A modeling system for top-down design of assembled products

by M. Mäntylä

The design of a mechanical product usually takes place primarily in a top-down fashion, where the designer first generates a rough, overall sketch of the product and its main components. Later, the designer refines the sketch to a detailed level while taking into account the relevant requirements posed by strength, cost, manufacturability, serviceability, and other similar considerations. Current computer-aided design (CAD) systems provide only limited support for this kind of work. For instance, they cannot deal with geometric or other information at varying levels of detail, nor do they capture explicitly geometric relationships among components intended to be joined together in an assembly. This paper describes early results of ongoing research on supporting top-down design of mechanical products and discusses the major requirements for CAD systems used for top-down design. A prototype design system is described that provides the following characteristics not usually found in ordinary CAD systems: structuring of product information in several layers, according to the stage of the design process; representation of geometric information about components at several levels of detail; and representation and maintenance of geometric relationships of components by means of a constraint-satisfaction mechanism.

1. Introduction

• *CAD and geometry*

Computer-aided design (CAD) systems have become accepted almost universally as central engineering design tools for a variety of applications in mechanical, civil, and electrical engineering. In these areas, CAD systems have boosted design productivity by enabling more

rigorous engineering analysis, enhancing the accuracy and quality of design documentation, reducing errors, and bridging the notorious gap between design and manufacturing. In some areas, such as the design of integrated circuits, the complexity of the designed objects could hardly be managed without the use of CAD systems and the related analysis tools.

The success of CAD systems is related to the fact that many important engineering design tasks are concerned primarily with the geometric shape of a product. This has resulted in the proliferation of various techniques of *geometric modeling*, intended for the capture, representation, and utilization of geometric shape information, which currently form the nucleus of CAD systems. After shape information of the product has been captured in a geometric model, CAD systems permit the utilization of a wide variety of analysis, visualization, and manufacturing planning tools through appropriate data conversion and interfacing modules, in addition to the production of engineering drawings and other design documentation. Geometric models can also be archived and re-utilized in later redesign projects, or transmitted to other CAD systems (e.g., to subcontractors) by using standardized CAD data interchange formats. The geometric modeling methodology required in the various tasks is to a large degree independent of the particular application, making it possible to construct generic geometric modeling tools and utilities that can be shared by several application-oriented modules of a CAD system.

Unfortunately, current CAD systems are far less attractive in applications where the shape information of the product plays a less prominent role or where the geometric information must be augmented with related nongeometric information. In fact, most engineering analyses belong to this group. For instance, in addition to shape information, engineering analysis with the finite element method (FEM) requires information on the intended use of the component being studied. In mechanical analyses, for example, this information may be translated into bending forces applied to the component.

• *Feature models*

The shortcomings of ordinary geometric modeling techniques outlined above and, hence, of ordinary CAD systems have led to a widespread interest in various kinds of augmented geometric modeling techniques that can capture and represent certain types of nongeometric information in addition to the basic geometric shape. During the last few years, so-called *feature models* have received particularly strong attention in the CAD research community. The basic motivation of feature models is the observation that most types of nongeometric information of interest in CAD applications seem to be naturally associated with groups of related geometric entities, instead of single entities such as points, lines, and arcs typically utilized in simple geometric modeling systems, or with solid blocks, cylinders, and spheres utilized in more advanced systems.

The basic approach of feature-based modeling systems is to replace simple geometric elements as the primary building blocks of product models with higher-level modeling elements that more directly correspond to shapes of engineering significance. The misnomer "feature" has become the trade expression used to denote such entities. Hence, the basic modeling vocabulary of a feature-based modeling system consists of a taxonomy of feature types; in mechanical engineering applications, such a taxonomy would include feature types such as holes, bosses, slots, and pockets. Product representations are constructed by creating instances of feature types and placing them in the model. Nongeometric information can be associated with the feature instances. On the basis of a feature model, geometric modeling techniques can be utilized to perform operations similar to those available in conventional CAD systems. In addition, various kinds of data or knowledge bases can be organized on the basis of the feature taxonomy to support applications that require special engineering information.

As might be expected, the success of feature modeling techniques is largely determined by whether a useful taxonomy of feature types can be identified and organized in a modeling system, and whether application-oriented data and knowledge bases can conveniently be organized on the basis of the taxonomy. So far, the best results have been demonstrated in applications where the domain of feature types of interest is somehow naturally limited. For instance, several researchers (including the author) have reported on feature-based machining process planning systems [1–7], where the family of interesting feature types is determined by the available machining processes and machining tools. In this case, feature types would be associated with a data base containing information on the available materials, machining tools, and fixtures.

So far, much less progress has been reported in the application of augmented geometric modeling techniques in actual design applications than in manufacturing-oriented applications such as process planning. One reason for the relative scarcity of convincing results is simply the difficulty of identifying and organizing a complete taxonomy of modeling entities of interest to a designer. However, there also seem to be more fundamental problems in the straightforward application of primarily geometric modeling techniques (including feature modeling) to design proper.

• *Role of geometry in the design process*

To understand the problems of applying geometric modeling techniques in design, we must take into account the varying roles that geometric information plays in engineering design. Typically, an engineering design process proceeds through a number of identifiable phases, each with somewhat different goals. During the early phases of design, decisions are made concerning the desired characteristics and the overall function of the product. In the later phases, the specifications generated during the early phases are refined to make sure that the product indeed fulfills the specifications and that it can be manufactured efficiently. Finally, the detailed descriptions are transmitted for downstream phases of the design–manufacturing process, such as process and assembly planning. Often, the composition of the design process in phases also corresponds to the organizational composition of the design personnel involved in the design process.

In most engineering design processes, the following three major phases of design can be distinguished:

1. *Functional design*, where the primary focus of the designer is on specifying the desired outcome of the design process.
2. *Conceptual design*, where the primary focus is on the selection of a collection of components and subassemblies, and the specification of their relationships such that, together, they will deliver the desired function.
3. *Detail design*, where the primary focus is on refining the individual components to a sufficient level while taking into account all restrictions and requirements posed by the applicable engineering analyses (strength, heat transfer, electrical properties, etc.) and by the available manufacturing technologies.

Of the three major phases, functional design (also called *strategic design*) is least concerned with product geometry, being involved instead with a broad variety of issues, including marketability, suitability for company strategy, and competitive situation. Nevertheless, some overall geometric restrictions and constraints (such as desired size and weight of the product) may already be set at this stage.

As we reach the phase of conceptual design, geometric product information receives more designer attention. As observed by Libardi, Dixon, and Simmons in their valuable survey of computer-aided assembly design [8], most designers follow the *top-down design* approach while carrying out conceptual design. In this approach, the designer begins with an abstract specification of the product and decomposes it into subsystems and subassemblies until the level of primitive components and parts is reached. In order to facilitate preliminary analysis of the design, *abstract geometry* is often introduced, even during the early stages of conceptual design, with the understanding that the geometric information is still incomplete and subject to change during the later phases. The abstract geometry typically focuses on the overall geometric arrangement of the major parts and subassemblies and leaves unspecified the exact geometric details of the individual components and their mechanical linkages. Nevertheless, it is not uncommon that some parts of the design are refined to a much greater degree than others, even during early design, if they are critical to delivering the desired function of the product.

Detailed geometry becomes the focus only in the detail design phase, where the aim of the designer is to optimize the design under the constraints set by the desired performance of the design, various engineering analyses, and the ability of the product to be assembled and manufactured. In this phase, drastic changes to the abstract geometry introduced during conceptual design are not uncommon. For instance, manufacturability issues may dictate that the number of distinct components introduced during the top-down decomposition process be reduced by a composition process in which previously separate parts are combined into a single component that serves multiple subfunctions in the design. This bottom-up optimization of the design may continue later, during production engineering of the product; for instance, manufacturing cost may be decreased by replacing a subassembly of machined parts with a single cast part.

• *Shortcomings of ordinary geometric models for design*

Ordinary CAD systems are primarily geared toward the generation and utilization of the detailed geometric information. Generally, they do not offer much support for other uses of geometric information during the design process, such as the introduction of abstract geometry for preliminary studies, the generation of altered or idealized geometric information according to the viewpoints of various engineering analyses, or the drastic restructuring of the geometry that may be required during the bottom-up composition stage.

One reason for the relative weakness of ordinary geometric modeling techniques in these applications is that geometric models typically can represent the nominal shape of the product at only a single level of abstraction. That is, they cannot make a distinction between the essential geometric information of a part (such as that concerning mating surfaces between major components) and the unessential geometric information (such as that concerning fillets and fairings). This results in a form of *overspecification*: The essential geometric

information must be embedded in a formally complete model, even though the detailed geometry should still be treated as vague or incomplete. While offering improvement in several other respects discussed above, feature models are also too low-level to provide a good basis for supporting all phases of the design process. In particular, *geometric features* that directly correspond to some surfaces or volumes of the part are often oriented toward certain detail design solutions or manufacturing techniques. If they were used for recording the results of early design phases, premature commitment to certain detail solutions would again result.

Ordinary geometric modeling techniques also lack the capability of recording the progression of the design process during the various phases and tasks. They do not preserve the abstract geometric information created during the conceptual design phase as an idealization of the detailed geometric models of the detail design phase, nor do they link the abstract geometry with the still more abstract functional specifications the geometry implements. As a consequence, geometric modeling systems do not, in general, capture the *design intent* of the designer, i.e., the reasons for a particular geometry being in the model in the first place. This limits the usefulness of geometric models as a resource for redesign and also makes difficult the interpretation of geometric models in manufacturing planning applications.

- *Scope and goals of the research*

The problems of ordinary geometric modeling techniques for design applications have been widely recognized, and various higher-level models have been proposed for representing incomplete or vague assemblies and parts, or assemblies and parts on a symbolic level, without reference to geometry [9, 10]. Graphlike symbolic structures have been used by several authors for the design of mechanisms; references [8, 11, 12] provide good reviews of current research into modeling techniques for assembly design and modeling. Nevertheless, these early results do not generally indicate how the various levels of information should be integrated into a model that not only provides a good basis for design but also can be interrogated and manipulated during the later phases of the design process.

The research reported in this paper (begun while the author was a World Trade Visiting Scientist at the IBM Thomas J. Watson Research Center in 1989) is pointed precisely in this direction. The overall goal is to *develop modeling techniques that can support conceptual design of mechanical assemblies according to the top-down approach.* We are also interested in *modeling techniques that permit the restructuring and idealization of geometric information*, as may be required during detail design. Another subgoal of the research is to make it possible for later applications such as process planning or assembly planning to *investigate and manipulate the model information* in the course of their analyses of the product.

The main body of this paper describes the current status of this ongoing research. It is organized as follows: First, in Section 2, we elaborate on the detailed goals of this research. Section 3 provides an overview of the prototype modeling system developed during the research. Sections 4, 5, and 6 give additional detailed information on the modeling techniques chosen for the prototype, namely *hierarchical part-of graphs*, *geometric feature models*, and *geometric constraint management*, respectively. Finally, directions for further work are indicated, and conclusions are given.

The problems of top-down design are illustrated further in the Appendix.

2. Requirements for a CAD system for top-down design

The abstract specification of a design can usually be captured in a structure consisting of the major components or subsystems of the desired product and their desired interfaces, relationships, and constraints. In some design domains, the specification may be expressed by means of numerical performance parameters and expressions; generally, however, nonnumerical, qualitative specifications are also needed. We believe that *the design system should provide facilities for modeling the designed object on a purely abstract level—as a structure of model entities, their relationships, and their properties.* We also believe that some kind of *general annotation mechanism for abstract models* is required.

After the initial specification, the design process proceeds from the abstract to the concrete. Abstract concepts are decomposed into more concrete ones, and new interfaces, relationships, and constraints are created among them. At some stage, it becomes useful to use a geometric representation to express the desired relationships between the concepts introduced. The initial geometry can be mainly dimensionless, concentrating instead on the general geometric arrangement of the main components and their geometric interaction. Except for a few values critical for delivering the desired function of the design, the actual dimensions and coordinate values are unimportant; the geometry mainly serves to specify the geometric constraints among the parts. We believe that *the design system should support the creation of abstract geometry, where important and less important characteristics of the geometry are explicitly distinguished from each other, and the designer can choose the level of detail of the representation according to the particular requirements of the design task.*

During the later stages of the design process, new, increasingly concrete concepts and their relationships are introduced, and the abstract geometry is modified to take them into account. In this process, the aspects of the abstract geometry which initially were treated as being unimportant are refined and more precisely defined. The critical aspects of the geometry, which were already specified, are observed as rigid constraints. Hence, the *previously fixed characteristics of abstract geometry must be treated as further design constraints for the future refinement of the less detailed and unspecified characteristics.* At the same time, the abstract geometry must not unnecessarily limit the freedom of the designer in the later stages.

One important aspect of top-down design is that it permits the designer to concentrate on one subproblem of the design at a time. During the design process, the focus of the designer shifts from conceptual design to the basic design of the various subsystems involved in the conceptual solution and, finally, to the detailed design of each subsystem and each component. The sequence of focus changes can be interpreted as an instance of the design methodology that the designer applies to this design; therefore, to enhance the value of the resulting model, the shifts of focus should be made explicit in the model. We believe that *the design system must support focusing on some particular aspects of the design and, in particular, capture the sequence of focus changes in the model representation.*

Iteration is, of course, an important practical characteristic of engineering design: All successful products are redesigned at some time, the most successful ones the soonest. As exemplified by the case study presented in the Appendix, a redesign may penetrate deeply into the previous design and result in a completely reworked product. We believe that *the design system should capture and preserve the design history for later examination and redesign.* Full support of iterative redesign also requires that *the design system support several alternative elaborations from a common starting point.*

In addition to the goals emphasized above, the design system should also possess several other desirable, if less fundamentally important, characteristics. For instance, the capability of introducing special technological knowledge into the system is helpful. The focus of the present work is not on applying artificial intelligence techniques to construct "intelligent CAD systems"; nevertheless, the accommodation of such techniques in the system should, of course, be possible if and when they become available.

We may summarize the desired capabilities of a modeling system for top-down design of assemblies as follows. The modeling system should

1. Provide support for representing the assembly information on several levels of abstraction (e.g., function, overall geometry, and detail geometry).
2. Represent design intent by preserving the "reason" for the existence of various model entities by means of the sequence of stages and the history of the design. In particular, the system should make it possible to retrace the steps from geometry to the functions that the geometry implements.
3. Make explicit the level of commitment of the designer to various properties of the design. In particular, strongly committed details must be treated as constraints for following design steps. Similarly, the system should support default geometry that represents parts of the design when no commitment to a particular solution has yet been made.
4. Support redesign in various ways, e.g., by supporting parallel refinements of a single starting model. Naturally, it should always be possible to retrace the steps to an earlier design stage.
5. Provide documentation tools for capturing functional specifications, for use by later phases in the design–manufacture process, and for redesign. For instance, it should provide a general annotation mechanism.

Some of the above requirements are domain-independent and are valid for top-down design in, e.g., software development and IC design. Mechanical design is made different from such domains by the special role of geometry: In mechanical design, geometry acts both as a medium for expressing design goals and constraints and as a medium for expressing the result of the design, whereas in software engineering different representations can be used at different levels of the design. Hence, the interface between geometry and other design representations must be more complex than the interfaces between various design object representations in those other disciplines.

3. Overview of the prototype modeling system

A prototype modeling system was built as our initial attempt to achieve the overall goals set forth in the previous section. Because of the experimental nature of the prototype, it was desirable to use a software development environment that provided for rapid prototyping and development of software, had good support for graphical interaction, and allowed extensive reorganization of the developed software as it became necessary. To achieve these goals, the Smalltalk V/286[1] programming environment [13] was chosen. We call the current implementation of the system *WAYT*, for *Why-Are-You-There?*

[1] Smalltalk V/286 is a registered trademark of Digitalk, Inc.

Smalltalk is the grandparent of various object-oriented programming environments. Following the style of this environment, the prototype system consists of three major model structures, each realized as a Smalltalk class hierarchy, and three related, interlinked programs ("Browsers"). The major components are the following:

1. *Design Browser* [14], which can be used to represent *hierarchical part-of graphs* at several levels of detail, and with several parallel elaborations.
2. *Geometry Browser* [15], which can be used to associate *geometric feature models* with the design entities in a design model created by means of the Design Browser.
3. *Constraint Browser* [16], which can be used to create and edit *geometric constraints* describing the mating conditions and other positional and dimensional restrictions among the parts of an assembly.

In the following sections, each of the major components is described in turn, and its relation to the overall goals of the system is explained.

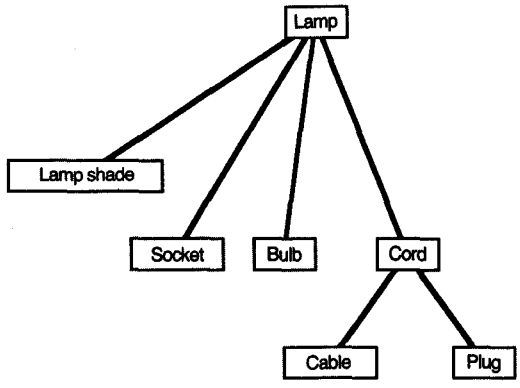

Figure 1

Simple part-of graph of a lamp.

4. Hierarchical part-of graphs

• *Concepts*

One of the major requirements for a modeling system for top-down design is a facility for conceptual, nongeometric modeling, which is mainly concerned with decomposing the product to be designed into subcomponents and their interfaces. To satisfy this requirement, WAYT makes use of so-called *part-of graphs.*

A part-of graph is a way of representing the breakdown of some entities of interest into more primitive entities. For instance, the entity "book" consists of the lower-level entities "front matter," "chapters," and "back matter," which, in turn, consist of still more primitive entities such as "title," "section," and "index." Technically, a part-of graph consists of a set of *nodes* representing the entities of interest, connected by directed *arcs* representing the "part-of" (or "consists-of") relations between the nodes. **Figure 1** is a simple part-of graph that represents the breakdown of a table lamp into more primitive components.

A more general formulation of part-of graphs results if we allow an entity to have several parallel, alternative breakdowns into more primitive entities, and a component entity to appear in all or just some of the breakdowns. Each breakdown is itself a part-of graph. We may interpret the parallel breakdowns as varying "views" of the entity represented. For instance, **Figure 2(a)** shows a "structural" view of the table lamp of Figure 1; observe that some new nodes have been included, and the part-of relationships have been modified. In a similar way, **Figure 2(b)** gives a "functional" view of the lamp.

Hierarchical part-of graphs in WAYT result from organizing the different views of a part-of graph into a tree structure and enforcing consistency rules among them. A view of a part-of graph may itself have any number of views, each representing a particular refinement of the parent view. A subview cannot eliminate any nodes that it has received from its parent view. However, it can introduce new nodes and part-of relationships into the graph, and it can modify the part-of relationships of the nodes it has received from the parent. For instance, the views of Figure 2 are subviews of Figure 1. This hierarchical relationship is shown in **Figure 3.** Observe that all nodes of Figure 1 are present in the views of Figure 2. The design models of the Appendix give further examples of part-of graphs and views.

• *MultiTree data structure*

WAYT represents hierarchical part-of graphs by means of instances of two Smalltalk classes:

1. Nodes are represented with instances of class MultiTreeNode. A MultiTreeNode records the set of component nodes, the parent node (if any), and some attribute information such as the label of the node and its position on a video display.
2. Views are represented with instances of class MultiTree. A MultiTree consists of attribute information and references to its MultiTreeNode(s).

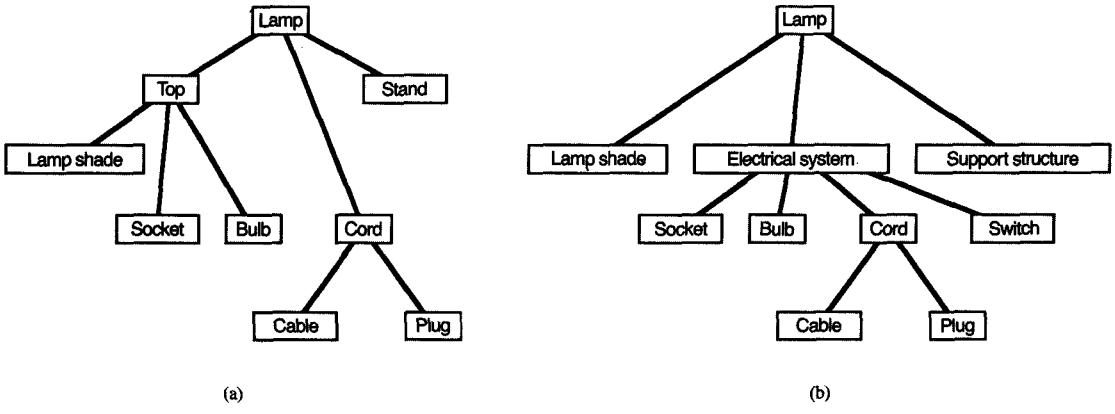

Figure 2

(a) Structural and (b) functional view of the lamp.

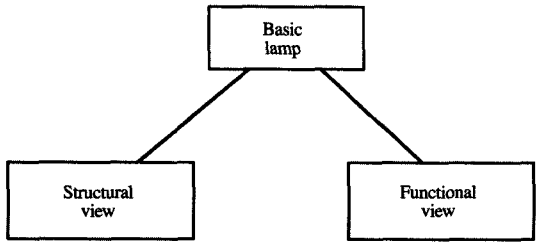

Figure 3

Hierarchical relationship of views of the lamp.

A node appearing in several views can either be represented as a single shared MultiTreeNode, or be explicitly copied in each of the views. Explicit copying is used if the views associate different structure or attribute information with the node; as we shall see, this capability is used to provide support for abstract geometry in WAYT. Further details of the data representation are given in [14].

• *Design model entities*

The MultiTree classes outlined above were intentionally left void of any particular semantics; hence, they should be capable of modeling any kind of hierarchical decomposition of things consisting of other things. According to the principles of object-oriented programming, the actual specialization of MultiTree classes for assembly design modeling was done by deriving new subclasses of the existing classes. These classes add new instance variables and functionality to the basic classes. The design model classes currently used in WAYT are given in **Figure 4**. In the figure, indentation is used to indicate the class structure. Instance variables of each class are listed in parentheses.

In fact, WAYT views are instances of class DesignContext, which adds the capability of recording annotations and a geometric model to the view. The abstract class MultiTree has no direct instances. The abstract class DesignEntity can store annotations and a list of geometric constraints, in addition to the instance variables of the basic class MultiTreeNode. The nodes actually used in WAYT are DesignFeature, which is used to represent a range of conceptual entities such as a subassembly, a component, or a significant geometric detail of a component, and DesignRelation, with its subclasses that represent various kinds of binary relationships between entities. Mechanical joints are modeled as a special kind of relation. The joint classes provide methods that "automatically" insert the geometric features needed for realizing the joint.

• *Design Browser*

The Design Browser is an interactive editor that provides access to the above entities. A sample display of the browser is shown in **Figure 5**. The browser includes

```
MultiTree(name treeRoots superTree subTrees)
    DesignContext(annotation model)
MultiTreeNode(name loc parent children tree superNode subNodes)
    DesignEntity(annotation constraints)
        DesignFeature(relations geometry)
        DesignRelation(from to)
            DesignConstraint()
                GeometricConstraint()
                    Joint(fromFeature toFeature
                          fromJointFeature toJointFeature)
                        SimpleJoint()
                            PlanarJoint()
                                ScrewJoint(auxParts)
                                    CSHoleCBHoleJoint()
                            CylindricalJoint()
                        CompoundJoint(nU nV deltaU deltaV)
```

Figure 4

Design entity classes of WAYT.

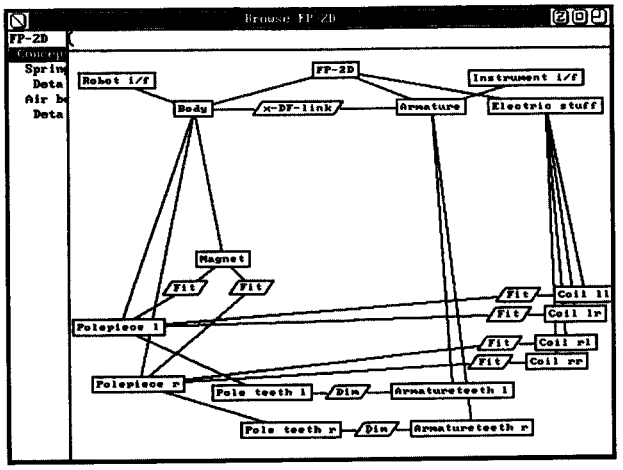

Figure 5

Design Browser.

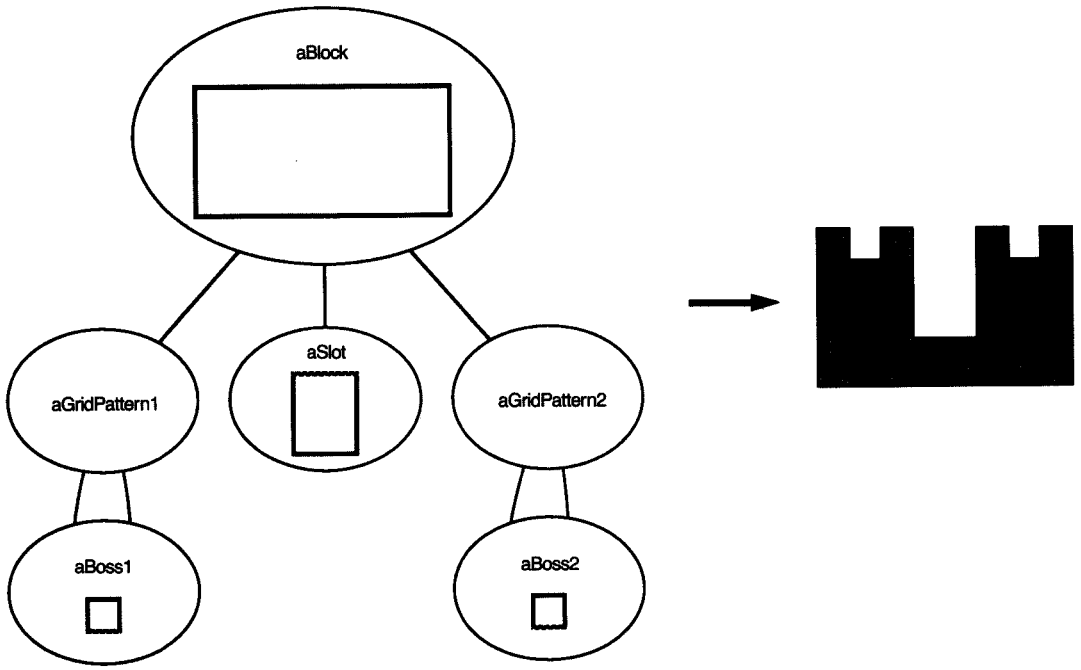

Figure 6

Simple feature model.

```
GeometricModel (name designContext parts
                constraintNetwork annotation)

GeometricEntity(copyOf)
  GeometricFeature(designEntity relations xLeft yTop
                   xDim yDim originX originY
                   otherVariables direction anchors)
    CompoundFeature()
      Combination()
        StepHole(length)
          CounterSinkHole()
      PatternFeature()
        GridPattern(nU nV deltaU deltaV)
    SimpleFeature()
      Hole(radius length)
        Boring()
          Sink()
            ConeSink(sinkAngle bottomRadius)
        FreeHole()
          BlindHole()
            ConeBlindHole(bottomAngle)
            PlaneBlindHole()
          ThruHole()
      PrismaticFeature(width length)
        Block()
        Boss()
        Relief()
        Slot()
          ThruSlot()
    Surface()
  GeometricRelation(from to)
    ConsistsOf()
    OnTheBottomOf()
    PatternOf(nU nV deltaX deltaY)
```

Figure 7

Geometric feature classes.

operations for adding, removing, and manipulating views and nodes. Currently available views are indicated in a list (partially visible) on the left side of the display; the selected view is displayed in a large window on the right side. The browser can initiate new, parallel instances of itself for editing lower-level views, thus simulating a parallel design environment in which several designers are concurrently working on various aspects of a design, or a single designer shifts from one aspect of design to another.

5. Geometric feature models

The design entities of Figure 4 can directly represent only the logical structure of a product. Another class hierarchy that implements a feature-based representation of simple two-dimensional objects is used to represent the geometric information of the product.

- *Feature taxonomy*

The feature model used in WAYT is based on a more elaborate model used in HutCAPP, a feature-based process planner constructed by the author and others at the Helsinki University of Technology [6, 7]. In this model, all geometric objects are represented by means of a tree structure, in which the nodes correspond to various kinds of volume features. The collection of feature types includes both subtractive features that correspond to material removed from the parent feature (e.g., slots) and additive features that represent material added to the parent (e.g., bosses). Arcs of the tree represent various kinds of geometric relationships between the features.

A simple feature model of a part discussed in the Appendix is shown in **Figure 6**. The part is described in terms of six features: a rectangular block feature (aBlock), a slot feature (aSlot), and two pattern features (aGridPattern1 and aGridPattern2), each consisting of two instances of a boss feature (aBoss1 and aBoss2, respectively).

The feature types are implemented in a straightforward fashion as Smalltalk classes; the current collection of feature and relationship classes included in WAYT is listed in **Figure 7**. As indicated in the figure, features are either simple features (e.g., holes) or compounds consisting of simple features (e.g., a countersink hole, which consists of a cone sink and a through-hole). Pattern features consisting of several instances of a feature are also included. Some intermediate classes, such as StepHole and Boring, are included for future growth and compatibility with HutCAPP. The class GeometricModel is included for simplifying the interfacing of design models and geometric feature models.

Instance variables of the various classes contain mainly the "natural" parameters of each feature; for instance, holes are expressed in terms of the radius and the length of axis. Similarly, all prismatic features are expressed in terms of length and width. Instance variables originX, originY, and direction indicate the location of the origin of the feature and its orientation.

- *Inheritance of geometric models*

Recall that the copies of a design entity appearing in several views are represented in terms of separate nodes in the underlying data structure and that each copy can have its own associated attribute information. WAYT makes use of this to implement the requirement of supporting abstract geometry.

The approach chosen for WAYT is based on the inheritance of geometric information from parent view to subviews. If a copy of a design entity is not directly associated with geometric information (i.e., its instance variable geometry is empty), the geometry of the copy of

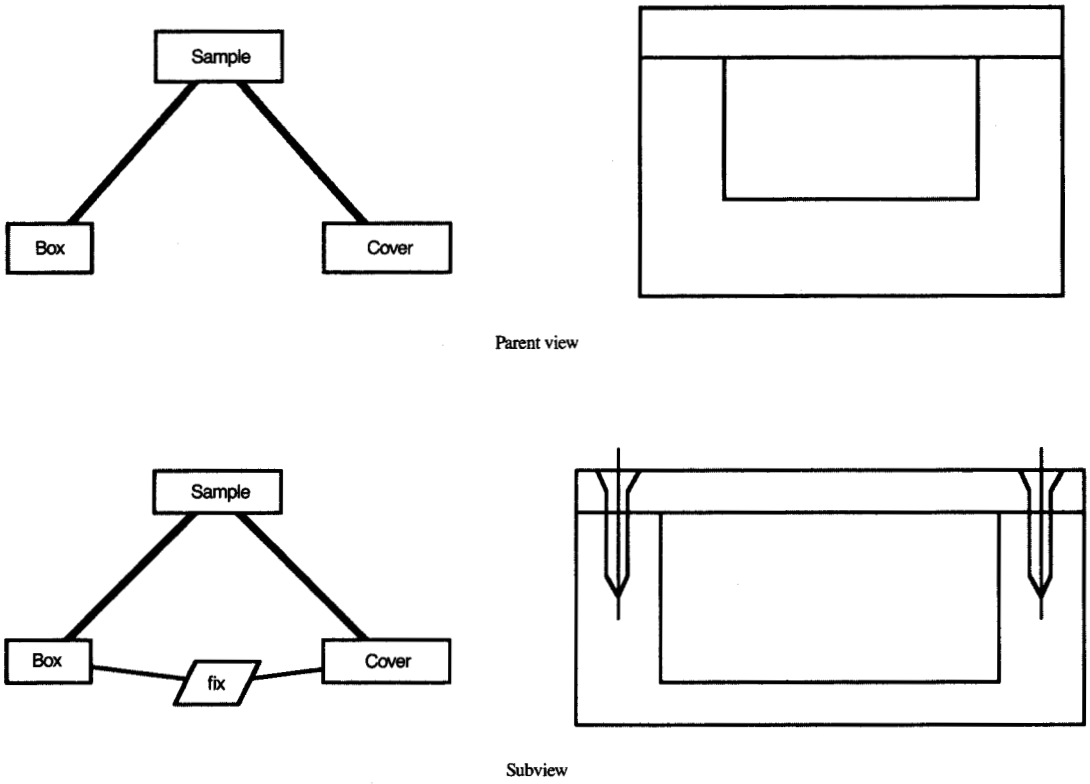

Figure 8

Inheritance of geometry.

the design entity in the parent view is used (if such a copy exists). If the parent's geometry is empty, the search continues to higher levels.

If the geometry inherited in this fashion is modified in a subview (e.g., new features are inserted in the geometry), WAYT creates a copy of the geometry inherited from the parent view and assigns it as the geometry of the appropriate node in the subview. In order to enforce the compatibility between the original geometry and the copied geometry, the copied features are tagged with a reference to the corresponding original geometry. In particular, when copied geometry is edited, features tagged as copies cannot be deleted. Similarly, if the designer changes geometry which has been copied elsewhere, a confirmation is first requested. Consequently, the designer can add only new features to inherited geometry. Naturally, the designer can also change the dimensions and locations of inherited features (within the degrees of freedom provided by geometric constraints, as detailed in the next section).

A simple example of geometry inheritance is shown in **Figure 8**, which consists of a parent view and a subview. The parent view consists of three design entities, Sample, Box, and Cover. The geometry associated with these nodes is shown on the right. In the subview, shown below, a relationship entity (called fix) representing a mechanical joint between the box and the cover has been added to the design model. The geometry of the parent view has been copied as the geometry of the subview, and new features have been added to represent the details of the mechanical joint between the two parts. Observe that some dimensions and locations of the inherited geometry have also been changed.

- *Geometry Browser*

The actual manipulation of geometric feature models is done with the Geometry Browser. With this browser, it is possible to create, modify, and delete geometric feature instances. The browser can edit several parts simultaneously (for instance, both geometries associated

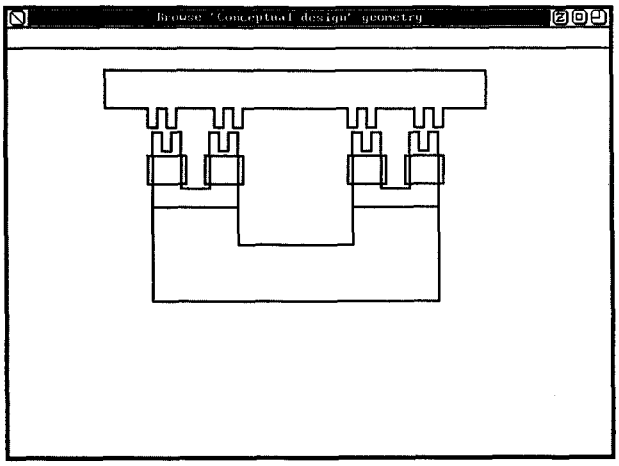

Figure 9

Geometry Browser.

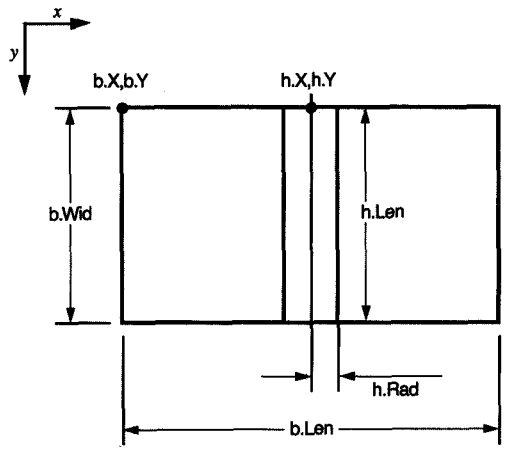

Figure 10

Block and through-hole.

with a DesignRelation, or all geometries associated with a whole DesignContext). The browser is typically started by the Design Browser for editing (or creating) the geometry of a design model node; if necessary, the geometry of a parent view is first copied.

A sample display of the Geometry Browser is shown in **Figure 9**.

6. Geometric constraint management

The geometric feature representation and the geometry inheritance mechanism described in the previous section are not sufficient for satisfying the requirements set in Section 3 of this paper. The copied geometry is protected against deletion only; the dimensions and locations of the various parts are all changeable in the subview. Furthermore, it is not possible to express the needed mating conditions between parts, nor the detailed relationships among the features of a part.

However, WAYT manages all required tasks, and some others, by means of *geometric constraints.* In particular, geometric constraints are used to represent and manage the following:

1. Dependencies between instance variables of geometric features.
2. Dependencies between features forming compound geometric features.
3. Dependencies between geometric attributes of features within a part.
4. Mating and other dependencies between parts.
5. The degree of commitment that the designer has to any of the dependencies or constraints of the above types.

• *DeltaBlue: An incremental constraint-satisfaction algorithm*

Many of the dependencies and constraints listed above can be represented in terms of simple equations involving geometric attributes of the parts considered. For instance, the constraints that a through-hole h must reside at the center of a rectangular block b and extend all the way through the block can be translated to the following collection of equations (see **Figure 10** for nomenclature):

$$h.Y = b.Y,$$

$$h.X = b.X + b.Len/2,$$

$$h.Len = b.Wid.$$

Actual assignments of dimensions and locations to the entities can be modeled as further constraints:

$$b.X = 4,$$

$$b.Y = 3,$$

$$b.Len = 18,$$

$$b.Wid = 10,$$

$$h.Rad = 1.$$

The problem of enforcing such constraints while still providing the user with facilities for changing the numerical values of the constrained geometric attributes is best solved by means of a *constraint-satisfaction algorithm* that can represent numerical constraints of the above type and construct a procedure for satisfying the constraints (if possible). A review of constraint-

satisfaction techniques is beyond the scope of this paper; for an overview, see [17]. Instead, we limit the discussion to the particular algorithm chosen for the present work.

DeltaBlue [18, 19] is a simple algorithm for constraint satisfaction that is applicable to acyclic linear numerical equality constraints; this simple domain was deemed sufficient for the prototype. DeltaBlue is based on constructing an explicit representation of the numerical equations corresponding to the constraints. The internal representation is a multigraph whose nodes represent the constrained values; in the above example, each geometric attribute would be represented as a variable. Arcs of the multigraph represent the constraints. For instance, the multigraph representation of the example of Figure 10 is shown in **Figure 11**.

Every arc of the multigraph is associated with alternative procedures that can satisfy the corresponding constraint when executed. For instance, the constraint

$$a = b + c$$

would typically be associated with the procedures

$$a := b + c,$$

$$b := a - c,$$

$$c := a - b,$$

where := denotes assignment, in contrast to =, which denotes the equality constraint. Given the multigraph, DeltaBlue will find a sequence of procedures that, when executed sequentially, will result in a set of values for which as many constraints as possible are satisfied.

The algorithm is based on the concept of "strength." All constraints have a strength attribute given by the creator of the constraint (in our case, WAYT). The strength of a variable whose value is (yet) undetermined by constraints is the absolute minimum; otherwise, the strength of the variable is the minimum of the strength of the constraint that determines the value of the variable and the strengths of the input variables of the corresponding procedure.

DeltaBlue works by considering one constraint at a time and constructing an intermediate solution at each step. When a new constraint is added to the multigraph, the algorithm selects the weakest associated variable as the potential output. If the constraint is stronger than the variable, the constraint can be satisfied by assigning a value to the variable by means of the corresponding procedure. Other variables act as input variables for the procedure. Otherwise, i.e., if the constraint is weaker than the weakest associated variable, the constraint cannot be satisfied. If a previously satisfied constraint is overridden, the algorithm will try to satisfy it recursively. A detailed description of the algorithm is given in [19].

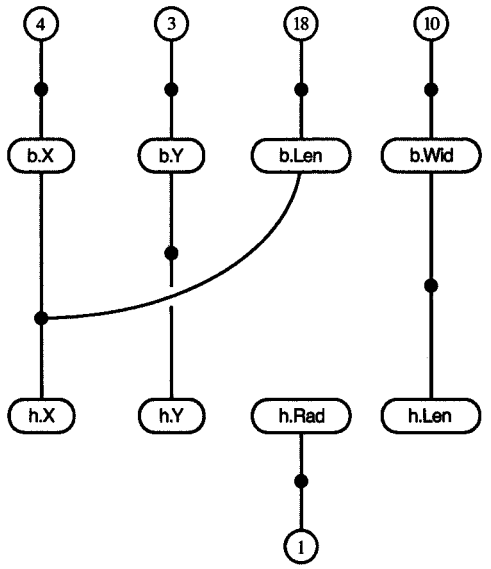

Figure 11

Multigraph representation of constraints of Figure 10.

A solution to the constraint graph of Figure 11 is shown in **Figure 12**. In the figure, arrowheads are used to indicate the chosen procedure of each constraint, and numbers indicate the evaluation order of the procedures.

WAYT capitalizes on several particular properties of DeltaBlue. First, the procedure outlined above can be shown to find a best possible solution to the constraint graph in the sense that the constraints are satisfied in decreasing order of their strength. Hence, DeltaBlue also behaves properly in overconstrained situations. Second, the incremental nature of the algorithm is well suited for interactive use, where constraints are added and removed by the designer. Third, as we shall see, the concept of the strength of constraints can be readily utilized for modeling the degree of commitment to a constraint held by the designer.

On the negative side, DeltaBlue does not provide support for constraints that correspond to inequalities. For instance, in the above example the additional useful constraint

$$b.Len > 2 * h.Rad$$

cannot be represented or enforced.

- *Constraints and features*

The constraint mechanism is integrated with other components of WAYT by making all instance variables

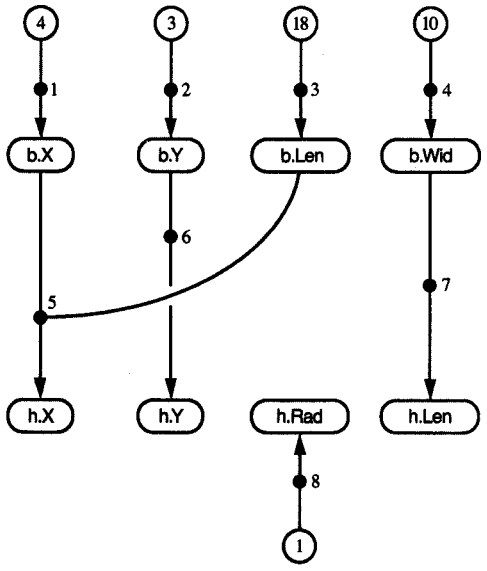

Figure 12

Solution of constraints of Figure 11.

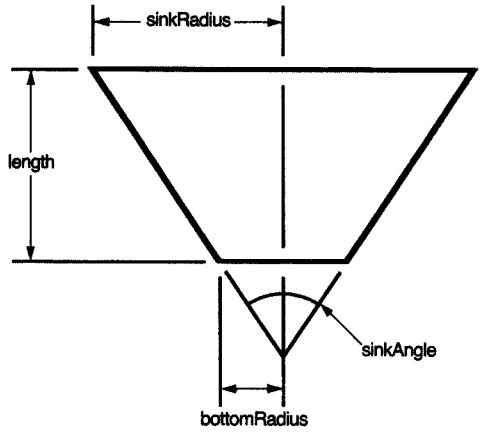

Figure 13

Natural parameterization of ConeSink.

representing dimensions and locations of geometric features instances of a class that represents constrained variables instead of plain numbers. This opens the door to using geometric constraints for expressing various aspects of the desired behavior of features.

One useful application of constraints is the relaxation of the fixed parameterization of geometric features implied by Figure 7. For instance, instead of using the top left point, it might sometimes be more convenient for the designer to use the center point of a rectangular block to specify its position. With the help of the constraints, it is straightforward to state that

$$b.Xcenter = b.X + b.Len/2,$$

$$b.Ycenter = b.Y + b.Wid/2,$$

and then let the user assign values to Xcenter and Ycenter instead of X and Y.

A more complex example is given by the cone sink, which has a redundant set of "natural" parameters (see **Figure 13**), bound by the following constraint:

$$length = \frac{sinkRadius - bottomRadius}{\tan(sinkAngle/2)}$$

By including this expression as a constraint, WAYT can let the user select any three of the four parameters that may be convenient.

WAYT also uses constraints to express the dimensioning of features with respect to one another. In simple cases, the constraints are inserted "automatically," with no further user input. For instance, if the user creates a slot within a block, the constraint that the slot must reside along the edge of the block is generated by WAYT. Joint features in mating parts are also constrained to reside in the proper relative positions with respect to each other. In more complex cases, the user must add the constraints himself.

Finally, WAYT also uses constraints for representing the positioning of parts with respect to one another.

Figure 14 gives an example of positioning constraints for three parts discussed in the Appendix. The two parts PL and PR, drawn in darker gray, are constrained to reside on top of the third part, M. The length of the slot S must combine with the lengths of PL and PR as shown. These positioning constraints are expressed with the following:

$$M.X = PL.X,$$

$$PR.X + PR.Len = M.X + M.Len,$$

$$S.X = M.X + PL.Len,$$

$$S.X + S.Len = PR.X.$$

Specification of constraints relating two parts requires that a relation node connecting the respective design entities be defined. Certain often-used constraints (e.g.,

"part A is positioned on the top edge of part B") can be inserted simply by selecting the needed constraint from a menu. Other constraints must be edited by means of the Constraint Browser described below.

Naturally, parametric editing of the geometry is possible through a simple menu-based user interface or with the Constraint Browser.

• *Inheritance of constraints*

In order to protect abstract geometry defined in a parent view against changes, the system makes a local copy of it before any modifications are performed. Naturally, the copied geometry should still behave according to the constraints that have been set in the parent view(s). To accomplish this, the original geometry and its copies share their geometric instance variables. For example, even though a feature and its copy are two different entities, the x-coordinates of both the feature and its copy are represented with the same single entity. Consequently, all constraints that relate to the x-coordinate are still in effect in the subview and for the copied geometry.

Of course, constraints set for a copied geometry in a subview should not affect the original geometry in the parent view. To provide the expected behavior, all constraints set in a view are stored in the corresponding local copies of the nodes of the view; in particular, all constraints related to a single part are stored in the corresponding DesignFeature, while constraints expressing relationships between parts are stored in the corresponding DesignRelation. As a result, the "local" constraints defined in a subview can be removed from the constraint graph when the parent view is browsed. Hence, at any time, only those constraints defined in the currently active view and its parent views are in effect.

• *Level of commitment*

The relative importance of constraints is modeled naturally with the help of the strength concept of DeltaBlue. When a value is assigned to a geometric variable, WAYT currently makes a distinction among three levels of commitment to the value:

1. *Anchored value* The designer is strongly committed to a particular value of a variable. This is modeled by means of a "strong" equality constraint of the type aVariable = constant.
2. *Default value* The designer has assigned a value to the variable, but is not committed to this particular choice. This is modeled by means of a "weak" equality constraint.
3. *Don't care* The designer is not interested in the value of the variable and has not provided any explicit value for it. The value of the variable may be determined by constraints, if any.

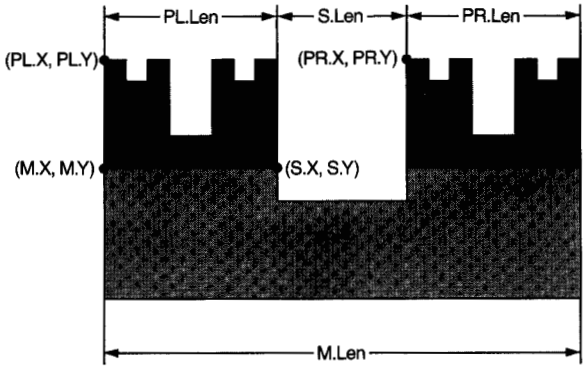

Figure 14

Horizontal positioning constraints.

These concepts are easily modeled with the strength attribute of constraints. The strength attribute is recorded as a nonnegative integer, where 0 denotes the absolutely weakest strength. By convention, we use the value 999 as the absolutely strongest strength.

In the root view of a design model, strong and weak constraints are assigned strengths 999 and 1, respectively. In order to protect anchored values while providing access to default values, the strengths of strong and weak constraints in a subview are set to $s - 1$ and $w + 1$, where s and w are the strengths of the strong and weak constraints in the corresponding parent view, respectively. Hence, a subview cannot override anchored values set by any of its parent views. However, it is capable of either assigning a new default value to a weak variable by overriding the existing constraint (in which case a subsequent subview can override the new default value) or assigning an anchored value for the variable (in which case the new value cannot be overridden).

Currently, WAYT treats all constraints relating several variables as being "strong" ones. It would be straightforward to extend the above classification to those constraints as well.

• *Constraint Browser*

Browsing and editing variables and their constraints is accomplished with the Constraint Browser, a simple editor for DeltaBlue constraints and variables.

A sample display of the browser is given in **Figure 15**. The browser displays a scrollable selection list of variables on the left edge of the display. The two windows on the

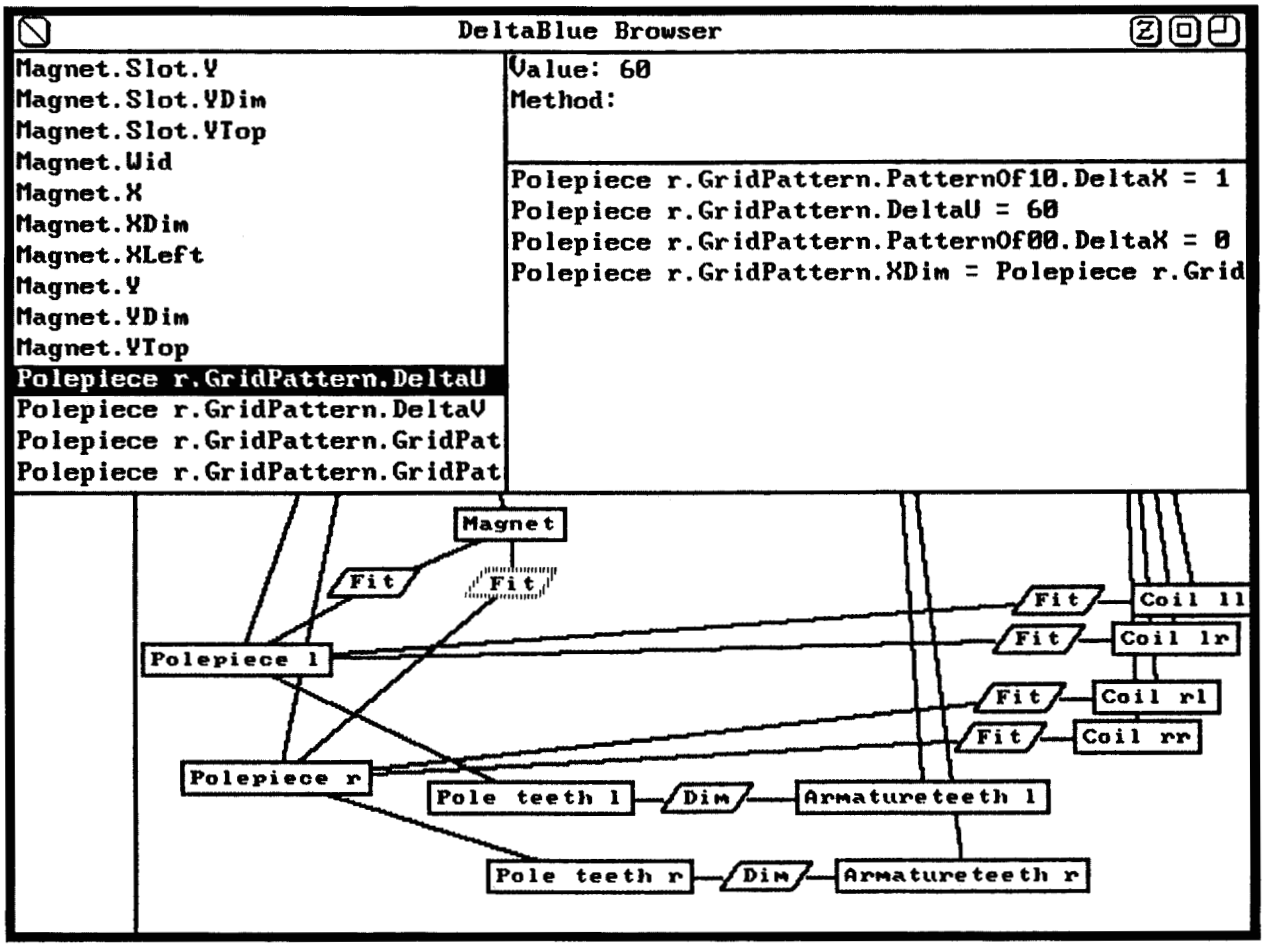

Figure 15

Constraint Browser.

right display the value of the currently selected variable, the method chosen for assigning the value (if any), and a scrollable list of the constraints currently referring to the variable. The browser provides operations for adding and removing constraints and variables.

The selection list of variables is set according to the entity whose constraints are browsed. For a DesignFeature, variables of the geometry of the design entity are listed, while for a DesignRelation, the variables of both parts are shown. It is also possible to invoke the browser while editing the geometry of a part with the Geometry Browser; in this case, only the variables pertaining to the currently selected feature are shown.

7. Directions for further work

The research described in this paper is still under way, and there are several areas where further work is required or contemplated.

The multitree data structure is based on several simplifying assumptions. Any sharing of design entities among parallel views is based on inheritance—that is, two views cannot share an entity unless they inherit it from a common parent view. This is clearly too restrictive. The current implementation of the multitree representation and its editing protocols also assumes that a single set of features is sufficient for expressing all views of the design. This also may be too restrictive; in particular, the model manipulation protocols must be relaxed and generalized in the future.

Another problem with the current view mechanism is that the "existence constraint" of design entities and geometric features is always strict: A subview must include all inherited nodes and their geometries. With the help of a more elaborate mechanism for dealing with views, the modeling expressiveness of WAYT could be greatly increased. An alternative mechanism is offered by

various truth maintenance systems (TMS), which would make it possible to model explicitly the constraints for the existence of an entity in the model. Outstanding examples of TMS in advanced CAD systems are given in [20, 21]. Reference [22] describes the use of a TMS in combination with a powerful logic-based constraint-satisfaction system.

The restriction to two-dimensional geometry and to a fairly simple collection of geometric features is obviously a severe limitation in the current system. In the future, we contemplate interfacing WAYT with HutCAPP [6, 7], which provides a three-dimensional version of a feature library similar to the one used in WAYT and also includes access to a powerful solid modeling system [23].

An original goal of WAYT, not explicitly mentioned above, was to provide an "intelligent explanation" of why some geometric feature is included in the model. For instance, a geometric feature might serve the purpose of implementing a geometric joint, and the joint would implement a mechanical linkage, which in turn would implement a design requirement for certain relative motion of parts. While the basic information for answering such queries is to a large extent available in the model, no such functionality has yet been implemented.

Other areas where improvement would be possible and desirable include the following:

1. *Constraint editor* Currently, the constraint-satisfaction algorithm knows about constraints only from a fixed (albeit easily programmable) selection. With the help of a computer algebra system, it might be possible to allow constraints to be specified simply in terms of their defining equations and to construct the satisfaction procedures on-line.
2. *Constraint satisfaction* A more elaborate constraint-satisfaction and propagation mechanism (such as in, e.g., [24, 25]) will be needed eventually.
3. *Geometric transformations* WAYT currently deals only with rotations that are multiples of 90 degrees. This obviously should be relaxed.
4. *Relationships of parallel views* Currently, a DesignFeature of one view must correspond to single DesignFeatures in other views. This may not be sufficient for expressing conveniently more complex mechanical assemblies.
5. *Cutting and pasting* The user interface can be improved in several ways in order to make it more convenient and efficient. The most fundamental improvement would be a facility for "cutting and pasting" design entities and geometric features from one design to the other, or from a prescribed library of standard solutions. The main difficulty with this extension is the preservation of the constraints defined for the moved entities.
6. *Large models* The current implementation does not scale well to large models, where thousands of parts must be managed. While the basic algorithm for constraint satisfaction should deal gracefully with a large number of constraints, another technique should be used for massive changes where large numbers of constraints are manipulated in a single step.

8. Conclusion

We have described a modeling system for the design of assembled products that possesses several characteristics not found in current commercial or research systems. These include

1. Support for hierarchical views of the designed object.
2. Support for abstract geometry.
3. Support for geometric constraints for modeling the design intent and the mating of parts of the assembly, and also for parametric design of feature models.

The techniques used to implement these characteristics are well known and relatively simple to implement and interface with existing modeling systems. We believe that future CAD systems will commonly include techniques such as those described herein.

Despite the fact that the implementation of Smalltalk used is based on interpretation, the system is quite responsive. The heaviest computational load occurs when the current view is changed and many constraints must be removed or reinserted. The largest model constructed with the system so far has 716 variables and 608 constraints. On a PS/2[2] Model 80 personal computer, it takes nearly 50 seconds to remove or reinsert all constraints. Ideally, WAYT would be implemented as a compiled module of a general-purpose mechanical CAD system; in this case, a speed improvement of one order of magnitude can be expected.

WAYT, as currently implemented, is directed more at capturing the design process than at being an active design tool. Nevertheless, even a limited-purpose system such as WAYT can be useful. We believe that understanding the issues involved with the representation of functional and conceptual design information forms a sound basis for providing active support during early phases of design.

Appendix: A case study of top-down design

This appendix illustrates the modeling problems of top-down design with the design process of a real product: an instrument designed and manufactured at the Thomas J. Watson Research Center. The design study illustrates the design principles, central design phases, and major design

[2] PS/2 is a registered trademark of International Business Machines Corporation.

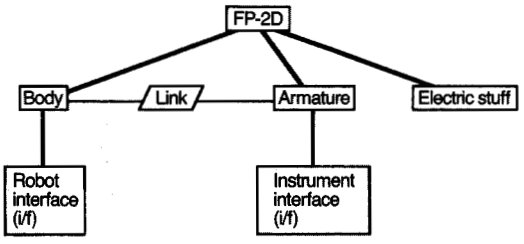

Figure A1

Functional model of generic positioner device.

decisions regarding the device; detailed information may be obtained from the original documentation[3] [1–5].

A simplified design model for the instrument is developed, compatible with the modeling facilities described in the main body of this paper.

• *Design principles and functional specification*

The *Fine Positioner* (FP) is a robotic planar positioning device intended for various manufacturing, laboratory measurement, and instrumentation tasks. It can be positioned within 0.2 μm of the desired point over a total movement range of 1.8 mm. The three-degree-of-freedom version of the FP also provides 1.75 degrees of rotation. The FP is expected to be used as the terminal wrist of an ordinary robot arm that provides the coarse motion. Since its initial design in 1984, the FP has gone through extensive design and engineering changes and has now reached a state where it can be mass-produced.

A *positioner*, in general, is a device that moves an *armature* with respect to a *body*, according to the instructions of external control. In turn, the armature is connected to the device being positioned, e.g., a measurement probe.

The preceding description provides a superficial statement of what the product to be designed must be like. There are two major external interfaces: the linkage with the robot arm and the linkage with the control unit. Similarly, there are two major internal parts linked with each other: the body and the armature.

The general functional specification of the positioner can be translated into the diagram shown in **Figure A1**. The diagram uses three constructs:

1. Rectangles containing text represent a hierarchy of entities corresponding to the product, its subassemblies, its components, and, ultimately, its geometric features. For simplicity, we use the term *part* to denote these entities.
2. Unlabeled arcs drawn with heavy lines represent hierarchical *part-of relationships* between parts.
3. Thin arcs with parallelogram label boxes represent binary *constraints* between two parts. We use these entities liberally to represent all interface, dimensioning, assembly, and kinematic constraints.

The specification shown in Figure A1 attempts to capture the generic properties of all conceivable positioners. We speculate that such a specification could be contained in a large library of engineering idioms included as a resource of a future-generation CAD system.

A particular specification of a particular fine-positioner product is a refinement of the generic specification of Figure A1. Such a refined description would indicate the required relative motion for the two major parts, the desired speed and acceleration of the motion, the payloads that must be handled, and the desired size and weight of the product. In the case of the FP, the desired motion was initially specified to be planar (x–y) translation. Later, rotation around the z-axis was also desired. An important practical design constraint for the FP was a requirement for contamination-free operation: Because the FP is intended for delicate manufacturing or laboratory measurement tasks, it must not itself produce noise, vibration, or particles that might harm the product being assembled or the experiment being performed.

• *Conceptual design*

The overall goal of the design of the FP was to find an implementation for the body and the armature such that the desired relative motion could be produced and the desired external interfaces could be provided. This design task was complicated by the additional design constraint of contamination-free operation, which precluded several otherwise possible solutions for realizing the relative motion (e.g., hydraulic linkages).

The principal solution chosen was to use a direct-drive electromagnetic actuator. A permanent *magnet* is used to create a magnetic flux path between the body and the armature. In **Figure A2**, the thick lines indicate the flux. The flux can be steered by means of *coils* wound around magnetic *pole pieces*. As shown in **Figure A3(a)**, when a suitable positive current is applied in the coil, the flux on the left side of the pole piece is increased and the flux on the right side is decreased. The increased magnetic flux forces the armature to move with respect to the body so that the small bosses, or *teeth*, in the armature and the pole pieces become aligned on the left side. At the same time, the teeth on the right side become disaligned. With

[3] R. Hammer, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, personal communication, 1989.

a suitable negative current, the opposite takes place [**Figure A3(c)**]. When no current is applied, the rest position [**Figure A3(b)**] is assumed. Intermediate positions can be very accurately achieved by controlling the current.

We can summarize the above design in the diagram shown in **Figure A4**, which is an amplified version of Figure A1. (Figure A4 is a redrawing of Figure 5 of the main body of the text.) In the diagram, new entities representing the magnet, the two pole pieces, and the teeth in the pole pieces and armature have been inserted. *Electric stuff* now includes the four coils for controlling the current. The model also includes constraints representing the dimensional relationships among the magnet, the pole pieces, the coils, and the teeth in the pole pieces and in the armature.

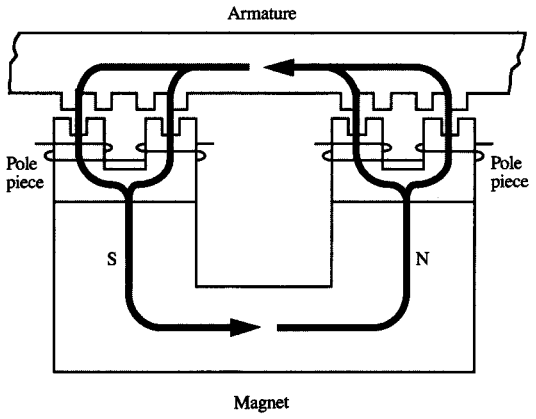

Figure A2

Magnetic flux path.

• *Engineering solution 1: Spring suspension*

The conceptual design of Figure A4 does not yet specify how the armature is linked with the body so that the proper distance between the teeth is maintained while the desired relative motion between the body and the armature is allowed. The first prototype of the FP attacks this problem by means of a *spring suspension* [2]. The armature and the body are connected with a pair of flexible springs, one on the left side and one on the right side of the assembly. The springs must be flexible enough to allow the desired movement and rigid enough to maintain the proper separation between the teeth. The actual three-dimensional design makes use of two pairs of springs for providing independent flexing along both x- and y-axes.

The design model shown in Figure A4 has been augmented to include two new parts corresponding to the springs; see **Figure A5**. Observe that the springs are organized as "parts of" the link constraint between the body and the armature. New constraints representing the mechanical joints of the springs with the armature are also included.

Next, the body assembly is detailed, with several new components. A simplified geometric representation of

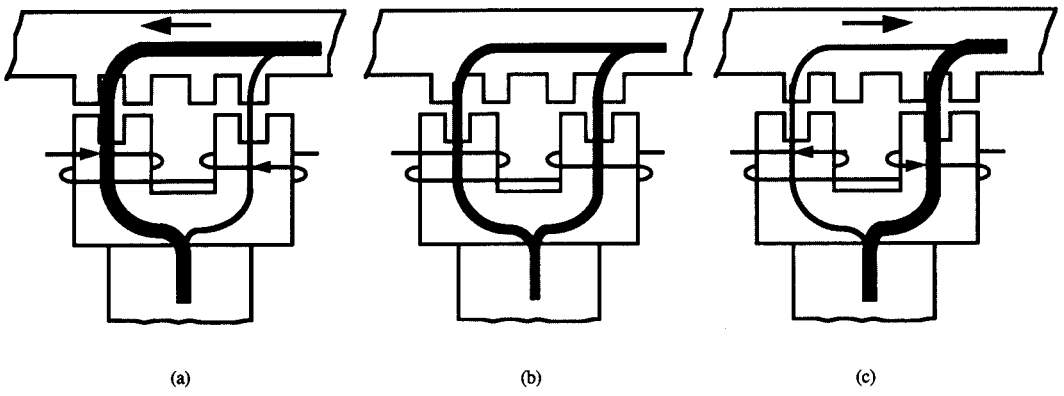

Figure A3

Steering the flux.

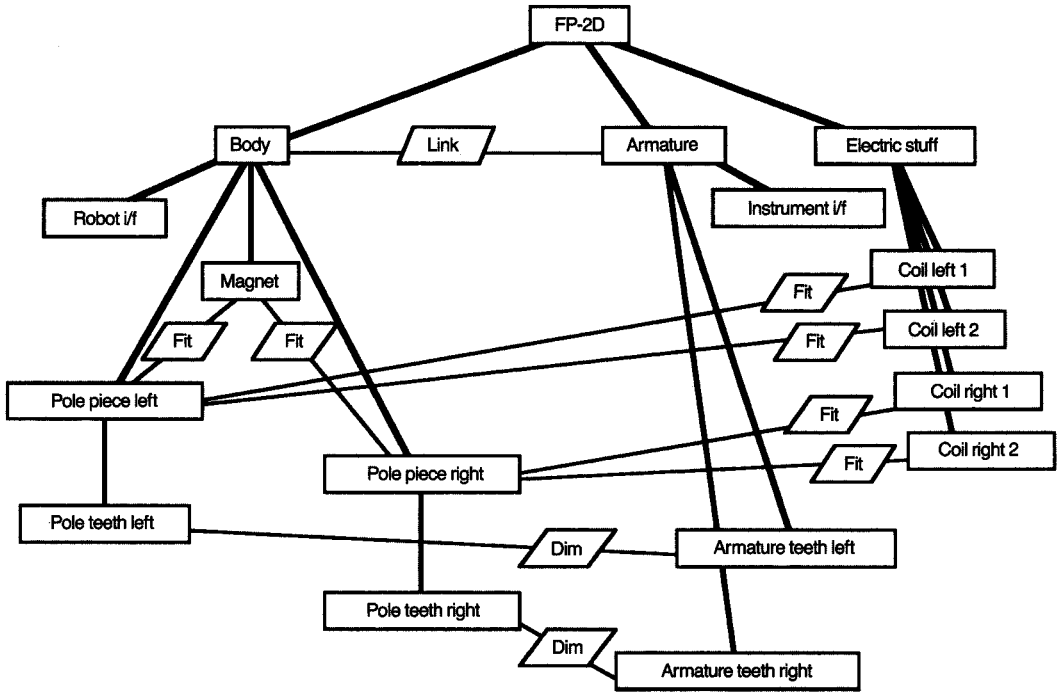

Figure A4

Refined model of the FP.

this design is shown in **Figure A6** (see also Figure 9 in the main body of the text), while **Figure A7** gives the corresponding design model. See [3] for information on the analogous three-dimensional design.

The model of Figure A7 must still be detailed to complete the design. The dimensions, tolerances, materials, surface finish, and plating of all components must be determined to ensure the correct operation of the device. The final shapes of all components must also be selected while taking into account the ability of the design to be manufactured and assembled.

• *Engineering solution 2: Air bearing*

The original design of the FP was successful in that a series of working prototypes were manufactured and the basic design principles validated. Nevertheless, the design had a number of problems:

1. The spring suspension introduces a small amount of unwanted motion along the z-axis. Although this motion is predictable, it nevertheless is a nuisance for development of control algorithms.
2. The air gap between the teeth is implemented with a long kinematic chain of components and linkages. Consequently, the prototype is difficult to assemble and adjust.
3. It is not possible to generalize the design to allow small rotations around the z-axis without even greater assembly and adjustment problems [4].
4. The product has too many parts; the part list of [3] includes 89 parts.

The spring suspension can be identified as the source of the first two problems. To improve the design, it must be eliminated. As a result, the design process must be restarted from the result of the conceptual design stage of the FP, shown in Figure A4.

An alternative engineering solution to the suspension problem, which has all the desired characteristics and also avoids the problem of adjusting the gap between the

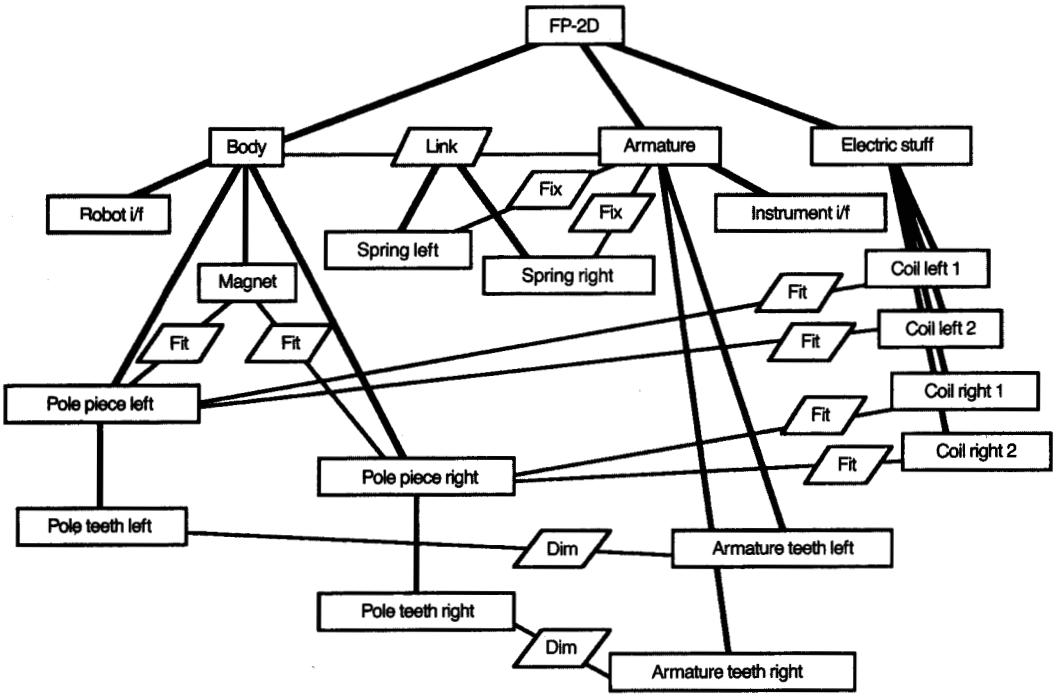

Figure A5

Fine positioner with spring suspension.

teeth, is to use an *air bearing* between the body and the armature. An air bearing requires two highly planar mating surfaces. They are separated by a thin layer of air blown through channels and orifices created in the surfaces. Hence, the various components on the top surface of the body of the FP must be integrated into a single unit, with a planar, polished top surface. Similarly, the bottom surface of the armature must become a flat, polished surface.

Instead of being separate components, the pole pieces and the pole teeth are now inserted into a new component, the *stator plate*, by milling slots into nonmagnetic material, inserting the teeth parts, made of magnetic material, tightly into the slots, and covering and polishing the result. Similarly, the armature teeth are inserted into the armature plate. The magnet is integrated into a new component, the *flux return plate*. Because of the larger production volume anticipated, it is worthwhile to design a *housing* into which the other parts can be

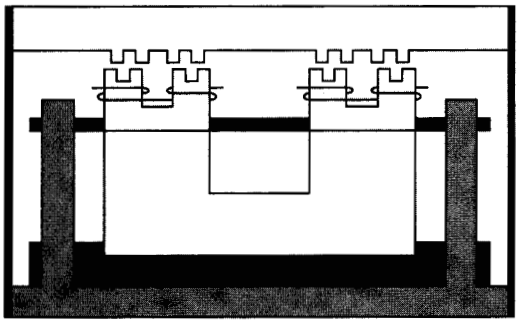

Figure A6

Geometry of the body.

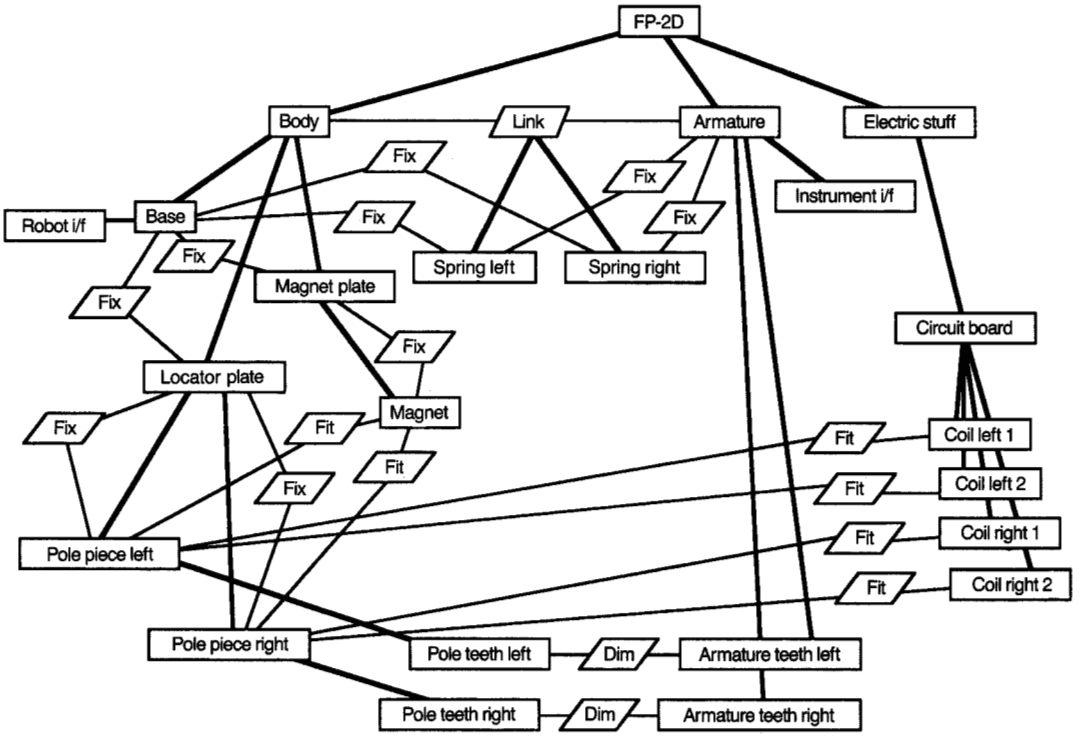

Figure A7

Detail design model of spring suspension solution.

assembled. The housing also provides the interface with the robot arm. Finally, some new components are needed for maintaining the air flow. A design model that includes the above refinements is shown in **Figure A8**, while **Figure A9** shows the corresponding geometric arrangement of the major components.

Again, the detailed geometric shapes of all components, their materials, the mechanical joints between the components, and the auxiliary parts needed for the assembly must still be determined. See [5] for detailed information.

• *Notes*

The case study illustrates well several properties of design processes that are also discussed in the main paper. Throughout the case study, we have presented the progress of design in terms of a structure consisting of the major subsystems, components, and vital details of the components. The important interfaces, relationships, and constraints among the components are highlighted in the design model. The design models are organized according to the hierarchical part-of graph representation discussed in the main paper. Observe that Figures A4, A5, and A7 are all consecutive refinements of Figure A1, while Figure A8 is an alternative subview of the view of Figure A5.

Abstract geometry is introduced to capture information on the desired relationships among the components of the design and to represent vital dimensions and tolerances. Observe how some aspects of the geometry are specified at a finer degree of detail than others; for instance, the geometry of the teeth and the relationship between armature and pole piece teeth is already represented accurately in Figure A2. As the design progresses, we may observe how the focus of the attention of the designer shifts from one aspect to another: from conceptual solution to suspension to body geometry to geometric detailing.

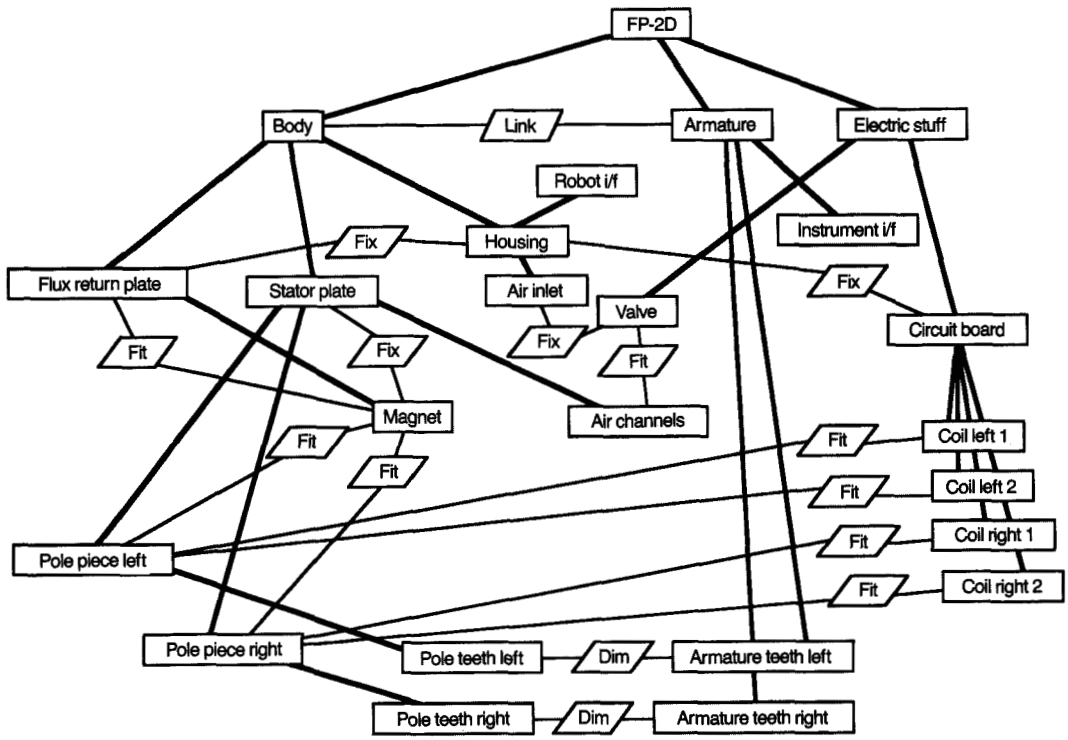

Figure A8

Refinement of design for air bearing solution.

The FP also illustrates the special problems of redesign. Although the two product generations do not share a single component, their actual design follows a similar outline: The problems of armature–body linkage design and body assembly design are solved as separate steps. Interestingly, during the redesign of the FP, an intermediate prototype that featured air suspension but still had a body assembly design similar to the original prototype was designed and built.

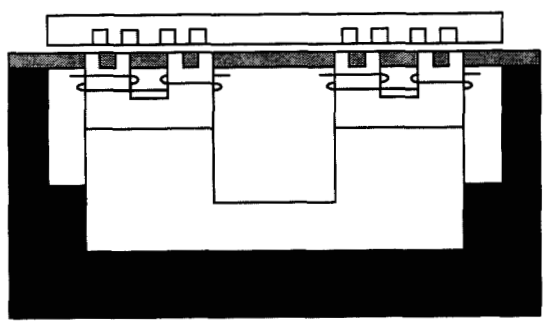

Figure A9

Geometry of the air bearing solution.

References for Appendix

1. R. L. Hollis, Jr., "A Planar XY Robotic Positioning Device," IBM Thomas J. Watson Research Center, Manufacturing Research, Yorktown Heights, NY, 1985.
2. R. L. Hollis, Jr., "Precision X–Y Positioner," U.S. Patent 4,509,002, 1985.
3. R. L. Hollis, Jr., "Geometric Design Processor (GDP) for Assembly Modeling: Fine Positioner Assembly Sequence," IBM Thomas J. Watson Research Center, Manufacturing Research, Yorktown Heights, NY, July 1984.
4. R. L. Hollis, Jr. and B. Musits, "Electromagnetic X-Y-Theta Precision Positioner," U.S. Patent 4,514,674, 1985.

5. R. Hammer and R. L. Hollis, Jr., IBM Patent Disclosure No. YO887-0604, July 1987, describes an air-bearing-supported three-degree-of-freedom fine positioner.

Acknowledgments

This work is dedicated to the memory of Dr. Markku Tamminen. The author gratefully acknowledges the helpful discussions and comments of colleagues at IBM Research during the early phases of this research, including Jarek Rossignac, Erik Jansen, and Ramesh Srinivasan. The help of Bob Hammer during the case study work was indispensable. The Fine Positioner as a target for design study was suggested by Lee Nackman. The author also thanks the anonymous referees for their insightful remarks on an earlier version of the paper. The research presented here was done while the author was visiting the IBM Research Division at the Thomas J. Watson Research Center. He wishes to thank IBM Research and IBM Finland for making the visit possible, and Helsinki University of Technology, which granted the necessary leave. Financial support was also received from the Helsinki University of Technology Foundation.

References

1. S. Ansaldi, L. Boato, M. Del Canto, F. Fusconi, and F. Giannini, "Integration of AI Techniques and CAD Solid Modelling for Process Planning Applications," *Proceedings, Computer Applications in Production and Engineering (CAPE '89)*, F. Kimura and A. Rolstadas, Eds., North-Holland Publishing Co., Amsterdam, 1989, pp. 351–364.
2. Y. Descotte and J.-C. Latombe, "GARI: A Problem Solver that Plans How to Machine Mechanical Parts," *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI-81*, Vancouver, B.C., 1981, pp. 766–772.
3. R. Fridshal, "Automatic Generation of NC Instructions from Geometric Models," *Proceedings of the Symposium on Computer Integrated Manufacturing*, ASME Winter Annual Meeting, New Orleans, December 1984.
4. K. Iwata and N. Sugimura, "A Knowledge Based Computer Aided Process Planning System for Machining Parts," *Proceedings of the Sixteenth CIRP International Seminar on Manufacturing Systems*, Tokyo, 1984, pp. 83–92.
5. D. A. Nau and M. Gray, "SIPS: An Application of Hierarchical Knowledge Clustering to Process Planning," *Integrated and Intelligent Manufacturing*, PED-Vol. 21, Proceedings of the Winter Annual Meeting of ASME, Anaheim, CA, December 7–12, 1986; ASME, New York, 1986.
6. M. Mäntylä, J. Opas, and J. Puhakka, "A Prototype System for Generative Process Planning of Prismatic Parts," *Modern Production Management Systems*, Proceedings of APMS '87, A. Kusiak, Ed., North-Holland Publishing Co., Amsterdam, 1987, pp. 599–611.
7. M. Mäntylä and J. Opas, "HutCAPP—A Machining Operations Planner," *Proceedings of the International Symposium on Robotics and Manufacturing Systems (ISRAM)*, ASME, New York, 1988, pp. 901–910.
8. E. G. Libardi, J. R. Dixon, and M. K. Simmons, "Computer Environments for the Design of Mechanical Assemblies," *Engineering with Computers*, Vol. 3, Springer-Verlag, New York, 1988, pp. 121–136.
9. R. J. Popplestone, "The Edinburgh Designer System as a Framework for Robotics," *Proceedings of the 1987 IEEE International Conference on Robotics and Automation*, Computer Society Press, New York, 1987, pp. 1972–1977.
10. F. L. Krause, M. Bienert, F. H. Vosgerau, and N. Yaramanoglu, "Feature Oriented System Design for Geometric Modeling," *Proceedings of the Conference on Theory and Practice of Geometric Modeling*, University of Tübingen, FRG, October 3–7, 1988; North-Holland Publishing Co., Amsterdam, 1989, pp. 483–498.
11. J. J. Shah, P. Sreevalsan, M. T. Rogers, R. Billo, and A. Mathew, "Current Status of Features Technology, 2nd edition," *Report R-88-GM-04*, CAM-I, Inc., Arlington, TX, August 1988.
12. A. Wilson and I. D. Faux, *The Modeling of Assemblies for Design and Manufacture (Revised)*, Study on behalf of CAM-I, Inc., Geometric Modelling Project, Dorset Institute of Higher Education, Bournemouth, England, February 1988.
13. *Smalltalk/V 286 Tutorial and Programming Handbook*, Digitalk, Inc., 9841 Airport Boulevard, Los Angeles, CA 90045, May 1988.
14. M. Mäntylä, "The Design Browser—A Hierarchical Part-Of Graph Browser," *Research Report RC-15393*, IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY, January 1990.
15. M. Mäntylä, "The Geometry Browser—A Feature Modeler in Smalltalk," *Research Report RC-15394*, IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY, January 1990.
16. M. Mäntylä, "DeltaBlue—The Implementation of a Constraint Satisfaction Algorithm," *Research Report RC-15395*, IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY, January 1990.
17. L. A. Barford, "Representing Generic Solid Models by Constraints," *Technical Report TR-86-801*, Department of Computer Science, Cornell University, Ithaca, NY, December 1986.
18. B. N. Freeman-Benson, J. Maloney, and A. Borning, "An Incremental Constraint Solver," *Commun. ACM* **33**, No. 1, 54–63 (January 1990).
19. B. N. Freeman-Benson and J. Maloney, "The DeltaBlue Algorithm: An Incremental Constraint Hierarchy Solver," *Technical Report 88-11-09*, Department of Computer Science, University of Washington, Seattle, November 1988.
20. P. Struss, "Multiple Representation of Structure and Function," *Expert Systems in Computer Aided Design*, J. Gero, Ed., Proceedings of IFIP 5.2 Working Conference on Expert Systems in Computer-Aided Design, Sydney, Australia, February 17–20, 1987; North-Holland Publishing Co., Amsterdam, 1987, pp. 57–84.
21. H. Suzuki, H. Ando, and F. Kimura, "Synthesizing Product Shapes with Geometric Design Constraints and Reasoning," *Proceedings of IFIP WG 5.2 2nd Workshop on Intelligent CAD*, Cambridge, England, September 1988; North-Holland Publishing Co., Amsterdam, to appear.
22. M. Inui, Y. Jinno, and F. Kimura, "Design Knowledge Inheritance—A New Approach Toward CAD System for Variant Products," *Preprints of the Third IFIP WG 5.2 3rd Workshop on Intelligent CAD*, H. Yoshikawa and F. Arbab, Eds., Osaka, Japan, September 26–29, 1989; North-Holland Publishing Co., Amsterdam, to appear.
23. M. Mäntylä, *An Introduction to Solid Modeling*, Computer Science Press, Rockville, MD, 1988.
24. A. Borning, "Thinglab—A Constraint-Oriented Simulation Laboratory," *Technical Report SSL-79-3*, XEROX Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, CA 94304, July 1979.
25. K. El Dashan and J. P. Barthes, "Implementing Constraint Propagation in Mechanical CAD Systems," *Intelligent CAD Systems II*, V. Akman, P. J. W. ten Hagen, and P. J. Veerkamp, Eds., Proceedings of the Second Eurographics Workshop on Intelligent CAD Systems, Veldhofen, The Netherlands, April 11–15, 1988; Springer-Verlag, Berlin, 1989, pp. 217–227.

Received December 1, 1989; accepted for publication February 2, 1990

Martti Mäntylä *Helsinki University of Technology, Otakaari 1 A, SF-02150, Espoo 15, Finland.* Dr. Mäntylä received his Ph.D. in 1983 from the Helsinki University of Technology. From 1983 to 1984 he was a Visiting Scholar with the Computer Systems Laboratory at Stanford University, and in 1989 he was a World Trade Visiting Scientist at the IBM Thomas J. Watson Research Center. Dr. Mäntylä is currently a Professor of Information Technology with the Laboratory of Information Processing Science at the Helsinki University of Technology, where he is the head of a product modeling research team. His research interests include computer applications in engineering, CAD, CAM, computer graphics, user interfaces, and data base management. Dr. Mäntylä is an associate editor of the *ACM Transactions on Graphics* and a member of the ACM, the IEEE Computer Society, and the Eurographics Association. He is also a member of Working Groups 5.2, 5.3, and 5.10 of the International Federation for Information Processing (IFIP).