a real problem solved by the **Cognition MA1000, a constraint-**based design system [Lìghi88j.

3. EXISTING CONSTRAINT-BASED DESIGN SYSTEMS

In this section we will first review the previous work on **constraint-based and parametric** design systems, and then look at various kinds of constraints **that have been identified. Some** of the concepts and terms introduced in this section will be described in section 4.

3.1. Existing Constraint-Based **and** Parametric **Design Systems**

3.1.L Constraint-Based Design Systems

Sutherland's Sketchpad [Suihcrland63J system pioneered both the use of interactive **computer** graphics and constraint-based design. The ìormcr of course, has become very **popular, while** the latter has still not received widespread usage. Sutherland **introduced the notion of *merges*** (a way of recursively cquivalcncing objects) as well as the solution techniques of *propagation of values, propagation of degrees of freedom* and *relaxation.* **Models could be built** graphically, only mathematical equality relations were handled, and **no explanation** capabilities were provided. ThingLab [Borning79] was an updated version of **Sketchpad,** adding a modern object-oriented language (Smalltalk), the notion *of paths,* compilation of the solution procedure, and an enhanced graphical user interface.

Steele **and** Sussman [Stcclc78] discuss a language for **the construction of hierarchical constraint networks** using value propagation, *alternate views* (redundant constraints), **and** relaxation. They briefly discuss manipulations of algebraic constraints. Steele's thesis [Steele80] goes into value propagation in depth, adding explanation capabilities. **Gosling** [Gosling83] described a system for simple line drawings. He added symbolic **mathematical** capabilities through a simple rcwnic-rulc system and developed a breadth-first **value** propagation algorithm.

Gossard and students examined *variaiional geometry* or *symbolic dimensioning,* **based upon** Newton-Raphson solution techniques. Light's system [Light80] examined **the Jacobian of the** system of nonlinear constraint equations to detect over- and under-constrained **situations. Lin** [Lin81] proposes segmenting the system of equations in an attempt to improve the efficiency of the solution. Serrano [Scrrano87j emphasizes constraint management, the identification of

over- and under-constrained systems of constraints, solution of inequalities and the interactive development of constraint networks. He used a graph theoretical approach to identify subsets of equations to solve. The Cognition MA1000 system [Deitz88] and the Premise Design View system arc commercial products that have evolved from this line of research.

Though it is not advertised as such, the Edinburgh Designer System [Popplestone84] can be viewed as a constraint-satisfaction system that integrates several solution techniques via a blackboard architecture. This system uses an *assumption-based truth maintenance system* to record alternate designs.

Juno [Nelson85] is a system oriented more towards producing pictures than doing engineering design. Its most interesting feature is that it maintained graphical *(direct manipulation)* and textual representations, and updated one of the representations when a change was made to the other.

Gross [Gross86] describes the Constraint Explorer, a constraint-based system for architectural design. He emphasizes the constraint-based approach as a theory of design, but does not give a detailed technical description of his system.

A Prolog-based system is described in [BrudcrlinS6]. In this system, geometric construction rules expressed in Prolog arc matched against the current set of objects and constraints. A plan to satisfy the constraints is thus generated, and then applied. Multiple solutions to a constraint problem can be found, but some problems involving simultaneous equations cannot be solved.

Leler [Lclcr88] describes Bcrtrand, a constraint language based upon *augmented term rewriting*. This approach provides a uniform way of defining symbolic and numerical constraints. However, it has limited power to deal with the large sets of non-linear equations that arise in engineering applications.

Ascend [Picla89] is a constraint-based design language and environment designed for engineering applications. Its Pascal-like language was influenced by ThingLab, but adds *arrays* of objects, automatic unit conversions, and constructs for refining the class of an object. Ascend supports non-linear equation solving and numerical optimization, and provides *degrees-of-frcedom analysis* capabilities to aid in problem formulation.

In the area of representation, Woodbury [\Voodbury87] presents an approach to geometric reasoning that relates geometric models, features (which are subsets of those models), abstractions of features, and constraints defined upon abstractions.

3.1.2. Parametric Design Systems

Rossignac [Rossignac86] described a parametric approach to imposing constraints on constructive solid geometric models by having the user indicate the transformations (constraints) to apply. MacCallum and Duffy [MacCallum87] describe DESIGNER, an essentially parametric system that permits a user to construct a dependency network of design relationships. The system can operate in a goal-oriented mode,- backward-chaining relationships to calculate goals.

ICAD and Wisdom Systems are commercial vendors that offer similar parametric design

systems. They arc both object-oriented extensions to Lisp that provide *part-whole hierarchies,* and come with pre-defined objects and development tools. The Pro/Engineer system from Parametric Technologies combines feature-based solid modeling with parametric design. Iconnex sells a commercial system that supports parametric design.

3*2. Constraints In The Design Process

This section lists different kinds of constraints previously identified by other researchers.

Sriram and Maher [Sriram86] discuss the role of constraints in the design process, as embodied in their knowledge-based structural design programs ALL-RISE and HI-RISE. They distinguish five kinds of constraints:

- *synthesis constraints* - due to geometric requirements and design heuristics, used in a generate and test mechanism,
- *interaction constraints* - compatibility conditions between subsystems,
- *causal constraints* - used to test the feasibility of a preliminary design,
- *parametric constraints* - used to propagate values,
- *evaluation constraints* - *soft* constraints, used to rank alternative designs.

Brown and Breau [Brown86b] discuss four kinds of constraints that arose during the work on DSPL (Design Specialist and Plans Language) [Brown86a], a knowledge-based design language:

- *implicit constraints* - constraints that have become absorbed into the synthesis procedure,
- *in-placc constraints* - a test constraint used in a generate and test procedure,
- *inherited constraints* - an in-placc constraint that arises due to the recognition of the design object as being of a more general, previously explored type,
- *accumulation constraints* - constraints which examine the combined results of two or more subsystems, initiating and guiding tradeoffs between the subsystems.

4. CURRENT WORK

Our current work on constraint-based design systems is being performed as part of the CASE *(Computer Aided Simultaneous Engineering)* project. CASE is a research project at the EDRC that focuses on integrating a variety of life-cycle concerns into the design process. CASE is composed of multiple *Design Agents* and *Design Critics.* Design Agents are responsible for the synthesis of an aspect of a design in response to a set of specifications. Design Critics arc self-activated analysis tools that track the progression of the design, evaluate an aspect of the design, and communicate relevant results to the design agents. The initial domain of CASE is window regulator design, and represents a joint effort of CMU and General Motor's Fisher Guide Division. CASE currently has one automated Design Agent, a design synthesis module, and three Design Critics, a structural (finite element) critic, clearance/interference critic, and a tolerance critic. Sec [Rchg88] and [Talukdar88] for a more complete description of CASE. The constraint-based system described here is currently used as the design synthesis module of the CASE system.

The initial domain of the CASE project is the design of *window regulators,* the mechanisms

used to raise and lower the window in an automobile door. This domain was chosen because it provides a good impedance match between what we know how to implement and our research objectives (i.e. it is simultaneously simple and challenging). In addition, this is an area where new designs (or at least variations on old designs) are made frequently, yet the process has traditionally been done without automation.

The problem is to design a mechanism that transforms the rotational motion provided by an occupant into a linear movement of the window glass. Electric window regulators are also of interest to us, but will not be discussed in this paper. The two stages of design are 1) choosing a design conflguraiion and 2) dciermining the dimensions of the parameters for the configuration. The different configurations correspond to different mechanisms used to achieve the desired motion. There are two kinds of configurations: *sector regulators* and *tape-drive regulators.* There are three kinds of sector regulators: *single-arm, cross-arm,* and *fixed-point.* This paper is concerned only with single-arm sector regulators, as sketched in Figure 4-1. Sector regulators contain a *spindle* which is attached to a crank inside the passenger compartment and a *pinion gear* inside the door. The pinion drives a *sector gear,* which rotates about the *pivot,* and is auached to a *lift-arm.* The pinion and sector gears are attached to the *back-plate,* which in turn is atiachcd to the door frame. As the sector gear turns, the end of ihe lift-arm, which is constrained to slide in a *channel* auached to the *glass,* moves along an arc, driving ihc glass up or down. A *spring* is used to offset the weight of the glass.

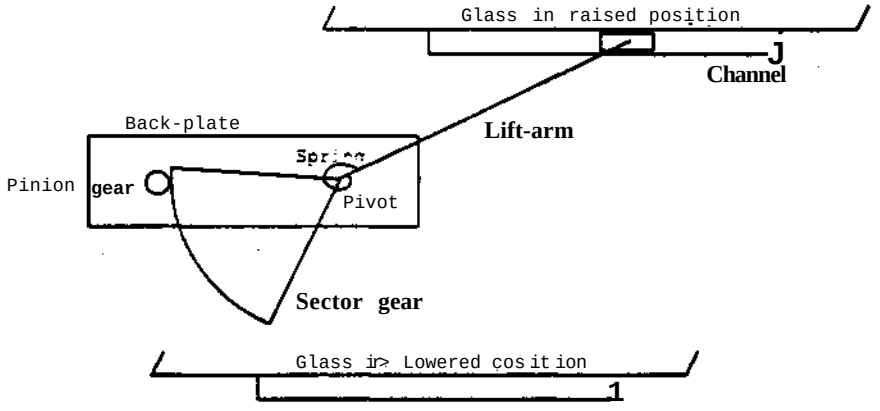

Figure 4-1: Single-arm window regulator

The constraints on the problem include:

- **fixed spindle** (pinion gear) location,
- fixed raised and lowered glass positions,
- **bounds on** the number of handle turns to fully raise or lower the window,
- **a limit on the** torque necessary to raise or lower the window,
- **the torques to** raise and lower the window should be about the same,
- **physical** limitations on the springs,

- the cost should be minimized.

The choice of a design configuration can be made with knowledge obtained by experience with previous designs, and is driven primarily by cost constraints. In the current system the user must choose the configuration. The problem of deciding upon the dimensions of the parameters is more difficult and tedious, requiring computations based on geometric and force analysis.

Upon examination of the problem, we realized that numerical optimization techniques were the appropriate means for determining the parameter values. In industry, however, such techniques are currently not used to solve this and similar problems, due primarily to difficulties in using optimization packages. In particular, it can be difficult and/or tedious to pose a design problem for numerical optimization.

Our approach to this problem was two-fold. First, we developed DOC *(Design Objects and Constraints)* a general constraint-based, object-oriented language capable of representing a large class of objects and constraints, including those appearing in the window regulator design problem. Then, we used DOC to develop WoRM *(Window Regulator Mechanism design),* a program dedicated to the design of window regulators.

4.1. DOC

DOC is our testbed for constraint systems research. As our primary concern is the semantics of the language, not the syntax, we wanted an implementation language suited for rapid prototyping of our ideas, leading to the choice of Common Lisp with Flavors object-oriented extensions. The design of the current system, DOC.O, has been influenced primarily by ThingLab; we believe that the object-oriented design of ThingLab will provide a good basis for our research.

DOC.O handles numerical constraints, solves systems of non-linear equations, performs numerical optimization and has rudimentary explanation facilities. The details are described in the following two sub-sections.

4.1.1. The DOC Language

In this section, we present the DOC language. A BNF syntax of the DOC.O language is provided in Figure 4-2. Figure 4-3 shows how the * object is defined. Other basic arithmetic objects (e.g. +, -, /) arc similarly defined. We begin with an introduction to the fundamental concepts used in DOC, and then describe *functional merges,* a new extension to constraint languages.

DOC borrows from the Skctchpad/ThingLab notions of *objects, constraints* and *merges.* An object is an entity with a list of *parts,* constraints and merges. If you consider an object as a record data structure, parts arc fields in the record that contain other objects. There are *compound objects,* which contain other objects as parts, and *simple objects,* which contain a *value* which is an atomic Lisp type: a number, symbol or string. Compound objects are used to form a *part-wfwle hierarchy* consisting of nested objects. A specification of a part or a part of a part, etc., is called a *path.* Paths arc similar to directory paths in Unix, except that only parts at or below the current object can be referenced.

```
<object-def> : :*■ (defobject <class-naxne> (<dass-name>*)
                     [:parts (<part-spec>*)]
                     [:constraints (<constraint>*)]
                     [:merges (<merge>*)]
                     [:functional-arguments (<part-name>*)]
                     [:functional-value <part-name>]
                     [:macro-object? <l-bool>])

<dass-name> : := any lisp symbol

<part-name> : := any lisp symbol

<part-spec> ::= (<part-name> <class-name>)

<path>      ::• <part-name> | <part-name>.<path>

<constraint> ::= «rule> <c-method>*)

<rule>      : := <l-expr>

<c-xnethod> : := (<part-name> <expression>)

<merge>     : := <p-merge> | <f-merge>

<p-merge>   : := (= <path> <path>) |
                 (= <path> <l-expr>) |
                 (s <l-expr> <path>)

<f-merge>   ::= (<class-name> [<f-expr>*])
<f-expr>    ::» <f-merge> | <path> | <l-value>

<l-bool>    ::= t | nil
<l-expr>    ::= any lisp s-expression
<l-value>   ::= any lisp constant I
                any lisp global variable
```

Figure 4-2: BNF of the DOC.O constraint language

```
(defobject * 0
  :parts ((a numeric) (b numeric) (c numeric))
  :constraints i:<<= a <* b c))
                 (a {*' b c))
                 (b (>f a c))
                 (c U' a b))))
```

Figure 4-3: The • object

As **in** Flavors, objects exist in a class hierarchy. When a *class* **of** objects **is defined, the *super classes*** (parents) of the class are specified. The class hierarchy is used **to provide** *inheritance* of parts, constraints, merges, and *class methods* **(procedures). At present only single** inheritance is implemented (i.e. at most one super class can be specified).

Constraints are associated with objects and provide a way of using Lisp functions to build computational relationships. Constraints consist of a single *rule* and zero or more *constraint methods,* expressed in terms of parts. The rule is used as a predicate to determine the truth value of the constraint (i.e. if the constraint relationship is satisfied or violated), and can be any Lisp expression (including numerical equality and inequality). A violated constraint is called a *contradiction.* Constraint methods provide procedures for computing a part given other parts, consistent with the rule. For example, if the rule is *a • b* • c, then there are three methods of solution: *a := b* c,b := a I* c, and *c :=al b.*

Merges are equivalence relations between parts. When two parts are merged, they are both replaced by a single equivalent object When merging compound objects, the parts in both objects are recursively merged. Two objects can be merged only if they are of the same class or if the class of one is a descendent of the class of the other *(conformable aneestry* in Ascend terminology). DOC distinguishes *primitive merges* from *functional merges.* Primitive merges specify that two objects be merged. Functional merges provide a way of stating complex constraints in a functional notation.

To understand functional merges, consider Figure 4-4, which shows our version of the ubiquitous ccntigradc-fahrcnhcii converter. (The == constraint merges its two arguments; it is detailed below.) The functional merge allows ==, *, and -, which are previously defined objects, to be used in a Lisp-like notation. Figure 4-5 shows how this simple object would be defined without the use of functional merges. Functional merges allow the construction of arbitrarily complex expressions that are easily written and understood because they look just like Lisp s-cxprcssions. An additional benefit is that the system can now know that there is one complex expression, not many simple ones. Thus explanations and other inference procedures can operate at the appropriate level.

```
(defobject c-f ()
  :parts ((c numeric) (£ numeric))
  :merges ((»« (* c 1.8) <- f 32.0))))
```

Figure 4-4: The c-f object

```
(defobject ugly-c-f ()
  :parts ((c numeric) (£ numeric) (tl *) (ml -))
  :merges ((« c tl.b)
           (= 1.8 tl.c)
           (= f ml.b)
           (» 32.0 ml.c)
           (= tl.a ml.a)))
```

Figure 4-5: The ugly-c-f object

Functional merges differ from functional languages (such as Lisp) in two ways: 1) the functional notation is expanded into a constraint network, and 2) the causality of computation is not limited to that implied by the functional notation. The value returned by the highest level functional merge is ignored; a *returned value* is that part of an object defined to be its *functional value* when used as a functional merge.

The functional value and arguments can be denned in one of two ways. By default the returned value is the First part specified in the object. The *default functional arguments* are obtained by sequentially matching up the 2^{nd} through (7+7/* parts of the object to the *i* arguments used in the functional merge. Alternately, the `:functional-value` and `:functional-arguments` sections of the object definition can be used to explicitly state which parts are used as the returned value and arguments, respectively. A possible enhancement would be to allow multiple sets of functional arguments and functional values to be defined. Each set would have its own functional name, which could be the same as the class name. This would, for example, allow the * object to be used both for multiplication and division; currently we define two separate objects.

The purpose of the `:macro-object?` section of the object definition is to allow an object used as a functional merge to contribute its merges and constraints without keeping the object around in the part-whole hierarchy. Currently our only use for this is in the = object, shown in Figure 4-6. This object was needed as a way of merging the returned values of two functional constraints without keeping a lot of meaningless == objects around. Note the `:functional-arguments` section - without it == would take *b* as an argument and return *a.* Currently this facility works only for the merges of an object, not its constraints.

```
(defobject == ()
  :parts ((a object) (b object))
  :merges ((= a b))
  :functional-arguments (a b)
  :macro? t)
```

Figure 4-6: The == macro object

4,12. DOC Internals

Each DOC object maintains a list of neighbors, inputs, and outputs for all objects and constraints. For an object, these lists contain constraints; for a constraint they contain objects. The list of neighbors represents the *undirected constraint graph* and the inputs and outputs represent the *directed constraint graph.* DOC has explanation facilities for identifying *upstream* and *downstream* (relative to the directed graph) objects and constraints.

Currently, two solution techniques are used in DOC: *value propagation* and *numerical optimization.* The value propagation algorithm uses the constraint methods to perform computations. When an object is modified, its neighboring constraints are notified. If they can choose a constraint method \o*fire* they do so, modifying their outputs, thus propagating changes. This algorithm handles *redundancies* in the network, and detects and marks *contradictions.* No attempt at backtracking is made when a contradiction occurs, instead a *global solution technique* (e.g. Ncwton-Raphson) is invoked if possible. The value propagation algorithm also marks objects and constraints to ensure that cycles do not occur.

An object can be *locked* or *unlocked.* A locked object is one that must be fully specified by the user, i.e. the system will never try to derive a value for it. An unlocked object can be given a value by the user, but the system is free to change the value to satisfy the constraints. Once a *model* (a high-level compound object) has been built up out of DOC objects, the user can lock or unlock any object. This provides the mechanism for changing the causality of the

constraint solution. Currently locking/unlocking is only applied to simple objects. We plan on experimenting with locking and unlocking compound objects as a means of easily controlling the scope of solution techniques.

The notion *of firmness* is used to guide the value propagation. An object is firm if it is locked, or computed by a constraint whose inputs are all firm. Thus the firmness itself must be propagated through the network when the type of an object is changed. The use of firmness allows the value propagation algorithm to avoid inadvertent conflicts and/or backtracking.

When the type of an object is changed, the change in firmness of the object is propagated. If the object has become unlocked, then a *degrees of freedom propagation* is performed to see if any contradictions in the network can be resolved. The degrees of freedom propagation essentially looks for a contradiction that can be resolved, and then tries to resolve it with a value propagation.

For numerical optimization, DOC uses a subsystem called Opt, which is a generic interface to optimization codes. Currently Opt talks only to OPTDES.BYU [Parkinson84], a program capable of nonlinear, continuous and discrete valued optimization. We currently optimize over continuous variables only and are in the process of extending the system to handle discrete variables. Opt can also be used to solve sets of nonlinear equalities and inequalities.

The interface between DOC and Opt also makes use of the value propagation algorithm. When an optimization is requested, DOC reduces the problem by choosing a small set of variables from which value propagation can be used to solve for the remaining variables. At present this choice of variables is based upon problem-specific information, though we expect to develop general algorithmic and heuristic techniques.

4.2. WoRM

DOC was used to create a library of mechanism components, i.e. *joints* and *links* [Tilove83, Rehg88]. Links represent the rigid parts of a mechanism and joints represent the constrained interaction of the links (both higher and lower order pairs are represented).

Figure 4-7 contains two sample objects selected from WoRM (the complete model will be presented in a future paper), a *unary-link* and a *prismatic-joint.* A unary-link represents a part with one attachment point, represented by a coordinate system. A prismatic-joint represents a joint with one translational degree of freedom. It is implemented as a subclass of *line-segment,* another object that contains two attachment points (called *csl* and *cs2)* represented with coordinate systems. The prismatic-joint adds another coordinate-system, *cs3,* whose angle is aligned with that of the line-segment, and is constrained (via the *linear-combination* functional merges) to lie on the line segment. Links and joints are connected by merging their attachment points.

To solve the window regulator problem, we modeled the mechanism using the library of mechanism components. The model contains two lift-arm assemblies (containing the sector, lift-arm, slider, and associated joints), one constrained in the lower position, the other constrained in the upper position, with objects representing common dimensions merged. Objects representing specifications (e.g. spindle and glass positions) are locked. The

```
(defobject unary-link ()
  :parts ((cs coordinate-system)))

(defobject prismatic-joint (line-segment)
  :parts ((i numeric) (cs3 coordinate-system))
  :merges ((*» a cs3.a)
           (linear-combination csl.x cs2.x cs3.x i)
           (linear-combination csl.y cs2.y cs3.y i)))
```

Figure 4-7: The unary-link and prismatic-joint objects

objective function is the minimization of the sum of the back-plate and lift-arm lengths.

A user interface based on the X window system was added to the DOC model to create WoRM. The user interface consists of three *windows.* The *input window* lists a subset of the objects in the model, displaying the name of the object, its value, and its type. The user can change the value and the type of any object displayed. The *diagram window* displays a stick-figure model of the system. The *status window* displays information about the current state of the model, including the list of contradictions. When the user makes a change, it propagates throughout the network and then all windows are updated to reflect the effects. An optimization can be invoked from the input window. By changing the types of the objects, the user is effectively re-formulating the optimization problem. Presently, the user cannot interactively add, delete or edit the object definitions, though these changes can be made in the textual definition of the objects. WoRM is currently being tested at Fisher Guide.

The advantages of the constraint-based approach to this problem become apparent in the following two situations: the first involves a change in specifications and the second involves using one model to design different objects.

The process of developing a window regulator design involves interacting with other engineers designing the other parts of the door. As a result, conflicts arise and one (or both) engineering group(s) may have to alter their design(s). For instance, the interior designers may decide to change the spindle location. But the engineer may want to keep the other parts of the design constant. With WoRM the solution is to unlock the spindle coordinates, lock the pivot coordinates, and then rotate the spindle about the pivot. The user is free to lock and unlock any object; there is no problem if the system becomes over-constrained unless a contradiction is introduced. Even then, the user is notified of the existence of the problem and can track the source of the problem with the explanation facilities.

Most designs use off-the-shelf springs, i.e. a spring is selected from a fixed list of existing springs. Sometimes it is desired to design a new spring, to obtain a more optimal design. In this case, the geometry of the window regulator may be locked, and the parameters of the spring allowed to vary continuously, or both the geometry and spring parameters may be unlocked. Thus, WoRM can be used to design the spring as well as the window mechanism geometry. To do this with a parametric system usually requires writing two separate models.

5. RESEARCH ISSUES

In this section we present a list of research issues related to constraint-based design systems. **Some of these** ideas represent long term goals whose solution **depends on advances in** basic **areas such** as knowledge representation and intelligent inferencing; others **are** much closer **to** implementation.

5.1. Generalizing Constraint Systems

Constraint-based systems to date have been very limited in **what they do; they have** essentially been numerical equation solvers with front-end facilities **for problem formulation. Leler** [Lelcr88] points out that constraint satisfaction systems **are not meant to be general** purpose problem solvers; they are supposed to be *support systems* **that perform the little** computational problems that arise in the context of **complex problem solving activities. We agree** with this viewpoint, yet we also feel that there is **room for generalization, Le. that the** scope of the *little computational problems* handled by constraint systems **can be made** expandable.

We envision a constraint management system as an **intermediary between an *intelligent designer*** (human or automated) and various *aspects* and *operators.* **Aspects are design** representations of arbitrary form, e.g. numerical and symbolic equations, **geometric models,** finite element models, etc. Operators are design procedures **that modify or transform aspects.** The role of the constraint system is to record the *desired relationships* **amongst the aspects,** and to be able to invoke well-defined operators in **order to maintain the relationships. The** designer is responsible for choosing representations and specifying constraints.

The *well-defined operators* mentioned above include those that have different **kinds of input** and output aspects (e.g. finite element analysis) and those that operate **on one aspect (e.g.** numerical equation solvers, discrete constraint satisfaction algorithms). **Operators can be** heuristic rule systems, procedural algorithms, and could even include random or **probabilistic** techniques. Operators are *well-defined* if they can be encoded in **a** procedure. **Operators that are** not well-defined are not encoded for the following reasons: decision-making in **novel** situations (it may not be worth the effort to encode the knowledge), difficulty **of knowledge** acquisition, or lack of resources (perhaps the encoding of the method simply **hasn't been done** yet). The location of the boundary between the designer and the constraint system **is** not fixed; as more knowledge is acquired and encoded in one form or another, more **of the** computations/decisions can be made by the constraint system.

Thus, a generalized constraint system will provide a *substrate* for intelligent **design systems. When used by a** human designer it will provide support in the areas of **representation, calculation and** documentation, and will thus have great utility in and of itself. **However we** also feel that it will be useful to fully automated design tools. The automated **design tools that** have been built to date (e.g. [MahcrSS]) have had need for these facilities. **They solved this** problem by implementing a subset of these capabilities tightly coupled **to their intelligent** decision making facilities. We suggest that advances in automated design systems can **be** facilitated by partitioning these systems into separate constraint management and intelligent decision making units.

We have identified two promising areas for investigation: 1) separation of solution **technique**

from problem specification, and 2) constraints as representation.

5.1.1. Separation Of Solution Technique From Problem Specification

Just as expert systems emphasized the need for separation of knowledge and inference mechanisms, the specification of a constraint problem should be kept separate from its solution technique. This separation is useful because it permits **reasoning about the problem and** allows the selection of the appropriate solution techniques **(operators). The primary problem** with parametric systems (and other procedural languages) is **that they intennix the problem** specification with the technique for its solution.

Most existing constraint languages have fixed solution techniques. **Some,** like **ThingLab and** Ascend have multiple techniques, others, like Bertrand, are built **around one technique. An** exception is CommonLog [Holman86], which allows for multiple **solution methods. Not only** must **a** generalized system have a variety of techniques, but it must be **easy** to **add new** techniques. The system should be designed to allow new techniques **to be added, just as DOC** allows new types of objects and constraints to be defined.

The solution techniques referred to above include general ones, such **as numerical equation** solving. However they also include very specific ones such **as** mechanism synthesis procedures, as well as techniques specific to a single problem - **conceptually they are all just** techniques. This allows techniques that can operate **upon higher-level (compound) objects, not** just elementary objects such as numbers. Also, this approach **allows *global* (e.g. Newton-Raphson)** and *local* (e.g. value propagation) solution techniques **to be treated** in **a common** manner. How to best choose which techniques to use on a particular problem is still **an open** question.

The interaction of multiple techniques is an area of concern. Interactions occur when **the** solution of the problem requires multiple techniques. As an example, consider the choice of window regulator configuration. The appropriate technique for this decision is based **upon** heuristic rules obtained from experienced designers. Once this choice is made, the remainder of the problem is solved with numerical optimization. A general constraint-based system would be capable of integrating these two techniques.

Because they arc procedural, parametric systems have **a** computational **advantage over** constraint systems. Constraint systems could benefit from an infusion of procedural **and** heuristic techniques. In certain situations it may be desirable to *superimpose* procedural **and** heuristic knowledge over the declarative knowledge in a constraint (or set of constraints), to facilitate computation. Such a technique adds a parametric flavor to the system, i.e. causality is explicitly stated, while retaining the declarative constraint statements. Thus the **best of both** worlds can be realized: the flexibility of a constraint-based approach with the explicitness of solution of a parametric approach.

Additionally, sometimes a heuristic or procedural solution technique is **a** faster **way, or even the only** reasonable way to solve a problem with limited resources. The constraint system should have mechanisms for capturing such techniques and using them when appropriate.

The notion of aspects and operators comes from the TAO graph approach of CASE [Talukdar88]. TAO graphs consist of aspects as nodes, and operators **as arcs** between **the**

nodes. Operators can be analysis procedures or merely translators between two aspects. WoRM, a constraint-based system, represents one operator. We are interested in investigating the use of constraints as applied to the TAO graph itself, and recursively to its aspects. This is complicated greatly by the difficulty of translation between representations.

5.1-2. Constraints As Representations

In order to achieve the goals of the generalized constraint system it will be necessary for constraint systems to deal with more than just numerical constraints. They must be able to *represent* relationships at various levels of abstractions, from high-level functional constraints down to low-level numerical constraints (including the types of constraints discussed in section 3.2). These levels of abstractions are not wholly independent, but are related in various ways. A generalized constraint language should provide mechanisms for stating and operating upon these *constraints among constraints.*

For example, the constraint *attached-rigidly* can be used to represent the geometric relationship between two parts in a design, and is sufficient for certain design tasks (e.g. one could determine if two arbitrary parts are rigid with respect to one another through the transitive nature of the *attached-rigidly* relation). Other design tasks may require more specific information, e.g. to perform tolerance analysis one would need to know just how two parts are rigidly attached - welds, rivets, etc. This lower level information can also be expressed as constraints, and can vary (as different manufacturing techniques are evaluated) without violating the constraint *attachcd-rigidly.* At the level of geometry, the *attached-rigidly* constraint is expressed as equations between numerical design objects.

It is worthwhile to consider the representation of constraints, much as the representation of geometry [Rcquicha80] and knowledge [Brachman8S] have been given much attention. The work on functional representation (e.g. [Lai87]) falls along these lines. Additional research is needed to understand how various representations are related, and how constraints can operate upon different representations.

As an issue of representation and computation, high-level constraint language constructs are needed. The functional merge construct introduced in this paper is an example of such a construct. In [Borning85] Borning presents constructs for functional, mapping, and recursive constraints. As an example of one kind of high-level construct that we seek, consider the two lift-arm assemblies that were described in section 4.2. The specification of the two assemblies and the merges identifying the common elements was quite tedious. An appropriate construct would greatly simplify this task.

Finally, we see applications of constraint management techniques to the following problems: 1) the representation of objects at different levels of resolution, 2) representations of partially specified objects, and 3) constrained topological modification.

5-2. Other Issues

Three additional research issues are discussed in this section: 1) building and debugging constraint models, 2) qualitative reasoning, and 3) a two level approach to constraint systems.

5J.I. Building and Debugging Constraint Models

A major problem with constraint-based systems is the construction and debugging of models. The main problem in numerical constraint systems is dealing with under- and over-constrained systems. When a system is under-constrained, the system should provide guidance in determining what information is missing, and how it should be provided. Over-constrained systems show up through redundant and contradictory constraints. Redundant constraints should be identified and removed from the solution procedure. The system, not the user should be responsible for identifying and isolating redundancies.

Contradictory constraints must be identified and maintained, and assistance should be provided in eliminating the sources of the contradiction. Until a satisfactory design is achieved, there will likely be contradictions in the model. Current systems are very poor at this; they require that contradictions be resolved for proper operation. Constraint-based systems should provide ways of permitting calculations to continue in the face of contradictions. Control mechanisms are needed to facilitate this.

More powerful explanation capabilities arc required. For example, explanations stated in high-level terms corresponding to uscr-lcvcl concepts could be generated from additional semantics imposed on the objects in the object-par^-constraint network.

Methodologies for building objects should be developed. Objects should be designed for use in different situations. These methodologies will necessarily be influenced by the level of constructs available in the language.

5.2.2, Constraints and Qualitative Reasoning

The recent work in qualitative physics, particularly qualitative mechanics [Nielsen88, Faltings88, Joskowicz88J represents techniques for reasoning from basic principles (physics, geometry). This technology is quite distinct from constraint technology, but we see applications in the area of intelligent constraint satisfaction.

One can think of designing a physical object with geometric constraints as being similar to exercising a mechanism. As a part is moved, other parts will be forced to move. However, a mechanism must move according to static, kinematic and dynamic constraints, while a designed object must satisfy constraints derived from functional and structural considerations. When a change is made to the design, the result should be reasonable as an engineering change, not as the motion of a mechanism. Simply solving systems of equations will not always provide this result. Qualitative methods could provide the necessary control over causality that achieves results that are more intuitive than numerical techniques. In addition, the explanation capabilities of qualitative methods could facilitate the construction of models.

5*23*. A Two-Level Constraint System

A design system based upon constraints could be accessed on two levels. Level-one would provide complete access to the full capabilities of the design system. Level-two access would provide a customized interface used to solve specific design problems.

Using a level-one system, constraint-based models can be built and solved interactively. The user of this system should be well versed in the capabilities of the system. When confronted with a new problem, the full power of the system is needed to try different approaches,

explore the design space, and develop a design methodology.

When the design methodology is developed, the level-one system can be used to develop a level-two system that is capable of solving a specific type of problem. Thus level-two systems are parametrized (and perhaps compiled) versions of the general system, where the parameters and solution techniques were identified through the use of the general system. Note that a parametric system is still not equivalent to a level-two constraint system because: 1) while its interface is parametrized, the level-two system still has access to the declarative constraints and can explain its results, 2) multiple level-two systems (each parametrized in a different way) can be built from the level-one system, and 3) the level-one system provides support for building and modifying level-two systems that purely parametric systems cannot provide.

6. FUTURE WORK

Many difficult issues were raised in the previous section. Here we list the subset of issues that we are currently pursuing.

There are various ways that one could approach the problem of integrating different solution techniques. One way that has received attention in the past is the use of blackboard architectures [Popplcsionc84]. We plan, instead, on building upon the object-oriented structure of DOC. There will be a message-passing protocol that will utilize both class and part-whole hierarchies to identify individual sub-problems, select appropriate operators for each sub-problem, and then execute the operators. Inheritance of methods will allow, for example, the most abstract class to implement general, weak, algorithmic methods, while particular objects will have heuristic, domain-specific techniques that understand the semantics of the objects they are processing. Class methods (which handle messages) will be written using any one of various programming techniques, for example, Lisp procedures or rule languages.

This approach will be tested by integrating several techniques for discrete-valued (symbolic) problem solving with the numerical techniques already present in DOC. Once we have a framework for handling symbolic and numerical constraints we will investigate ways of relating symbolic and numerical constraints at various levels of abstractions.

Other problems that we are working on include: developing additional high-level constraint language constructs, specifying and implementing a procedural interface to DOC, mechanisms for controlling the scope of solution techniques, and generalized explanation facilities.

Finally, to provide a better understanding of constraints, we are compiling a taxonomy of constraint types, and arc developing more formal, general and complete definitions of constraints in terms of their representational and computational properties.

7. SUMMARY

This paper was intended 10 review previous work **on constraint systems, including a detailed look at DOC and WoRM, and to present our views on what future constraint systems could and should be. Many of the goals stated here arc long-term objectives, while others represent ideas much closer to implementation. We hope that our work contributes to additional understanding and interest in constraint systems so that their potential can be realized in *real world* design situations.**

8. REFERENCES

[Boming79] **Borning, Alan Hamilton, *ThingLab - A Constraint-Oriented Simulation Laboratory,*** PhD Thesis, **Stanford University, 1979.**

[Borning85] **Borning, Alan Hamilton, *Constraints and Functional Programming,*** **Technical Report No. 85-09-05, Computer Science Department,** University of Washington, **Seattle, 1985.**

[Brachman85] Brachman, Ronald and Lcvcsque, **Hector J., *Readings In Knowledge Representation,*** Morgan Kaufman Publishers, **1985.**

[Brown86a] Brown, D; Chandrasckaran, **B; *Knowledge and Control For A Mechanical Design Expert System,*** **Computer, July 1986.**

[Brown86b] Brown, David C. and **Brcau, Robert, *Types of Constraints In Routine Design Problem Solving.***

[Brudcrlin86] **Brudcrlin, Beat, *Constructing Three-Dimensional Geometric Objects Defined By Constraints,*** **1986 Workshop on Interactive 3D Graphics,** October 1986.

[Dcitz88] Dcitz, Daniel, *Tools For Total Quality,* **Computers in Mechanical** Engineering, July/August 1988.

[Faltings88] **Fallings, Boi, *A Symbolic Approach to Qualitative Kinematics,*** **Technical** Report 88-02, Swiss Federal Institute **of** Technology, **1988.**

[Gosling83] Gosling, James, *Algebraic Constraints, **PhD** Thesis,* **Carnegie Mellon** University, May 1983.

[Gross86] Gross, Mark Donald, *Design As Exploring Constraints,* **PhD Thesis, MIT,** February 1986.

[Holman86] Holman, Cara; Borning, Alan; **Kahn, Kenneth; Miller, Mark; *Constraints and Logic Programming;*** **University of Washington Technical Report** 86-12-01, December 1986.

[Joskowicz88] **Joskowicz, Leo, *From Kinematics To Shape: An Approach to Innovative Design,*** Proceedings **of AAAI '88, August 1988.**

[Kimura87] **Kimura, F.; Suzuki, H.; Ando, H.; Sato, T.; Kinosada, A; *Variational Geometry Based On Logical Constraints and its Applications to Product Modeling,*** Annals of **the CIRP, Vol. 36/1/1987.**

[Lai87] **Lai, K. and Wilson, W.R.D; *FDL - A Language For Function Description and Rationalization in Mechanical Design,*** **Proceedings of the 1987 ASME** Computers **In Engineering Conference.**

[Leler88] **Lclcr, Wm.; *Constraint Programming Languages, Their Specification and Generation,*** Addison-Wcsley, 1988.

[Light80] **Light, Robert Allan, *Symbolic Dimensioning In Computer- Aided Design,*** Master's Thesis, MIT, **February 1980.**

[Light88] Light, Robert **Allan, Personal communication, December 14,1988.**

[Lin81] **Lin, Vincent C, *Three-Dimensional Variational Geometry In Computer-***

Aided Design, Master's Thesis, MIT$_f$ May **1981.**

[MacCalIum87] **MacCallum, KJ. and Duffy, A.;** ***An expert system for preliminary*** *numerical design modeling,* Design Studies, Vol. 8 No. 4, October 1987.

[Mackworth85] Mackworth, Alan, *Constraint Satisfaction,* University of British Columbia Technical Report 85-15, September 1985.

[Maher85] Mahcr, Mary Lou and Fcnvcs, Steven J.; *HI-RISE: A Knowledge-Based* ***Expert*** *System* ***For The Preliminary Structural Design Of High Rise*** *Buildings,* Report No. R-85-146, Department of Civil Engineering, Carnegie Mellon University, January 1985.

[Nelson85] Nelson, Greg, *Juno, A* ***Constraint-Based Graphics System,*** **1985 ACM** Siggraph Conference Proceedings.

[Nielsen88] Nielsen, Paul, *A Qualitative* ***Approach to Mechanical Constraint,*** Proceedings of AAAI ^{f}88, August 1988.

[Parkinson84] Parkinson, A.R.; Balling, R.J.; Free, J.C; *OFTDESJBYU: A Software System For Optimal* ***Engineering Design,*** **ASME Computers in** Engineering Conference, August 1984.

[Piela89] Picla, Peter; *ASCEND: An* ***Object-Oriented Computer Environment for*** *Modeling and Analysis,* PhD Dissertation, Carnegie Mellon University, 1989.

[Popplcsionc84] Popplcstonc, Robin J.; *An* ***Integrated Design System For Engineering,*** 1984.

[Rehg88] Rchg, Jim; Elfcs, Alberto; Talukdar, Sarosh; Woodbury, Rob; Eiscnbcrgcr, Moshc; Edahl, Rick; ***CASE: Computer-Aided Simultaneous*** *Engineering.*

[Rcquicha80] Rcquicha, Arisiidcs A.G., ***Representations for Rigid Solids: Theory,*** *Methods and Systems,* ACM Computing Surveys, Vol. 12, No. 4, December 1980.

[Rossignac86] Rossignac, Jaroslaw R., *Constraints* ***in Constructive Solid Geometry,*** 1986 Workshop On Interactive 3D Graphics, October 1986.

[Serrano87] Serrano, David, *Constraint* ***Management in Conceptual Design,*** **PhD** Thesis, MIT, October 1987.

[Simon81] Simon, Herbert, *The Sciences of the Artificial,* The MIT Press, 1981.

[Sriram86] **Sriram, D. and Mahcr, M.L,** *The* ***Representation and Use of Constraints*** *in Structural Design,*

[Stcclc78] Stccle, G.L.; Sussman, GJ., *Constraints,* AI memo no. 502, November 1978.

[Stcelc80] Stccle, Guy Lewis, *The Definition* ***and Implementation of a Computer*** *Programming Language* ***Based On Constraints,*** **PhD Thesis, MIT,** August 1980.

[Suthcrland63] Sutherland, **I.E.,** *Sketchpad: A* ***Man-Machine Graphical Communication*** *System,* Technical Report No. 296, MIT Lincoln Laboratory, January 1963.

[Talukdar88] Talukdar, Sarosh; Rchg, Jim; Woodbury, Rob; Elfes, Alberto; *Upgrading Design Systems,* **AAAI** 1988.

[Tilovc83] Tilovc, Robert, *Extending Solid* ***Modeling Systems for Mechanism Design*** *and Kinematic Simulation,* IEEE Computer Graphics and Applications, May/June 1983.

[Woodbury87] Woodbury, **R,** *The Knowledge* ***Based Representation and Manipulation of*** *Geometry,* PhD Thesis, CMU, December 1987.