AN OBJECT-ORIENTED PRODUCT STRUCTURE FOR ASSEMBLY MODELLING

Winfried van Holland and Willem F. Bronsvoort
Faculty of Technical Mathematics and Informatics
Delft University of Technology
Delft, The Netherlands
e-mail: (W.vanHolland/Bronsvoort)@cs.tudelft.nl

ABSTRACT

Assemblies are combinations of single parts with relations between them, and therefore assembly modelling and single part modelling are highly related to each other. This paper describes an object-oriented product structure that can be used in single part modelling as well as in assembly modelling.

In single part modelling, the concept of feature-based modelling is now widely accepted, whereas in assembly modelling, the concept is only in its infancy. An object-oriented modelling concept can make use of the similarities between both feature modelling concepts, but can also handle the differences. This results in a unified way to model both single parts and assemblies.

The concept of object-oriented modelling is briefly discussed, and the data structures for single parts and assemblies are described.

1 Introduction

One of the disadvantages of geometric modelling is that only geometric information can be stored by the designer, and most of the other product information known by the designer cannot be stored in the model. Both the geometric information and the other information are very important during process planning. In case of geometric modelling, the other information must be retrieved, using difficult procedures, from the stored geometric information only. For example, in milling, where different shapes are milled with specific milling operations, it is hard to automatically retrieve these shapes from the geometry only.

These problems can be solved by not using only geometric models, but by using *feature models* instead. Features represent both geometry and functional information for a specific application. All relevant information can be stored within a feature model, and can be used by several analysis and planning tools. An additional benefit of using feature models, instead of geometric models, is that features are on a higher abstraction level than geometric entities, and this level is closer to the way of thinking of designers and engineers (Bronsvoort and Jansen 1993).

Every discipline can make use of an own set of features to represent the product model, e.g. the designer uses design features and someone from manufacturing uses manufacturing features. They all have their own *specific view* on the product model. How these different views on one product model can be implemented is described by (Bronsvoort et al. 1996, de Kraker et al. 1996).

Another discipline is assembly, where single parts are combined into products. Also in this discipline the concept of features can profitably be used (van Holland and Bronsvoort 1995*a*, van Holland and Bronsvoort 1995*b*, van Holland and Bronsvoort 1996). Although design and manufacturing features can be used to retrieve some information needed in assembly planning, these features only give information on single parts. We, therefore, extend the feature concept, and store assembly-specific information in so-called *assembly features*. We distinguish two types: *handling* and *connection features*, storing information specific for handling a component, respectively on connections between components.

A way to model features in a computerised environment, is to make use of object-oriented modelling. In object orientation, both data and operations on these data can be represented by so-called classes, and instances of these classes, the objects. The feature types can be represented with classes, and the feature instances can be represented with objects of these classes. In this way, the object-oriented concept can very effectively be used in feature modelling.

Section 2 will first briefly describe the concepts of object-oriented modelling. Section 3 shows how these concepts can be used in feature modelling for single parts. Section 4 discusses the object-oriented data structures used in assembly modelling. Combinations of the data structures for modelling single parts and modelling assemblies are presented in section 5. The paper will end with section 6 giving some conclusions.

2 Object-oriented modelling

This section will give a brief description on some issues in object-oriented modelling. For a detailed description on object-orientation see (Stroustrup 1993) and (Gorlen et al. 1991).

In object-oriented modelling, abstract data types are used. An *abstract data type* is a user-defined data type that encompasses data elements along with the operations that can be performed on them. Most programming environments do not support these abstract data types, but separate the data elements and the operations that can be performed on them. An advantage of combining data elements and operations is that it is easier to change available data structures and to add to new structures. An additional benefit is that object-oriented models are closer to the way of thinking of a programmer.

Much of the value of object-oriented modelling results from *inheritance*. Start with an already developed set of object types, or *classes*, and extend them for new applications by adding data elements and operations to form new classes. Do not write new classes from scratch, but inherit data and operations from useful *base* classes. Add new functionality by describing how the new or *derived* class differs from the base classes. Figure 1 shows such a *class hierarchy* for 2D objects in a 2D drawing environment. The base class, the 2D shape class, contains, for example, functions to draw or move a 2D shape on a screen. The derived classes inherit these functions, so they can also be drawn or moved on a screen, and they add some other specific operations or data types. For example, the difference between the rectangle class and the derived square class is the modification operation, where the latter class restricts the modification by defining that width and height must always remain the same.

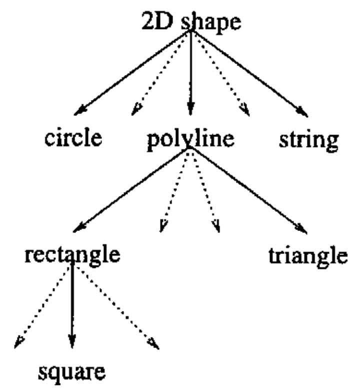


Figure 1: Class hierarchy for 2D objects

It is allowed for a class to derive data elements and operations from more than one base class; this mechanism is called *multiple inheritance*. Figure 2 shows an example of multiple inheritance. When a combined data structure for a string and a rectangle around it is needed, and data structures for a string and a rectangle have been defined, then the new class, the bordered string class, can be created by inheriting the data elements and operations from both rectangle class and string class.

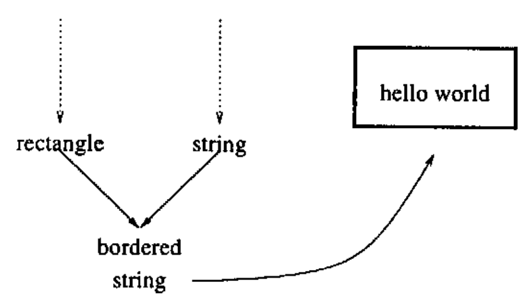


Figure 2: Multiple inheritance

How object-oriented models can be used in developing data structures used in feature modelling, is shown in the following sections.

3 Feature modelling for single parts

Feature modelling is now commonly used in modelling single parts. A *feature* is defined here as physical part of an object mappable to a generic shape and having functional significance. Features with significance for the designer are called *design features* or *form features.* Strictly speaking, there is a difference between form features and design features; a form feature contains only additional information on the shape of the feature, whereas a design feature can also contain design specific information.

A single part is represented by several *instances* of form features. Each type of feature instance is represented by a generic feature class. So, if a single part has multiple holes in it, then each hole is represented by an instance of the feature hole class. Every feature inherits from the base feature class. This class contains a data structure in which the geometry and topology of the feature and methods on these data structures can be described. Each derived feature class, e.g. the through hole feature class, describes the shape type of the feature, its geometry and topology. Each instance of these classes, describes the exact shape, with specified attributes. Detailed descriptions of form feature classes can be found in (Ovtcharova et al. 1992), and an example is given in figure 3.

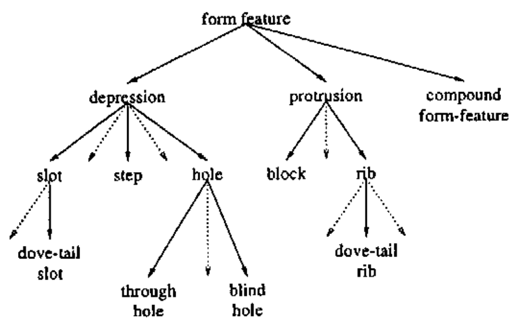


Figure 3: Class hierarchy for form features

A special class is the compound form-feature class, where new form features can be created by taking combinations of other form features.

To define the exact position and orientation of the instances, relations, or constraints, are placed between these features. Therefore a base relation class is defined, and derived classes are defined to describe special constraints, e.g. the mating plane-plane class and offset plane-plane class. Details on constraints can be found in (Dohmen et al. 1996), and a brief example is given in figure 4. New constraints can be defined in the compound constraint class by taking combinations of other constraints.

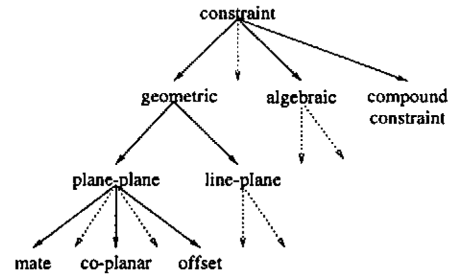


Figure 4: Class hierarchy for constraints

To define a complete single part, both instances of features and instances of constraints must be specified. A constraint solver is used to satisfy these constraints, and to calculate the resulting position and orientation of the feature instances. To combine features and constraints, a feature model class is defined. This class defines methods for adding instances of features and instances of constraints, and defines operations for calculating the actual geometry of the defined single part. In figure 5, a single part is shown with instances of features and instances of constraints.

4 Feature modelling for assembled products

An assembled product consists of combinations of, possibly similar, single *parts.* These parts are not always directly assembled into the complete product, but mostly, and for several different reasons, *sub-assemblies* are created as stable entities. These sub-assemblies can further be used to assemble other sub-assemblies or the complete product. Both a single part and a sub-assembly are stable entities (with respect to transport), and can therefore be assembled onto other entities; these stable entities are called *components.* The already assembled components are called a *partial assembly.* A partial assembly can thus be a single part (when assembly has just been started), an instable group of components (during assembly), a sub-assembly or the complete product.

Between components there exist all kinds of relations, representing a certain function between the

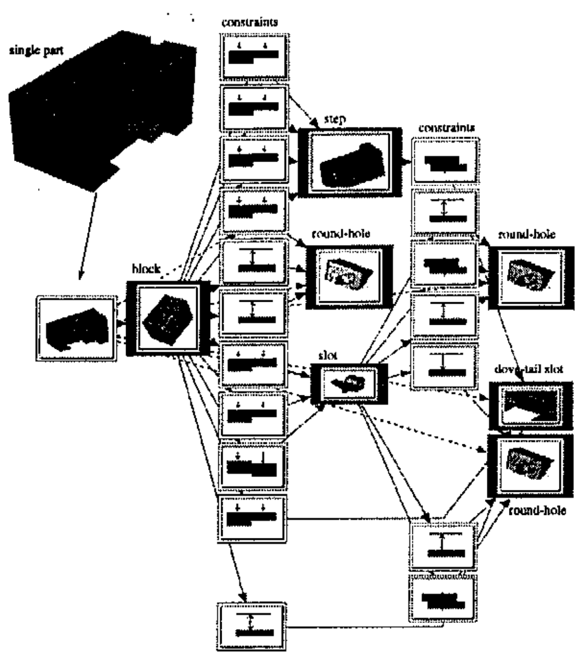


Figure 5: Form feature model of a single part

components, and prescribing the position and orientation between the components. Because these relations are highly dependent on the shapes of the involved components, they are called *connection features.* A connection feature is an example of an assembly feature. An *assembly feature* is defined as a feature with significance for assembly processes. A connection feature contains assembly-specific information on a connection between components. Another assembly feature is the *handling feature,* containing information on how to handle a component. See for details on assembly features (van Holland and Bronsvoort 1995*b*).

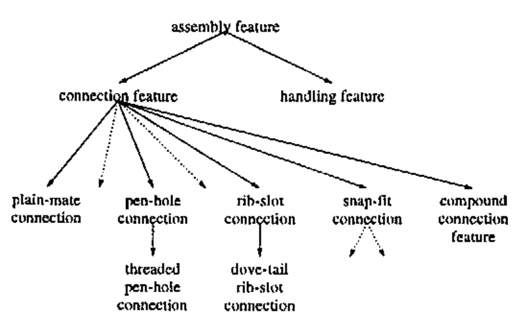


Figure 6: Class hierarchy for assembly features

A class hierarchy for the assembly features is given in figure 6. In this class hierarchy, also a compound connection feature class is defined, to create new connection features by combining other connection features.

In a partial assembly, the same type of component can be available on several places in the partial assembly, e.g. several bolts to fasten a plate. For each different type of component, we introduce a *generic* component, describing the geometry and topology by its design features. The generic component does not describe a position and orientation in the product; these are described by an *instance* of a generic component. In this way a product can have several instances of a generic component, but for every type of component it will have only one generic description. Each instance can have different connections in the product, represented with connection features. An object of an instance component has an attribute element in its data structure pointing to the represented generic component. Figure 7 shows the class hierarchy used to describe the different components.

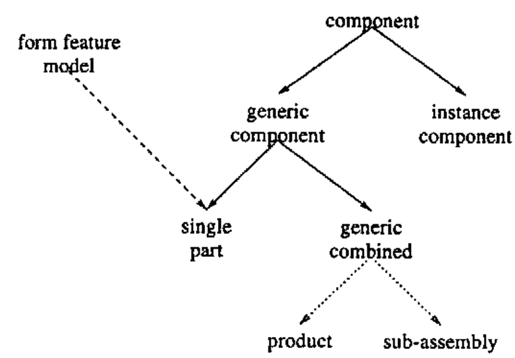


Figure 7: Class hierarchy for different components

There are two derived classes from the generic component class: the single part class and the generic combined class. The difference is that the single part class is also derived from the feature model class (the dashed line in figure 7), i.e. it represents the feature model of a single part; these components cannot be subdivided into smaller components. The generic combined class represents the components that consists of combinations of instances of components and connection features between them. Both product and sub-assembly are components that can be subdivided. It is hard to give a difference between these types. A product described by one person, e.g. a complete engine, can be seen as a sub-assembly by another, e.g. the engine for a complete car. That is the reason why no different classes are introduced for product and sub-assembly (the dotted lines in figure 7); both are represented by the generic combined class.

During modelling of a product or sub-assembly, components are assembled onto a partial assembly. This partial assembly is also a combination of instances of components and connection features. A partial assembly differs from a product or sub-assembly, in the sense that it is not known whether the combination of components is a stable entity. We did not introduce a new partial assembly class, because the partial assembly can also be represented by the generic combined class. An attribute in the data structure of the generic combined class contains whether it represents a stable entity, i.e. sub-assembly or product, or it is not known whether the entity is stable, i.e. a partial assembly.

In figure 8 an example is given of a generic combined component, consisting of three instances of two different generic components, and two connection features.

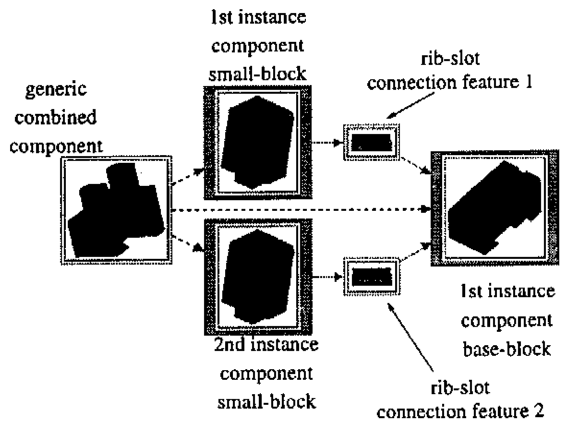


Figure 8: Assembly model of a sub-assembly

5 Combining single part modelling and assembly modelling data structures

Both in single part modelling and in assembly modelling, data structures are used to represent shapes, relations between the shapes, and structures for combining these shapes and relations. In single part modelling, these are represented by, respectively, form features, constraints and the feature model. In assembly modelling, these are represented by, respectively, instances of components, connection features and generic combined components. Because of the similarities, new classes are defined to create a uniform environment for modelling single parts and assemblies. The new classes are: combined class, related class and the relation class; they are shown in figure 9.

The relation class is introduced as a base class for all objects that represent a relation between entities, i.e. the constraints and connection features. The related class is introduced as a base class for all objects involved in relations, i.e. the form features and the instances of components. The combined class is introduced as a base class for classes in which sets of related objects and relations between them are specified, i.e. feature models and generic combined components. Using these three base classes, a uniform way in modelling single parts and assemblies is created. This uniform modelling concept can be used to combine both modelling environments into one environment, where both single parts and assemblies can be modelled.

6 Conclusions

In this paper a uniform object-oriented data structure is presented for modelling single parts and assemblies.

The object-oriented models are easier to extend by using the inheritance mechanism. New classes derive already made data elements and operations on these elements from available base classes. Only the differences between base class and derived class must be specified.

The form feature class hierarchy and constraints class hierarchy are presented to be used in generating feature models representing single parts. Both the feature classes and constraint classes can be extended by using the compound class definitions.

The component class hierarchy and assembly feature class hierarchy are presented to be used in generating assembly models representing products or sub-assemblies. Assembly features are divided into handling and connection features, for representing information on handling a component respectively about connections between components. Here a compound connection feature is defined for extending the connection feature classes.

The combined, related and relation classes combine the similarities between the modelling environments for single parts and assemblies, to create one modelling environment for both single parts and assemblies.

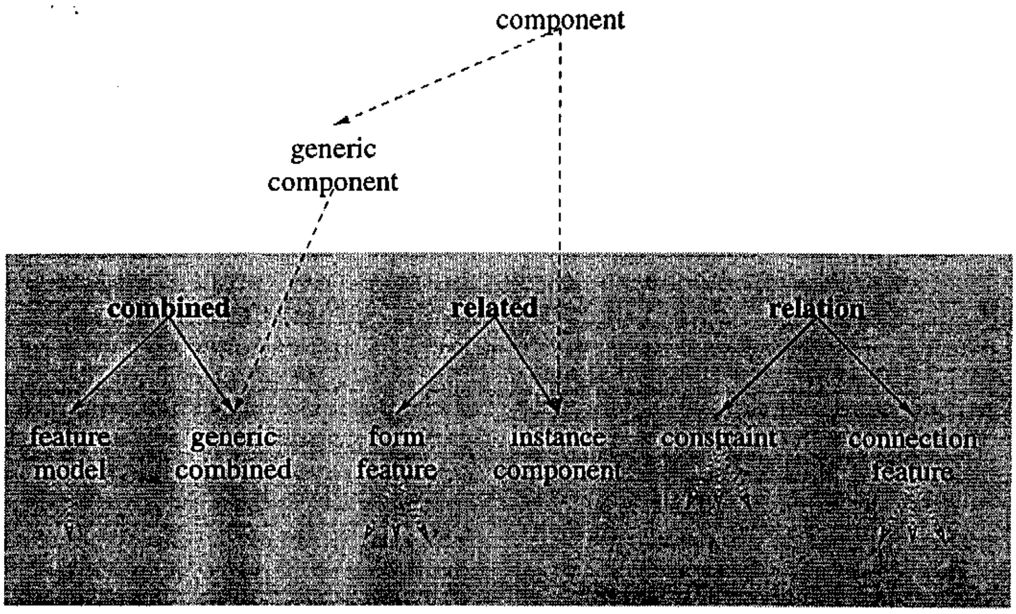


Figure 9: Combined data structure for modelling single parts and assemblies

The presented object-oriented models have been implemented, and are already used in several assembly process planning modules. These modules have shown the profitability of using the feature concept in assembly.

REFERENCES

Bronsvoort, W. F. , Bidarra, R. , Dohmen, M. , van Holland, W. and de Kraker, K. J. (1996), Feature modelling for concurrent engineering, *in* 'Proceedings of the Symposium on Tools and Methods for Concurrent Engineering', Technical University of Budapest, pp. 46–55.

Bronsvoort, W. F. and Jansen, F. W. (1993), 'Feature modelling and conversion - Key concepts to concurrent engineering', *Computers in Industry* **21**(1), 61–86.

Dohmen, M. , de Kraker, K. J. and Bronsvoort, W. F. (1996), Feature validation in a multiple-view modeling system, *in* 'Proceedings of the 16th ASME International Computers in Engineering Conference', Irvine, California. To be published.

Gorlen, K. E. , Orlow, S. M. and Plexico, P. S. (1991), *Data Abstraction and Object-Oriented Programming in C++*, John Wiley & Sons Ltd, Chichester, England. ISBN 0-471-92346-X.

van Holland, W. and Bronsvoort, W. F. (1995*a*), Assembly features and visibility maps, *in* A. A. Busnaina, ed., 'Proceedings of the 15th ASME International Computers in Engineering Conference', Boston, pp. 691–697.

van Holland, W. and Bronsvoort, W. F. (1995*b*), Assembly features in modelling and planning, *in* M. Tichem, T. Storm, M. M. Andreasen and K. J. MacCallum, eds, 'Proceedings of the WDK Workshop on Product Structuring', Delft University of Technology, Delft, pp. 195–206.

van Holland, W. and Bronsvoort, W. F. (1996), Extracting grip areas from feature information, *in* 'Proceedings of the 16th ASME International Computers in Engineering Conference', Irvine, California. To be published.

de Kraker, K. J. , Dohmen, M. and Bronsvoort, W. F. (1996), Multiple-way feature conversion - opening a view, *in* 'IFIP WG5.2 Workshop on Geometric Modeling', Airlie, Virginia.

Ovtcharova, J. , Pahl, G. and Rix, J. (1992), 'A proposal for feature classifiction in feature-based design', *Computer and Graphics* **16**(2), 187–195.

Stroustrup, B. (1993), *The C++ Programming Language*, 2nd edn, Addison-Wesley Publishing Company, Reading, MA. ISBN 0-201-53992-6.

An Object-Oriented Product Structure for Assembly Modelling

Winfried van Holland
Willem F. Bronsvoort

TUDelft
Delft University of Technology
Faculty of Technical Mathematics and Informatics

2^{nd} WDK Workshop
on
Product Structuring

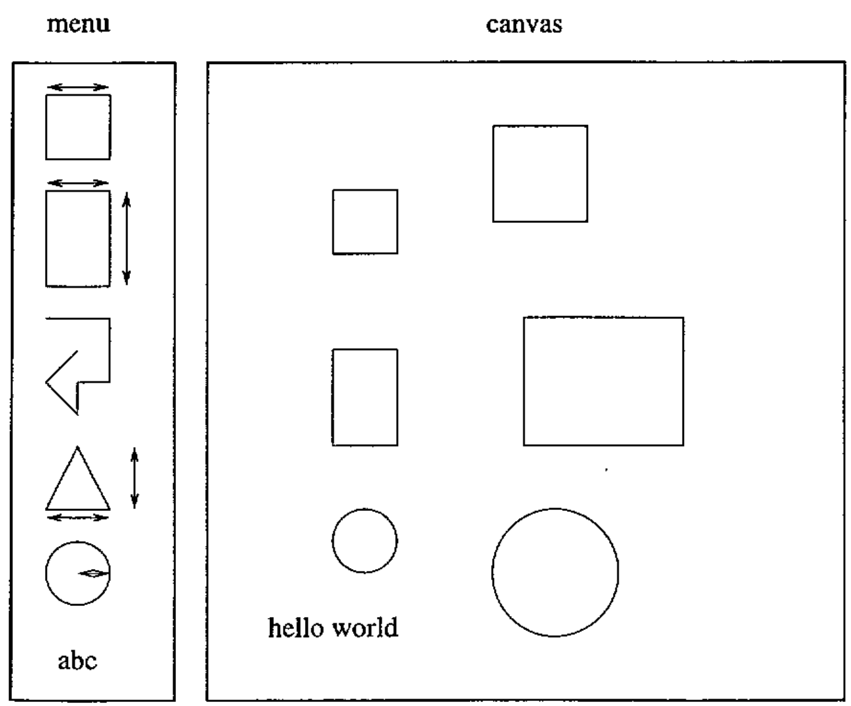
Delft, June 3-4, 1996

overview

- object-oriented modelling
- feature modelling for single parts
- feature modelling for assemblies
- combination of single part and assembly modelling
- conclusions

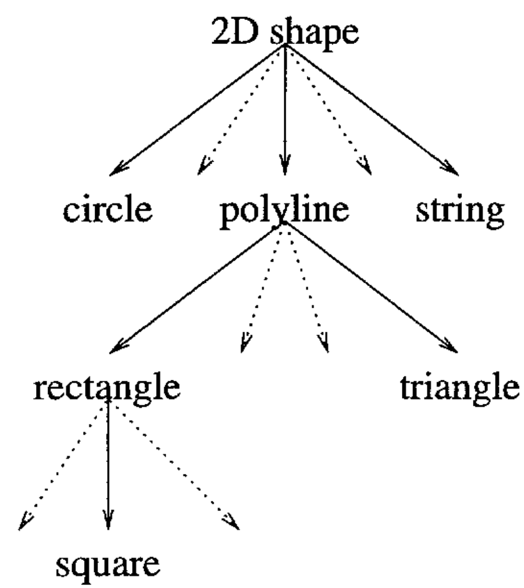
object-oriented modelling

combine both data elements and operations on these elements in one structure



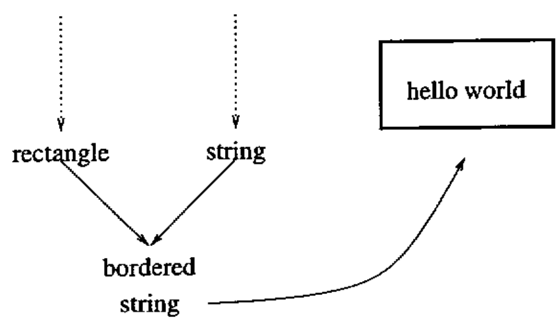
2D drawing tool

do not write new classes from scratch, but *inherit* data and operations from useful *base classes*, add only *differences* in functionality between base class and *derived* class



class hierarchy

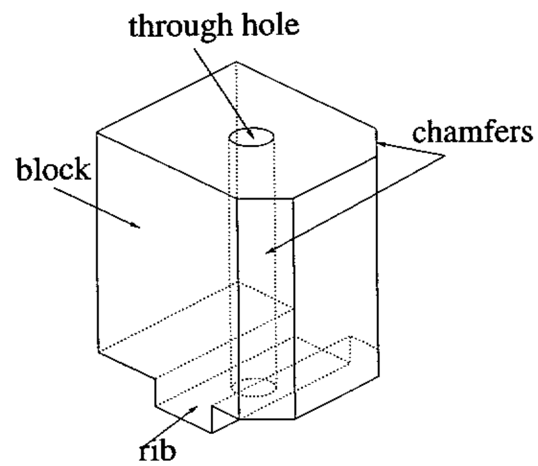
multiple inheritance



derive data elements and operations from more then one base class

form features

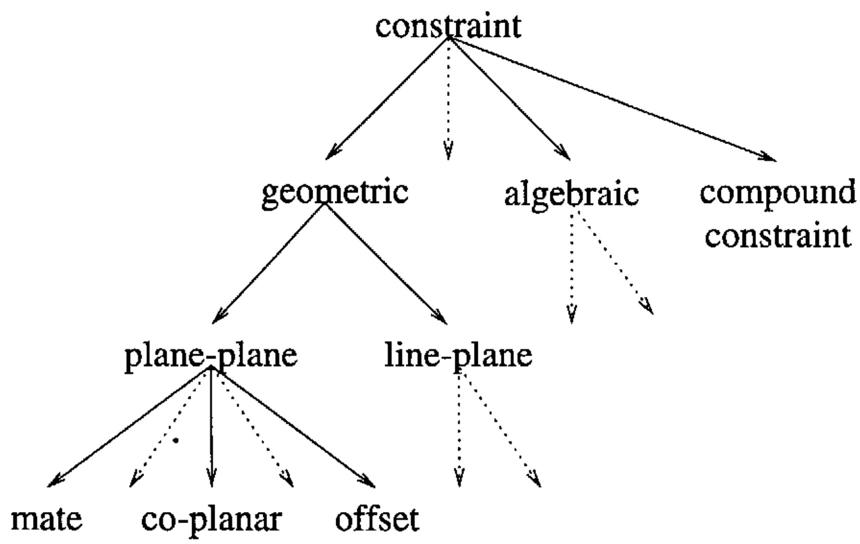
a feature is a *physical part* of an object, mappable to a *generic shape* and having *functional significance*



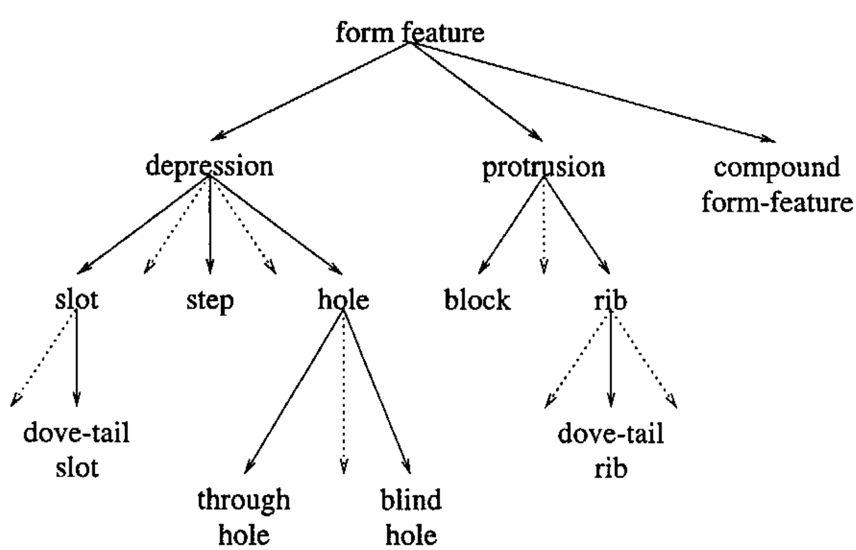
single part

a *feature model* combines *features* and *constraints* between them in one data structure representing the product model

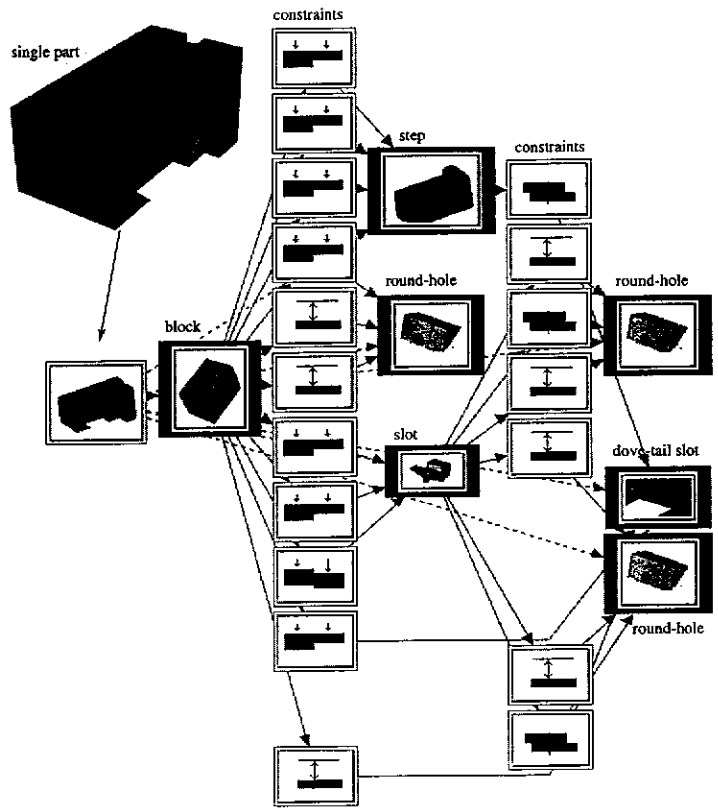
constraint class hierarchy



feature class hierarchy

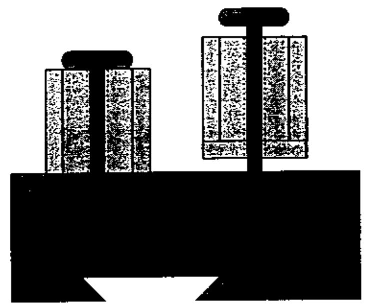


single part example



feature model for a single part

assembly modelling

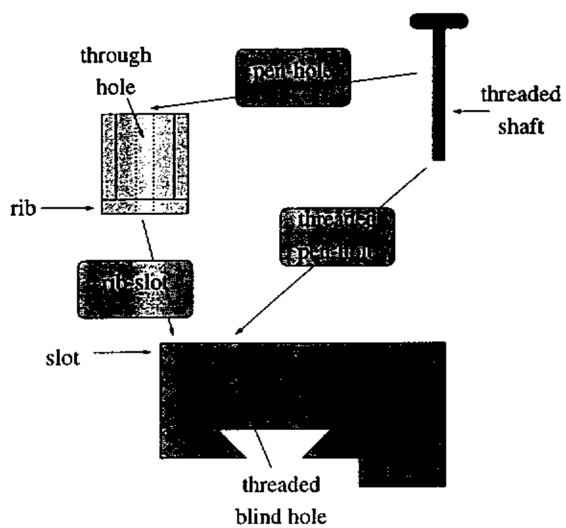


- components
  - part
  - sub-assembly
  - partial assembly
  - product
- relations between components

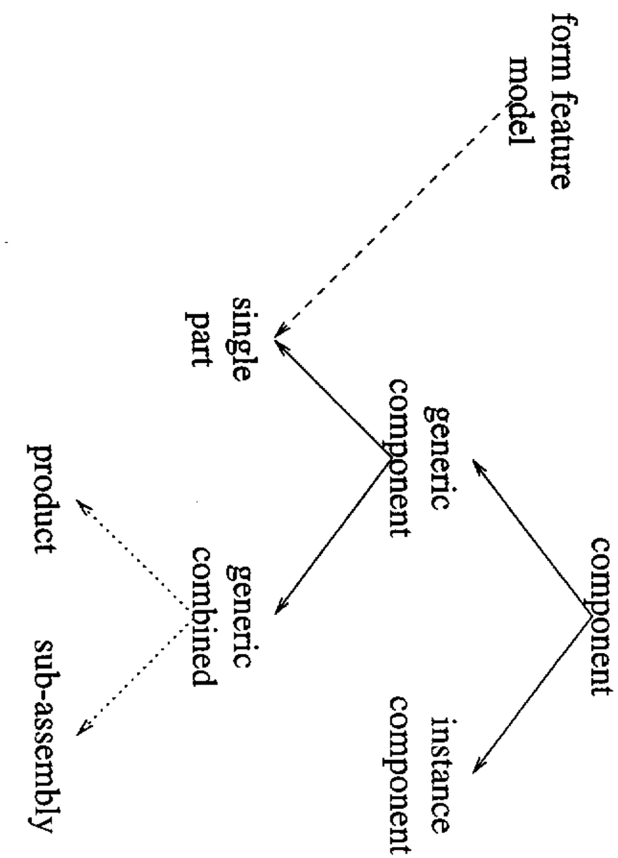
connection features

contains specific assembly information on connections between components

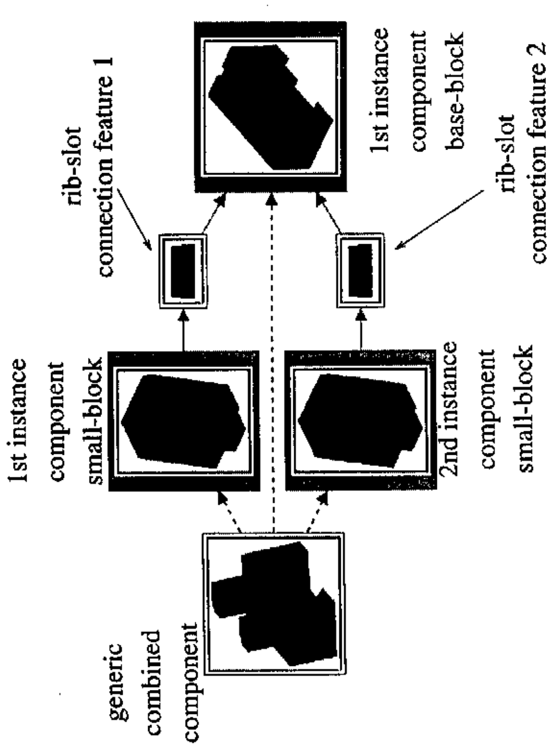
- insertion and final position, insertion path
- internal freedom of motion
- contact areas
- tolerances



component class hierarchy

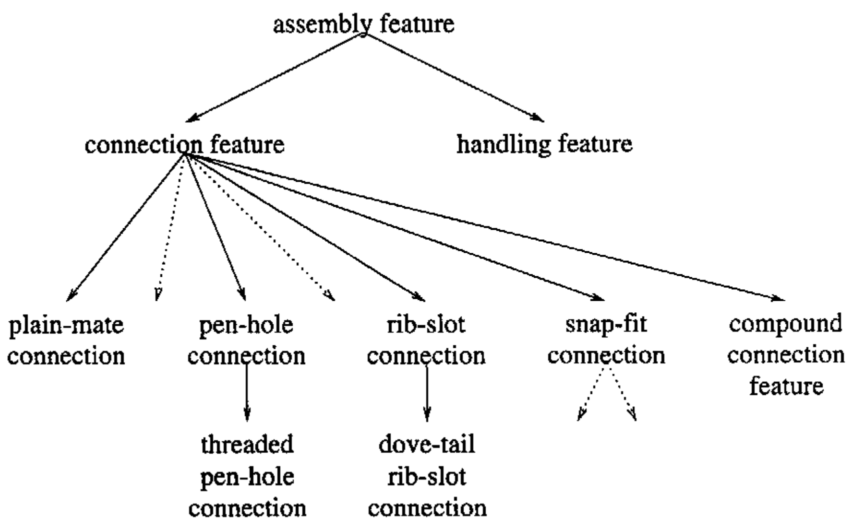


assembly example



assembly model for a sub-assembly

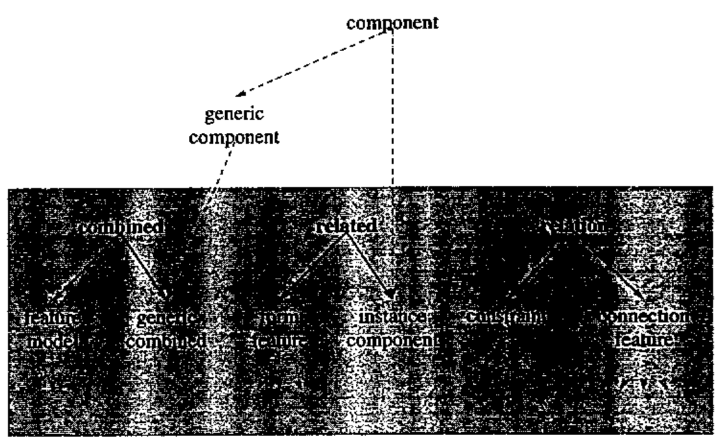
assembly feature class hierarchy



combine data structures

combine data structures for single part and assembly modelling

- uniform data structure
- integrate single part and subassembly modelling



conclusions

- easier to extend object-oriented model using inheritance
- both feature models for single parts and assemblies are closer to way of thinking of the designer/engineer
- combine both structures to a uniform object-oriented data structure for modelling single parts and assemblies
- already used in several assembly process planning modules