SUPPORTING COLLABORATIVE DESIGN BY TYPE RECOGNITION AND KNOWLEDGE SHARING

SUBMITTED: January 2002
REVISED: June 2002
PUBLISHED: August 2002 at http://www.itcon.org/2002/11
EDITOR: Abdul Samad Kazi

Jos P. van Leeuwen, PhD. MSc.Eng.,
Faculty of Architecture, Building, and Planning, Eindhoven University of Technology, The Netherlands
email: J.P.v.Leeuwen@tue.nl - http://www.ds.arch.tue.nl/jos

Sverker Fridqvist, PhD. Arch.,
Faculty of Architecture, Building, and Planning, Eindhoven University of Technology, The Netherlands
email: S.Fridqvist@tue.nl - http://www.ds.arch.tue.nl/sverker

***SUMMARY**: In collaborative design projects, designers are required to share and identify design knowledge, which is an aspect of design that can benefit significantly from formalisation of design knowledge. On the other hand, working with formalised design knowledge should not impede the creativity in design by restricting the design process and limiting the freedom of manipulating design information. The research project reported in this article provides an approach to design modelling that manifests a high level of flexibility and extendibility of the formalised design knowledge. The design process is supported in this approach by tools that help identify the design rationale through type recognition and by Internet-based services that allow designers to share design models and design knowledge in a well-structured manner.*

***KEYWORDS**: Design Support System, Collaborative Design, Feature-Based Modelling, Case-Based Reasoning, Internet Application.*

1. INTRODUCTION

Since construction projects are one off and short lived, communication principles and methods need to be solved outside them. Product modelling is an area of ongoing research in the construction industry to meet this need, in which international projects such as STEP and IFC have achieved important results. A number of research projects have concentrated on developing frameworks for product modelling that specifically aim at supporting design tasks in the construction industry. The present project is based on the experiences the authors gained from the Feature Based Modelling project (van Leeuwen, 1999) and the BAS•CAAD project (Ekholm and Fridqvist, 1998, Fridqvist, 2000).

The modelling requirements of the design phase differ from those of the production phase. Production modelling refers to concrete objects and established production practices and can be implemented through a set of pre-defined classes. Design modelling is less explicit and especially in the early stages it tends to use highly abstract concepts that only vaguely refer to concrete objects. Additionally, models evolve and transform during the design phase. This makes fixed classes too constricted for design work, although they are very suitable for production models (van Leeuwen et al., 2001, de Vries et al., 2001).

Building design is mostly done by teams that consist of individuals from many firms that are geographically dispersed. This makes face-to-face collaboration hard to achieve, and produces an incentive to use information technology for communication.

The range of building products, materials, and construction methods available today is very large and continuously changing, making it an impossible task for any individual to keep an overview even in the personal field of profession. Internet has become a key resource for finding this kind of professional information, but provides it in a very unstructured and therefore unmanageable manner.

2. OBJECTIVES

This article reports on a research project that focuses on the support of collaborative design. A requirement for collaborative design is the ability to identify and share design knowledge. Sharing design knowledge is common practice, though it is often done in an informal way. Describing design knowledge formally should greatly improve the effectiveness of sharing it; the meaning of the knowledge can be made more explicit and its interpretation less ambiguous. Also, formalisation provides a foundation for automated search and comparison of sources of design knowledge.

The presented research project aims to develop the technology of Internet based *Design Knowledge servers (DesKs)*. These provide design knowledge that is modelled using Feature Based Modelling (FBM), an information modelling technology to represent design concepts in a flexible and extendable way that is suitable for modelling during design (van Leeuwen and de Vries, 2000).

An Internet based system of design knowledge servers can be utilised by designers to collect and systemise information produced and published by the various bodies of the construction industry. Designers can search design knowledge servers and evaluate the available design knowledge regarding the applicability for their needs. Designers can also share design solutions through the same network of servers, making their work available to partners in the design project or sharing it with an even wider audience.

The research project has three main facets:

- To develop the so-called Feature Manager core module that provides the software functionality to model and retrieve design knowledge. This module is prepared to function in an Internet environment.
- To develop Feature Type Recognition, the technology that enables comparison of data-structures in the FBM methodology. This technology will be an integral part of the Feature Manager core module.
- To develop the technology of design knowledge servers and the services they provide for sharing and searching design knowledge. This implements the software interface to the Feature Manager and makes its functionality available to clients of the design knowledge servers.

3. FEATURE BASED MODELLING

The knowledge modelling approach in this project is built upon *Feature Based Modelling* (FBM). FBM supports modelling of both typologies and instances of design concepts in a flexible manner. The entities in this approach that represent design concepts are called *feature types* and *feature instances*. They are structured and formalised in a way that allows the designer to have full control over the definitions and structures used to formalise the design concepts. This technology allows communication of highly abstract concepts as well as concrete data. Additionally, it supports a layered approach to modelling, which is essential both to standardisation and to publishing product data.

Feature Type Recognition (FTR) is the process of finding feature types that correspond to a specific configuration of feature instances. FTR has many potential areas of application, such as case retrieval, product finding, translating models between schemas, and certain types of analysis, all of which would be important capacities of a design knowledge server.

Feature Based Modelling allows highly complex concepts to be modelled and expressed in terms of more basic ones, while Feature Type Recognition can find a complex concept that encompasses and includes a set of simple concepts. It is estimated that sufficiently basic concepts may be less problematic to agree upon internationally than the more complex ones found in national building classification systems, and that this can form a basis for translation and communication.

3.1 Construction documentation and modelling

Traditionally, construction documentation focuses on the building as a material thing that is to be produced. The production phase involves many people that need instructions, and they cannot all get it personally from the designer. Primarily, the documents allow the work leadership to be delegated to many people, and thus is a enabling factor for all greater construction projects.

In modern times construction documents are also used to delegate some design activities, mainly of technical nature. Here, the documents serve not to guide the production of the building, but the production of the design of some elements or systems that are parts of the building.

Traditional construction documentation is separated into drawings that express the shape of the design, and text documents that contain all other information related to the production of the building. With the advent of computer-based production and handling of documents, an interesting possibility to merge all documents into a single entity has become feasible. Such a united documentation is called a *product model*, since it contains all information necessary to produce something. Thus, the product model concept is still focused on production.

The idea of product modelling has been extended to incorporate also the period after production, i.e. to document and guide maintenance activities. While the production phase focuses on assembling parts and putting materials in place, i.e. on matter, the maintenance phase is focused on keeping the function intact. For instance, maintenance of an office building may require the interior walls to be repainted, the computer network cables exchanged, etc. Thus, the functions remain while the material components that provide the functions change. In other cases the functions of parts may change when a building is taken to use for other activities while the parts remain the same. A maintenance model should support this function – provider duality (EPISTLE, 2000, Ekholm, 2001).

Another extension of product modelling is to preserve design information. Since a product model may contain all information about a product, it could be used to store also the information collected during design, including the rationale behind design decisions. Much of the design information cannot be documented by traditional means and is thus lost. Since the design phase is focused on function and the means to obtain function, this information could be useful in the maintenance phase if it could somehow be forwarded through the production phase.

To get the most out of product modelling, it should be extended not only to the design phase, but *through* it. This is where feature based modelling comes into play. In general, product modelling only focuses on describing the product, but not on the process of creating the description, i.e. on the *design process*. Thus, a product model *schema* may very well support modelling all aspects of a well-known building design, but nevertheless be quite useless for actually creating the design. The reason for this is that many product modelling schemas adopt a class centred modelling approach.

A *class centred* schema (Garrett and Hakim, 1994) tries to implement one model object *class* for each different *kind of real or abstract thing* that is to be modelled. For building modelling, this would render classes for different kinds of walls, floor slabs, windows, doors, heating and ventilation components, and so forth. While this approach serves product modelling well, it may cause problems for the designer (Eastman and Fereshetian, 1994, Eastman et al., 1995, Galle, 1995, Junge et al., 1997, van Leeuwen and Wagter, 1997).

3.2 Property-oriented modelling

The design process is often described as a search to find a satisfactory solution to a *design problem*. This is, however, only partly true, since the problem is mostly not very well defined. Thus, the design process also involves defining the problem. In other words, design is not only to answer the questions, but also to find the proper questions to ask.

While a suitably defined class centred schema could represent both questions and answers, it would still not support the search aspect of the design process. The reason is that any schema has a limited set of classes, and a class centred schema would thus allow only a specific set of questions and answers. In a class centred schema, classes need to represent many aspects of a thing, or else the number of classes would become unmanageably large. Thus, each time the designer chooses an object to be part of his model, the schema forces him to ask many questions at once in order to provide the many answers required to set all the parameters of the object. This would restrict the search in ways that would plausibly make the design process less efficient.

What is needed to support design efficiently is a system that allows the designer to move freely between subjects and to focus on any question that he believes is the most important at that moment, in the order that he finds most suitable. The feature based modelling approach is one such system. It is *property oriented*, instead of class centred, since it separates modelling of properties from the things that have these properties (van Leeuwen et al., 2001).

A property can be explained as a characteristic that is attributed to a thing. Examples of properties are: colour, shape, mass, location, and function. Many different kinds of things can have a property in common, but no two kinds of things share all properties.

A property-oriented modelling system allows the designer to initially create an empty object to represent his design, and then to add properties to that object to reflect the different things he needs to express about the design. Alternatively, the designer can just collect the desired properties of the subject of design and add them to the model, while postponing the decision to which objects these properties are attributed. Property-oriented modelling allows the designer more freedom to decide upon the context of properties and how properties work together in achieving the required functionality of the designed object, which is an important capability during the course of design. Thus, a property-oriented system does not impose any order or other restrictions on the design process.

The term ‘classification’ denotes two related but different activities: a) to define conceptual classes based on observed or expected similarities and differences among items; and b) to sort or order items according to such classes. Classification in the first sense is fundamental to property-oriented modelling, where classes are defined through collections of properties. The main difference in approach between class centred and property-oriented systems lies within the moment of and responsibility for the formalisation of the classes. Class centred systems define these schemas at the time of development of the system; they are the responsibility of the software engineer, whereas property-oriented systems allow the designer to define classes.

While a property-oriented approach to modelling design information offers a high level of flexibility to express the semantics of design concepts, it is still important to analyse the function and meaning of properties on a more generic level. A generic classification of properties is essential for the development of automated reasoning about construction artefacts through their properties. Ekholm (2002) proposes a generic classification of properties of construction objects. The purpose of this classification is not to determine a complete list of properties of construction objects, but to determine classes that represent a particular kind of knowledge about construction objects. The knowledge that a property belongs to a particular class can be used in the development of computer support for design and modelling.

3.3 Delayed determination of semantic content

The property-oriented approach implements delayed determination of the semantic content of the model objects, as opposed to the class centred one, where the semantic content is predominantly determined at instantiation time. The property-oriented approach allows the designer to create a model object to represent only the existence of a thing (present or future), while postponing the specific definition of the properties of that thing. As the model evolves, the designer adds property objects to the model object and thus changes its semantic content. This calls for a mechanism where the current semantic status of a model object can be analysed.

Within the context of a property-oriented approach, pre-defined classes can still be used to speed the design process by enabling the designer to add many properties to his design at once, as an organised collection. Pre-defined classes are also the basis for communication between the design and the production phases, i.e. to generate building production documentation. These can be class centred product models, or traditional drawings and text documents. Classes can be used as properties themselves in collections of properties of higher-level classes. In a property-oriented approach, instantiation of the classes results in objects to which additional properties can be attributed independently of the original class definition.

3.4 Basic concepts of feature based modelling

The basic concept of FBM is *feature*. The feature of FBM is derived from the *form features* that some mechanical 3D CAD systems use. A form feature captures a generic feature of the shape of things, such as slot, hole, chamfer, etc. Thus, form features raise the semantic level of modelling. FBM extends form features to a generic concept, intended to cover everything that can be said about a product.

FBM is divided in two sub-areas, general and particular models. A general model represents a general concept, a *kind* of thing, while a particular model represents an *individual* thing. General concepts are modelled with *feature types*, while individuals are modelled using *feature instances*.

In FBM, Features represent both classes and properties. Whether a specific feature is to be interpreted as a property or as a class depends on its context, i.e. its relations to other features as defined through components. Features can also be class and property at the same time, regarded in different contexts.

3.5 Extendibility

Feature based modelling is not only property oriented, but it is also *extendible*, meaning that the user can create new definitions for classes and properties as needed. While a property-oriented system allows the designer to make decisions in *any order*, extendibility furthers this freedom by allowing him to define *what* decisions. Extendibility also extends the designer's freedom to decide when to make decisions, by not requiring newly created properties or classes to be more than empty shells. Thus, it is possible for a designer to state that he intends to design a house that is beautiful and inexpensive, by creating the three features *house*, *beautiful* and *inexpensive*, and assigning the two latter as properties to the first. At a later stage, the designer can specify what he meant with all of these by adding more features to their definitions. For instance, he may indicate that the house is beautiful because of a brick façade and stained glass windows.

3.5.1 FBM classes

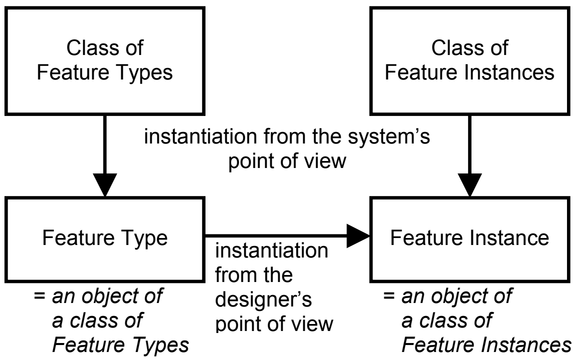
Class of Feature Types
Class of Feature Instances
instantiation from the system's point of view
Feature Type
instantiation from the designer's point of view
Feature Instance
= an object of a class of Feature Types
= an object of a class of Feature Instances

FIG. 1. Two kinds of instantiation in FBM: for the designer: instantiation from types; for the system: instantiation from classes.

In the FBM framework, extendibility is obtained by the definition of a meta-level of classes that define the possible content of both feature types and feature instances. The meaning of the term 'class' in this and the following paragraphs is as defined in object-oriented contexts and should not be confused with the more philosophical meaning of the word 'class' used in previous sections of this article. Both the types and the instances are created as objects in the system, although to the user it appears as if the instances are created as objects of the types (see FIG. 1). This approach allows the user to define new types (as objects in the system) at runtime and immediately use the new type in creating instances of that type. The system is not affected by the extension of its data schema in this manner because it can deal dynamically with the newly introduced definition of the type and its relationships to other types.

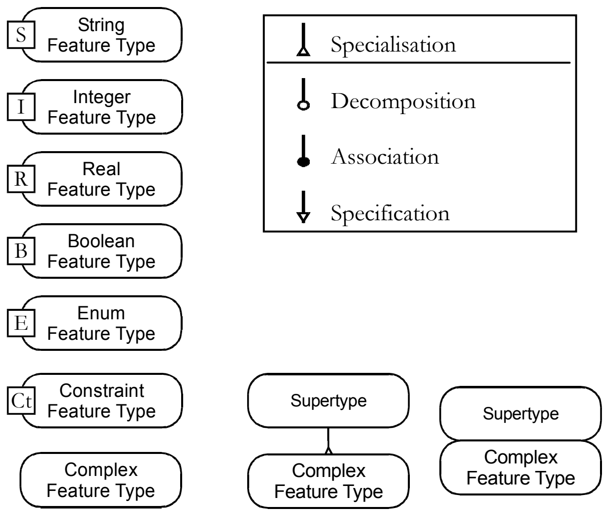

FIG. 2. Graphical notation of feature types.

Since each class of feature type has a parallel class of feature instance, much of the following text will only mention *feature*. FBM defines nine classes of features (i.e. nine classes of types and nine classes of instances). Among these, six are used to store simple data:

- String features – store text strings.
- Integer features – store integer numbers.
- Real features – store floating point numbers.
- Boolean features – store Boolean values, i.e. *true* and *false*.
- Enumeration features – sometimes something may be in one of a finite number of states. An enumeration feature type defines a list of state names, and an enumeration feature instance can take one of these state names as its value.
- Geometrical Features – intended to capture shape information in relation with a geometric engine, but not currently implemented.

Complex features organise collections of features to create more complex model structures. Since also complex features may be included this way, arbitrarily complex structures may be created. FBM provides three different ways to link an organising complex feature with an organised feature.

- Specification – the organised feature *specifies* (a property of) the organising complex feature.
- Decomposition – the organised feature is a *part of* the organising complex feature.
- Association – the organised feature is *related to* the organising complex feature without being a part.

The linking element provided by FBM is called *component*, and it can have one of three *role types,* named as the kinds of links listed above, to reflect the type of link it provides. Within their context in a complex feature, components are identified by their *role name*.

In addition to components, two complex feature types can be linked by the *specialisation* relation. Specialisation is an important concept in modelling, since it allows building kind-of hierarchies and showing similarity between different kinds. Using specialisation it is possible to define that *Batavus Bologna* is a kind of *bike*, which is a kind of *vehicle*, and that *Gazelle Primeur* is also a kind of *bike*. This tells us that a *Batavus Bologna* is similar to a *Gazelle Primeur* in all ways that are defined in *bike*.

In addition to the seven *static* feature classes listed above that can represent various kinds of data, FBM provides two *dynamic* feature classes to further enhance its modelling power. Both include program code as part of their definitions, and both are defined as feature types but operate on feature instances.

Constraint features are used to represent constraints between the components within a complex feature type. Constraints may either refer to the modelled thing, or to the model. In the first case, a constraint feature may represent a design requirement that involves some kind of computation, i.e. it is not sufficient to use a static feature. An example is a requirement that a floor area should not exceed a certain size. In the second case, a constraint feature may define constraints on the model itself, for instance that the different legs of a table model must be unique individuals. The functionality of constraint features has been partially implemented.

Handler Features are intended to add dynamic capabilities to feature models in relation with events that occur in the context of the models, but they are currently not implemented.

Feature types and feature instances are arranged into *namespaces,* which are nested structures that uniquely identify features together with their names. This is a development from the original specification, which used *libraries* and *models* to manage collections of feature types and instances, respectively, and to identify individual features.

3.5.2 Delayed determination of semantic content in FBM

Each feature instance becomes associated with a feature type when it is created and inserted into the model. In case of a complex feature, the type defines a set of components. However, it is not mandatory to follow this definition when adding components to the instance. The user is free to choose more or fewer component instances than what is defined by the type. Moreover, the user is free to add component instances that have no definition at all in the type. Because of this, the model is likely to differ semantically from its instantiated type after a while. An instance's semantic content is defined through its components and may therefore not be very clear from a human point of view if many components were added to the original instantiation of the type.

However, the explicit meaning of the model can be enhanced by feature type recognition, which analyses the semantic content of a feature instance in terms of pre-defined feature types.

4. FEATURE MANAGER

The project presented here has developed a *feature manager*. The feature manager module is the core of feature based modelling and has no user interface of its own. Instead, separate user interface modules have been developed that connect to the manager through an Application Programming Interface (API) utilising messages and events. The feature manager provides several main functions:

- Providing the functionality for the definition, creation, and modification of feature types and feature instances.
- Maintaining the *semantic consistency* of the database of feature types and feature instances.
- Providing persistence. While the feature manager provides access to the data through an object model, it persists the data to a relational database.
- The manager provides the functionality for *feature type recognition*.
- For evaluation purposes, design models may need to be completed with default data. The manager can serve users with *automatically completed data* while leaving the model as is.

4.1 Maintaining semantic consistency

The feature manager ensures that feature types and instances within its control are consistent with the definition of FBM. It is also capable of various kinds of semantic checking, based on the semantics inherent in FBM.

The feature manager is implemented in an object-oriented approach, with classes for the different feature types and instances. These classes implement the consistency checking for the manager.

Through its classes, the manager is capable of evaluating how various changes to the data might impact the semantic consistency, and it will reject all changes that would lead to inconsistency. Thus, it is ensured that the database is always consistent at the level of features. However, higher level consistency checking, concerning the user-defined meanings given to the various features, can naturally not be maintained or checked by the manager. It is the *Constraint feature type* that serves this purpose.

Constraint feature types allow the user to define certain relationships that should be fulfilled between the feature instances that the constraint feature is referring to. The manager does not impose this type of constraints, or resolve conflicts between them; it merely provides a way for the designer to designate the constraints. The reason is that defining conflicting constraints is part of the design process, and should be allowed by a design support system. However, experiments with a constraint-checking and constraint-solving agent that co-operated with a previous implementation of a feature manager have shown that this kind of agents can provide a significant support to the designer in finding suggestions for solutions of the constraint violations (Kelleners, 1999).

4.2 Persistence

The term *feature data* designates data that describes feature types or feature instances. The manager persists the feature data into a relational database and can also import from and export to files containing XML-Schema's and XML documents. However, feature data can also be downloaded from URLs or retrieved from the so-called design knowledge servers (see section 6).

Feature data is uniquely identified by the *name* of the type or instance and the *namespace* it is contained in. The namespace can be related to the current document of feature data or to a URL. It may happen that a referenced feature type or instance is changed after the referring feature data was saved. This may result in 'dangling references', i.e. references indicating features that do not exist. The manager has functionality to cope with this situation gracefully, and it will not result in a corrupt database. *Versioning* might prevent such errors altogether. For this purpose, versioning functionality is implemented at the object level, for both feature types and feature instances.

4.3 Automatic model completion

The manager has functions to provide a user with defaults for model data as specified by the types, even if the model is not complete in itself. Thus, evaluation functions can always query a model according to its type, and get answers. Temporary feature instances are created, based on default values provided by the types. These *implicit component instances* are not considered part of the model, and are not persisted. The manager removes them automatically when they are not further needed, e.g. when the designer adds the particular instances explicitly.

The manager interacts with the front-end applications through messages and events. It supports the user interfaces with consistency-checked data for various purposes, providing context specific selections of feature data, for example when modelling relationships between instances. Thus, the user interface does not need to present selections that would be rejected as user input.

5. FEATURE TYPE RECOGNITION

The semantic content of feature instances is initially set at creation time through associating the instance with a feature type (see also section 3.5.2). Subsequent addition of components to the instance, that supersede or violate the definition of the type, may render this classification inadequate. While the instance's set of components may fully define its semantic contents, there is a need to make such rather implicit classifications fully explicit. Feature type recognition analyses the semantic content of a feature instance in terms of its components, and on this basis it establishes references to feature types that provide higher-level semantic definitions.

Feature Type recognition (FTR) utilises the structure of FBM, where higher-level concept definitions are based on lower level ones in tree-like structures that in practice become entangled to form lattices. In principle, FTR works by breaking a complex definition down into its components, and then reassembling them again by following a different tree-like path through the lattice. In practice, recognition is to match a feature instance's set of components with those of one or several feature types.

FTR can be used for two principally different purposes. The first is to find a type that *classifies* an instance according to its current state. Secondly, with a different set of matching weights FTR can also find types that *extend* an instance in various ways. This could be used to suggest ways to further develop a design.

5.1 FTR algorithm

The algorithm has the following steps:

- Inspect all the component features (step 2 in FIG. 3).
- Collect the types of all the component features (step 3.1 in FIG. 3) and the supertypes of these types (step 3.2 in FIG. 3).
- For each type t from step 2, collect the types where t is a component (step 4 in FIG. 3), this is the list of candidate types.
- For each of the types from 3, collect the subtypes where the component is inherited; i.e. not re-defined (step 5 in FIG. 3).
- For each of the types collected in 3 and 4, evaluate how well its components match the components of the FTR-subject. Then order the items according to the matching evaluation. This results in the list of candidates for the final decision.

Ad 2. The reason for including the supertypes is that it is always allowed to use a more specific feature instance for a component than is defined in the corresponding type. An example: When modelling a door, the 'door' type may define that a 'door handle' should be one of its parts. However, the designer may want to specify a specific make of door handle. Thus, the actual instance is more specific than the 'door' type prescribes. If we are about to apply FTR to the door instance, we need to consider the generic 'door handle' to be able to find the 'door' feature type.

Ad 4. The subtypes of the first found types are added to the list of candidates. The reason for this is that we want to find the most specific feature type that matches the feature instance. Thus, we need to include subtypes of the type candidates found in step 3, as long as they don't prescribe a more specific component than the one we have got. The reverse relation between an inherited component and a type is not explicitly maintained by the manager; thus the relation needs to be established in this way.

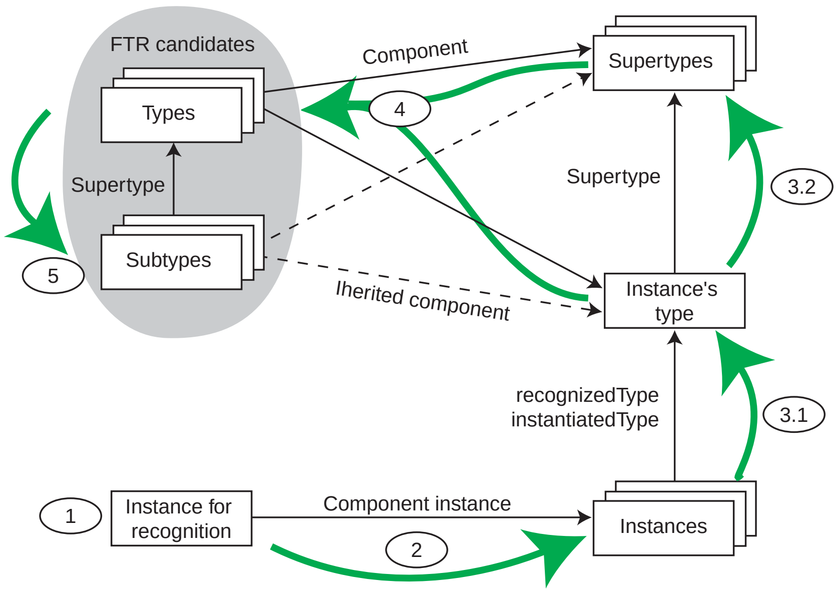

FIG. 3. The FTR algorithm.
Legend: Rectangles represent feature types and instances. Thin straight arrows represent relations defined by FBM. Thick curvy (green) arrows represent how the FTR process traverses the feature data.

The algorithm uses a set of criteria for ordering the candidates, all based on the components. Components are compared if both *role type* and *role name* are identical:

- Compare how well the sets of components match.
  Is there a one-to-one correspondence of components?
  Is the type under-specified, i.e. has the instance some components that the type doesn't have?
  Is the type over-specified, i.e. has the type some components that the instance doesn't have?
- Compare the prescribed filler types type of the candidate's components with the actual fillers of the instance's components. How far, supertype-wise, are the candidate's types from the fillers' types?
- Compare the cardinality of the type's components with the number of fillers of the instance's components'.
  Is the type component's maximum cardinality higher than or equal to the actual number of fillers? Only the maximum value is compared, since a number of component instances lower than the minimum value is considered implicitly filled.

The final outcome of the FTR process depends on what weights are given the different matching criteria. At the time of writing the effects of this setting has not been thoroughly studied and will be evaluated in the next phase of the project. Plausibly, different settings would be preferable for different purposes.

FBM defines three different *role types* for components, representing different kinds of relations between features. This opens up the possibility to do FTR on a subset of the components, and only to consider components with some of the role types. The modelling aspect can thus to some degree be selected for the recognition process. For instance, if only decomposition components were considered, the feature instance would be classified according to the decomposition aspect, while the functional aspect would only consider association components. In addition, components of a complex feature type are given *role names*. Also with this part of the definition of the component various scenarios can be pursued in the recognition process. More strict searches would not consider components with differing names, but the usage of a thesaurus or linked classification tables such as those provided by the LexiCon project (Woestenenk, 2000) can be used to widen the search. In this manner, the recognition process will include types that have a semantic match though using different terminology.

The FTR function allows the user to choose the namespaces where to look for types. This will add to the aspect selection provided by choosing a subset of role types, narrowing the search to a collection of types within a namespace that represents specific design aspects.

6. DESIGN KNOWLEDGE SERVERS

Part of this research project focuses on providing the FBM framework functionality as services on the Internet. These services can be used by designers to store and share design information within a global context or within the scope of a design project. The research project develops and investigates the technology and the organisational aspects that are required for designers to build up a common, but distributed, searchable design knowledge base. The knowledge base can be used as a common source for designers or, e.g., as a central or distributed storage and reference means for design projects.

6.1 DesKs as storage place for design information

For support of collaboration in design, Design Knowledge servers (DesKs) can be used as centralised or distributed repositories to store design data that must have shared accessibility for the design team. They provide the facilities to define typologies and to create design models. DesKs function as nodes in a network, where each node represents a particular facet of the collaborative design project. For example, each participant in the project can run their own node in the network to store design data and provide it to the other participants.

Outside the particular project scope, the facility of shared repositories of design information also allows publication of information through the DesKs. This makes these information sources available to the design community or to selected users. From within the project scope, these published information sources can be easily accessed through the services available from the DesKs. Potentially, this means that relations are facilitated from project data to, for example, product datasheets from manufacturers or building codes supplied by governmental organisations. The only requirement is that the latter sources of information are also made available through the DesKs technology.

6.2 DesKs as vehicle for sharing design information

Apart from functioning as a storage location to facilitate the sharing of design information, the DesKs also provide tools to search the stored information, using amongst others the FTR algorithms described in section 5. These searches can be based either on samples of data from the current design project, or other forms of input in queries to the data store. Directory lists of DesKs, made available through the DesKs, enable the designer to expand the search to multiple distributed stores of design knowledge worldwide.

6.3 Applications of DesKs

The Design Knowledge servers form a network of nodes that can function in a variety of scenarios.

Collaborative Design

In large scale design and construction projects, where collaboration between multiple designers is a crucial issue, currently available document management systems fail to provide sufficient data-awareness: they allow the designers to share documents but cannot assist in interpreting the meaning of the contents of documents. Product models do provide the required higher-level data definitions for the content of design documents, but they are too inflexible to be useful in the design stages.

The design knowledge servers can be used in this scenario to develop the design solutions by gradually building up formal definitions for the design concepts (specific feature types) in combination with the application of more standard design concepts (generic feature types). Project specific design concepts are stored in the participants' nodes in the DesKs network and are managed under the responsibility of the participants. In a similar fashion, the model representing the design case is created as a collection of feature instances, distributed over the network of the participants under their own responsibility. In relation to the various functions of individuals in the project team, user-roles are assigned that determine the individuals' access rights to the data.

DesKs services:	Storage and retrieval services for feature data
	User access management based on membership in project teams
	Access to standards made available through ‘generic’ DesKs
Content:	Generic design concepts (generic feature types)
	Project specific concepts (specific feature types)
	Project description data (model of feature instances)
Providers:	Generic DesKs (providing feature databases for standardised and general design information)
	Project designers
Consumers:	Project designers
	Client of the project
	Project contractors and other participants

Design Knowledge Commerce

In this scenario, there is a contract between a party needing a design solution to a particular problem and a party able to provide the solution. The parties have been brought together through a search session on one or multiple DesKs, where the searching party has input the design problem into the search algorithm and the providing party had made the design solution available on a DesK. On the basis of the contract between the parties, rights are assigned to the acquiring party to use the provided design knowledge.

The commercial scenario described above involves many complications in terms of, e.g., copyrights, multiple uses of the provided knowledge, re-use of knowledge provided by third parties (composite solutions), etc. These complications need to be addressed before this scenario can actually function in a commercial setting. The same scenario can, however, be envisioned in the context of, for example, designers' associations. In this scenario designers subscribe to the provided services, acquiring the right to use all knowledge available while feeding the servers with their own knowledge.

DesKs services:	Storage and retrieval services for feature data
	User access management based on individually acquired rights
	Search facilities on the basis of problem descriptions
Content:	Feature data (typologies and case-models) for a large variety of design problems
	Design problem descriptions, related to
	Design solution descriptions
Providers:	Organisations or individuals providing design services (commercially)
Consumers:	Designers who want to buy solutions to a particular design problem
	Construction organisations who want to buy design expertise for a particular project
	Product manufacturers who want to buy design expertise for the development of new products

(Historical) Design Reviews

The ability for design critics and architectural historians to make their observations of design rationales in historic or new buildings available in a formalised manner would allow them to share this knowledge with the architectural design discipline. This would offer a tool for very direct feedback into today's design practice. It also would allow formal comparisons of design approaches and alternative design solutions.

DesKs services:	Storage and retrieval services for feature data
	User access management based on ownership of data
	Search facilities on the basis of keywords
Content:	Typologies representing the design rationale of historical or new designs
	Instance models representing the particular design cases
	Additional descriptive data, for example review comments from design critics
Providers:	Architectural design critics
	Architectural historians

Consumers:	Architectural design critics Architectural historians Practicing architectural designers Architectural students

Product Databank

Providing product data in a ready-to-use format is a very feasible application area for computer support in design. The construction industry has already achieved several successes in this area. The Dutch national building-documentation system (NBD) provides this kind of information in a digital format, but this format is generally too loosely defined to be utilised directly in computer supported design tasks. The international developments in the STEP community on standardisation for parts descriptions aims to provide a much more detailed definition of how part information is to be provided. Another example is the approach followed by CAD-O-Theek to provide product information in a CAD-ready format, but on the basis of specific requirements posed by the manufacturers and products and by the CAD systems to be used.

The scenario of applying DesKs for providing product data to the building design community follows a similar path: manufacturers can provide their own specific typologies and samples of products and materials in feature types and instances. Product information made available through design knowledge servers can be used directly in the design process.

DesKs services:	Storage and retrieval services for feature data User access management based on ownership of data Search facilities on the basis of keywords and on the basis of performance specifications
Content:	Feature data (types and instances) describing typologies and samples of materials and products for the construction industry
Providers:	Manufacturers Resellers
Consumers:	Practicing designers

6.4 Functionality of the DesKs

The DesKs are nodes in a network where communication takes place on a peer-to-peer basis, using the HTTP and SOAP protocols. The FBM modelling functionality, the server functionality, and dedicated client functionality are jointly implemented in an application called the DesKs WebNode. A WebNode can be used as a modelling tool, as a server for sharing design knowledge, and as a client to access remote design knowledge on other nodes. There two kinds of access to these nodes: browsing access and enhanced access by the dedicated client functionality.

The browsing access is meant for exploration of the contents provided by the DesKs. If browsing the DesKs leads to the wish to actively utilise the design knowledge, enhanced access will be needed to fully utilise the DesKs functionality. A session with the WebNode in client mode allows the user to work remotely with the data stored at a WebNode in server mode. The WebNode application handles multiple client and server sessions simultaneously.

The main tasks of the DesKs are listed below.

- Provide a storage place for shared design knowledge.
  This is a server-located store of feature data, containing a set of namespaces. The server runs an FBMcore module to provide access to the store.
- Maintain a DesKs directory.
  The DesKs directory provides a list of servers that may be of interest to the clients of the current server. The server provides simple maintenance facilities for this directory and expands the directory on the basis of performed searches requested by clients. In case of failure to provide the appropriate design knowledge, the directory is used to redirect the search request.
- Provide read-access to feature data.
  This includes: authenticated browsing access to the feature namespaces that are available in the

server-located feature store; provision of summary information for each of the available feature namespaces, for human interpretation, e.g. using a short description or set of keywords; and browsing access to the server's DesKs directory.

- Provide enhanced access to feature data.
  The WebNode client provides remote access to the feature namespaces managed by a server. It sends a request for a particular collection of data from the namespace and the server responds by providing remote access to the data-objects that it manages. The server responds to: requests for a particular element of feature data, identified by its name and namespace; queries in a QBE form; queries for a pattern match (using various ways and levels of matching); and queries using a set of keywords used to search descriptive text that is available with feature data and namespaces. On failure to provide an adequate response to a request, the server forwards the request to (a selection of) the DesKs available in the DesKs directory.
  The client may also require write-access to the feature namespaces; the server accepts posts of feature data and updates the feature namespace with this posted data, providing that the user has sufficient privileges.

- Manage user-access to the server.
  The server allows anonymous access and authenticated access. It administrates users and groups of users for a variety of purposes such as project-teams, design knowledge-subscriptions, etc. The users and groups can be granted various privileges such as browse, read, add, change, delete. Access-security can be set on cascading levels. Higher-level security settings (e.g. at the server level) can be loosened at lower levels (e.g. at namespace level). User security settings loosen group security settings, allowing a member of a group to have additional privileges.

7. IMPLEMENTATION

The project is developed using XML technology and implemented in Microsoft's .NET framework. However, while the .NET framework supports much of the project's functionality, it is not a prerequisite. Since XML is system-independent, the functionality of the project could be developed on any system that supports XML. XML technology not only provides the functionality required for accessing data across the Internet, it also provides excellent support for the dynamic modelling paradigm that is inherent to the FBM methodology and allows import and export of feature data through XML schemas and documents.

The .NET framework facilitates reuse of software in a variety of application scenarios and provides good support for developing the integration of various technologies. The FBMcore module, including the FTR technology, is implemented as a class library in C# that provides an object model for access to the feature data. The object model is persisted by the FBMcore module using a relational database. The class library of the FBMcore module is referenced in the development of the DesKs WebNode application, which is a stand-alone Windows application. The modelling functionality of the WebNode application provides both a dialog-based user-interface to the object model and a graphical user-interface, allowing diagramming and drag and drop kind of interaction with feature types and feature instances in namespaces. The server portion of the WebNode application implements the remote access to the FBMcore's object model and the additional functionality for searching feature data. The server functionality is implemented through .NET's Remoting technology which applies the SOAP (Simple Object Access Protocol).

8. DISCUSSION

This article reports an ongoing project, and final results will be available only after testing and evaluation. However, preliminary results show that creating design knowledge servers that form an Internet-based network through peer-to-peer communication is a viable concept. Feature type recognition has also been demonstrated to function as expected.

We expect the combination of Feature Based Modelling, Internet, and Feature Type Recognition to provide a powerful and dynamic modelling environment able to support the complete span of modelling and communication needs within the life cycle of a building. More specifically, the solution of dispersed servers communicating via the Internet allows models to be built upon published information in a layered manner, and supports distributed modelling in collaborative design processes. Feature Type Recognition will provide the power of these servers to assist searches for information as well as translation between different modelling practices.

In addition to developing a prototype system, this research project identifies and evaluates the implications that the use of such a system may have on how people work together on a design project. It addresses organisational questions, such as who owns the design knowledge, who is responsible for it, etc. Finally, it investigates various aspects of sharing design knowledge and will produce a set of recommendations towards the application of this technology in the area of civil engineering and architectural design.

The Design Knowledge servers will be accessible for evaluation and testing at: http://www.DesignKnowledge.info.

9. ACKNOWLEDGEMENTS

This research project was initiated at the faculty of Architecture, Building, and Planning at Eindhoven University of Technology, The Netherlands. It is partly carried out at the department of Civil Engineering of the Instituto Superior Técnico, Lisbon, Portugal. The project is co-financed by the Fundação para a Ciência e a Tecnologia of the Portuguese Ministry of Science and Technology.

10. REFERENCES

Eastman C. M. and Fereshetian N. (1994). Information models for use in product design: a comparison. Computer-Aided Design Vol. 26, No 7, pp 551-572, July 1994.

Eastman C. M., Assal H., and Jeng T. (1995). Structure of a database supporting model evolution. In: *Modelling of buildings through their life-cycle. Proceedings of CIB workshop on computers and information in construction* (eds. Fisher M., Law K., and Luiten B.) Stanford University, Stanford, Ca, USA, August 21-23.

Ekholm A. and Fridqvist, S. (1998). A dynamic information system for design applied to the construction context, *Proceedings of the CIB W78 workshop The life-cycle of Construction IT*, 3 – 5 June 1998, Stockholm, Sweden.

Ekholm A. (2001). *BSAB och klassifikation för produktmodellering och design, Slutrapport förstudie 2001-04-10*, AB Svensk Byggtjänst (in Swedish).

Ekholm A. (2002). Principles for classification of properties of construction objects, *Proceedings of the CIB W78 workshop Distributing knowledge in Building*, 12 – 14 June 2002, Aarhus, Denmark.

EPISTLE (2001). EPISTLE Core Model version 4.0, http://www.stepcom.ncl.ac.uk/epistle/specifications/ecm/ecm_400.html (visited Dec. 12, 2001).

Fridqvist, S. (2000). *Property-Oriented Information Systems for Design, prototypes for the BAS·CAAD System*, PhD thesis, Lund University, Sweden.

Galle P. (1995). Towards integrated, “intelligent”, and compliant computer modeling of buildings. Automation in Construction. Vol. 4, No 3, pp. 189-211, 1995.

Garrett JR J. H. and Hakim M. M. (1994). Class-centered vs. Object-centered Approaches for Modelling *Engineering Design Information.Proceedings of the IKM-Internationales Kolloquium über Anwendungen der Informatik und der Mathematik in Architektur und Bauwesen*, pp. 267-272, Weimar, Germany, March 16-18, 1994.

Junge R., Steinmann R. and Beetz K. (1997). A dynamic product model. In: *CAAD futures 1997, proceedings of the 7th International Conference on Computer Aided Architectural Design Futures* (Ed. Richard Junge) Dordrecht: Kluwer Academic Publishers.

Kelleners, R.H.M.C. (1999). *Constraints in Object-Oriented Graphics*. PhD thesis, Eindhoven University of Technology, The Netherlands.

van Leeuwen J. P., and Wagter H. (1997). Architectural design by features. In: *CAAD futures 1997, proceedings of the 7th International Conference on Computer Aided Architectural Design Futures* (Ed. Richard Junge) Dordrecht: Kluwer Academic Publishers.

van Leeuwen, J.P. (1999). *Modelling Architectural Design Information by Features, an approach to dynamic product modelling for application in architectural design*, PhD thesis, Eindhoven University of Technology, The Netherlands.

van Leeuwen, J.P. and B. de Vries (2000). Modelling with Features and the formalisation of early design knowledge, *Proceedings of the Third European Conference on Product and Process Modeling in the building and related industries*, 25 – 27 September 2000, Lisbon, Portugal.

van Leeuwen, J.P. and A.J. Jessurun (2001). XML for flexibility and extendibility of design information models, *Proceedings of CAADRIA 2001*, 19 – 21 April 2001, Sydney, Australia.

van Leeuwen, Jos, Ann Hendricx, and Sverker Fridqvist (2001). Towards Dynamic Information Modelling in Architectural Design, *Proceedings of the CIB W78 workshop 2001*, 30 May – 1 June 2001, Mpumalanga, South Africa.

de Vries, B., et al. (2001). The VR-DIS Research Programme, Design Systems group, *Proceedings of the Computer Aided Architectural Design Futures Conference 2001*, 8 – 11 July 2001, Eindhoven University of Technology, The Netherlands.

Woestenenk, K. (2000). The LexiCon: an update, *Proceedings of the Third European Conference on Product and Process Modeling in the building and related industries*, 25 – 27 September 2000, Lisbon, Portugal.