

A Feature-Based Description Technique for Design Processes: A Case Study

Henri Achten, Jos van Leeuwen
Eindhoven University of Technology
Department of Architecture, Building, and Planning
Eindhoven
The Netherlands

ABSTRACT

In order to develop appropriate tools for decision support in design processes, it is necessary to found them on an understanding of design. Analytical techniques of design processes that have a direct relationship with tool development can enhance design support systems development. The paper focuses on a design support system in the VR-DIS research program. The aim of this research program is to develop insight in the architectural design process and to establish design tools for architects working in Virtual Reality. The basic approach for data modelling in VR in this research is based on an extension of the Feature Based Modelling paradigm taken from design in mechanical engineering. The computer model of the design in the system is a Feature-based model. This paper describes design processes in terms of changes in the Feature-based model of the design. For this purpose, a case of a house design is used. Drawings in the conceptual design phase up to the preliminary design phase are studied. Each state of the drawings is described in terms of a Feature-model. Particular design actions such as creation of spaces, definition of architectural elements, and changes during the design process can be expressed in terms of changes in the Feature-model. Because of the use of Features, the changes can be formalised in the VR-DIS system. The description in terms of Features offers an analytical tool that leads to a functional brief for design support tools. The paper ends with a discussion of implications and future work.

1 DESIGN SUPPORT FOR ARCHITECTURAL DESIGN

In order to develop appropriate tools for decision support in design processes, it is necessary to found them on an understanding of design. Design, as a problem-solving process, involves activities of searching information, analysing, manipulating, and structuring information, generating new information, and evaluating and communicating information. These are not sequential activities, but take place in cycles (Markus 1969), (Maver 1970). (Lawson 1990) argues that designers tend to switch in an ad hoc manner between different activities, resulting in concurrency of activities with no predictable sequence. Design has a very dynamic nature which should be supported by design aid systems.

In the Design Systems group, a research program has been initiated called VR-DIS, meaning Design Information System and Distributed Interactive Simulation in Virtual Reality. The goals and projects of this program have been reported in (de Vries et al. 1997). The possible advantages of the VR-DIS program compared to

conventional CAD systems are discussed in (de Vries and Achten 1998). They propose that VR technology shows the best performance in the early design stage, using tools to create and evaluate (abstract) design models based on a three dimensional dynamic representation, and that it has the most potential in those areas where traditional CAAD has a poor performance. For this purpose, it is necessary to develop a brief which defines the required functionality for such a system. The performance specification should be formulated in terms that can easily be transformed computationally. Feature-Based Modelling, an approach currently under investigation and development in the group, provides such a formalism. If design processes can be described in terms of Features, then mapping between findings of the analysis and the functional brief can become quite straightforward. We have chosen to analyse concrete design cases in terms of Features. Before we discuss the case, some basic properties of Features will be introduced in the next section.

2. PRINCIPLES OF FEATURE-BASED MODELLING

The approach chosen in the VR-DIS program to model information in the research, is based on Feature-Based Modelling (FBM), a technique of modelling product information that originates from areas of Mechanical Engineering. The background and history of these techniques have been discussed and summarised in early papers by (Cunningham 1988), (Shah 1991; 1994), and (Bronsvort 1993; 1996). FBM has been reviewed for its relevance to architectural design in (Van Leeuwen et al. 1996; 1997). The main conclusions from the latter reviews are that concepts of FBM are very relevant for modelling architectural information in a broader sense. In the VR-DIS program, the following definition of the term Feature is employed (Van Leeuwen 1998a):

A Feature is a collection of high-level information, possibly emerging during design, defining a set of characteristics or concepts with a semantic meaning to a particular view in the life-cycle of a building.

This definition reflects four important aspects of Feature modelling in the architectural context: (i) a Feature has high-level information with semantic meaning. (ii) Both physical and non-physical characteristics and concepts can be defined. (iii) Definition and use of emerging Features during design is supported, and (iv) A Feature relates to a particular view in the life-cycle of a building.

(Van Leeuwen 1998a) provides a Feature modelling framework for the development of information modelling systems for support of architectural design. The framework defines how Features are to be modelled. It distinguishes between three layers of information definition (figure 1):

The top layer describes the classes of Feature types that can be defined in the system on the basis of the Abstract FeatureType. These are: Simple FeatureType, Enumeration FeatureType, Geometric FeatureType, Complex FeatureType, Constraint FeatureType, and Handler FeatureType.

The middle layer consists of Generic Feature Types (which are basic in the Building & Construction industry) and Specific Feature Types (which are specific for the design project at hand) that are by structure defined in the Meta-layer. There is no principle distinction between Generic Feature Types and Specific Feature Types.

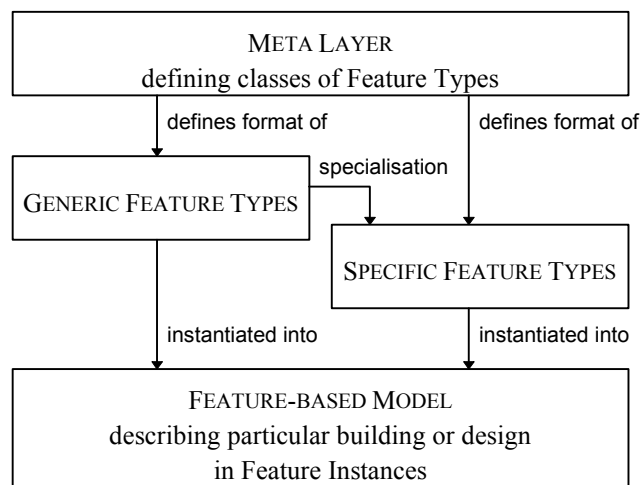
Both types are instantiated into Feature Instances, which are represented in the third layer of the framework. Feature Instances contain actual building information forming Feature models.

Feature Types can be seen as conceptual pieces of knowledge, whereas Feature Instances represent the actual state a concept has in the design. Take for example, a Feature Type called 'UnitPrice,' defined to represent information about cost. It is of the class Simple FeatureType, having a basetype 'real,' and a unit 'guilder per unit.' A Feature Instance of the Feature Type 'UnitPrice' could be 'RentalPrice_office1,' with a value of 600 guilders a month. In a design there can be many instances of the Feature Type 'UnitPrice.'

Feature models are rather flexible in that they support alteration of specific Feature Types during the design process. They are extensible through support of defining new Feature Types and Feature Instances. Also, it is possible to define relations between Feature Instances. This dynamic character of Feature modelling seems to be in accord with the dynamic nature of design.

For Feature modelling, a tool has been developed by J. Jessurun to define Feature Types and Feature Instances and to manipulate them (see figure 2; figure 3). In time, Feature manipulation of the design is envisioned to be an integral part of the

Figure 1: **Infrastructure of Feature-Based Modelling with three Layers of Abstraction (Van Leeuwen and Wagter, 1997)**



VR environment. Work by (Coomans 1997) is aimed towards this development. For now, the Feature tool is used for Feature definition. Features can be represented in a graphical way (Van Leeuwen 1998a) or in a textual way, the so-called Feature Type Definition Language (Van Leeuwen 1998b). In this paper, we will be using the textual representation.

3. METHODOLOGY OF THE CASE STUDY

The VR-DIS system requires functionality in terms of Feature modelling, as this kind of data structuring has been chosen for the system. The goal of the case is to analyse a design process in terms of Features, in order to acquire statements for the system functional brief. The first case of this research is also used to set up and critique the methodology of the analysis.

3.1 Case Material

The case is a small house design, which is followed from start to end. It is an actual design executed by an architects office. In the architects office, AutoCAD is used from the very start of the design work. Drawings during the design process are made

Figure 2: Feature Tool Showing Feature Type Definition Section

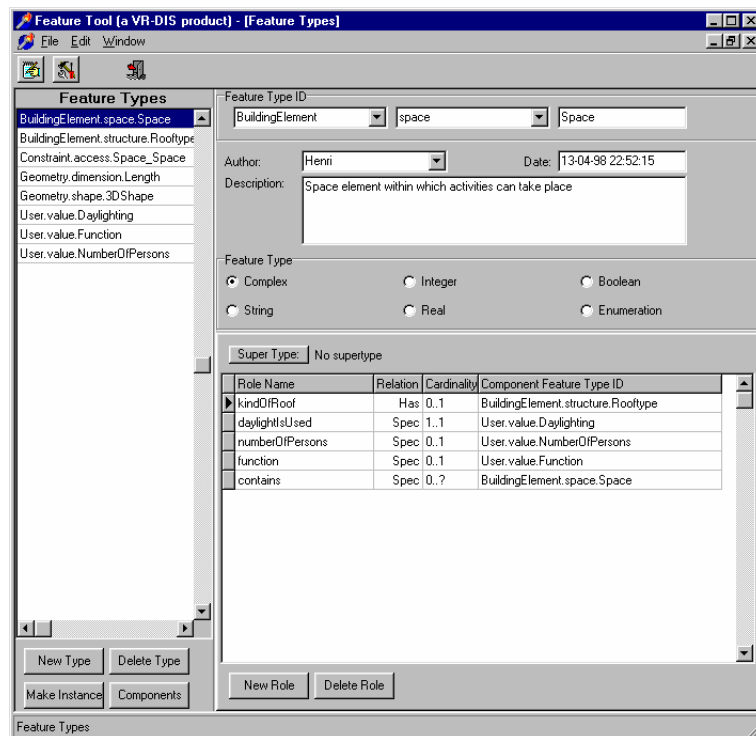
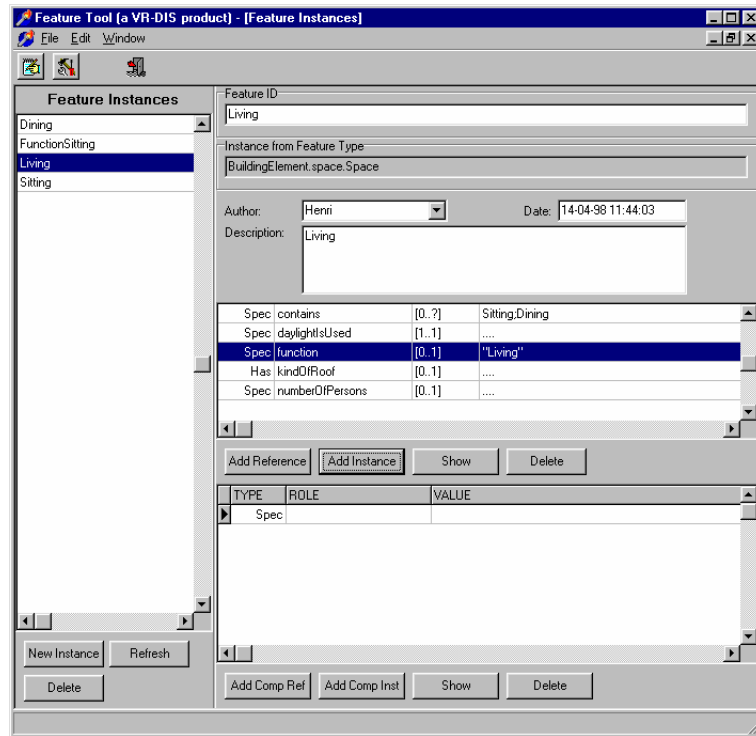


Figure 3: Feature Tool Showing Feature Instance Section



as new copies rather than changing or revising old drawings. In this way, key phases of the design process are available for analysis. The material available for analysis consists of: one print A4 with the brief termed Brief, four prints A4 termed Compilation 1 through 4, and four prints A2 termed Print 1 through 4. Compilation 1-4 have drawings made during the design process. Prints 1-4 have the finalised design.

Compilation 1 through 4 contains 30 drawings made during the design process. Each single drawing is taken as a step in the design process for which a Feature model can be established. The transition from one step to the next therefore, represents the design decisions taken from that phase to the next.

We define as a phase: one single drawing. Notation: phase n , $n \in \mathbb{N}$.

We define as a step: the transition from phase $n \rightarrow$ phase $n+1$.

In order to include the brief as included in document Brief in the phase definition, we define the brief as phase 0.

3.2 Establishing The Design Process

The sequence of drawings is established on the work by (Achten 1997), and is consulted with the project architect for verification and required changes. We found

that the first proposed sequence did not require many changes to represent the actual sequence of the design process. In particular, two major sequences 10-18 and 21-30 had to be switched without internal changes since they were misunderstood in their time order. The restructuring of the design process can be avoided in new cases when each drawing is time-coded when established.

In the reconstructed sequence, each drawing was described on the basis of its constituent elements. In an interview, the project architect was asked to verify the text and to make changes if appropriate.

From the work, it appeared that analysis through the sequence of drawings and description per drawing is a productive way to extract design information. We have to note here, that verification in such a way after the design process can lead to post-rationalisation, as termed by (Darke 1979), and to unwarranted agreement if the description seems plausible enough. These aspects can be met when such comments are made during the actual process. However, the main purpose of the research is focused on the design functionality of VR-DIS rather than producing a precise description of the design process.

3.3 Feature-Modelling Of The Case

The analysis is carried out as follows:

1. In phase n that is subject of analysis, identify the elements that have to be defined as Features. These elements are concepts used in the design, and typically are nouns in the text.
2. If the elements are new, define complex Feature Types and simple Feature Types accordingly. An element is defined as complex Feature Type when it cannot be defined as a simple Feature Type (string, integer, real, Boolean, or enumeration). If the element in question already has been defined as a Feature Type, record which changes have taken place and determine whether these should be included in the Type. If such changes do not alter existing instance definitions, then changing the Feature Type definition does not cause inconsistencies with the current Feature model. However, since the Feature Type should hold more general knowledge, it requires a careful balance what to include in the definition.
3. In the case of new elements, define Feature Instances based on the Feature Types above. In the case of existing Feature Types, record changes in Feature Instances.

3.4 Example Of Feature-Modelling In The Case

The brief of phase 0 provides a lexicon of design elements that play a role in the building design. In the example we will focus on the concept of space. Spaces such as hall, toilet, and wardrobe are elements of the lexicon.

3.4.1 Feature Type Definition In The Case

Rather than defining a Feature Type for each kind of space (such as a Feature Type ‘hall’ for the hall, ‘toilet’ for the toilet, etc.), we will define a Feature Type space of which the various spaces in the brief are instances.

The text in the brief notes the following aspects of space: function (such as bedroom and bathroom), contained elements (such as stair and toilet), visual relationship (such as kitchen closed with respect to living), access relationship (such as doors to garden and bathroom), daylighting (such as daylighting in kitchen), adjacency (such as scullery between garage, kitchen and bathroom), roof type (such as no glass roof), and number of persons (such as guest room for two persons).

Determining which aspects are to be included in the definition of the Feature Type Space and which aspects are to be defined in other Feature Types is not straightforward. If the aspect concerns only the space itself, and does not refer to other elements, then it can be included in the type. Following this rule, constraint-like relations such as visual relationship are better defined outside the Feature Type. Function, contained elements, daylighting, roof type, and number of persons are within the particular space and therefore are included in the type definition.

The Feature Type space is defined accordingly, and results in the following (see figure 2 for the representation in the Feature tool):

```
complex BuildingElement.space.Space {
  TypeDate {4/13/98}
  TypeAuthor {Henri}
  TypeDescr {"Space element within which activities can take place"}
  Spec BuildingElement.space.Space contains[0..?];
  Spec User.value.Daylighting daylightIsUsed[1..1];
  Spec User.value.Function function;
  Has BuildingElement.structure.Rooftype kindOfRoof;
  Spec User.value.NumberOfPersons numberOfPersons;
}
```

The first line identifies the Feature Type class, which is ‘complex’ in this case. The text ‘BuildingElement.space.Space’ is the Feature identification (ID) which is a unique description of the Feature. The ID is constructed as follows: Library.section.Feature (van Leeuwen 1998b). The next three lines stating ‘TypeDate,’ ‘TypeAuthor,’ and ‘TypeDescr,’ are inherited from the abstract class FeatureType. Each Feature class therefore, has these properties. In the following, we will abbreviate by leaving out these properties. The next five lines define the aspects

of the Feature Type space as identified above. They are contained elements, daylighting, function, rooftype, and number of persons respectively. Each line has a three part structure: relation, FeatureID, and role. Four of the relations are specifications since they further define the space. The rooftype relation is a decomposition since it is a part of the space. The FeatureIDs refer to Feature Types that are related to the Feature Type Space. Their definitions follow next. The role describes the role of the Feature in the definition. The numbers in brackets (for example '[0..?]') indicates so-called cardinality, which indicates how many instances of this role are allowed or required in a Feature Instance.

In order to complete the Feature Type definition of space in this phase, the Feature Types User.value.Daylighting (we will leave out the Library.section part for abbreviation where appropriate, as well as the TypeDate and TypeAuthor part), Function, Rooftype, and NumberOfPersons must be defined:

```
string BuildingElement.structure.Rooftype {
}
boolean User.value.Daylighting {
  TypeDefault {True}
}
string User.value.Function {
}
integer User.value.NumberOfPersons {
  TypeDefault {1}
  TypeUnit {"person"}
}
```

The other aspects we have identified above, will include the Feature Type Space in their definition. We provide here an example of the Feature Type Constraint.access.Space_Space which defines the access constraint between any two spaces:

```
complex Constraint.access.Space_Space {
  Spec BuildingElement.space.Space access[2..2];
}
```


3.4.2 Feature Instance Definition In The Case

The Feature Types define the concepts used in the design. They have to be instantiated into the particular Features used in the design. In the case of Feature Type Space, this means making instances of spaces such as hall, living, kitchen, veranda, etc. We will provide an example of the Feature Instance living (see figure 3 for the representation in the Feature tool).

```
BuildingElement.space.Space Living = {
    contains[1] = Dining
    contains[0] = Sitting
    function = FunctionLiving
}
BuildingElement.space.Space Dining = {
    function = FunctionDining
}
BuildingElement.space.Space Sitting = {
    function = FunctionSitting
}
User.value.Function FunctionDining = {
    Value {"Dining"}
}
User.value.Function FunctionLiving = {
    Value {"Living"}
}
User.value.Function FunctionSitting = {
    Value {"Sitting"}
}
```

In subsequent phases the Feature model is established on the basis of the previous one, which enables to track changes during the design process.

4. THE CASE: HOUSE DESIGN

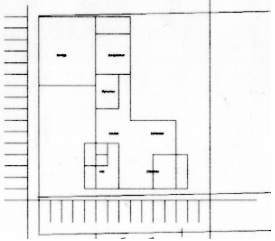
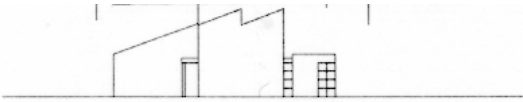
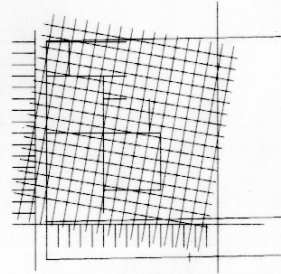
The case starts with the brief as noted on the document Brief. It is a one page document made by the architect-in-chief of the office consulting with the client. The brief is as follows:

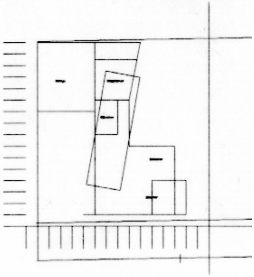
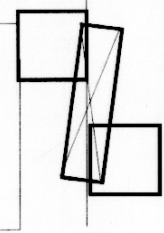
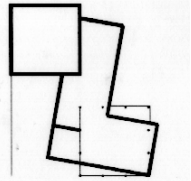
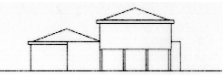
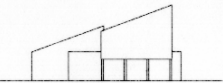
- The hall has stairs, toilet, and wardrobe. The stairs should not be in the living room, have no platform, and should not be straight.
- The living room has a separate sitting and dining.
- The kitchen is closed with respect to the living. It has a relation with the garden. No table and chairs. Daylighting.
- Veranda sitting. Relation with living. No glass roof.

- Scullery between garage, kitchen and bedroom.
- Bedroom with daylighting, and bathroom (containing bathtub, shower, two wash basins, and toilet pot). Bedroom has doors to garden and bathroom.
- Storey has guest room for two persons. Shower, toilet, and wash basin.
- Fireplace in living. Stone floor covering.

Table 1 presents the first eight drawings of the case, including brief textual statements on representation and basic design decisions.

Table 1: **Eight Drawings of the Case Study**

 <p>Figure 4: Phase 1</p>	<p>Phase 1: The functional brief is translated into spaces positioned in the site, giving an indication of the required space and the organisation of the ground floor plan. The main mass is located in the north west, leaving space for the garden. The module used is 1.20 m. It is a ‘sketch-module’ used by the architect.</p>
 <p>Figure 5: Phase 2</p>	<p>Phase 2: Most spaces are placed on the ground floor. The facade study shows the building as it is realized in the vertical plane, including window position and roof shape.</p>
 <p>Figure 6: Phase 3</p>	<p>Phase 3: The architect uses a grid from the start, establishing space dimensions. The use of a second grid is based on the location of the double sized garage in the upper left corner. Access to the garage in a straight line would make this element visually too dominant which is why part of the design is rotated on the basis of a second grid. The first grid = grid 1. The second grid = grid 2.</p>

 <p>Figure 7: Phase 4</p>	<p>Phase 4: The center part of the building is oriented towards grid 2. Spaces are placed according to the new grids to see how they work out.</p>
 <p>Figure 8: Phase 5</p>	<p>Phase 5: In this phase, the basic variant is introduced. The building mass consists of two squares between which a center part is defined, which is rotated relative to the squares. This basic variant is used throughout the following design process. Upper left square = square A. Bottom right square = square B. The center piece = center piece.</p>
 <p>Figure 9: Phase 6</p>	<p>Phase 6: Sub variant of phase 5. Square B is lifted from the ground floor (dots indicating columns). The center part has acquired an L-shape lying against square A. The L-shape is dominant with respect to the other squares A and B. The bottom-side of the L is aligned with bottom right corner of square B. In that point, lines of grid 1 and grid 2 intersect. Square A is kept intact. The elevated floor of square B is elaborated in phase 7 and 8.</p>
 <p>Figure 10: Phase 7</p>	<p>Phase 7: Facade study of phase 6, showing particular roof shape for this design.</p>
 <p>Figure 11: Phase 8</p>	<p>Phase 8: Facade study of phase 6, showing particular roof shape for this design.</p>

5. FEATURE-BASED DESCRIPTION OF THE CASE

We will not provide an exhaustive (and lengthy) list of the Feature model, but aim to demonstrate the basic approach by discussing the first two steps.

5.1 Phase 0

In this phase, the brief has to be translated into a Feature Model. We will discuss elements and constraints respectively here. With respect to the elements, section 3.4.1 and 3.4.2 show what particular examples of Feature Types and Feature Instances look like (Feature Type Space and Feature Instance Living).

5.1.1 Elements

The Feature Types and Feature Instances in this phase are shown in table 2.

Table 2: Feature Types and Feature Instances of Elements in Phase 0

Feature Type (super type)	Feature Instance
Space	Hall, Toilet, Wardrobe, Living, Sitting, Dining, Kitchen, Veranda, Scullery, Garage, Bedroom, Bathroom, Shower, GuestRoom
Door (ElementInWall)	DoorBathroom_Bedroom, DoorBathroom_Garden
Floor	FloorLiving
Material	MaterialFloorcovering, MaterialRoofVeranda
Roof	RoofVeranda
Stair	Stair
Garden	Garden
Chair (Furniture)	Chair
Table (Furniture)	Table
Fireplace (Heating)	FireplaceLiving
Bathtub (Sanitary)	BathtubBathroom
ToiletPot (Sanitary)	ToiletPotHall, ToiletPotGuestroom, ToiletPotBathroom
WashBasin (Sanitary)	WashBasin1_Bathroom, WashBasin2_Bathroom, WashBasinGuestroom
Daylighting	DaylightingBedroom, DaylightingKitchen, DaylightingVeranda
Function	FunctionBedroom, FunctionHall, FunctionDining, FunctionKitchen, FunctionSitting, FunctionLiving
Storey	StoreyGroundFloor, StoreyFirstFloor
NumberOfPersons	NumberOfPersonsGuestroom

The need for unique instance FeatureIDs leads to the concatenation of descriptions, such as ‘ToiletPotGuestroom.’ Obviously, these could be any encoding, but the current naming style gives clarity at this moment. The Feature Types and Feature Instances define the nouns used in the brief. The super types enable to state for example, that a space has ‘sanitary’ without being specific what kind of sanitary exactly is meant.

5.1.2 Constraints

In phase 0, a number of statements are made about relations between elements that can be considered as constraints: ‘stair should not be in the living room,’ ‘kitchen is closed with respect to the living,’ ‘kitchen relation with the garden,’ ‘no table and chairs in the kitchen,’ ‘scullery between garage, kitchen, and bedroom,’ ‘bedroom has doors to garden en bathroom,’ and ‘fireplace in living.’

Some of these constraints concern spatial relations. These can be expressed as: A _adjacent_ B, A _in_ B, A _above_ B, and their logical opposites A _NOTadjacent_ B, A _NOTin_ B, A _NOTabove_ B, with A and B Feature Types.

Other constraints concern access from one element to another. These access constraints can be expressed as: A _access_ B and the opposite A _NOTaccess_ B. Since this relation is reciprocal, only one _access_ constraint has to be defined. The Feature Type definition of equal Feature Types in an _access_ constraint can be established by using cardinality. In cases where there is no reciprocity, or where there are unequal Feature Types in the relation, cardinality can not be used, since the theory of Feature Modelling does not support strict orders in the Feature Types included in the cardinality list.

The third type of constraint is expressed in the statement ‘kitchen is closed with respect to the living,’ meaning that there is no visual relation between the kitchen and living space. The constraints A _visual_ B and the opposite A _NOTvisual_ B are reciprocal, and can be defined in the same manner as the _access_ relation.

The ‘fireplace in living’ constraint can be expressed in two ways: (i) as a Fireplace_in_Living instance of a Heating_in_Space Feature Type, or (ii) by establishing an association-relation in the Feature Instance Living with the instance Fireplace. We have chosen here for the first option, since more heating elements are bound to occur in more spaces, which makes it reasonable to define a constraint Feature Type for this relation.

Table 3: Constraint Feature Types in Phase 0

Spatial constraint Feature Type	Spatial constraint Feature Instance
Space_adjacent_Space	Kitchen_adjacent_Living, Veranda_adjacent_Living, Scullery_adjacent_Garage, Scullery_adjacent_Bedroom, Scullery_adjacent_Kitchen
Space_adjacent_Garden	Kitchen_adjacent_Garden, Bedroom_adjacent_Garden
Storey_above_Storey	Storey1_above_Storey0
Furniture_NOTin_Space	Furniture_NOTin_Kitchen
Heating_in_Space	Fireplace_in_Living
Stair_NOTin_space	Stair_NOTin_Living
Access constraint Feature Types	Access constraint Feature Instances
Space_Space	Bedroom_access_Bathroom
Space_Garden	Bedroom_access_Garden, Kitchen_access_Garden
Visual constraint Feature Types	Visual constraint Feature Instances
Space_NOTvisual_Space	Kitchen_NOTvisual_Living

At this point, phase 0 has been expressed in terms of Features. The textual Feature Type and Feature Instance list is included in Appendix I after section 8.

5.2 Phase 1

This phase shows a key transition between the functional brief and a first composition of spaces (see figure 4 in table 1). Significant changes with respect to phase 0 consist of assigning shape and dimensions to spaces, and locating them in the site by means of a grid.

5.2.1 Shape And Dimensions Assigned To Spaces

Phase 1 shows a number of spaces that have been given dimensions by virtue of drawing shapes in a grid. The new notion of shape can be implemented as part of the existing Feature Type Space or by defining a new Feature Type for shape, which is associated with the Feature Type Space. When considering phase 1 only, the first option would suffice. However, the notion of shape is very basic in architectural design, and many other kinds of shapes may occur later in the design, each with their own intrinsic properties. Therefore, we have chosen to define a supertype 2DShape of which Rectangle is a subtype. Length, Point, and Coordinate also are new Feature Types that have not yet been defined, and that are required to complete 2DShape:

```

complex Geometry.shape.2DShape {
  TypeDescr {"General shape definition with point of reference"}
  Spec Geometry.topology.Point referencePoint[1..1];
}
complex Geometry.shape.Rectangle(Geometry.shape.2DShape) {
  TypeDescr {"Rectangular shape with dimensions and reference point"}
  Spec Geometry.dimension.Length length[1..1];
  Spec Geometry.dimension.Length width[1..1];
}
real Geometry.dimension.Length {
  TypeDescr {"Measure along straight line in m"}
  TypeDefault {1}
  TypeUnit {"m"}
}
complex Geometry.topology.Point {
  TypeDescr {"Point in orthogonal axial system with x, y, z values"}
  Spec Geometry.topology.Coordinate xcoordinate[1..1];
  Spec Geometry.topology.Coordinate ycoordinate[1..1];
  Spec Geometry.topology.Coordinate zcoordinate[1..1];
}
real Geometry.topology.Coordinate {
  TypeDescr {"Coordinated along an axis in an axial coordination system"}
  TypeDefault {0}
  TypeUnit {"m"}
}

```

The above Feature Types are typical examples of Feature Types that would appear in a standardised library of Generic Feature Types. The Feature Type 2DShape is associated with the existing Feature Type Space (bold line shows addition to old Feature Type):

```

complex BuildingElement.space.Space {
  TypeDescr {"Space element within which activities can take place"}
  Spec BuildingElement.space.Space contains[0..?];
  Spec User.value.Daylighting daylightIsUsed[1..1];
  Spec User.value.Function function;
  Has BuildingElement.structure.Rooftype kindOfRoof;
  Spec User.value.NumberOfPersons numberOfPersons;
  Assoc Geometry.shape.2DShape shape;
}

```

5.2.2 Grid And Location In The Site

The shapes are drawn in a grid which is used for coordination. In the interview, the architect also stated that the grid provided a sketch-module of 1.20 m. Although the grid can be defined by stating its module, it also needs a point of origin relative to which coordinates are established (this also accommodates the use of multiple grids). Both the origin of the grid and the position of elements in the grid require a set of coordinates. We define therefore, on the instance level, a Feature Instance Origin (instance of Geometry.topology.Point) relative to which measures can be taken and grids positioned. The left-bottom corner of Grid_1 is placed on the Origin.

The positive x-axis is oriented horizontally and to the right of the Origin. The positive y-axis is oriented vertically and above the Origin, as is customary in architectural design. For the Feature Type Rectangle, the reference point is defined as the most left-bottom corner of the rectangle, width and length being measured in the orientation of the positive x and y axis.

These Feature Types and Feature Instances suffice to establish the state defined in phase 1. Kitchen, for example, is changed because of the additions to the Feature Type Space of which it is an instance, and the definition of its location and dimensions in the associated Rectangle. The Kitchen has dimensions 3.60 m x 3.60 m, and is located on (6.0, 6.0, 0):

```
BuildingElement.space.Space Kitchen = {
  Descr {"Kitchen"}
  daylightIsUsed = DaylightingKitchen;
  function = FunctionKitchen;
  shape = Rectangle_Kitchen;
}
Geometry.shape.Rectangle Rectangle_Kitchen = {
  Descr {"Rectangular shape for kitchen"}
  length = Length_Kitchen;
  referencePoint = ReferencePoint_Kitchen;
  width = Width_Kitchen;
}
Geometry.dimension.Length Length_Kitchen = {
  Value {3,6}
}
Geometry.dimension.Length Width_Kitchen = {
  Value {3,6}
}
Geometry.topology.Coordinate Coordinate_X_Kitchen = {
  Value {6}
}
Geometry.topology.Coordinate Coordinate_Y_Kitchen = {
  Value {6}
}
Geometry.topology.Coordinate Coordinate_Z_Kitchen = {
  Date {7-05-98}
  Author {Henri}
  Value {0}
}
```

Note that in phase 1 not all spaces mentioned in phase 0 are present, and that there are four spaces that have not been assigned a function name by the architect. The other instances of spaces that are included in phase 1 (garage, bedroom, scullery, hall, dining, veranda, and space_0 through space_4) are changed in the same manner as the Kitchen.

6. TOWARDS A FUNCTIONAL BRIEF FOR VR-DIS

The discussion of phase 0 and phase 1 has shown that it is possible to describe a design process in terms of Features. The VR-DIS system aims to combine Feature-based modelling with an innovative approach of an interface for design and design information. In a 3D immersive environment, the designer should be able to manipulate spatial elements of the design, as well as the underlying Feature model (Coomans and Timmermans 1997; 1998; Coomans and Achten 1998 explore the interface properties of such a system).

In order to establish a prototype of the VR-DIS system, we aim for a system that can accommodate design activities such as studied in phases 0 through 7 of the case. The system should be able to:

1. Aid in translating the functional brief into Feature Types and Feature Instances.
2. Check if all requested rooms and spaces fit on the site.
3. Check with the principal if the requirement list is complete.
4. Give a first impression of possible composition of rooms and spaces.
5. Give a first impression of the layout of the plan.
6. Give a first impression of the facades, including windows, doors, and materials.
7. Give a first impression of possible roof shapes.

A more detailed account of the design system is provided by (de Vries and Jessurun 1998).

7. DISCUSSION

The paper discusses a first approach towards describing design processes in terms of Feature Based Modelling as defined by (van Leeuwen 1998a). This approach has a number of significant properties that are relevant to researching design processes and establishing a functional brief for the VR-DIS project.

- Description by means of a Feature model is much more precise and comprehensive than a standardised textual description.
- Feature-Based modelling as defined by (van Leeuwen 1998a; 1998b) provides a flexible and extensible language with which the changing concepts of a design can be captured.
- The flexibility of definition however, also means that there are quite a number of ways in which to establish the Feature model during the design process. Often, the most economical definition becomes apparent only later in the process.
- A considerable number of Feature Types in the case can be generalised as Types that will be used time and again in architectural design. Defining these so-called 'generic Feature Types' in each separate design process does not make much sense. In principle, the Building & Construction Industry has to provide a standard set of Generic Feature Types.
- Working with the Feature Tool has brought to light that issues of control in the structure of the Feature Model will become important when the Feature Model grows in complexity and during the course of design. These aspects have not yet been addressed in the Feature Tool and will also play a role in the Feature view of the VR-DIS system (Coomans and Timmermans 1998).
- Consistency will be an important issue in the modelling system. This includes consistency, for instance, between the definition of various Feature Types. It also includes consistency between the Feature Type definitions and the instances of these types. The latter is especially important when type definitions are amended and the existing instances are to be maintained. These kinds of consistency can be implemented using the class of Handler Feature Types.

Future work consists of extending the case study to the remaining phases of the case (phase 8 through phase 31), and to analyse other cases in order to establish the Feature-Based description technique.

8. ACKNOWLEDGEMENTS

This research is performed within the VR-DIS research program, and builds on the Ph.D. research by van Leeuwen under supervision of Prof.ir. H. Wagter and Prof.dr. R.M. Oxman. Thanks are due to the architectural firm Architektenburo Jan Merks B.V., Veldhoven and architect ir. L. Stiphout for their cooperation in the case study. Also, we would like to thank dr.ir. Bauke de Vries and ir. Joran Jessurun for contributions to the Feature description and the production of the Feature tool.

APPENDIX I

Feature Type list of phase 0 (abbreviated Feature notation):

```
complex BuildingElement.space.Space {
  Spec BuildingElement.space.Space contains[0..?]
  Spec User.value.Daylighting daylightIsUsed[1..1]
  Spec User.value.Function function
  Has BuildingElement.structure.Rooftype kindOfRoof
  Spec User.value.NumberOfPersons numberOfPersons
}
complex BuildingElement.space.Storey {
  Spec BuildingElement.space.Space spacesInStorey[0..?]
}
complex BuildingElement.structure.Door(BuildingElement.structure.ElementInWall) {
}
complex BuildingElement.structure.ElementInWall {
}
complex BuildingElement.structure.Floor {
  Spec BuildingElement.structure.Material floorMaterial
}
string BuildingElement.structure.Material {
}
complex BuildingElement.structure.Roof {
  Spec BuildingElement.structure.Material roofMaterial
}
string BuildingElement.structure.Stair {
}
complex Constraint.above.Storey_above_storey {
  Spec BuildingElement.space.Storey above[1..1]
  Spec BuildingElement.space.Storey under[1..1]
}
complex Constraint.access.Space_Garden {
  Spec Ground.plot.Garden accessFrom[1..1]
  Spec BuildingElement.space.Space accessTo[1..1]
}
complex Constraint.access.Space_Space {
  Spec BuildingElement.space.Space access[2..2]
}
complex Constraint.adjacent.Space_adjacent_Space {
  Spec BuildingElement.space.Space adjacent[2..2]
}
complex Constraint.in.Furniture_NOTin_Space {
  Spec BuildingElement.space.Space doesNotHave[1..1]
  Spec Infill.furniture.Furniture isNotIn[0..?]
}
```

```

complex Constraint.in.Heating_in_Space {
    Spec BuildingElement.space.Space HasGot[1..1]
    Spec Infill.heating.Heating IsIn[1..1]
}
complex Constraint.in.Stair_NOTin_space {
    Spec BuildingElement.space.Space doesNotHave[1..1]
    Spec BuildingElement.structure.Stair isNotIn[1..1]
}
complex Constraint.adjacent.Space_adjacent_Garden {
    Spec BuildingElement.space.Space AdjacentToGarden[1..1];
    Spec Ground.plot.Garden AdjacentToSpace[1..1];
}
complex Constraint.adjacent.Space_adjacent_Space {
    Spec BuildingElement.space.Space adjacent[2..2];
}
complex Constraint.visual.Space_NOTvisual_Space {
    Spec BuildingElement.space.Space NOTvisual[2..2]
}
complex Ground.plot.Garden {
}
complex Infill.furniture.Chair(Infill.furniture.Furniture) {
}
complex Infill.furniture.Furniture {
    Spec User.value.NumberOfPersons NumberOfUsers
}
complex Infill.furniture.Table(Infill.furniture.Furniture) {
}
complex Infill.heating.Fireplace(Infill.heating.Heating) {
}
complex Infill.heating.Heating {
}
complex Infill.sanitary.Bathtub(Infill.sanitary.Sanitary) {
}
complex Infill.sanitary.Sanitary {
}
complex Infill.sanitary.ToiletPot(Infill.sanitary.Sanitary) {
}
complex Infill.sanitary.WashBasin(Infill.sanitary.Sanitary) {
}
boolean User.value.Daylighting {
    TypeDefault {True}
}
string User.value.Function {
}
integer User.value.NumberOfPersons {
    TypeDefault {1}
    TypeUnit {"person"}
}

```

Feature Instance list of phase 0 (abbreviated Feature notation):

```

BuildingElement.space.Space Bathroom = {
    contains[0] = ShowerBathroom
    Spec BathtubBathroom HasBathtub
    Spec ToiletPotBathroom HasToiletPot
    Spec WashBasin1_Bathroom HasWashBasin1
    Spec WashBasin2_Bathroom HasWashBasin2
}
BuildingElement.space.Space Bedroom = {
    daylightIsUsed[0] = DaylightingBedroom
    function = FunctionBedroom
}
BuildingElement.space.Space Dining = {
    function = FunctionDining
}

```

```

BuildingElement.space.Space Garage = {
}
BuildingElement.space.Space Guestroom = {
  contains[0] = ShowerGuestroom
  numberOfPersons = NumberOfPersonsGuestroom
  Spec WashBasinGuestroom HasWashBasin
}
BuildingElement.space.Space Hall = {
  contains[1] = WardrobeHall
  contains[0] = ToiletHall
  function = FunctionHall
  Spec Stair HalHasStair
}
BuildingElement.space.Space Kitchen = {
  daylightIsUsed[0] = DaylightingKitchen
  function = FunctionKitchen
}
BuildingElement.space.Space Living = {
  contains[1] = Dining
  contains[0] = Sitting
  function = FunctionLiving
}
BuildingElement.space.Space Scullery = {
}
BuildingElement.space.Space ShowerBathroom = {
}
BuildingElement.space.Space ShowerGuestroom = {
}
BuildingElement.space.Space Sitting = {
  function = FunctionSitting
}
BuildingElement.space.Space ToiletGuestroom = {
  Spec ToiletPotGuestroom HasToiletPot
}
BuildingElement.space.Space ToiletHall = {
  Spec ToiletPotHall HasToiletPot
}
BuildingElement.space.Space Veranda = {
  daylightIsUsed[0] = DaylightingVeranda
  function = FunctionSitting
}
BuildingElement.space.Space WardrobeHall = {
}
BuildingElement.space.Storey StoreyFirstFloor = {
  spacesInStorey[0] = Guestroom
}
BuildingElement.space.Storey StoreyGroundfloor = {
  spacesInStorey[7] = Hall
  spacesInStorey[6] = Living
  spacesInStorey[5] = Kitchen
  spacesInStorey[4] = Scullery
  spacesInStorey[3] = Bedroom
  spacesInStorey[2] = Bathroom
  spacesInStorey[1] = Garage
  spacesInStorey[0] = Veranda
}
BuildingElement.structure.Door DoorBathroom_Bedroom = {
}
BuildingElement.structure.Door DoorBathroom_Garden = {
}
BuildingElement.structure.Floor FloorLiving = {
  floorMaterial = MaterialFloorcovering
}
BuildingElement.structure.Material MaterialFloorcovering = {
  Value {"Stone"}
}

```

```

BuildingElement.structure.Material MaterialRoofVeranda = {
    Value {"No glass"}
}
BuildingElement.structure.Roof RoofVeranda = {
    roofMaterial = MaterialRoofVeranda
}
BuildingElement.structure.Stair Stair = {
    Value {"Not straight, no platform"}
}
Constraint.above.Storey_above_storey Storey1_above_storey0 = {
    above[0] = StoreyFirstFloor
    under[0] = StoreyGroundfloor
}
Constraint.access.Space_Garden Bedroom_access_Garden = {
    accessFrom[0] = Garden
    accessTo[0] = Bedroom
}
Constraint.access.Space_Garden Kitchen_access_Garden = {
    accessFrom[0] = Garden
    accessTo[0] = Kitchen
}
Constraint.access.Space_Space Bedroom_access_Bathroom = {
    access[1] = Bathroom
    access[0] = Bedroom
}
Constraint.in.Furniture_NOTin_Space Furniture_NOTin_Kitchen = {
    doesNotHave[0] = Kitchen
    isNotIn[1] = Chair
    isNotIn[0] = Table
}
Constraint.in.Heating_in_Space Fireplace_in_Living = {
    HasGot[0] = Living
    IsIn[0] = FireplaceLiving
}
Constraint.in.Stair_NOTin_space Stair_NOTin_Living = {
    doesNotHave[0] = Living
    isNotIn[0] = Stair
}
Constraint.visual.Space_NOTvisual_Space Kitchen_NOTvisual_Living = {
    NOTvisual[1] = Kitchen
    NOTvisual[0] = Living
}
Ground.plot.Garden Garden = {
}
Infill.furniture.Chair Chair = {
}
Infill.furniture.Table Table = {
}
Infill.heating.Fireplace FireplaceLiving = {
}
Infill.sanitary.Bathtub BathtubBathroom = {
}
Infill.sanitary.ToiletPot ToiletPotBathroom = {
}
Infill.sanitary.ToiletPot ToiletPotGuestroom = {
}
Infill.sanitary.ToiletPot ToiletPotHall = {
}
Infill.sanitary.WashBasin WashBasin1_Bathroom = {
}
Infill.sanitary.WashBasin WashBasin2_Bathroom = {
}
Infill.sanitary.WashBasin WashBasinGuestroom = {
}
Constraint.adjacent.Space_adjacent_Garden Bedroom_adjacent_Garden = {
    AdjacentToGarden = Bedroom;
}

```

```

    AdjacentToSpace = Garden;
}
Constraint.adjacent.Space_adjacent_Garden Kitchen_adjacent_Garden = {
    AdjacentToGarden = Kitchen;
    AdjacentToSpace = Garden;
}
Constraint.adjacent.Space_adjacent_Space Kitchen_adjacent_Living = {
    adjacent[1] = Living;
    adjacent[0] = Kitchen;
}
Constraint.adjacent.Space_adjacent_Space Scullery_adjacent_Bedroom = {
    adjacent[1] = Bedroom;
    adjacent[0] = Scullery;
}
Constraint.adjacent.Space_adjacent_Space Scullery_adjacent_Garage = {
    adjacent[1] = Garage;
    adjacent[0] = Scullery;
}
Constraint.adjacent.Space_adjacent_Space Scullery_adjacent_Kitchen = {
    adjacent[1] = Kitchen;
    adjacent[0] = Scullery;
}
Constraint.adjacent.Space_adjacent_Space Veranda_adjacent_Living = {
    adjacent[1] = Living;
    adjacent[0] = Veranda;
}
User.value.Daylighting DaylightingBedroom = {
    Value {True}
}
User.value.Daylighting DaylightingKitchen = {
    Value {True}
}
User.value.Daylighting DaylightingVeranda = {
    Value {True}
}
User.value.Function FunctionBedroom = {
    Value {"Sleeping"}
}
User.value.Function FunctionDining = {
    Value {"Dining"}
}
User.value.Function FunctionHall = {
    Value {"Entrance and circulation"}
}
User.value.Function FunctionKitchen = {
    Value {"Cooking and food preparation"}
}
User.value.Function FunctionLiving = {
    Value {"Living"}
}
User.value.Function FunctionSitting = {
    Value {"Sitting"}
}
User.value.NumberOfPersons NumberOfPersonsGuestroom = {
    Value {2}
}
}

```

9. REFERENCES

- Achten, H.H. (1997) Generic Representations - An Approach For Modelling Procedural And Declarative Knowledge Of Building Types In Architectural Design, *Bouwstenen* 46, Technische Universiteit Eindhoven, Eindhoven.
- Achten, H.H. and Oxman, R.M. and Bax, M.F.Th. (1998) Typological knowledge acquisition through a schema of generic representations, *Proceedings of Artificial Intelligence in Design '98*, Lisbon, Portugal, July 20-23, 1998 (forthcoming).
- Bronsvoort, W.F. and Jansen, F.W. (1993) Feature modelling and conversion, Key concepts to concurrent engineering, *Computers in Industry*, 21(1), pp. 61-86.
- Bronsvoort, W.F. and Dohmen, M. and Bidarra, R. and Van Holland, W. and De Kraker, K.J. (1996) Feature modelling for concurrent engineering, *Proceedings of the International Symposium on Tools and Methods for Concurrent Engineering '96*, Technical University of Budapest, Budapest, pp.46-55.
- Coomans, M.K.D and Timmermans, H.J.P. (1997) Towards a Taxonomy of Virtual Reality User Interfaces, *Proceedings of the International Conference on Information Visualisation (IV97)*, London, Great Britain, 27-29 August, 1997.
- Coomans, M.K.D. and Achten, H.H. (1998) Mixed Task Domain Representation in VR-DIS, *Proceedings of the 3rd Asia-Pacific Conference on Computer Human Interaction, APCHI98*, July 15-17, Hayama-machi, Kanagawa, Japan, 1998 (forthcoming).
- Coomans, M.K.D. and Timmermans, H.J.P. (1998) A VR User Interface for Design by Features, *Proceedings of the 4th Conference on Design and Decision Support Systems in Architecture and Urban Planning*, Maastricht, the Netherlands, July 26-29, 1998, (forthcoming).
- Cunningham, J.J., and J.R. Dixon (1988) Designing with Features: the origin of Features, *Proceedings ASME Computers in Engineering Conference, San Francisco*.
- Darke, J. (1979) The Primary Generator and Design Process, in Cross, N. (ed.) *Developments in Design Methodology*, John Wiley & Son Ltd.
- Lawson, B. (1990) *How designers think, the design process demystified, 2nd edition*, Butterworth Architecture, London.
- Leeuwen, J.P. van and Wagter, H. and Oxman, R.M. (1996) Information Modelling for Design Support. *Proceedings of the 3rd Design and Decision Support Systems in Architecture and Urban Planning Conference*, August 18-21, Spa, Belgium.

- Leeuwen, J.P. van and Wagter, H. (1997). Architectural Design-by-Features. *CAAD futures 1997*. Kluwer Academic Publishers, Dordrecht.
- Leeuwen, J.P. van and Wagter, H. (1998a) A Features framework for architectural information, *Proceedings of Artificial Intelligence in Design '98*, Lisbon, Portugal, July 20-23, 1998 (forthcoming).
- Leeuwen, J.P. van (1998b) *FBAIM and F(t)DL. Feature Based Architectural Information Model and Feature (type) Definition Language*, internal report.
- Markus, Th.A. (1969) The role of building performance measurement and appraisal in design method, in Broadbent and Ward (eds.), *Design methods in architecture*, Lund Humphries, London.
- Maver, Th.W. (1970) Appraisal in the building design process, in Moore (ed.), *Emerging methods in environmental design and planning*, MIT Press, Cambridge, Massachusetts.
- Shah, J.J. and Mäntylä, M. and Nau, D.S. (eds.) (1994) *Advances in Feature based manufacturing*. Elsevier Science, Amsterdam.
- Shah, J.J (1991) Assessment of Features technology, *Computer-Aided Design*, 23(5), pp. 331-343.
- Vries, B. de and Leeuwen, J.P. van and Achten, H.H. (1997) Design studio of the future, *Proceedings of the CIB W78 Conference "Information Technology Support for Construction Reengineering,"* Cairns, Australia, pp. 139-148.
- Vries, B. de and Achten, H.H. (1998) What offers Virtual Reality to the designer?, *Workshop 'The art of Design' at the Third Biennial World Conference on Integrated Design & Process Technology*, (forthcoming).
- Vries, B. de and Jessurun, A.J. (1998) An experimental design system for the very early design stage, *Proceedings of the 4th Conference on Design and Decision Support Systems in Architecture and Urban Planning*, Maastricht, the Netherlands, July 26-29, 1998 (forthcoming).