Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

J.L.H. Rogier

A component class for design objects

Computer Science/Department of Interactive Systems Report CS-R9060 October

A Component Class for Design Objects

Jan Rogier
Department of Interactive Systems
Centre for Mathematics and Computer Science (CWI)
Kruislaan 413, 1098 SJ Amsterdam, The Netherlands

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

This paper discusses an architectural design support system. The system is based on a strategy of incremental spatial decomposition of a design object. It provides mechanisms for the application of strategies such as functional decomposition and re-use of design object knowledge. First we discuss two basic strategies used in architectural design, viz. decomposition and application of prototypes. We then discuss the architecture of a design system that supports both strategies. By means of an example, we show how the system actually supports an architectural design process and how the system is used to communicate with external, autonomous design support expert systems. Such systems provide special facilities for drawing, calculating and other tasks performed during a design process, and are interfaced by means of our design system. The last part of the paper contains an evaluation of our work. We propose topics for further research for our project: the IIICAD project. The aim of our system is to provide a sound basis for the tools developed in the IIICAD project and at the same time to be an interface to already existing tools like expert systems for FEM modelers, drawing machines etc.

CR Categories and Subject Descriptors: D.1, J.6.
Key Words & Phrases: intelligent CAD, knowledge representation, design strategy, object-oriented programming.

1. Introduction

One of the main problems in knowledge based design support is the representation scheme of a design object description. Such a representation scheme must fulfill a dual purpose. In the first place, it is an integrated description of the design object, constructed during the course of the design. In the second place, it is used in the process of re-using design object knowledge in other designs. Only a representation scheme that can be used for both purposes, forms a strong basis for the construction of a knowledge based design support system.

Both intentions of the above mentioned design object representation scheme lead to different requirements. The identification of these requirements is based on the analysis of design. In design we identify two basic strategies. A first strategy is incremental decomposition of the design object. It supports the evolution of a design object description from global to detailed, using stepwise refinement [9]. We call every increment in such a decomposition a *design step.* A second strategy is the application of existing design object descriptions to support a design step. Such existing design object descriptions are used as *prototypes* for new designs.

Report CS-R9060
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

With the above introduced decomposition strategy we capture both the design process itself and the result of a design process. The result is a hierarchical design object description. There are several methodologies for object decomposition. We make a distinction between *functional* and *spatial* decomposition. Functional decomposition uses the functional specification of a design problem. It decomposes a design problem into comprehensible sub-problems and accordingly it provides a functional decomposition of a design object [1]. Spatial decomposition aims at building the structure of a design object. With spatial decomposition, a design object is described as an assembly of spaces with sub-spaces.

The application of existing design object descriptions as prototypes, supports the actual declaration of a design object. A prototype provides the user with a previously built spatial decomposition structure, which serves as a template for further declaration. The user can change this template in order to make it fit to a functional specification. A prototype contains in general more information than supplied by a given functional specification for the design object description. Therefore, a prototype can be used to extent the functional specifications.

In this paper we propose a design supporting system based on spatial decomposition of a design object. In the next section we introduce and elaborate the concept of spatial decomposition and the application of existing design object descriptions as prototypes. In §3 the system and its functionality is described. §4 contains an example that illustrates the application of the system. The last section evaluates the system and proposes topics for further research.

2. Architectural Design Strategies.

2.1. Functional and spatial decomposition

An architect's design problem is to assign both space to function and function to space. In both cases he applies a strategy based on *decomposition.* We define *functional* decomposition as assignment of space to function, and *spatial* decomposition as an assignment of function to space. A special case of spatial decomposition is called *material* decomposition, which is a known strategy in assembly based design [2].

Functional decomposition results in a functional specification of different parts of a design object model. A functional decomposition is associated with a specific field of interest. The concept of functional decomposition for architectural design is discussed in detail by Alexander [1]. He solves a design problem by sub-dividing it into different sub-problems, corresponding to different sub-functions. Each of these problems is tackled by an independent sub-system. The design solution is found by integrating the results from each of the sub-systems. Straight on application of a strategy based on functional decomposition, results in a number of aspect models of a design object. These aspect models exist independently from each other. The major disadvantage of the application of functional decomposition, is that it generates a problem of integrating different aspect models into a single, coherent design object model.

Spatial decomposition describes an object as an assembly of spatial parts, which may have sub-parts. Spatial decomposition is devoted to the design object solution. Spatial decomposition combines function with space in a sense that any given (sub-)object is characterized by the fact that the space it occupies is devoted to one function. Such spaces are disjunct, but may overlap resulting in sub spaces resulting in more than one function. An example spatial decomposition is material decomposition. Material decomposition only considers spaces filled with a material.

The integration of functional and spatial decomposition in a design supporting system, and the possibility to incorporate material decomposition in this concept, is based on the next strategy.

- A space is devoted to one complex functions. Aspect models can be produced as required by internal and external applications. These aspect models are reductions of the spatial decomposition which still have the property that the relevant functions are uniquely determined through spatial references. Aspect models therefore are subsets of a model based on spatial decomposition.
- A space is made from one composite material. This composite material has unique properties. Further decomposition of a space into its sub-spaces causes a more detailed material specification of the design object.

We elaborate this strategy as follows: With functional decomposition, as proposed by Alexander [1], for each function, a sub-system or sub-model of the design object is generated. Each function poses constraints on spatial parts or sets of spatial parts. The same spatial part is referred to by different aspect models. For example, a 'wall' of a 'building' will have both a function in the construction of that building and a function of being a shelter for the inside of the building from the outside. All functions of a part has become one unique composite function, as soon as they are associated to a specific spatial part. A part therefore has one function.

Physically speaking materials are mutually exclusive, therefore material decomposition is a special case of spatial decomposition. Material, in architectural design is defined on a high semantic level. On this level all material is composite material. This composite material has unique properties that are relevant in designing and are not directly to be derived from the different sub-materials. As an example consider the case of a 'brick' under 'water'. In this case there are, logically speaking two different materials in the same location (e.g. 'water' and 'stone'). 'Wet stone', however, has unique properties which cannot directly be derived from the properties from 'water' and 'stone' independently. That kind of unique properties is important in designing. Wet stone is for instance a bad material to use for walls in a sleeping room.

Taking the whole strategy into account, both functional decomposition and material decomposition can be expressed in terms of spatial decomposition. We agree that a strategy based on functional decomposition is a commonly accepted approach for specifying a design object. However we claim that a design object description based on functional decomposition is insufficient when used to support the design process. The reason is that a strategy based on functional decomposition results in a number of independent aspects models which causes a problem of integrating these models into a single design object description.

In our approach functional decomposition is applied using autonomous applications that uses local representations of a design object description. These local representations are derived from the same integrated design object description which is based on spatial decomposition. All autonomous applications contribute to this central design object description by extending the complex functions that are associated with the different spatial components. A strategy of material decomposition is supported by associating a single composite material to each spatial part that belongs to the design object description. This composite material may either be solid or not, which allows 'spaces filled with air' to be part of the decomposition. The description is especially aimed at reuse of design knowledge. As a side-effect it is possible to link our approach with international developments on standardization of product data. This is done by combining our representation with *standard vocabularies.* A standard vocabulary is a set of all primitive entities and operations, used in a specific knowledge domain. E.g. some members of a geometric vocabulary are the primitive entities: *block* and *cylinder* and the operations: *calculate volume* and *calculate the conjunction of two primitives.*

3. Design Support System

In this section we introduce a design system which supports the design strategy described above. Each component of the design system is explicitly accessible. The system is used as a design support tool rather than as a design expert system. This apprentice like behaviour has a twofold reason. The main purpose of our implementation is to illustrate the applicability of the design strategy, presented in §2. The second reason is that we believe that a system should never be more than a tool in supporting the design process. Instead of hiding information from the user, as is often done in so-called expert systems, we state that the purpose of a design system is to expose information. For more detailed considerations on this approach we refer to the work of Bijl [3].

3.1. System architecture

The design support system, described in this paper, is built from a set of components (Fig. 1) It contains a kernel, a database, a knowledge base and a set of external application modules which are accessed by means of application interfaces. The kernel is used to construct a design object model, based on spatial decomposition. This model contains references to entities that belong to the application interfaces and that are stored in what we call *standard vocabularies.* Standard vocabularies are used to access external applications. Applications are used to support functional decomposition of the design object. Support of prototype based declaration of a design object description is done with the kernel. With this kernel a knowledge base and database are accessed. The database contains previously built models of design objects, the knowledge base contains descriptions of standard components and manufacturing features. A detailed discussion on the system's architecture can be found in [6, 7, 9].

3.2. Object-oriented and logic approach

We use an object-oriented approach both for structuring the implementation of the experimental system and for structuring the model of the design object. The implementation of the experimental system is based on an object-oriented approach in order to get efficient code. Besides it provides a simple and comprehensible structure that is easy to explain. The structure of the model of a design object is based on an object-oriented approach, because this approach best fits on a strategy based on spatial decomposition.

Code efficiency in object-oriented programming is reached by using a fixed hierarchy of classes with subclasses. Each class contains the description of the data template of instances and all functions that may be applied on instances (methods). The hierarchy of classes is an *is-a*

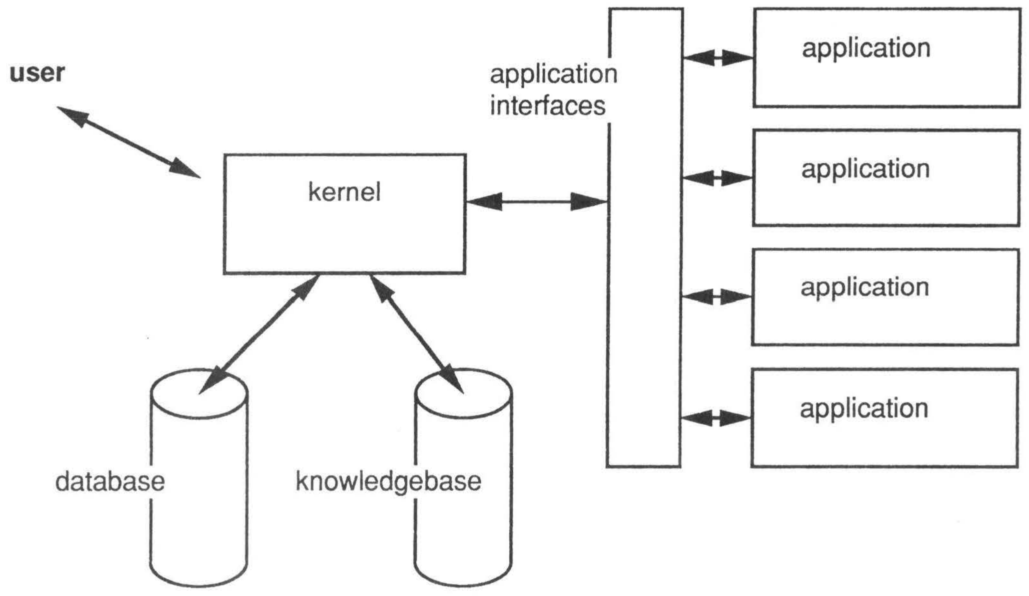


Fig. 1. System's Architecture

hierarchy. Every subclass is a specialization of its superclass. For our system we aim to limitate the number classes that are accessible to the user. In this paper we mainly discuss the class *component* from which part of a model of a design object is an instance. These instances contain references to instances of other classes that are used to define application interfaces.

The model of a design object is also based on an object-oriented approach. In this case however we use different hierarchical approaches. We use an *part-of* hierarchy to describe the decomposition structure of the design object. We use an is-a hierarchical organization of parts to generate prototype descriptions for new parts. This made us implement all parts of a design object as instances of a single class: component. All instances of this class are defined by an extendible number of attributes. Virtual is-a hierarchies of classes are specified by predicates over the set of all instances of the class component. Such a specification allows for a flexible hierarchical organization. We call procedures, that generate virtual is-a hierarchies, *grouping* functions.

3.3. Grouping

A group is a temporal class. A group contains a subset of all instances of the class component that meets a set of predicates. As classes groups are organized in hierarchies. These hierarchies describe is-a relations between classes and sub-classes. We prefer to use the word *group* instead of *class* in order to express the temporal existence of this entity. During their life time however, a group provides the same features as are provided with classes in common object-oriented approaches. Groups are generated by means of grouping procedures that contain a set of rules applied on the contents of all instances of the class component.

Grouping results in the creation of a template. This template contains all attribute names and their types of value [4]. When attributes refer to other instances of the class component or descriptions of relations between values of attributes, these are also included in the template.

Generation of a template is controlled by procedures that describe how existing instances should be interpreted. From these candidates the common set of attributes serves as a basic template, while the non-common attributes cause a system controlled interview of the user. As a default mechanism the ***group identifier*** of a part is used to select appropriate candidates for generating a data template for a new instance. A group identifier is a user declared name that expresses the class to which a user thinks that an instance belongs. To give an example, 'living room', 'bed room', 'kitchen', 'wall', 'door' etc. are all spatial parts of a design object 'house'. They are represented by instances of the class component. 'Living room' and 'bed room' both belong to the group 'room'. What a room is, is described in a procedure. (All instances that have the string "room" as value for the attribute 'group-identifier', belong by default to the group 'room'.) This procedure generates a subset of instances from the class component that is structured in a hierarchy based on their set of attributes (Fig. 2).

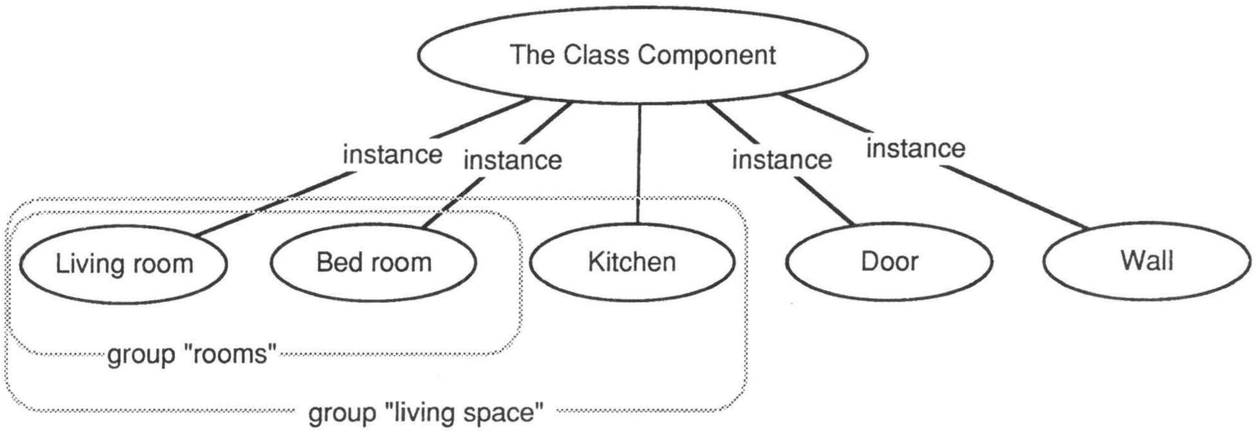


Fig. 2. Classes, Groups and Inheritance

3.4. Unique identification

Each part of a design object is represented with an instance of the class component. This instance can be uniquely identified. Instances of the class component have two identifiers. They have a *name* and they have a *reference* (compare Fig. 4). The name is a string of characters, is local to the instance and does not have to be unique within the system. The reference or pointer is unique in the system, but is only used by the system internally. The user has no direct access to it. Unique identification of an instance is based on identification from context. Every part (always represented by an instance of the class component) is sub-part of another part. On top of this (part-of) hierarchy is the design system itself. Sub-parts of the system are *projects* representing the initial object of a design problem (e.g. 'myhouse' etc.) (Fig. 3). Every other sub-part is part of another part. Although each part has a local name, the system uses a enumeration of names to identify a part by name. This enumeration expresses the part off hierarchy for a specific part, initially starting from the current context. This name may include backward referencing symbols and is as a mechanism comparable with directory identification in UNIX.[1]

[1] UNIX is a trademark of AT&T Technologies, Inc.

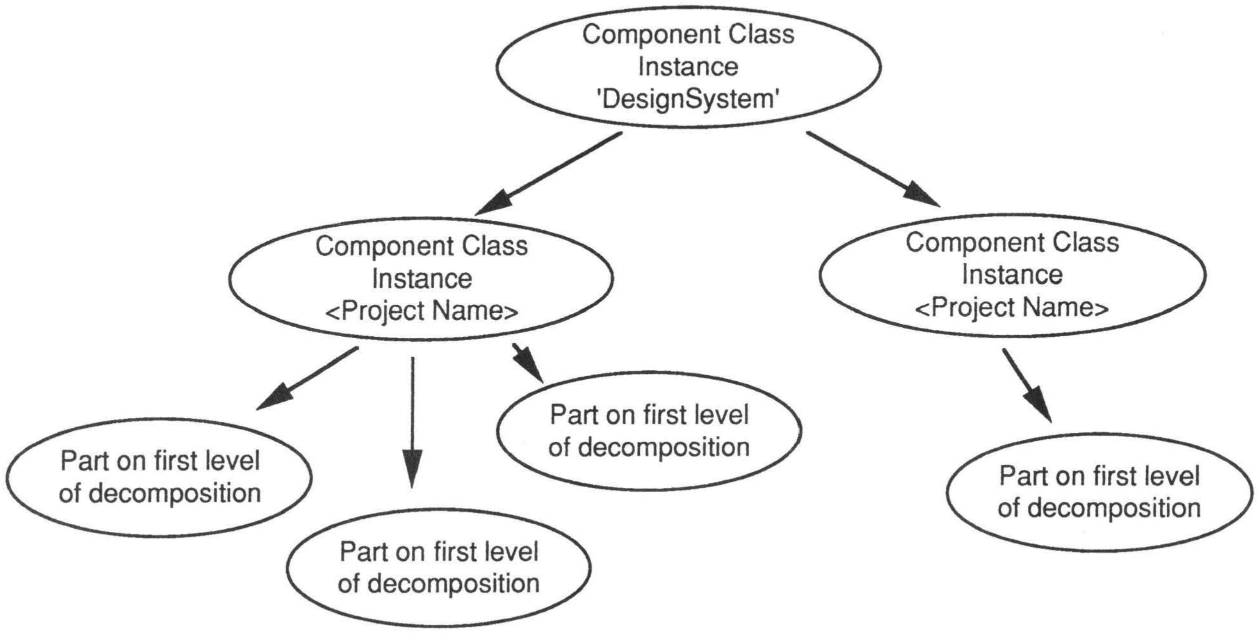

Component Class Instance 'DesignSystem'
Component Class Instance <Project Name>
Component Class Instance <Project Name>
Part on first level of decomposition
Part on first level of decomposition
Part on first level of decomposition
Part on first level of decomposition

Fig. 3. Has-part hierarchy organization

3.5. Data structure of instances of the class component

An instance of the class component has three variables. It has a name, it has a *group identifier* and it has a *dictionary* with attributes. These attributes serve three different tasks (Fig. 4).

1. They are used to define constant type properties of the part. For example, the attribute 'colour' has a constant value.
2. They are used to describe the decomposition structure of the design object. For example, an attribute 'leg_1' of a 'table' contains a reference to another instance of the class component.
3. They are used to describe relations between different parts or constraints on a part. For example, a reference to an expression that describes the relation between the 'lengths of legs' of a 'table', is a value of an attribute of this table.

Each of the above tasks is denoted by the type of the attribute value. This *type* denotes the attribute (e.g. constant typed attribute, attribute referring to a part, attribute referring to a formula expression).

3.6. Instance creation and declaration the attribute template

New parts are created as sub-parts of existing parts. This implies a top-down strategy in declaring the structure of the design object. Declaration of the attribute template of a instance can be done in three ways. i) The set of attributes can be declared one by one, ii) references can be made to predefined entities that belong to a standard vocabulary and have a fixed set of attributes, and iii) prototype based declaration strategy can be applied by making copies of either standard components, previously defined objects, or templates of objects that are generated using grouping procedures. Like in ii), creation and declaration of the attribute template are a single action.

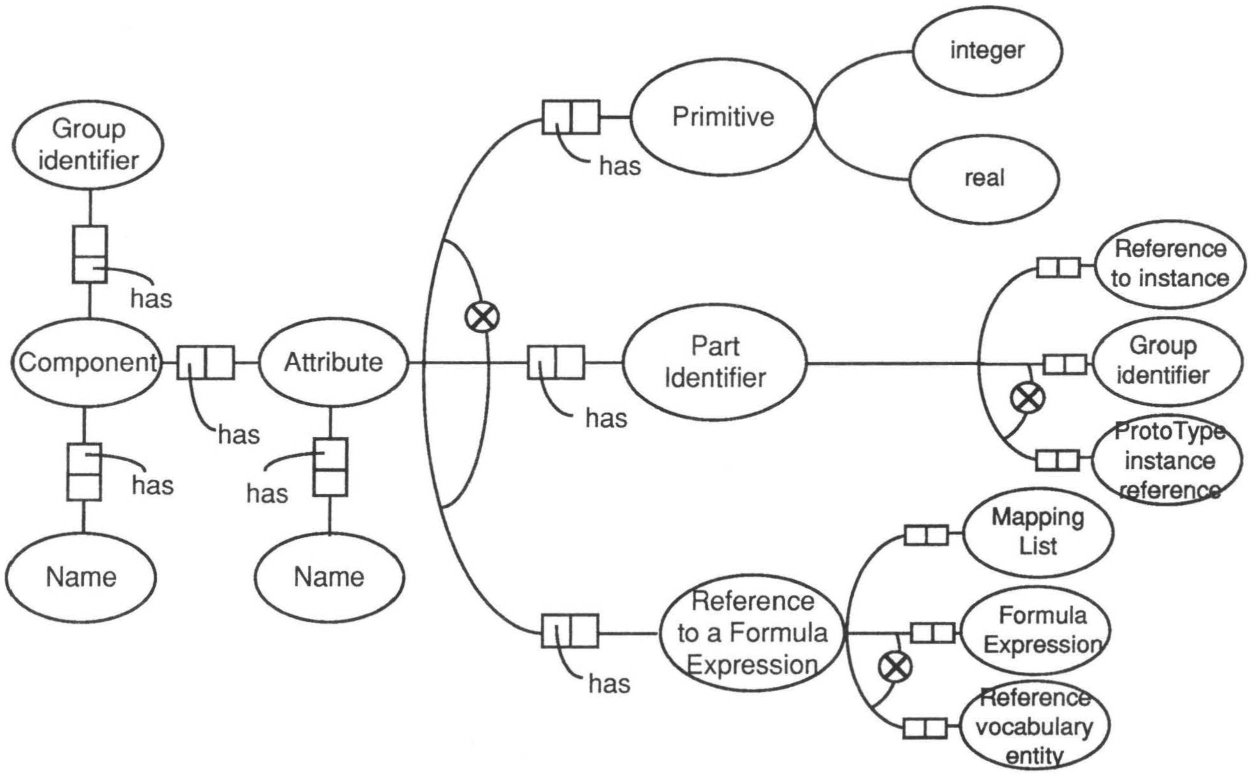


Fig. 4. Data model

The first approach is outrageous and will probably only be used in specific circumstances. The second approach is used to describe aspects of the design object. This approach has a two-fold purpose. It is used to detail the model of the design object and it is used to establish a link with external applications that evaluate this model. In the example we elaborate the application of a geometric vocabulary. The third approach provides a template that is generated with a procedure that evaluates the contents of the systems' database or knowledge base. In both cases instance creation and the generation of the attribute template are one and the same action. Standard components belong to the system's knowledge base. A template based on a standard component can not be changed structurally but only for a limited set of attribute values. The system's database contains previously built models of design objects. The contents of this database is evaluated by means of a (selection) procedure. This procedure is an intermediate value of a part-reference attribute. The result of applying this procedure is a attribute template. When the procedure results in the selection of a single instance, a deep copy of this instance is made including all constant values of attributes. When the procedure results in a set of instances the same procedure has to define a strategy to combine information in a single template.

4. Mathematical Dependencies between Attribute Values

Relations between parts are described by means of (formula) expressions. These expressions describe the (mathematical) relation between constant values of attributes of parts. The description of a relation itself is not part of the design object description, but is stored in an external library. The connection between the design object description and the relation is made by means of a reference and a *mapping list.* The mapping list describes the mapping of attribute-names used in the design object description on attribute-names used in the description of a formula expression. A formula expression has a twofold function. It is used to calculate values of attributes and it is used to verify values of attributes. The specification of a formula expression can be used to guide the detailization of a part. Each of the attributes of an expression has to be mapped on an attribute of the part description.

Formula expressions are stored in libraries. Sets of formulas can be combined into a single entity called primitives. Primitives are also stored in libraries. When these primitives has a twofold functions in a sense that they are used as basics in the communication with external applications, we call these libraries standard vocabularies. Apart from a set of primitives these vocabularies include functions used to access external applications. An example of an standard vocabulary is elaborated in the example.

5. Maintenance of Consistency

Maintenance of consistency is considered on two levels: on the level of the design object description itself, and on the level of the design object description in the context of the system's data base.

5.1. Design object description

Maintenance of the design object description is carried out by means of values of attributes and the evaluation of formula expressions. Each attribute of a design object with a constant value (e.g. 'height'), may have at most three values: a *set* value, a *calculated* value, and a *default* value. The first one is (explicitly) set by the user; the calculated value is the result of executing a formula expression; the default value originates from a prototype for the design object. The set value prevails above the calculated value, the default value is only used for informational purposes but plays no active role in calculation procedures. The consequences of adding and overwriting values are controlled by an user declared procedure, linked to the object description. From each of the attributes of an object is known (is declared) what will happen when its value changes. As a default mechanism every change will cause a warning procedure. We adapted this approach for experimental purposes only. It is subject to further research.

The existence of default values are used to express the difference between re-use of old designs and the use of so-called *standard components.* Copying of an 'old' design object results in a copy where all constant values are used as defaults (for informal query purposes only); applying a standard component results in an copy where all (or a predefined subset) constant values are interpreted as set values.

As soon as all values used in a formula expression has a value (either set or calculated elsewhere) the expression is used to verify the values of these attributes. If the values does not correspond, based of the evaluation procedure linked to the object, a warning is produced.

5.2. System's knowledge base

Maintenance of consistency on the level of the system's knowledge base is done by restricting the possibilities to refer directly to other object description. As soon as a design object description is used as a prototype for another design object description, it can not be changed any longer. Changes to be made to the prototype cannot be carried out, except by making a new object description that uses the existing one as a prototype. This approach causes problems on conceptual level. In this approach it is not possible to access the properties of number of objects except by incorporating them in a single project. So, for example, changing the 'colour' of a set of 'chairs' is not carried out by changing that property of the object that is used as a prototype, but by declaring that the whole set of chairs that belong to a project and have some common property that is defined on the level of the project; e.g. 'colour of all chairs' (see Fig. 5) The system's decision when to lock a design object description (e.g. prevent further elaboration of the description), is based on the existence of a prototype reference to that object.

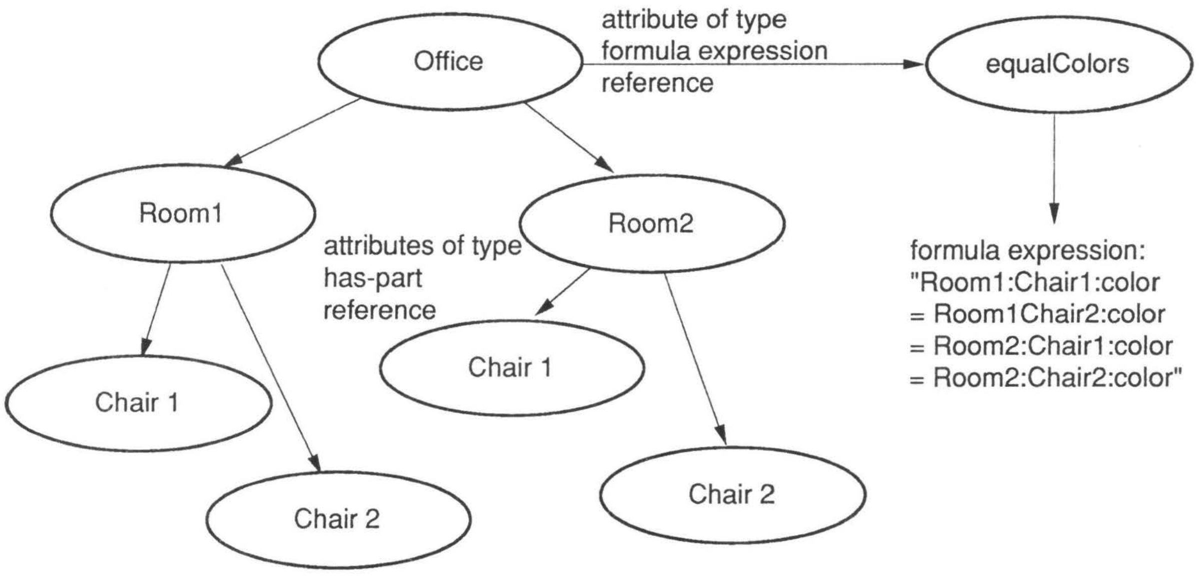


Fig. 5. Relations between different parts

All restrictions together make it possible to store only the new information about an object. All existing information is available for re-use but only (virtual) copies can be used in the context of a new design object description. As a consequence, the first project ever to be done on the system will cost a, relatively, enormous amount of space and specification time. On the longer term, and depending how wide the application field is, each new project will relatively occupy less space and probably less effort.

6. Example

This paragraph contains the description of an example application of the system for architectural design purposes. It describes the design of a simple cottage as people in Holland often build on their allotment. Using an example like this, is based on several considerations. In the first place the example results in an independent object which does not have to fit into a context. In the second place the object is relatively simple. In the third place a cottage is, compared to other buildings, relatively weakly defined which gives the designer a lot of freedom. This makes the example easier to explain. In order to simplify the example to a further extend, we will not focus on generating alternative solutions. The example is meant to illustrate how design can be supported in our approach, not to generate the best cottage ever designed. Another restriction is, that we will concentrate on the description as if it was the first one ever made with our design system. Explanation of the strategy of *prototype application* by means of an example, makes this example unnecessarily complex and will not elaborate this topic.

The example is described in three sub-paragraphs. The first paragraph contains a abstract description of what happened in the design process. It also compares this design process with more complex design processes. The second subparagraph describes the design process in terms of spatial decomposition. The third paragraph focus on a specific design step. In this paragraph is described how an external application will be used to support the process.

6.1. Design process

In our system we apply a design strategy based on spatial decomposition and which contains the following tasks (Fig. 6). The first task is to specify global constraints on the design object. The second task is to translate these constraints into properties of the design object. The third task is to decompose the design object into parts. This task is further sub-divided into several sub-tasks in during which further decomposition takes place. The fourth task is to make links between the decomposition structure and items that belong to libraries of standard components and libraries of *features*. Standard components are the lowest level of decomposition of a design object. Features are descriptions of manufacturing processes which are identified by means of their result. An example of a standard component in our case is a 'door' with a 'door-frame,' an example of a feature is a specific 'wood joint.'

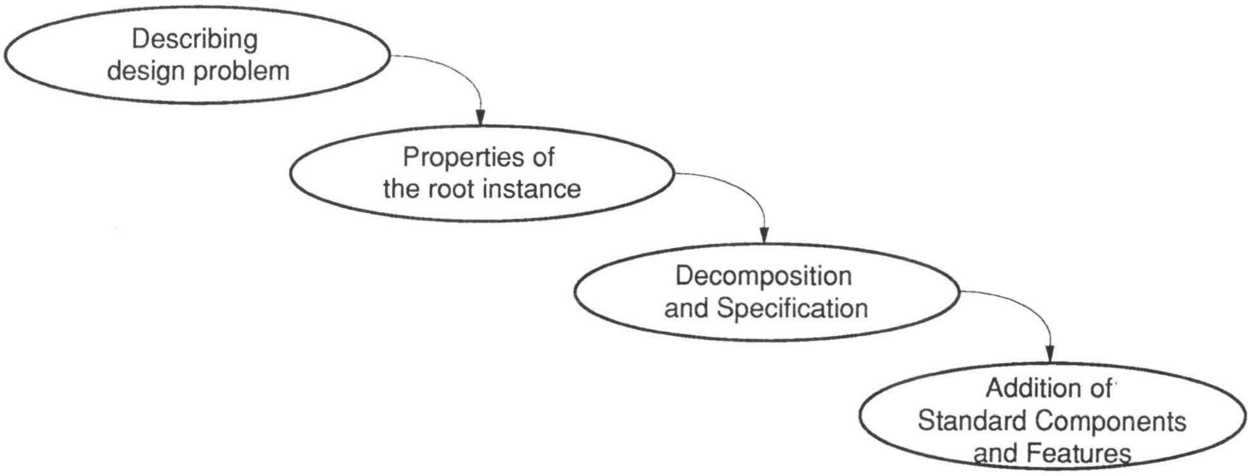

Describing design problem
Properties of the root instance
Decomposition and Specification
Addition of Standard Components and Features

Fig. 6. Design process structure

Although subdivision into different tasks insinuates that the process as a whole is sequential, the process is in fact iterative. Which tasks are performed, depends on the amount of information that is actually present at a specific moment. So, for example, cost analysis is only useful as soon as information is present about materials and features, while constructional aspects can only be taken into account, when information is present about the possible location of construction elements. Evaluation of different aspects, in our approach, takes place by means of external applications. These applications internally build a representation of the design object description (e.g. a sub-set of information) and produce information that can be added to the model of the design object. The process is illustrated in Fig. 7.

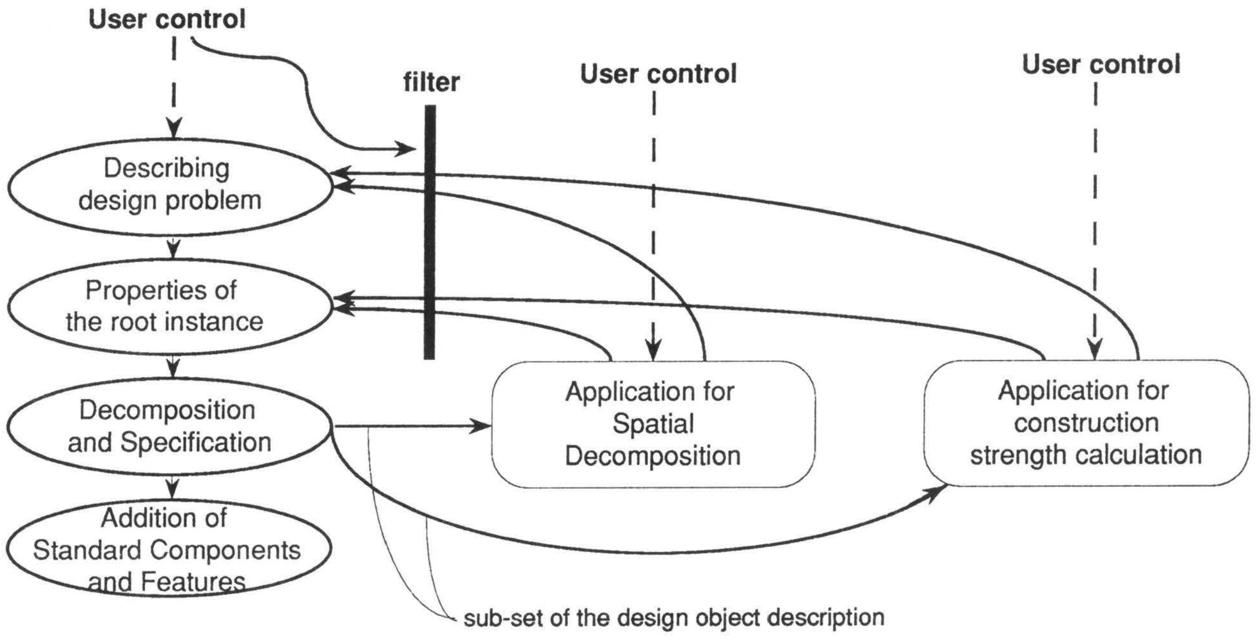

User control
filter
User control
User control
Describing design problem
Properties of the root instance
Decomposition and Specification
Addition of Standard Components and Features
Application for Spatial Decomposition
Application for construction strength calculation
sub-set of the design object description

Fig. 7. The role of external applications

6.2. Design object description

For the sake of simplicity, we skip the iteration issue in this approach and describe the process as if it took place in a single sequence.

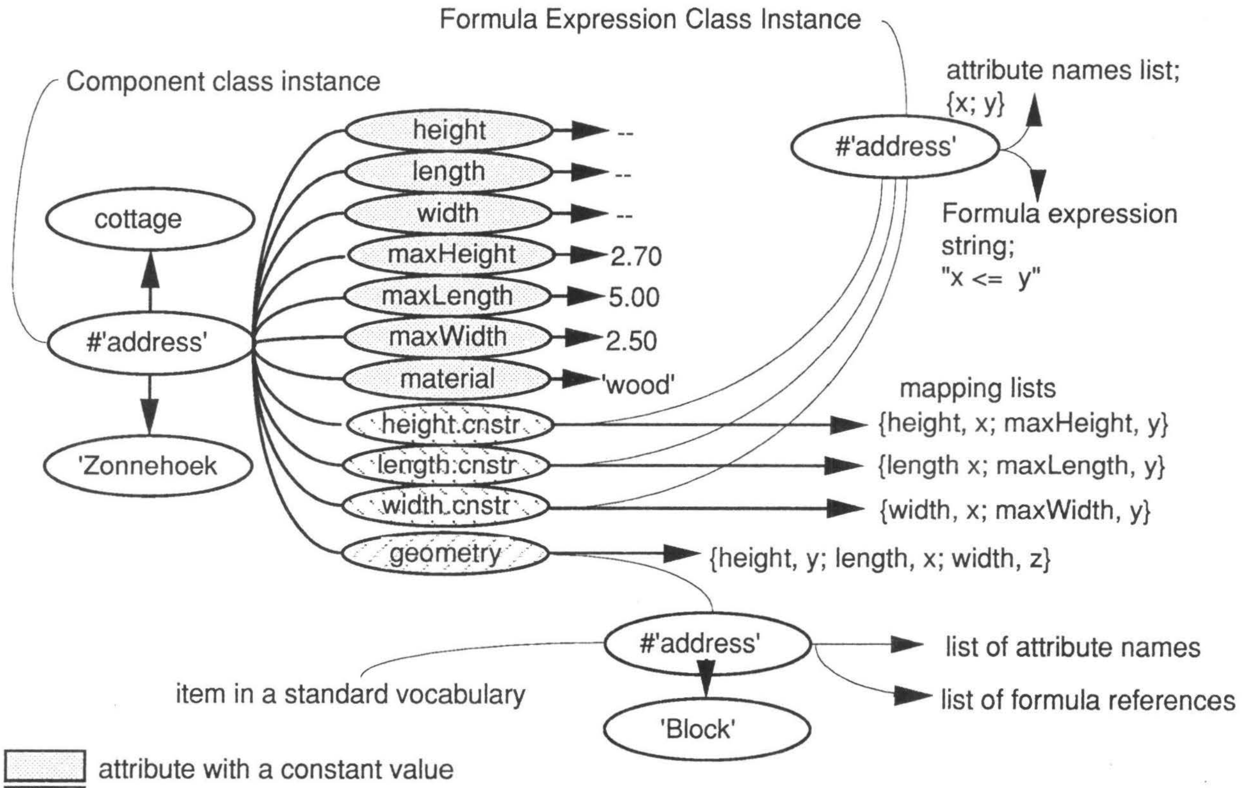
6.2.1. Task 1. The first task in the design process is taken by collecting some global properties of the design object:

- The most important restriction on cottages is that they may not be used for permanent living.
- They have to be built from 'wood' or another non-permanent material, they may not have permanent sleeping facilities (you may stay in it for a couple of days but not permanently), and they have to be built on about 12.5 square meter (measuring 2.5 by 5 meters) and may be not higher that 2.7 meter. The backgrounds of these limitations are found in the origin of such a cottage, being a dry storage place of garden tools, and a shelter to protect a gardener from rain.
- They may however have a kitchen and a bathroom including a facility like a shower. Although cottages on allotments are not intended for permanent living, they more and more are used as summer houses. So we specify some more requirements based on its function as

a summer house.

- There should exist a spatial distinction between what can be called a 'quiet area' and a a 'noisy area,' separated from each other by a 'walking space.'
- Another consideration is that the bathroom and the 'quiet area' should be separated by at least two internal walls.

6.2.2. Task 2. We start design support with initiating a new project called 'De Zonnehoek'. This name identifies a new instance of the class component, the root instance of the new project. Attributes of this object are some global properties; 'height', 'length', 'width', 'material', and some functionality in the sense of constraints on the values of attributes. For representational purposes a geometrical entity is used that belongs to a geometrical vocabulary. This assignment is carried out by means of a *geometrical attribute.* At the same time initial values of all attributes are declared. The declaration procedure has a result illustrated in Fig. 8.



▒ attribute with a constant value
▨ attribute with a reference to a formula expression as value
▧ attribute with a with a reference to an item in a standard vocabulary as its value

Fig. 8. The data structure in use

6.2.3. Task 3. The next task in the design process is to decompose the basic object into parts and parts of parts. All parts describe spaces that in a later task will be or be not 'filled' with material. The result of this task is a entity-relationship description of the design object. It contains all relations and some values used as boundary indications of values of attributes. In the process of detailing, the entity relationship model is used as a frame [5]. 'Zonnehoek' contains the non-material 'sitting space' (representing the 'quiet area'), 'walking space' and 'working space' (representing the 'noisy area'). It has four walls, a floor and a roof. The floor plan of the object is illustrated in Fig. 9. Generation of this floor plan lay out, takes place by means of an external application (in our case; the mind of the designer). Computational support of this sub-task will be discussed in the third sub-paragraph.

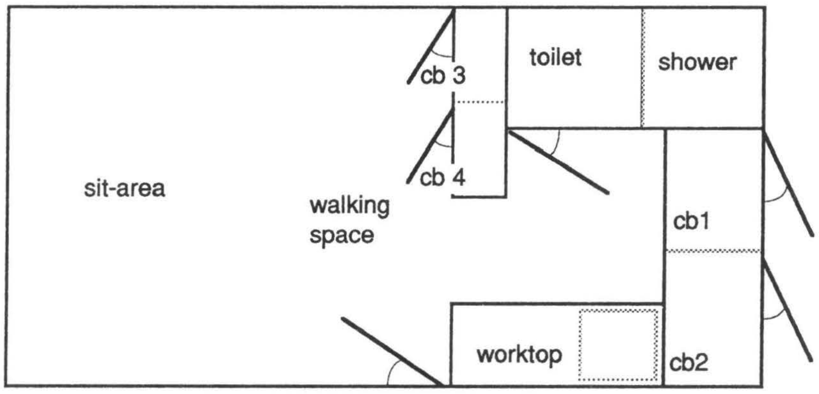


Fig. 9. Floor plan

Also during this task, the model is extended with formula expressions that describe the relation between values of attributes of parts. Relations between values of attributes of different parts or relations between values of attributes of the root object and attributes of parts, are expressed on the level of the assembling part. So the expression "(Zonnehoek:)height = floor:height + walkingSpace:height + roof:height", is specified on the level of the root object 'Zonnehoek' and not on the level of the parts. A part of the decomposition tree is illustrated in Fig. 10.

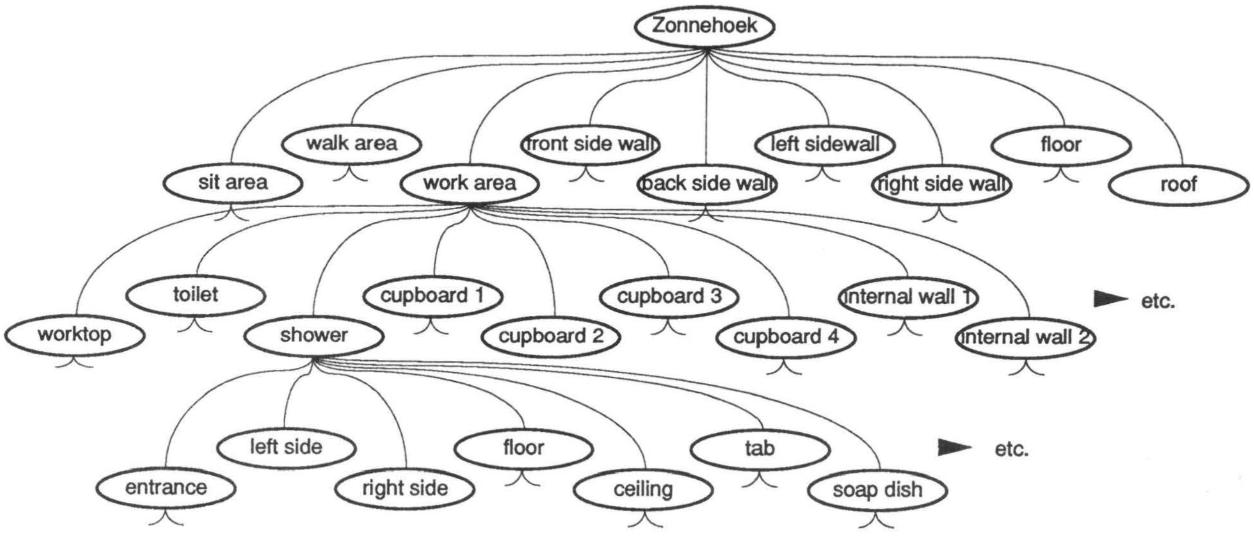


Fig. 10. Part of the has-part hierarchy

6.2.4. Task 4. The next task is to 'fill' spaces with material. When a more complex design object was chosen, this would have been the time to use a construction strength evaluation application. In the current description, all places where construction elements can be located are identified and an algorithm is applied to find the optimal location of certain construction elements. In our case (e.g. a cottage on an allotment) we will simply assume that all walls together will be strong enough to carry the weight of the ceiling and will make the total building stable (designer's experience). 'Filling' with material on the one hand takes place by further detailing the design object description, and on the other hand by choosing solutions from a library with standard elements. In our case we assume that 'doors' and 'windows' may be found in a catalogue and does not have to be designed. On a more detailed level each of the materials for construction, e.g. 'wooden beams' and solutions for 'wood joints' may also be found in a library with standard solutions. Selection of all these standard elements provide our frame with concrete information from bottom-up (e.g. all values of attributes). The last phase in this task therefore integrates a top-down and bottom-up approach in the declaration of values of attributes until the design object description is fully specified. The result of this task is a complete and detailed design object description that can be justified in the context of the consulted external applications (in our case the designer's mind) and be represented geometrically.

6.3. Spatial reasoning

This sub-paragraph gives an impression how spatial decomposition of a design object may be supported by means of an external applications. Although the basics of such a spatial reasoner are implemented, the application itself is not yet operational. The current illustration is therefore hypothetical. The process of spatial decomposition takes place on a design object description as is illustrated in Fig. 11, and results in a description as is illustrated with Fig. 11 (in 3D). The relation between these two descriptions is established, based on a definition that a *space* can be represented as a *space enclosing a surrounding space.* So the original 'Zonnehoek' was geometrically represented by the entity *block.* This block is decomposed into a block with a shell that is itself a space. Decomposition of that shell will result in the construction of a set of (at least) six blocks that enclose the internal block (see Fig. 12). In this case, the relation between the enclosing blocks and some parts of the object 'Zonnehoek' are obvious. All blocks can be used to describe the geometrical properties of parts of 'Zonnehoek' being its four walls, its floor and its roof. The same strategy is applied with spatial sub parts of 'Zonnehoek'. Each of its sub parts is represented by a block that is transformed to a set of blocks.

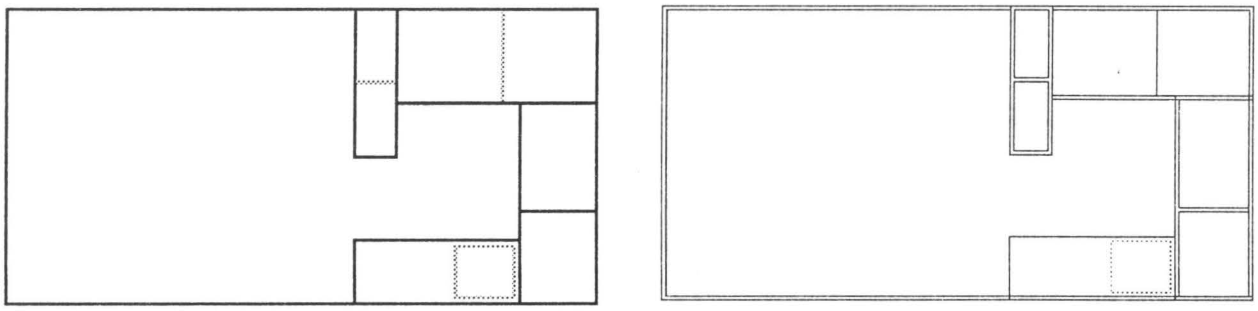


Fig. 11. Design description evolution

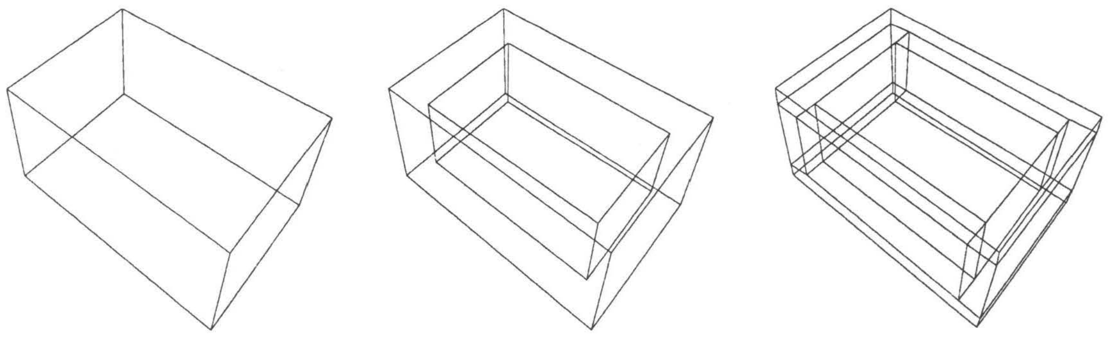


Fig. 12. Materialization of a block

The advantage of the approach is obvious when the relation between representations of spatial parts are described by means of set operations. The seven blocks representing the geometry of the object 'Zonnehoek', have by definition one ore more parts of their surfaces in common with others (in the case of the internal block: all) and do not coincide with each other. The relation between each of these representations with the geometrical attributes of sub parts is described as follows. On the level of the global definition (e.g. one block instead of seven) the space is completely subdivided into non-coinciding block (spaces) that represent the geometry of the sub parts. ('Zonnehoek' is subdivided into 'a quite area', a 'walking space' and a 'noisy area'). On the level of transformation of one block into seven, both horizontally (between subparts) as vertically (between a sub-part and a parent part) there may exist overlap (as is illustrated in Fig. 13) but no overlap may take place between an internal space of a sub part and an enclosing space of a parent part (no external wall halfway the interior of a toilet).

This consideration has some consequences: In the first place the user has to provide a description of the relation between the geometrical representation and its transformation. This can take place by means of labels that identify such relations (see Fig. 14 *exterior, interior* and *global).* In the second place, the user has to supply rules concerning overlapping spaces (e.g. ''no overlap may occur on both 'bathroom enclosing space', 'working space area' and 'walking space area','' indicating that the bathroom may not lie against the walking space wall, or anther rule: ''overlap of the spaces enclosing the toilet and the shower results in removal of this space''). These relations can be expressed using set operators: ''conjunction of 'bathroom' & 'walking space' & 'working space' empty'' or ''conjunction of 'toilet' & 'shower' is 'non-material'.'' In the third place optimization criteria should be supplied to the reasoner, saying for instance that the total amount of overlapping space should be maximized, which will say that the total amount of non-overlapping space (e.g. the internal walls) is minimized.

The actual consequence of this approach is the generation of a set of rules and labels linked to each of the parts of the design object describing the proposed interpretation of that part. So the spatial reasoner will look for a geometrical attribute with all parts. The value of this attribute is a reference to a term in a standard vocabulary; an indication about how to interpret this term (e.g. *globally, exterior, interior* or more specific, concerning each of the indicated properties); and a set of rules concerning the relations between parts in terms of set operations. Interpretation of the geometric attribute can be based on the existence of upper and lower bound for an attribute value. So existence of 'maxLength' indicates that 'length' should be interpreted as an external

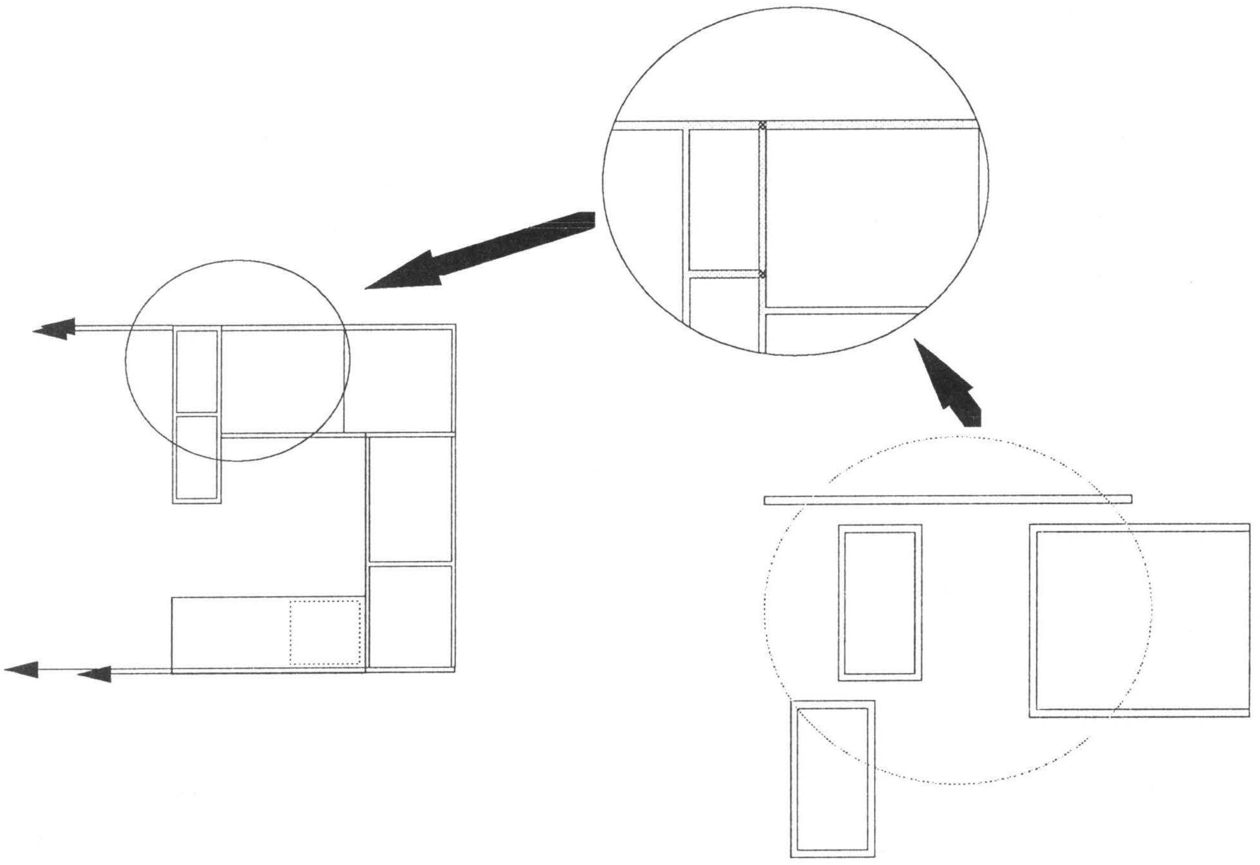


Fig. 13. Overlapping spaces

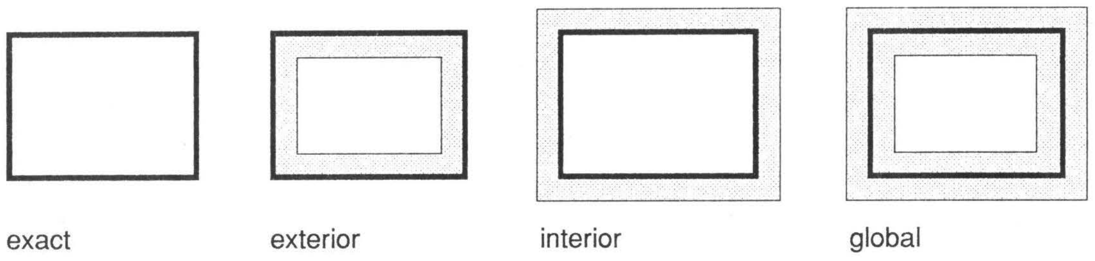


Fig. 14. Different interpretation of the geometric attribute value

measure, 'minLength' indicates that 'length' should be interpreted as an internal measure, while existence of both these attributes indicates that 'length' is a global value. Not that all these attributes, are properties of a space, and not of the object (sub-part) that should be located in that space. So the difference between 'maxLength' and 'minLength' does not indicate the 'thickness' of a wall (to be exact the thickness of two wall together) but a maximal thickness.

As a result and based on concrete values of attributes of parts, different lay outs may result from the application of the spatial reasoner. These results can be manipulated by adding rules or changing values, until a result is reached that is appropriate. The next task in the design process is to 'fill' all these spaces with actual material components which on their turn will provide a more detailed description of the spatial decomposition of the design object.

7. Evaluation and Conclusion

In this paper we have described a design support systems strategy based on spatial decomposition. This approach we think provides a sound basis for the representation of design object knowledge. The approach is based on two considerations about architectural design, mentioned in the first part of the paper: a commonly accepted approach of decomposing a design problem into several sub-problems, and a commonly accepted approach of re-using concrete design object (part) descriptions as templates for new designs. The paper discusses several considerations that lay behind the system's architecture and presents an example concerning architectural design. This example serves several purposes. It focuses on the approach of spatial decomposition. With it, the data structure is described as is generated in this process. It shows the use of a standard vocabulary (geometrical) in order to access an external application for geometric representation of the design object description, and for supporting an reasoning process called 'spatial reasoning'. It also introduces the concept of libraries with standard components (or features) and their functionality compared with, what was previously called *prototyping.*

Several subjects are not illustrated with the example. We consider these as topics for further research. Although the main focus of the paper aims towards the re-use of design knowledge, this topic is only discussed with the term *prototype application.* The example does not discuss this subject in order to prevent an explanation of a contents a a system's database. The twofold purpose of a formula expression in setting and checking an value remains a topics of further research. The current example simply assumes a maintainable model of a design object, but the exact consequences of using different of values for the same attributes has to be studied in more detail. The topic of using a standard vocabulary for accessing external applications is only globally identified. The example only uses a *geometrical vocabulary,* without going into details. This twofold shortcoming in elaboration of the example has several reasons. The main reason is that the purpose of a standard vocabulary is to access external application, and is therefore dependent from international standardization on access of external applications [10]. This discussion (international standardization of product model data) is been held and has not yet resulted into an internationally accepted standard. In the discussion about grouping of comparable part description, we mention procedures that force these (temporal) grouping. Although the purpose of this mechanism is explained, we did not elaborate this topic further. These procedures are based on the description of parts (e.g. their group identifier, their list of attributes names etc.). The default approach is to group all parts with the same group identifier and extract their common attribute (names). The syntax and semantics of more advanced procedures, that will simulate class hierarchies as are presented by 'Smalltalk-like' systems, remain a topic of further research.

With mentioning these topics it looks as if our system needs a considerable amount of further elaboration. Most of these topics however are covered by other products of the research project our system is developed in: the IIICAD project [8]. In this project a design language is made that will cover all the above topics and will be used on top of our system. The aim of our system was to provide a sound basis for the application of such a language.

References

1. Chr. Alexander, *Notes On the Synthesis of Form,* Harvard University Press, Cambridge, MA., 1964.
2. J.-P. Barthes and K. El Dahshan, ''Implementing Constraint Propagation in Mechanical CAD Systems,'' in *Intelligent CAD Systems II: Implementational Issues*, ed. P.J.W. ten Hagen and P.J. Veerkamp, pp. 217-227, Springer Verlag, Berlin, 1989.
3. A. Bijl, ''Strategies for CAD,'' in *Intelligent CAD Systems I: Theoretical and Methodological Aspects*, ed. P. ten Hagen and T. Tomiyama, pp. 2-19, Springer Verlag, Berlin, 1987.
4. H. Lieberman, ''Using Prototypical Objects to Implement Shared Behavior in Object Oriented Systems,'' *OOPSLA'86*, vol. 21, no. 11, pp. 214-223, ACM Sigplan, New York, 1986.
5. M. Minsky, ''A Framework for Representing Knowledge,'' *Readings in Knowledge Representation*, pp. 245-263, Morgan Kaufmann Publishers, Los Altos, 1985.
6. J. Rogier and F. Tolman, ''De formulering van een ontwerpparadigma voor bouwkundig ontwerpers,'' *Ontwikkelingen rond industriele automatisering*, pp. 3-395, SAMSOM, Alphen a/d Rijn, 1989.
7. T. Tomiyama and P.J.W. ten Hagen, ''Organization of Design Knowledge in an Intelligent CAD Environment,'' in *Expert Systems in Computer Aided Design*, ed. J. Gero, Amsterdam, 1987.
8. P.J. Veerkamp, V. Akman, P. Bernus, and P.J.W. ten Hagen, ''IDDL: A Language for Intelligent Interactive Integrated CAD Systems,'' in *Intelligent CAD Systems II: Implementational Issues*, ed. P.J.W. ten Hagen and P.J. Veerkamp, pp. 58-74, Springer Verlag, Berlin, 1989.
9. B. Veth, ''An Integrated Data Description Language for Coding Design Knowledge,'' in *Intelligent CAD Systems I: Theoretical and Methodological Aspects*, ed. P. ten Hagen and T. Tomiyama, pp. 295-313, Springer Verlag, Berlin, 1987.
10. P. Wilson and P.R. Kennicott, *STEP/PDES Testing Draft; St. Louis Edition 4.1.2,* ISO, St. Louis, 1987.