This paper is published as: van Leeuwen, Jos and Bauke de Vries (2000). Capturing design knowledge in formal concept definitions. DDSS 2000. Proceedings of the 5th Conference on Design and Decision Support Systems in Architecture and Urban Planning, Nijkerk, The Netherlands, August 22-25, 2000.

Capturing design knowledge in formal concept definitions

Jos van Leeuwen
Bauke de Vries
Eindhoven University of Technology
Department of Building and Architecture
Design Systems group
Eindhoven
The Netherlands

ABSTRACT

For support of creativity in architectural design, design systems must be provided with information models that are flexible enough to follow the dynamic way of designers in handling early design information. This paper discusses a framework for information modelling using Features that answers this need. One of the characteristics of the framework is that designers can define the formalisation of their own design concepts into types of Features. The definition of these Feature Types can be done in different manners; three scenarios for this procedure are presented and discussed.

1. SUPPORTING CREATIVITY IN DESIGN

1.1 Dynamic nature of Design

Architectural design can generally be regarded as a process of problem solving. These problems, however, are normally ill-defined, or 'wicked' as (Cross 1984) characterises them. These wicked design problems require a very dynamic behaviour from designers, not only in searching for design solutions, but also in searching for the design problem (Coyne at al. 1991). Tasks such as analysis, synthesis, and evaluation do not occur in neat cycles, but designers tend to switch in a rather ad hoc manner between the different stages and tasks in design and often perform these tasks concurrently (Lawson 1990).

Clearly, information is not treated as static data during such a dynamic design process. Content and structure of design information is invariably subject to change, which places a significant requirement on the development of computer support for design tasks: formal information models for design must be as flexible and dynamic as the design process itself. They must evolve, as design evolves. The evolution of design information models has been subject of research by (Eastman et al. 1995) and at the Design Systems group at Eindhoven University of Technology (van Leeuwen et al. 1995, 1997, 1998, 1999; van Leeuwen 1999). Similar issues are addressed also in the work by (Ekholm and Fridqvist 1997, 1999) and (Hendricx 2000).

The following conclusions can be drawn when considering the dynamic nature of design:

1. Design is a process of problem-solving and often concerns problems that are initially not well-structured;
2. Information related to design problems and solutions is dealt with in different ways, related to the approach of solving the design problem. Design involves creativity through combination of these approaches:
   a) Selection of an existing solution to a similar design problem involves matching information related to the problem and the existing solutions.
   b) Creating a new solution to the design problem involves generating new information that defines the solution to the problem.
   c) Combining existing information in order to find new relations or structures in concepts and ideas that lead to design solutions involves analysis, re-interpretation, and re-structuring of existing design information.
   d) Altering the design-problem in order to find a suitable solution means analysis, re-interpretation, and re-structuring of the information related to the problem, possibly even adding to, or dropping parts of the design problem.
3. Activities in design do not take place in a predictable order, the information dealt with in design activities cannot be foreseen: the content and structure of required information or generated information cannot be presupposed;
4. Individual designers, as well as the sector of design and the Building & Construction industry as a whole, are under constant development, with new knowledge, concepts, techniques, methods, products, materials, and styles emerging. Conceptual information models must evolve along with this development, in order to accurately represent the changing domain of design and B&C.

The above conclusions lead to the statement of new requirements on information models that are to support the dynamic nature of design. These requirements are denoted by the terms *extensibility* and *flexibility*, both ensuring the possibility for an information model to evolve along with the development of a design.

1.2 **Modelling with Features**

The Feature-Based Modelling technology (FBM) has been developed initially in the area of mechanical engineering (Shah and Mäntylä 1995). As in the approach of Product Modelling (PM), FBM has started from the objective to generate semantically rich models of engineering data. Yet, the approach followed in FBM is different. PM approaches have developed conceptual models that represent the data structures to be used by their applications. These conceptual models largely aim at the later stages of design of the product, and are used to communicate the as-designed information between the various participants. Historically, the starting-point in FBM has been formed by geometry models, from which it was attempted to recognise the semantics of design. These semantics were then modelled using so-called Features. However,

because much of the design information available during design cannot be recognised from geometric models, the design-by-Features approach was developed, where the semantically rich Features formed the primitives in building up the geometry. Combinations of both design-by-Features and Feature recognition joined the advantages of both approaches (DeMartino et al. 1994; Ovtcharova and Vieira 1995).

The result of this historical development from Feature recognition to design-by-Features, and eventually to the combination of both, has been that models consisting of Features are not, as in most PM approaches, predefined in large data structures. Features are defined as relatively autonomous entities of information that are given a position and relationships in the model only at design-time, not at the time of development of conceptual models. Also, the collection of Features available to designers is not assumed to be complete: designers can define and add their own Feature types to their collection of design tools. These characteristics of Feature-Based Modelling are very appealing to the dynamic architectural designer who is struggling with ill-defined design problems at the early stages of design.

Research on a Feature-Based Modelling approach for architectural design has led to the development of a theoretical framework with the following characteristics (van Leeuwen and Wagter 1997; van Leeuwen 1999).

1. Features are used to represent the semantics of a building design;
2. Features are the formal definition of characteristics or concepts of design;
3. Features are applied to multiple levels of abstraction of modelling the design (as opposed to the original FBM area, where Features are used only to describe the level of parts);
4. Features can be Generic Features, shared by the domain of architectural design, or Specific Features, which are defined for a particular view, e.g. a particular design style;
5. Types of Features can be defined by designers as the need to formalise a design concept arises;
6. Features form interrelated structures in a Feature model, using the relationships that are defined at the level of Feature Types, or by adding occasional relationships at the instance level.
7. Libraries of Feature Types represent bodies of domain knowledge. These libraries can also include instantiated data, mixed with the typological definitions.

Of these characteristics of the Features framework, issue number 5, Types of Features can be defined by designers, is discussed in detail in the remainder of this paper.

2. STRATEGIES FOR FEATURE TYPE DEFINITION

2.1 **Identification of a concept**

Defining a Feature Type follows the decision to formalise a design concept. Therefore the first problem to address in Feature Type definition is: how to recognise and identify a concept? Two different points of view from which to approach this problem are discussed. The first point of view discusses how concepts can be acquired from sources of design knowledge. The second point of view presents the most common approaches in OO Analysis to classify a given knowledge domain. Both points of view must be considered in the processes of identifying concepts for the formalisation of design knowledge.

2.1.1 Design domain knowledge and vocabulary

The first point of view in the quest for concepts is taken from the body of knowledge in the domain of design. This knowledge, particularly in the complex discipline of architectural design, is not always readily available or easily accessible. Certain concepts in this body of knowledge are scientifically defined, such as the SI units (meter, second, Kelvin, Volt, etc.). These often are rather elementary concepts, which is to say that, in terms of information structure, they do not bare much complexity. Other, perhaps more complex, concepts may be defined in a less exact manner, but still be well conceived, such as industrial products of which all characteristics are known and available from manufacturers. The terminology used for these products and their characteristics forms the basis for defining the Feature Types that are to represent this kind of concept. A third kind of concept is perhaps the most important in design, especially in early stages. These concepts form the core of architectural design theory and methods. They represent elements of design that can be either concrete or abstract, tangible or intangible, exact or indeterminate. For this kind of concept, the vocabulary of the design domain may be a suitable starting-point for their formal definition into Feature Types. This vocabulary, in architecture, is not formally defined either, but many terms have traditional meanings that are generally accepted.

The first consideration in the process of identifying a concept should therefore be whether a term exists that covers the potential concept. Terms are normally used to indicate the names of, e.g., systems, structures, products, materials, functions, organisational units, et cetera. An analysis of the way the term is used should be projected onto the concept being identified and reveal if the term actually represents that concept or not. If an existing, accepted term cannot be found, there are four possible consequences:

- The potential concept needs some adjustment to fit a term that is reasonably close to describing the concept;
- The potential concept covers a combination of multiple terms;
- The potential concept introduces a new term in the design vocabulary;
- Any combination of the three options above.

Whether or not new terminology should be defined involves a trade-off between aspects such as:

- Acceptability of the concept in the design discipline. This may be an important issue when the concept serves, e.g., purposes of standardisation, regulation, or information exchange.
- Desired or allowed level of ambiguity of the defined concept. Because new terminology, as opposed to traditional terminology, is not naturally known, its introduction may result in various interpretations of the term, which have to be verified against the concept's definition. Any ambiguity in the formal definition of the concept will then allow variance in the interpretation of the term.
- Completeness and exactness of the definition. As a result of the previous aspect, the completeness and exactness of the definition of the concept cannot rely on knowledge inherently related to traditionally known terminology.
- Uniqueness of the concept in relation to existing vocabulary. Using new terminology allows a concept to be defined distinctly and independently from implicit meanings related to existing terminology. This can be a prerequisite when the uniqueness of the concept is to be stressed or when distinction from other concepts is necessary.

Closely related to design domain knowledge are the areas of design methodology and design theory. Design methodology, according to (Roozenburg and Eekels 1995), is the science that studies the structure, methods, and rules of design. Design methodologies are either developed while focusing on the design process as a whole, or intended for specific domains or phases in design. An example of the latter, given by Roozenburg and Eekels, is the morphological method, which relates characteristics and functions of a design with the variant components for that design in an array containing all conceivable solutions. For the identification of design concepts, it is interesting to look at the subjects used in specific design methods, especially those subjects that form an intrinsic part of the method.

A recently developed methodology for architectural design is presented as Generic Representations by (Achten 1997). This methodology involves an approach to the identification of design content in architectural graphic representations. Its hypothesis is that graphic representations made during the design process imply the design decisions that are made. The research shows how it is possible to extract such design decisions from the graphic representations, by inferring the declarative knowledge embedded in these representations. The methodology proposed by Achten involves using generic representations, and the design knowledge acquired from them, as a model for procedural decision making in design.

Many design methods, like the example given above, develop design aids, such as archetypes, design patterns, proportional or other measuring systems, rules for design schemata for instance for floor plans and elevations, and so on. These tools can be regarded as the design concepts that are applied in the context of the design situation at hand, using established procedures from the design method. The definitions of

these concepts are not always clear and explicit but may involve implicit knowledge about the usage and meaning of the concepts themselves and of the procedures for using them in design. The formalisation of this kind of concept into a Feature Type requires that all relevant knowledge be made explicit, which may involve formalisation of other concepts and knowledge about concepts that have not been identified explicitly before. Classification strategies will help to identify these.

2.1.2 OOA strategies for classification

The term classification in Object Oriented Analysis refers to the task of the software engineer to identify classes of objects in the domain for which software is to be developed. These classes then form the backbone for the design of procedures and data storage of that software. According to (Sowa 1984) there have only been three general approaches to classification:

1. Classical categorisation.
   The criteria for sameness of objects is formed by their properties: objects that have one or more properties in common belong to a category.
2. Conceptual clustering.
   First, the conceptual descriptions of classes are formulated, then objects are classified according to these classes using a 'best fit' method.
3. Prototype theory, or classification by example.
   The class is not defined conceptually, but by means of an example: a prototype. Objects are member of the class only if they sufficiently resemble the prototype.

In practice of Object Oriented Analysis, these approaches are combined and/or followed sequentially. Classification forms the main starting-point not only for the identification but also for the design of object classes. It supports the determination of structures of classes and of the structure of data and behaviour of these classes. As such, these approaches to classification are valuable also during the definition of Feature Types.

2.2 **Decisions in Feature Type definition**

Definition of a Feature Type is a procedure that is very similar to the definition of object classes in OO approaches for which many strategies and checklists have been described, e.g. (Booch 1994). Aspects that need to be considered when defining a Feature Type are the following:

- *Bottom-up versus top-down*
  A top-down approach allows the designer to represent the logical hierarchies that are found in the domain of architecture, whereas a bottom-up approach stimulates the re-usage of existing Feature Types.
- *Typical versus non-typical*
  Is the information to be formalised typical for the concept, or does merely it concern a characteristic of a particular instance of that concept? This has to do

with the reusability of the concept: when too much information is included in the Feature Type, then the reusability of the concept will be less: perhaps some less common characteristics should not be defined as part the Feature Type, but modelled as instance level relationships for particular Feature Instances only.

- *Wide structures versus deep structures*
  (Booch 1994, p. 140) discusses the subject of how to choose the inheritance relationships between classes of objects: deep inheritance trees tend to have classes that are less interdependent, but may not exploit all commonality; wide inheritance trees result in smaller individual classes, re-using other classes, but their complexity will be harder to understand. A similar problem exists with other relationships between classes, such as decomposition.
- *Presentation versus representation*
  This concerns the distinction between how a concept is presented to the user, e.g. on screen, and what actually comprises the concept. The latter kind of information must be modelled and stored and is used to generate the data for presentation.
- *Choice of relationship: specialisation, decomposition, association, or specification*
  For the definition of the relationships between Feature Types, these four relationships are available in the framework. Specialisation results in the definition of sub-types inheriting from super-types. The other three kinds of relationships are given a role name in the definition of a Feature Type by the designer.
- *Redundancy, completeness, and consistency*
  Although modelling a design information structure should aim at minimal redundancy and maximal completeness and consistency, it must be realised that the optimal configuration of information does also rely on aspects like reusability and practicability. Especially these two aspects will often justify certain levels of redundancy to exist in a collection of Feature Types. The pursuit for completeness should always be considered in the context and purpose for which a Feature Type is to be used and in relation with the amount of information that is likely to be available at the time of modelling or that designers are willing to provide. From the point of view of information management, consistency should always be pursued, yet in creative design, inconsistency may, to a certain degree, be acceptable. Moreover, the option to be inconsistent in dealing with information during design is often considered an important factor in creative processes.

2.3 **Scenarios for Feature Type definition**

Three distinct situations are recognised in which Feature Type definition may be initiated.

1. Feature Type definition from scratch.
2. Feature Type definition from a prototype.
3. Feature Type recognition.

2.3.1 Feature Type definition from scratch

The first situation in which Feature Type definition is initiated, is when a designer (or e.g. an organisation for standardisation) decides to formalise a concept that has not necessarily been modelled in terms of Features before. The formalisation of such a concept is started from scratch. For this approach, a procedure is described in the next few pages, that guides a designer through the various decisions to be made when defining a new Feature Type. This procedure leads to a selection of the appropriate class of Feature Type and assesses the definition of all its attributes, possibly resulting in the definition of other Feature Types or the instantiation of Feature Instances.

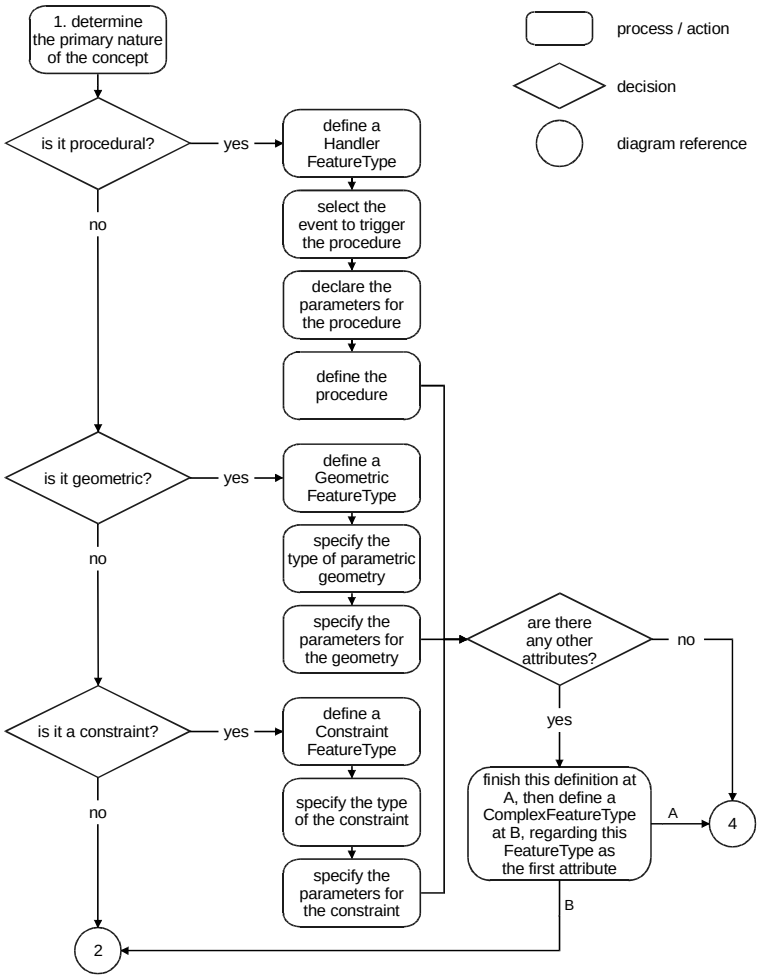

Figure 1. Diagram 1 of the procedure for Feature Type definition

The figures 1 to 4 show the procedure for formalising a concept and defining a Feature Type. This procedure assumes that the concept has been identified by the designer. It comprises four diagrams that guide the designer through a number of decisions regarding the contents of the concept. Diagram 1 starts with the

determination of the primary nature of the concept. It distinguishes procedural, geometric, and constraint concepts from all others. This distinction may lead to the definition of a Handler Feature Type, a Geometric Feature Type, or a Constraint Feature Type respectively. If none of these apply, one is to proceed with diagram 2 of the procedure.

A Handler Feature Type requires the selection of the event that is to trigger the procedure defined by the handler. The parameters for this procedure need to be declared, meaning that the types are specified of the Features that can be passed as parameters to this procedure. The procedure itself must be defined, using one of the procedural languages made available by the design system.

In case of a geometric nature of the concept, a Geometric Feature Type is defined. This type requires selection of the parametric geometry that it represents. The available kinds of parametric geometry depends on the geometry modelling engine that is integrated with the design system. The selected geometry provides the types of the parameters that must be provided by instances of the Geometric Feature Type, these parameters are given from within the context of the geometry modeller. The Geometric Feature Type further declares these parameters by providing the types of Features that can be passed as parameters.

For Constraint Feature Types, the type of constraint needs to be indicated, which depends on the availability of constraint solvers in the design system. Once the constraint type is selected, the parameters required by this constraint are known and the Constraint Feature Type further declares these as the types of Features that can be passed.

After the definition of any of the above Feature Types, there may be remaining aspects of the concept that have not yet been taken into account as parameters. If this is the case, the defined Feature Type is itself contained in a larger Feature Type, a Complex Feature Type, which must be defined next. First, the definition of the current Feature Type is finished by proceeding with diagram 4 of the procedure. After that, the definition of the containing Complex Feature Type can be started, which takes the Feature Type that has just been defined as its first attribute.

Diagram 2 shows how to proceed when the concept does not lead to the definition of either Handler, Geometric, or Constraint Feature Type. First, all the attributes of the concept that represent data to be stored by Features of this type are listed. For each of these data-attributes, it is considered whether or not the attribute is relevant to the majority of occurrences of the concept. This is not necessarily a very clear decision, as the term ‘majority’ already indicates, more so because the possible occurrences of the concept may not come into view clearly at this time. Nevertheless, it should be questioned if the particular attribute really contributes to the concept’s significance, or if it is relevant only for the one occurrence of the concept that the designer has in mind. If the latter is the case, the attribute should not be defined a part of the Feature Type, but rather be modelled as a relationship at the level of the Feature Instances.

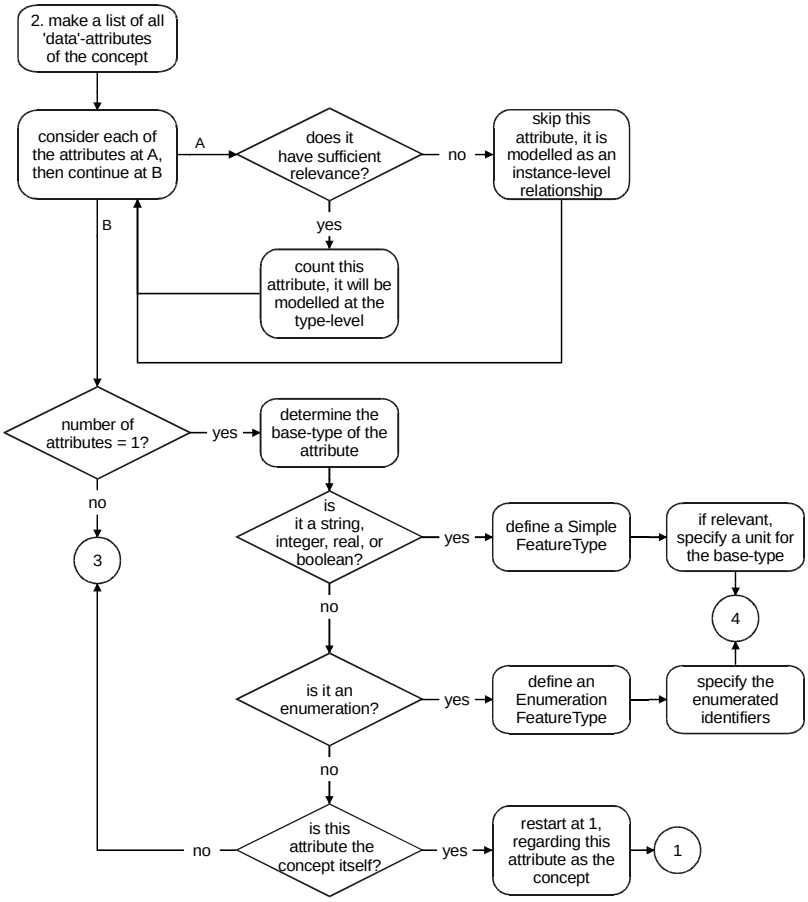

Figure 2. Diagram 2 of the procedure for Feature Type definition

The number of data-attributes that are considered relevant for the Feature Type's definition are counted. If this number is exactly one, then the base-type of this attribute must be determined. For attributes that represent a string, integer, real, or Boolean value, a Simple Feature Type is defined. For attributes that represent an identifier chosen from a given list of identifiers, an Enumeration Feature Type is defined. If neither of the above is the case, then the attribute itself represents a complex information structure, which must be represented by another Feature Type. Because this attribute is also the only data-attribute of the concept, it might in fact be that this attributes represents the concept itself. This is particularly true if no behaviour attributes are to be defined for this concept (see diagram 4), meaning that the concept exhibits no other characteristics than those represented by this attribute. Therefore, promoting this attribute to be regarded as the concept itself should be considered. If this is found to be the case, the procedure should be restarted, taking the notion of this attribute as the notion of the concept. Else, the procedure is continued at diagram 3, where the attribute will be the first and only attribute of a Complex Feature Type.

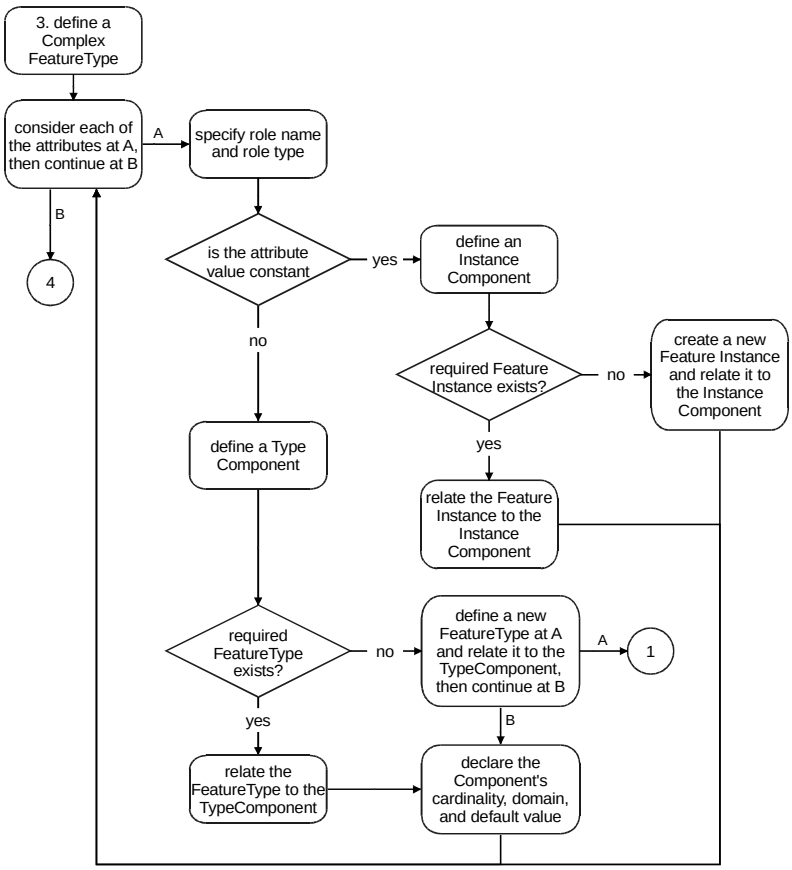

Figure 3. Diagram 3 of the procedure for Feature Type definition

If the number of relevant data-attributes of the concept is greater than one, then a Complex Feature Type needs to be defined and the procedure is continued at diagram 3. This is also the case if no attributes are found relevant. This may appear to be an odd case, formalising a concept that has no characteristics, yet the mere existence of a Feature Type with a given name may be sufficient to represent a particular design concept at certain stages in the development of a design or design theory. Perhaps later the content of the concept will become more clear and attributes will be added to the Feature Type that represents it. Also, behaviour attributes are yet to be dealt with, at diagram 4, which may give more meaning to the Feature Type being defined. Concepts with no data-attributes at all are modelled as Complex Feature Type that have no Components.

If the procedure leads to diagram 3, this means that the concept must be represented by a Complex Feature Type. All data-attributes of the concept are to be defined as components of the Complex Feature Type, which are given a role name and role type (decomposition, association, or specification). For every relevant attribute of the concept the question must be answered whether or not the attribute has a constant value for all occurrences of this concept. If this is true, the Complex

Feature Type will define an Instance Component, which is formed by a relationship to a Feature Instance. Possibly, this Feature Instance needs to be created.

For attributes with a value that varies for the different occurrences of the concept, a Type Component is to be defined for the Complex Feature Type. This is a relationship to another Feature Type, which, during instantiation, results in one or more relationships to Feature Instances. Possibly, the related Feature Type does not yet exist and must be defined in a new procedure started at diagram 1. For Type Components, the cardinality, domain, and default value must be specified.

After a component has been defined for each data-attribute of the concept, the procedure is continued at diagram 4 with the definition of the concept's behaviour.

The fourth and last diagram of the procedure for defining a Feature Type adds behaviour to the type's definition by means of adding event handlers. First a list is made of all the behaviour-attributes of the concept being formalised. As with the data-attributes in diagram 2, all those attributes are eliminated that bear relevance only to certain instances of the concept and are not significant to the intrinsic notion that the concept represents.

For each of the remaining behaviour-attributes, the event is specified that will trigger the particular behaviour, the event handler, of the instances of this Feature Type. Next, it must be determined if the parameters that are to be assigned to the event handler will be assigned in a similar manner for all instances of the Feature Type, or if each instance will assign the parameters in their own particular manner. If the way of assigning parameters does not vary per instance, the parameter assignment can be done at the level of the Feature Type, which results in relating a Handler Feature Instance, containing the parameter assignment, to the event handler. This Handler Feature Instance may need to be created in case it does not already exist.

In the case of per instance assignment of parameters, only the Handler Feature Type can be specified for the event handler. Again, this Handler Feature Type may need to be defined if it does not already exist. The actual parameter assignment is done during instantiation, when an instance of the specified Handler Feature Type is created.

After all the behaviour-attributes have been formalised into event handlers, the definition of the Feature Type can be concluded by specifying the domain for the instances of the type, and a default value. The kind of content of both domain and default value depends on the class of Feature Type that has been defined.

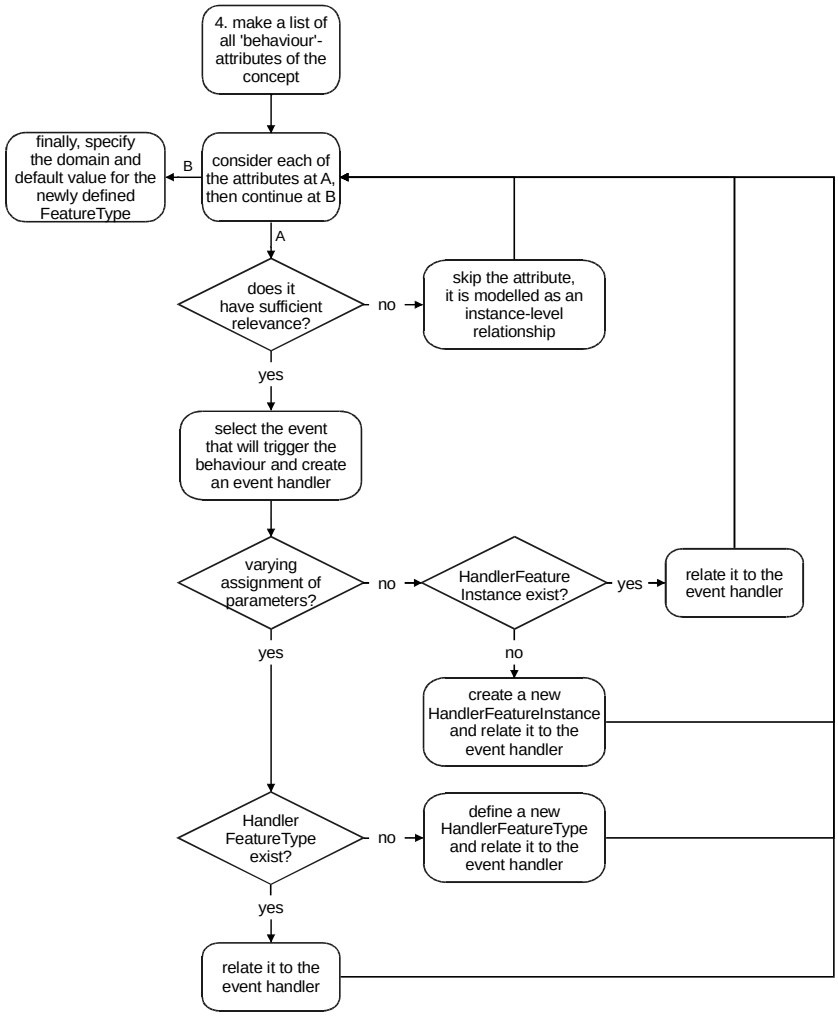

Figure 4. Diagram 4 of the procedure for Feature Type definition

2.3.2 Feature Type definition from a prototype

The second scenario, Feature Type definition from a prototype, is the situation where a particular pattern of information, modelled in structures of Feature Instances, is acknowledged by the designer as representing a particular concept that will recur during the same or other design cases. The definition of a new Feature Type can then be done on the basis of the structure of Feature Instances that was modelled using relationships at the instance level. The 'prototype' that the designer has built by creating this structure of Features is turned into a new complex Feature Type that defines the relationships as its components. From this point on, this scenario follows a procedure similar to that of the first scenario, as described above.

2.3.3 Feature Type recognition

The procedure of turning a prototype Feature structure into a Feature Type definition could also be initiated by a design system. Using pattern matching algorithms, a design system can search for recurring patterns of Features and relationships at the instance level. Once such a recurring pattern has been found, it may be proposed to the designer as a concept of design.

An important issue in the original area of Feature model in Mechanical Engineering is Feature recognition. In that area, Feature recognition has the meaning of recognising Features from a given geometric model. The geometry is analysed and searched for patterns of geometry that match the definition of known Feature Types. Once such a match is found, the geometry can be replaced by an instance of the found Feature Type. In this manner the geometric model, which is poor in semantics, is converted to a Feature model that provides all the additional information necessary to, for instance, manufacture the geometry of the designed product with the available machinery.

Architectural design systems may well benefit from a similar approach to designing elements of a building. Providing the designer with generic geometric modelling tools, the created geometry may be analysed and interpreted as a structure of Features that semantically enrich the geometry with detailed architectural information. This approach is, of course, limited to those Features that can actually be discriminated on the basis of their geometric representation. Using inference methods, these geometrically recognised Feature structures may eventually be enhanced with additional Features that are defined as relationships to the geometric Features. For example, once a wall Feature has been recognised from the geometry created by the designer, Features such as material, construction method, cost, maximum load, etc. may be inferred from the existence of the wall Feature and added to the list of relationships of that Feature.

Another kind of Feature recognition that can assist the designer in building up a consistent and semantically rich design model, is to try and recognise patterns of Features not from a geometric model but from the Feature model as it is being created. Here, it is not the bare geometry that is matched to definitions of Feature Types. Instead, in the Feature model the instance-level relationships between Feature Instances are analysed and compared to the structures of Feature Types in available libraries. In this manner, a given constellation of Features that are interrelated by the designer during modelling at the instance-level, can be replaced by an instance of a Feature Type that has been found to define the same relationships at the type-level. This facility of the design system supports the designer in creating consistent models and adding knowledge to the model that is implied by the design actions. The degree of similarity between found Feature structures and the relationships in a particular Feature Type should possibly be variant, allowing the designer some freedom in using accustomed terminology and including cases that look similar to defined Feature Types. Mainly the latter may well appear to be a stimulant to the designer, since the system is now encouraging the creativity of the designer and helping the development of the design as it proceeds.

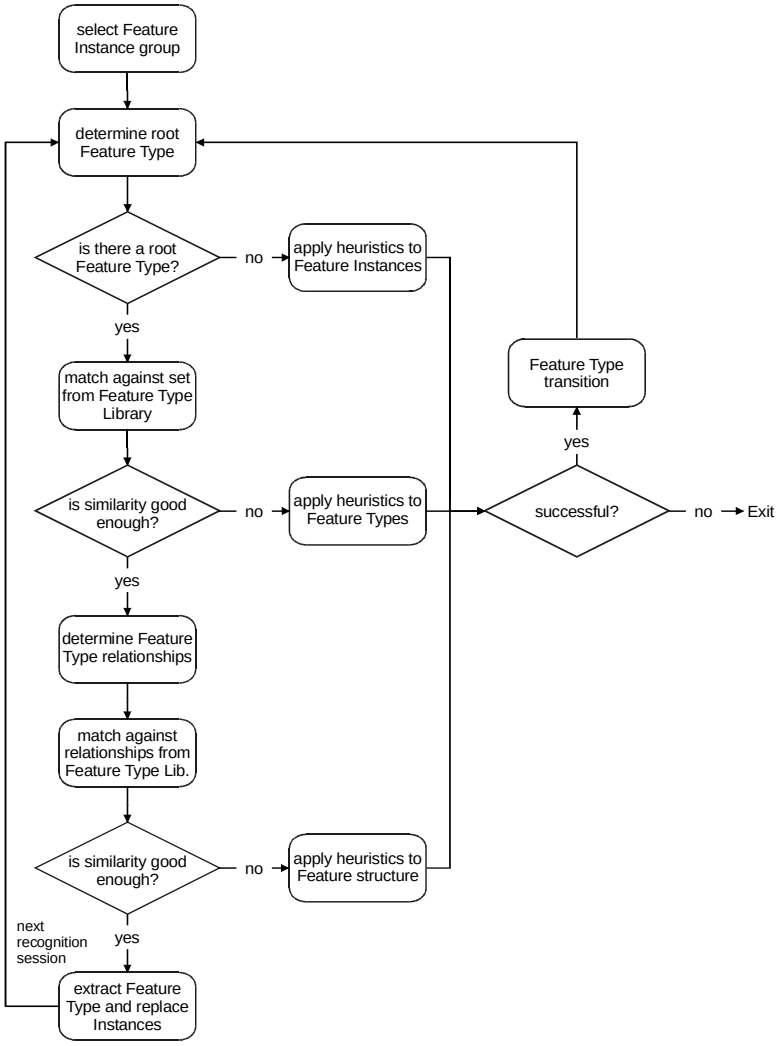

Figure 5. Procedure for computer aided Feature Type recognition

Figure 5 shows the procedure that is followed in case the user or the system requests a Feature recognition process to be executed. First a group of Feature Instances must be selected from which known Feature Types are to be recognised. Selection of this group can be performed entirely by the designer, supported by design system interaction or completely automatic by a design application. From the Feature Instance group the corresponding Feature Types can be determined.

In the Feature Type Library, some Feature Types are marked as a root type, namely those Feature Types that are considered a main architectural concept (e.g. wall, floor, space). The root Feature Type will be the objective of the recognition process. The question now is whether the selected group of Feature Instances contains an instance of such a root type: a root instance. In searching for a match between a possible root instance and the root types in the library, inheritance must be considered, meaning that a match is also made against sub-types in the library.

If this root instance cannot be identified, additional heuristics are needed to introduce an appropriate root instance. For example, four Feature Instances of an assumed Feature Type called Space Boundary could geometrically constitute a space. If this geometrical relationship is detected then a Feature Type Transition procedure is executed that infers an instance of the Feature Type Space. After that, the Feature Recognition procedure is restarted. Identification of the root instance in the selection is not necessarily a completely automated task, it can also assisted by the user.

Once a root instance is found or inferred, the Feature Types from the selected group of Feature Instances are matched against the structure of the root Feature Type in the Feature Type Library. Not all Feature Types related to the root Feature Type in the Feature Type Library are necessarily present in the selected group of Feature Instances. The relationships between the instances are not yet considered in this stage of the recognition process.

Now the question is raised whether the similarity match is good enough. This decision can either be taken automatically by the system, using thresholds for the number and severity of missing instances, or in discussion between the system and the user. If the similarity is too low then additional heuristics are required, for instance a thesaurus of Feature Type names to detect possible cross references between the used names. Feature Type Transition is executed and the Feature Recognition process is restarted.

If the Feature Type similarity is sufficient then the Feature Type relationships are determined from the selected group of Feature Instances. Considering the relationships in the Feature Type Library cluster, starting from the root Feature Type, they may:

– be absent in the selected group of Feature Instances,
– have a different (e.g. association instead of decomposition), or
– have a different role name.

First, a match is performed not taking these differences into account, just considering the topology of the structure. For this purpose graph matching techniques are used. Again, this successfulness of this match can be determined automatically by the system using thresholds or in discussion between the system and the user. Additional heuristics provide rules that can add or replace relationships in order to fit the selected Feature Instances in the structure of the Feature Type found in the Library. Since Feature Based modelling allows for describing a specific building concept in several ways, this process supports the conversion of different description styles to one generic style.

The Feature recognition procedure exits if the one of the heuristics fails. At that point there are several possible results of the recognition process:

1. One or more root instances have been identified or inferred and the structure of instance relationships found between Feature Instances in the model has been replaced by an instantiation of the structure found in the Feature Type Library;

2. One or more root instances have been identified or inferred but a proper match of the relationships in the model to Feature Types in the library could not be made. In this case, the user can decide to use the relationships modelled at the instance level to define a new Feature Type: this is scenario 2 described above in section 2.3.2 as Feature Type definition from a prototype. If a partial match could be made, then the user can alternatively decide to define a sub-type of the partially matched Feature Type.
3. No root types known from the Feature Type Library could be identified or inferred in the selected group of Feature Instances. Again, the user may decide to define a new type from the prototype instances, which in this case would also lead to the definition of a new root type.

3. DISCUSSION

The proposed strategy for formalising architectural design knowledge is in fact a design process in itself. It is the design of architectural design knowledge; design at a meta level. As such the meta design level process dangers from the same pitfalls as the architectural design process illustrated in the introduction of this paper, namely ill-defined problem, ad hoc process cycles etc. The three described strategies offer a style guide for architectural design knowledge modelling based on Feature technology. FBM allows for describing a building concept in different ways using (slightly) different Feature models.

Applications, though, that will share FBM data require a predefined Feature model structure. Without this structure or additional knowledge it is impossible to extract information from the Feature model of a design. Therefore generic Feature Type libraries are needed that contain standardised Feature model structures. In that sense generic Feature Type libraries serve the same goal as standardisation efforts in product modelling (e.g. STEP Application Protocols, Industry Foundation Classes). In contrast with the STEP AP's, a generic Feature Type library is dynamic, it can be update anytime leaving the existing Feature Type structure unchanged. Secondly Feature Type libraries can contain Feature Instances also. This is especially useful in case of specifying supplier's information with a limited variable domain (e.g. the width of a door is either 800 mm, 820 mm or 840 mm).

Inconsistency and incompleteness is an inherent characteristic of Feature Based modelling. This can be regarded both a pro and con of this modelling approach. In this respect, the following conclusions are drawn:

- Inconsistency and incompleteness is an elementary part of architectural design and thus a prerequisite for architectural knowledge modelling.
- Inconsistency and incompleteness are designer dependent. Apart from checking norms and standards there is no general rule a designer can count on for maintaining consistency. Also, incompleteness may be a target of a design process.

– FBM, as currently developed in this research, does not support any kind of strategy for maintaining consistency and completeness for a specific design part or design task. Future research must be conducted on this issue, as is done in the work of (Eastman et al. 1997a; 1997b).

4. REFERENCES

Achten, H.H. (1997) *Generic Representations, An approach for modelling procedural and declarative knowledge of building types in architectural design*, PhD. Thesis, Eindhoven University of Technology, Eindhoven, The Netherlands.

Booch, G. (1994) *Object oriented analysis and design*, second edition, Benjamin/Cummings Inc. Redwood City, CA.

Coyne, R.D., Rosenman, M.A., Radford, A.D., Balachandran, M., and Gero, J.S. (1991) *Knowledge-based design systems*, Addison-Wesley, Reading, Massachusetts.

Cross, N. (1984) *Developments in Design Methodology*, Wiley & Sons Ltd. Chichester.

DeMartino, T., Falcidieno, B., Giannini, F., Hassinger, S., and Ovtcharova, J. (1994) Feature-based modelling by integrating design and recognition approaches, *Computer-Aided Design* 26(8)

Eastman, C.M., Jeng, T.S., Assal, H.H., Cho, M.S., and Chase, S.C. (1995) *EDM-2 Reference Manual*, University of California in Los Angeles, Los Angeles.

Eastman, C.M., Parker, D.S., and Jeng, T.S. (1997a) Managing the integrity of design data generated by multiple applications, The theory and practice of patching, *Research in engineering design* 9: pp. 125-145.

Eastman, C.M., Jeng, T.S., Chowdbury, R., and Jacobsen, K. (1997b) Integration of Design Applications with Building Models, *Proceedings CAAD Futures '97*, pp. 45-59, Kluwer, Dordrecht, NL.

Ekholm, A., and Fridqvist, S., (1997) Design and modelling in a computer integrated construction process, The BAS-CAAD project, *Proceedings CAAD Futures '97*, pp. 501-518, Kluwer, Dordrecht, NL.

Hendricx, A. (2000) *A core object model for architectural design*, PhD thesis, Katholieke Universiteit Leuven, B.

Lawson, B. (1990) *How designers think, the design process demystified*, 2nd edition, Butterworth Architecture, London.

van Leeuwen, J.P., Wagter, H., and Oxman, R.M. (1995) A Feature based approach to modelling Architectural Information, *Modeling of buildings through their life-cycle*, CIB W78-TG10 publication 180, ed. Fisher, Law, and Luiten, pp. 260-269, Stanford University, USA.

van Leeuwen, J.P., and Wagter, H. (1997) Architectural design-by-Features, *Proceedings CAAD Futures '97*, pp. 97-115, Kluwer, Dordrecht, NL.

van Leeuwen, J.P., and Wagter, H. (1998) A Features Framework for Architectural Information, a case study, *Artificial Intelligence in Design '98*, ed. Gero and Sudweeks, pp. 461-480, Kluwer, Dordrecht, NL.

van Leeuwen, J.P. (1999) *Modelling Architectural Design Information by Features*, PhD. Thesis, June 1999, Eindhoven University of Technology, Eindhoven, NL.
Ovtcharova, J., and Vieira, A.S. (1995) Virtual prototyping through Feature Processing, *Virtual Prototyping - Virtual environments and the product design process*, ed. Rix, Haas, and Teixeira, pp. 78-90, Chapman & Hall, London.
Roozenburg, N.F.M., and Eekels, J. (1995) *Product Design: Fundamentals and Methods*, Wiley & Sons, Ltd. Chichester.
Shah, J.J., and Mäntylä, M. (1995) *Parametric and Feature-Based CAD/CAM*, Wiley & Sons, New York.
Sowa, J., (1984) *Conceptual Structures: Information Processing in Mind and Machine*, Addison-Wesley, Amsterdam.