Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

J.L.H. Rogier, D.B.M. Otten

Retrospective creation of virtual alternative hierarchies

Computer Science/Department of Interactive Systems Report CS-R9051 September

---

The Centre for Mathematics and Computer Science is a research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a nonprofit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands Organization for the Advancement of Research (N.W.O.).

Retrospective Creation of Virtual Alternative Hierarchies

Jan Rogier[1,2] and Daan B.M. Otten[1]

[1]Centre for Mathematics and Computer Science (CWI)
Department of Interactive Systems, Kruislaan 413
1098 SJ Amsterdam, The Netherlands
Email: rogier@cwi.nl, daan@cwi.nl

[2]Netherlands Organization for Applied Scientific Research (TNO)
Institute of Applied Computer Science (ITI)
Schoemakerstraat 97, 2628 VK Delft
The Netherlands

In this chapter we present an object-oriented strategy to support the creation of *geometric models*. These models are represented as *part-whole hierarchies* of objects. Creation of new models is done by copying and modifying previously existing models or parts of them. Past models are stored in the system's database and serve as *prototypes* for the new models. To support the selection of an appropriate prototype the system provides a view on its database presenting an *is-a hierarchy* of *object type definitions* that depends on:

- the currently available models in the system's database,
- a *functional specification* that represents the designer's intentions, and
- a set of *definitions* that are used by the system to interpret a functional specification.

From the hierarchy the user can either select an object type definition that might serve as a *template* for a new part or select one of the past models from which an object type definition originates. Since the 'is-a' hierarchy of object type definitions only exists virtually, is moreover directly controlled by the user and is highly flexible, as well as providing alternative hierarchies presenting alternative views on the system's database, we call our approach 'retrospective creation of virtual alternative hierarchies'.

CR Categories and Subject Descriptors:
D.1 [Programming Techniques]
J.6 [Computer Aided Engineering] - Computer Design (CAD)

Key Words & Phrases:
Computer Aided Design, Object Oriented, Generalization / Specialization, Aggregation / Decomposition, Prototypes

1. Introduction

We are currently developing a design system as part of a research project on intelligent CAD [Veth87a]. The feature that will be discussed in this chapter concerns the structure and use of the database of our design system for the reuse of existing models. The system's database has a two-fold purpose. Firstly, it is used to make models persistent, i.e., to store models and their relations which last beyond a program session. Secondly, it provides a mechanism to browse through the models in the database. Selected models can be used to create new models. The basis of our design system is a strategy called *extensional decomposition* [Rogier91a] and the use of *standard vocabularies* and *standard components*. Essentially, extensional decomposition is a design

Report CS-R9051
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

strategy that views every object as a set of components which are also objects in themselves. This is elaborated in Section 3.1. As regards the first purpose of our database, the system's database stores and maintains models as part-whole structures that result from the use of extensional decomposition. For the second purpose of the database, selected models from the database are organized in an 'is-a' hierarchy of object type definitions. From this hierarchy the user can select a single definition, modify it and use it as template [Lieberman86a] for creating or detailing the new model. Comparison of models is based on their extensional decomposition structure and the use of standard vocabularies and standard components.

The creation of an 'is-a' hierarchy of object type definitions is based on i) the models in the database, ii) a description of what the user wants concerning the object to be designed (called a *functional specification*), and iii) a set of definitions that are stored in the system and are used for the interpretation of a functional specification. The user has three means to influence the organization of object type definitions in a hierarchy:

a. Modification of the functional specification. Based on the functional specification the system selects a set of models that minimally meet the specification. All extra data with each model can be considered to be the specialization of the functional specification and is used to form the hierarchy of object type definitions.

b. Manipulation of the set of definitions used by the system to interpret both functional specification and past models. We make a distinction between two types of definitions: i) *categories*, being sets of standard components, and ii) *scenarios*, being sets of requirements for object types. The total set of definitions that is applied by the system is manually controlled by the user by declaring which definitions are 'active'. The current set of definitions will influence the interpretation of the functional specification and therefore influence the order of the hierarchy.

c. Controlling the effect of the constraints over properties. Scenarios also include constraints on physical properties of objects. These properties are organized in so-called *aspects*. Controlling the effect of constraints by making it possible to ignore, that is, deactivate, aspects will influence the contents of the selected set of past components. In turn this will influence the hierarchical order in the object type definitions.

The chapter is organized as follows. Section 2 describes the background to our system and gives a short description of the design strategies of the group of designers we are aiming at. In Section 3, the basis of our system, viz. extensional decomposition, standard vocabularies and standard components, is elaborated. Section 4 describes the architecture and the basis elements of our system. In Section 5, the way the system works is explained. It describes the selection of prototypes and the creation of the virtual hierarchy and template. In Section 6, a comparison with class-based object-oriented systems is made. Section 7 contains a summary and conclusions.

2. Background

The current generation of design systems take generic characteristics of specific classes of design objects as a basis for the structure of the system. They use a fixed reference model for a certain type of design object. This reference model is created by the designer of the system instead of by the user. Apart from the fact that it introduces a problem of miscommunication between system designer and user, this approach ignores the fact that i) generalization of characteristics is based on concrete examples and ii) reference models evolve in time. The limitations of such a system can only be removed by changing it fundamentally. The main reason is that the reference model forms an integral part of the system's structure. Instead of being of use as a supporting tool, systems that use a fixed reference model as an integral part of their structure dictates their view on a

class of objects to a user. As a result a user can only work within the boundaries set by the system.

According to Bijl's view on a system's role in design [Bijl87a], a system should not specify the boundaries within which a user has to design. The user should be free to use the system to design anything he wants with it. For this reason we took an approach where the user can create his own reference models based on what was previously designed with the system.

We identify three approaches taken by the user in which past models play an important role in the design process. These approaches are based on the identification via context, association and generalization.

a. With the first approach a user identifies a prototype based on its *context*. Identification takes place by referring directly to a past model. This approach is adequate as long as the number of models in the system is relatively small. It is often taken by users who do not have much experience or practice in a broad application field. Identification from context is supported by our database structure. No special tools are required to apply this approach. Applying it in fact results in copying and modifying an existing design object description. Due to the limitations of currently available design systems, this approach is often practiced manually.

b. The second identification approach is based on *association*. In this case the user wants to associate the current design problem with a previous one, but does not know exactly which previous model. First, he builds a functional specification of the current design problem. Then this specification is applied to all past models and produces a set of comparable models. From this set the user selects the most appropriate model which will serve as a template for the specification of the new model. According to Yoshikawa [Yoshikawa81a] this approach forms the basis of the design process.

c. The third approach is comparable to the previous one. However, in this case the user does not select a single model from a set but uses the whole set to construct a template. The template describes all generic properties of a certain type of object based on what the designer has stored in the system's database.

Our system aims at users that apply the following design strategy.

1. The user creates an initial functional specification.
2. The user specifies the criteria which should be applied by the system to interpret the functional specification in order to compose a set of comparable past models.
3. The user gets the database to search for past models that match the functional specification and to organize them based on their extra data. As a result a hierarchy is created that has the initial functional specification as the root and contains all selected models as specializations.
4. The user looks for a specialization in the hierarchy that will help him further detail the functional specification until he gets one that fully meets his intentions.
5. The user then repeats the steps from step 2.

3. The Basis of the Design System

The basis of our design system is extensional decomposition and the application of standard vocabularies and standard components.

3.1. Extensional Decomposition

Extensional decomposition is a design strategy that views every object as a set of components which are objects themselves. With the term *extensional* we want to emphasize that every component has two different identities. It has an identity as a unique entity and it has an identity as a set of components. These two identities have to be viewed independently. The essence of this twofold identity is explained with an example.

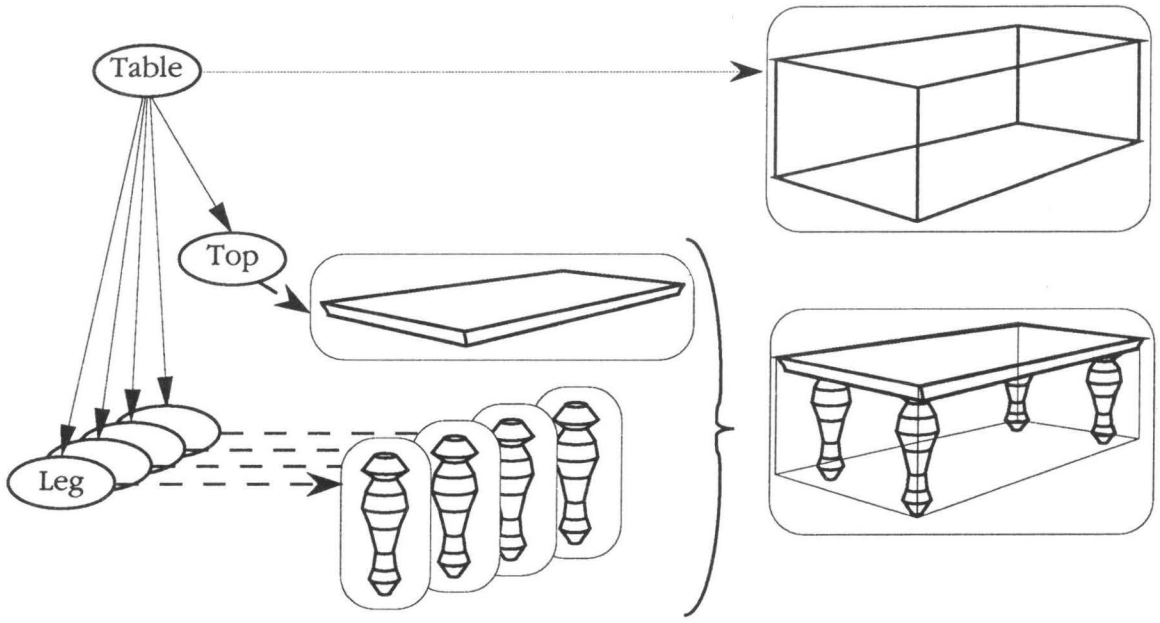

Figure 1 Twofold identity of a component

Figure 1 gives a rough impression of what we mean with a twofold identity. The table has a geometry that on its own level describes its rough shape as a block form, with a height, a width and a length. This shape describes the space that is occupied by the table. Apart from this, it has a shape that is a composition of the shapes of its components. Although the shape of both identities are probably related, they in fact exist independently. The height of a table may be defined as a value that should be equal to the sum of the length of its leg(s) and the thickness of its top. From a declarative point of view however, the height of a table may be the actual intended height. The other two values may be derived from descriptions of standard parts. Extensional decomposition puts the emphasis on the *incompatibility* of these values. This incompatibility between an attribute of a model and the attributes of its components are observed by the system. However in contrast to expert systems, the system does not change the functional specification but only notifies the user.

3.2. Standard Vocabularies

Standard vocabularies are sets of terms used to define aspects of an object (compare Wilson and Kennicot [Wilson87a]). They are used i) to support the process of declaring different aspects of an object, ii) as an intermediary to exchange data between different external applications, and iii) as a basis for comparison of models based on their properties.

3.3. Standard Components

Standard components are the primitive objects which define the lowest level of detail considered by a designer. Standard components are categorized. For instance, for an architect building a house, categories might be doors, windows and walls. For a construction engineer a wall itself is built up from more detailed components and is therefore not a standard component.

4. The Basic Elements of our System

4.1. The System's Environment

The architecture of our design system is one that contains a kernel modeller that communicates with a number of external, independently operating, special purpose applications (see Figure 2).

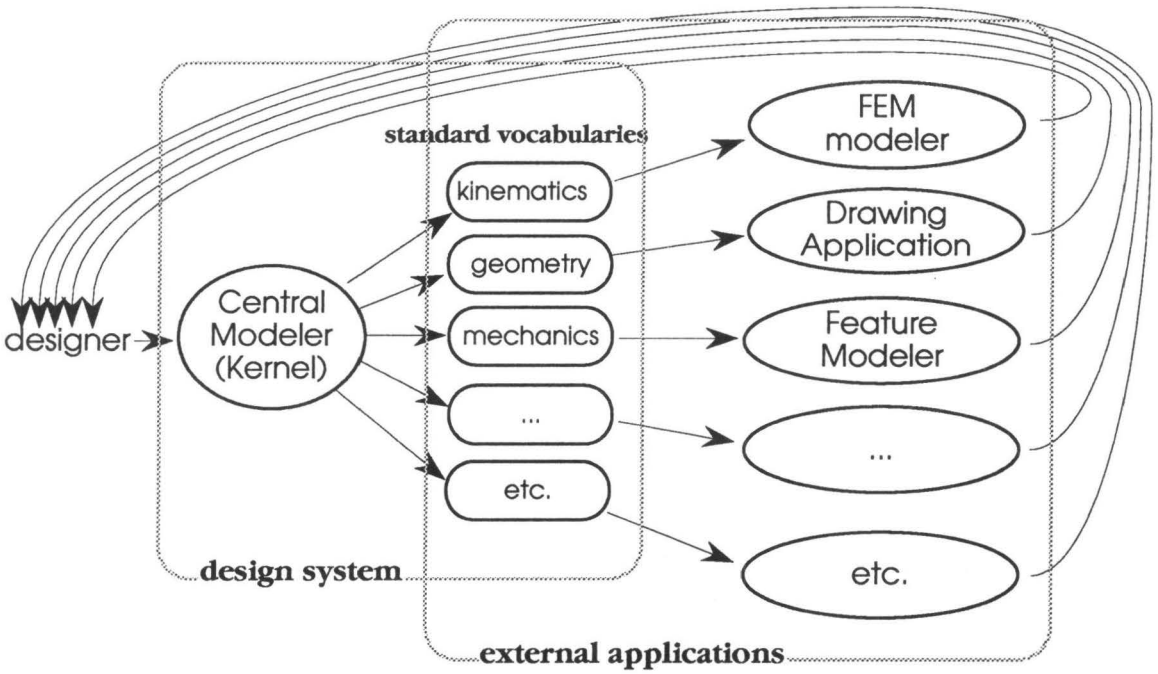

Figure 2 The kernel modeller in its environment

Data exchange between the kernel and applications is simplified by using standardized languages for different aspects. Such languages are vocabularies of terms. In our approach we make these vocabularies independent entities and establish explicit links between the terms in the model and the terms in these vocabularies. This approach has the advantage that both vocabularies and models can easily be extended.

4.2. Basic Primitives

Applying extensional decomposition leads to a part-whole structure of component descriptions. For the description of each single component we use the same basic primitive. Each object is described with a *set of attributes* (see Figure 3). Values of attributes can be of three different types. Firstly, they can be *constants*, e.g., when one wants to describe the height of an object.

Secondly, they can be *references to other components*. These attributes are used to describe the 'has-part' relations between objects. Thirdly, they can be *references to mathematical expressions*.

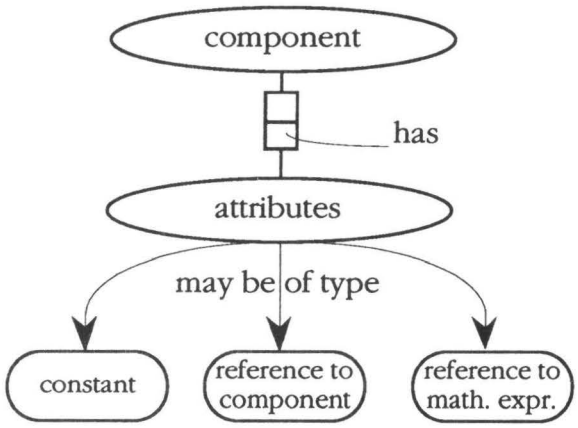

Figure 3 Basic data structure

References to mathematical expressions are used to describe the dependency relations between constants. They contain a reference to a *term* and a *mapping list*. A term is part of a *standard vocabulary*. The mapping list describes the mapping between each of the local attributes of a term and local attributes of the component. A standard vocabulary itself contains all terms used to describe an *aspect* (see Figure 4).

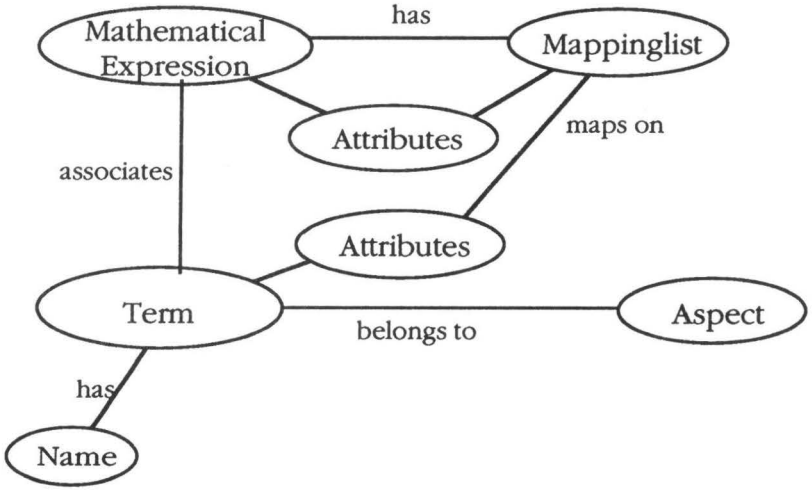

Figure 4 Basic data structure of dependency reference

4.3. Aspects, Categories and Scenarios

Aspects, categories and scenarios are the implementational counterparts of vocabularies of terms, sets of standard components and functional specifications, respectively. Categories may be specified as sets of other categories. Scenarios use the names of aspects and categories or other scenarios to define the requirements of object types. These requirements constrain the number of parts or the physical properties of an object.

5. The System's Operation

The system uses a functional specification to select and organize a part of the database. A functional specification is described by a scenario. It has a twofold purpose. On one hand, it is used to describe the user's intentions with the new object being designed. On the other hand it is used to select and organize items in the system's database. These two roles may be in conflict. The simplest example is when a functional specification is so accurate a description of the new object being designed that no items in the database meet all requirements. However, the second purpose of this functional specification is to collect as much data as possible from the system's database to simplify the declaration process. Although components may not exist that fully meet the constraints specified by the functional specification, there may be enough items that will partly meet them and contain valuable information. To facilitate identification of these components, modifying the functional specification itself is not allowed since this is in conflict with its first purpose.

For its second purpose the functional specification should be interpreted more globally. This requires mechanisms that control the interpretation of functional specifications. One mechanism will interpret the functional specification to *select* a set of appropriate components in the database. The other mechanism serves to *organize* object type definitions into a hierarchy. Because these hierarchies present merely a view on the database, we call them virtual hierarchies.

5.1. Virtual Hierarchies

We define a virtual hierarchy as one that is generated according to a functional specification applied to models in a database. A virtual hierarchy serves to create a template for the specification of a new component that will become part of the current model. Such a template describes a type of object, and includes all generic properties of a small set of comparable past models.

An object type definition is written in a scenario. Such a scenario contains rules that constrain the number of parts of specific types and the boundaries of other properties. To identify an object we say that an object belongs to a type if it at least meets the constraints in the scenario. Although an object may not exceed the boundaries set by the scenario it may however contain properties which are not mentioned in such a scenario. So if a table is an object that has a single top with at least one and at most four legs, a desk is 'a-kind-of' table because it meets these constraints even though it has drawers.

5.2. The Creation of a Virtual Hierarchy

The process of creating a virtual hierarchy of object type definitions (see Figure 5) is subdivided into two steps.

1. A set of components is selected that meets the functional specification as interpreted according to the selection criteria.
2. This set is transformed into a hierarchy of object type definitions based on the data about

extra parts. Objects with different numbers of the same type of parts result in subtypes at the same specialization level, extra parts result in object types at a lower specialization level. In our example the desk is a subtype at a lower specialization level of the table because it contains extra parts not mentioned in the object type definition of a table.

We shall now discuss the steps in more detail.

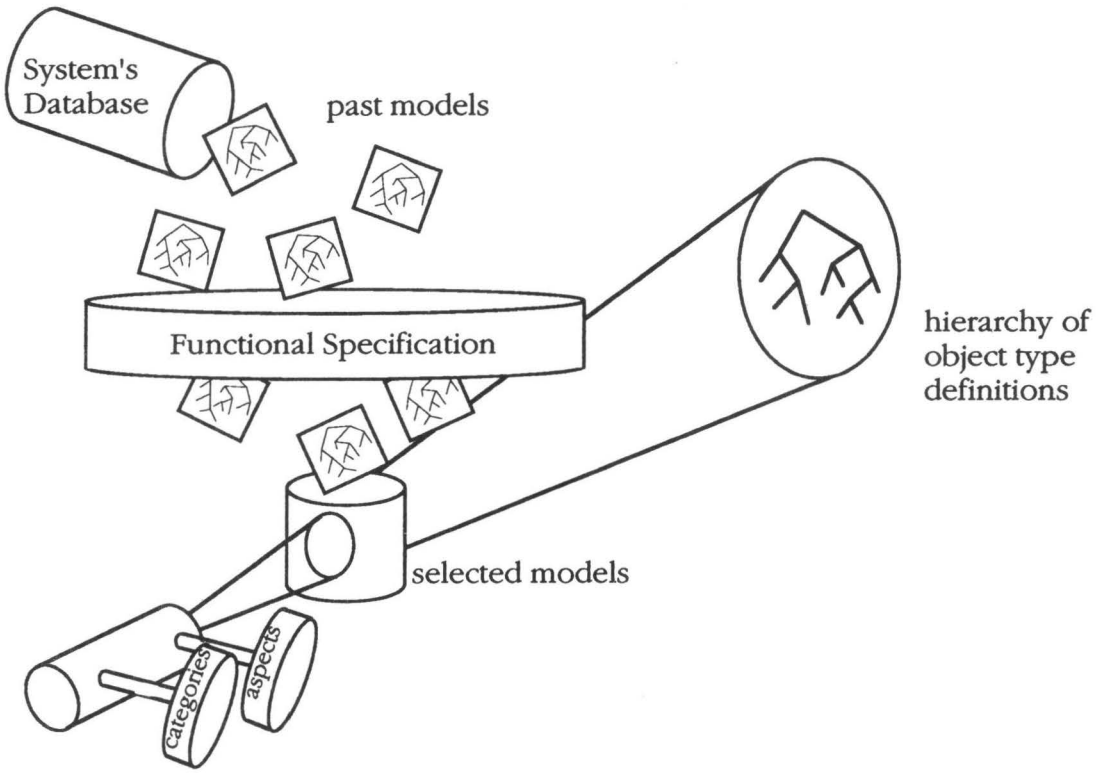

Figure 5 Creation of a virtual hierarchy of object type definitions

5.2.1. Step 1

Selection of a set of appropriate components is done with a functional specification which is interpreted by the system using certain criteria. All components in the database that minimally meet the constraints in this functional specification are selected. A functional specification is a scenario. It contains two types of constraints: constraints on physical properties and constraints on part properties.

a. Constraints on physical properties set boundaries for values of attributes of components. They are specified by i) a link with a term in an aspect vocabulary, and ii) constraining the local attributes of the term. When a constraint is applied to a past model the system will first look for the term linkage and then, based on the mapping list that is associated with this model, compare the value of a model attribute with a range that is specified in the constraint.

b. Constraints on part properties specify the type of a part and set boundaries on the number of parts of this type. The type is specified by using names of categories or object type definitions (other scenarios). Therefore, to evaluate whether a component belongs to a

certain type, it requires a process which evaluates whether its parts belong to the types in the scenario. This is a recursive process of type checking that boils down to verifying whether standard component parts belong to mentioned categories the predefined sets of standard components.

The interpretation of the functional specification is controlled by activation of aspects and categories. Deactivating an aspect leads to constraints on physical properties in object type definitions being ignored. This results in the selection of a larger set of components that meet the requirements of object type definitions. The relevance of this possibility can be illustrated with imagining a situation whereby a doll's house should play a role in detailing a floor plan for a real house. A doll's house will probably not meet the measurement requirements of a real house. Making the aspect 'geometrical-measurement' inactive however will make the doll's house selectable.

Deactivation of categories will lead to a more global interpretation of them. When a scenario (i.e., an object type definition) refers to a category that is marked 'inactive', the system will collect all standard components that belong to those categories that refer to this inactive scenario. For instance, we want to make a functional specification on a dwelling with one study and one bedroom. Assume beds and desks, which distinguishes studies from bedrooms, are subcategories of a category furniture. Deactivating the category desks, will make the scenario of a study able to select a bedroom. This will make a house with two bedrooms meet the constraints in the functional specification.

5.2.2. Step 2

A hierarchy of object type definitions is based on the comparison of all extra data with respect to the functional specification of the set of selected components. The comparison is based on the identification of the type of these parts. As more than one type can match a part, identification of the type results in a set of appropriate types. The system will then look for the type of part that is referenced by most models. This type is used to subdivide the set of selected models into subsets based on the number of parts of this type. This type defines the first level in the subtype hierarchy. Each subtype is then further subdivided. The process stops when the description of all elements in a subset matches in detail or when a subset contains a single element.

5.3. Use of the Virtual Hierarchy: creation of a template

Selecting an object type definition in the virtual hierarchy by the user results in the creation of a template. The process of creating a template is subdivided in the following steps.

1. All components that has led to the creation of the selected type definition are first collected.
2. A template is created that contains the initial part-whole structure as described by the selected type definition. It uses default names for all parts (names of attributes only have a local purpose and can always be modified by the user).
3. This template is extended with the attributes mentioned in the scenario, e.g., attributes like the height, width and depth of our new table.
4. It is also extended with the terms mentioned in the scenario, e.g., the block shape of the table.
5. All attributes linked to the selected terms will next be added, with arbitrary names (a composition of all the names of attributes as used in the different items in the selected set). For instance, any block has a volume and surfaces. Therefore when these are mentioned in one or more of the selected models, they will be added to the template.
6. As the attributes added in (5) might be used to refer to other aspects in a model, the terms in

these aspects will also be included in the template.

After this, step 5 and 6 will be re-executed until the sets of attributes and terms are not longer extended. For instance, when the length and width of a model are used for strength calculation then terms concerning kinematic aspects are added.

Note that the template stays interactive in a sense that removal of parts or attributes will lead to removal of dependencies.

6. Relation with Object-Oriented Systems

Object type definitions in our system can be compared with classes in class-based object-oriented systems [Borning86a; Lieberman86a] Both systems provide a method to modularize and organize data and to support the creation of new objects. However, there are some major differences:

a. In class based object-oriented systems there is a notion that enough information is available beforehand to define a class for a group of objects. Each individual object can be created by instantiation of the class. In our system, every instance is an autonomous entity whose properties might become part of an object type definition. The reason to apply this approach is that in our case the object is defined incrementally so that an object type definition cannot be given beforehand.

b. Class based systems assume that it is possible to create a unambiguous specialization/generalization hierarchy of classes. In our system, it is not possible to define a single specialization/generalization hierarchy. Firstly, the hierarchical order is dependent on an evolving database. Secondly, it is based on a functional specification whose interpretation is controlled by the user and therefore can lead to several alternative hierarchical organizations. The reason we took this approach is that the hierarchical order plays an important role in the identification of the intended object type.

c. In class based systems, the use of subclassing determines which properties of the superclass are inherited. In our system we decided that the user should choose which properties of the object type definition are actually inherited. The reason for this choice is that, even in the case where the system provides a suitable object type definition, this might not completely reflect the users intentions. It must therefore be modifiable.

7. Summary and Conclusion

In this chapter we discussed a method for creating an 'is-a' hierarchical view on a set of elements in a database that has a part-whole structure. The reason for creating such a view is that we would like to use an 'is-a' hierarchy to identify objects according to their type. In our view this plays an important role in design. Our design system supports a design strategy based on extensional decomposition. As a result the system's database has a part-whole structure. An essential item in applying a design strategy based on extensional decomposition however, is the selection of prototype objects based on a functional specification. The appropriateness of our design strategy can be deduced from a diversity of other design theories. There is the work of Yoshikawa [Yoshikawa81a] on his general design theory, and the work of Gielingh [Gielingh88a] on Product Modelling. Design strategies as described by Alexander [Alexander64a], Eastman [Eastman87a], Gero [Gero85a] or Brown and Chandrasekaran [Brown85a] also contain elements that justify our strategy.

We identified a functional specification with an object type definition. Type definitions however can be structured in 'is-a' hierarchies rather than 'part-whole' hierarchies of object data. Considering the structure of our database this made us look for a mechanism to transform a 'part-whole' hierarchy into an 'is-a' hierarchy for the purpose of supporting the declaration of elements which should become part of a 'part-whole' hierarchy. This chapter described the mechanism we have developed to this purpose. Using Yoshikawa's terminology we make a distinction between a functional (constraint based) and a attributive (product based) specification. Our system transforms existing attributive specifications into prototype attributive specifications by means of a functional specification as an intermediary.

On the basis of a functional specification we make a selection from a set of components that belong to the system's database. Based on their part-data, these components are generalized to object types. The set of types is organized into a hierarchy. The structure of this hierarchy is controlled by the user who manipulates the interpretation criteria of the system. From a hierarchy of object type definitions the user selects a single one. This leads to the creation of a template that is used for a prototype attributive specification.

The transformation of a part-whole hierarchy into an is-a hierarchy takes place *virtually*. We only present a view on an existing database with components which are existing objects. This view also only exists for the purpose of creating a single template and exists only during this process. This makes it *volatile*. To select the most appropriate template the user is free to manipulate the interpretations of the functional specification that lies above the hierarchy of object type definitions. He is able to create a number of *alternative* hierarchies based on the same specification. We call our approach 'Virtual Hierarchies Creation in Retrospect'.

References

[Alexander64a]
Alexander, C., *Notes on the synthesis of form,* Harvard University Press, Cambridge, Massachusetts, 1964.

[Bijl87a]
Bijl, A., *Strategies for CAD,* pp. 2-19, Springer Verlag, Berlin, 1987.

[Borning86a]
Borning, A.H., ''Classes versus prototypes in object-oriented languages,'' in *1986 Proceedings Fall Joint Computer Conference*, ed. H.S. Stone and S. Winkler, pp. 36-47, IEEE Computer Society Press, Washington, D.C., 1986.

[Brown85a]
Brown, D.C. and Chandrasekaran, B., *Expert system for a class of mechanical design activity,* pp. 17-19, North Holland, Amsterdam, The Netherlands, 1985.

[Eastman87a]
Eastman, C.M., ''Prototype integrated building model,'' *Computer Aided Design*, vol. 12, no. 3, pp. 115-119, Butterworth-Heinemann, 1987.

[Gero85a]
Gero, J.S., Radford, A.D., Coyne, R., and Akiner, V.T., ''Knowledge-Based Computer Aided Architectural Design,'' in *1984 Proceedings of IFIP WG 5.2 Working Conference on Knowledge Engineering in Computer Aided Design*, ed. Gero, J.S., North Holland, Amsterdam, 1985.

[Gielingh88a]
Gielingh, W.F., ''General AEC Reference Model (GARM),'' Report by TNO-IBBC, Rijswijk, 1988. October

[Lieberman86a]
Lieberman, H., ''Using Prototypical Objects to Implement Shared Behavior in object-oriented Systems,'' in *Object-Oriented Programming Systems, Languages and Applications*, ed. Meyrowitz, N., pp. 214-223, ACM Press, New York, 1986.

[Rogier91a]
Rogier, J., ''A Component Class for Design,'' in *Intelligent CAD systems 3: Practical experience and evaluation*, ed. Veerkamp P. and Ten Hagen P., Springer-Verlag, Berlin, 1991. forthcoming

[Veth87a]
Veth, B., ''An Integrated Data Description Language for Coding Design Knowledge,'' in *Intelligent CAD Systems I: Theoretical and Methodological Aspects*, ed. Ten Hagen, P.J.W. and Tomiyama, T., Springer Verlag, Berlin, 1987.

[Wilson87a]
Wilson, P. and Kennicot, P.R., *STEP/PDES Testing Draft,* St. Louis Edition 4.1.2, St. Louis, U.S.A., 1987.

[Yoshikawa81a]
Yoshikawa, H., ''General Design Theory and a CAD System,'' in *Man and Machine Communication in CAD/CAM*, ed. Sata, T. and Warman, E., pp. 35-57, North Holland, 1981.