Fridqvist, Sverker and Jos P. van Leeuwen. **2002**. "Serving building product information with design knowledge servers." In *Proceedings of the European Conference of Product and Process Modelling*. Portorož, Slovenia: 9 – 11 September 2002, 301-306.

Serving building product information with design knowledge servers

S. Fridqvist and J.P. van Leeuwen
Eindhoven University of Technology, Eindhoven, Netherlands

ABSTRACT: Building product information is a valuable resource for building design, but the number of different producers and products makes it hard for a single person to keep informed of all available products. A solution to this problem would be computer-based product information repositories, from which designers could fetch up-to-date information. The Design Knowledge Server (DesKs) approach has properties that make it function as such a product information repository. Compared to other approaches, it provides enhanced functionality that will benefit especially architects in early design phases. The paper introduces the main characteristics of DesKs and describes the way design information is modelled in this approach. It focuses on how a technology called Feature Type Recognition (FTR), which is part of this development, can be used to find product information.

1 INTRODUCTION

To find applicable components to become parts of a design is one of the main tasks of a designer (Coyne et al. 2001). This is true for all kinds of design, for mechanical engineers as well as for architects. Alas, the multitude of producers making similar products makes it an impossible task for any individual to maintain a good overview of it all. This has promoted the idea of creating computer-based repositories for component information to assist designers to find the products they need.

Building design generally involves many persons that are geographically dispersed. To enable all actors to work within one environment would greatly improve collaboration. The design knowledge server (DesKs) technology is a means to achieve such a common environment. Additionally, its basic functionality allows a DesKs server to act as a product information repository.

2 BUILDING PRODUCT INFORMATION

Building products are manufactured to become parts of buildings and are offered for sale on a general market, i.e. they are mostly not designed and manufactured only for a specific building. Building product information is published by the manufacturers to advertise the products, and to guide building designers in their choice of components for their design.

Traditionally, building product information has been distributed as printed matter, featuring texts, pictures, and drawings. With many similar products from several vendors on the market, it is hard for the individual designer to have knowledge of them all. This has created a market for specialised 'information brokers', who collect, organise and distribute building product information. However, even with the aid of brokers, designers have to study the printed information to learn what specific products may apply to the current design.

With the advent of CAD as the dominant medium for describing building designs, many producers have started to supply information also in CAD formats. This enables the designer to insert symbols or 3D-representations of products into drawings and models of the building to be (Coyne et al. 2001). Additionally, the Internet has enabled a direct channel from building product manufacturers to designers. However, brokers still have a mission to fill to organise the material, and the designer still must manually study the information to be able to find and select a product. This area would benefit from further computer assistance (Augenbroe, 1998). If product information were provided in a suitable format, computers could provide automated searches for products with the properties required for a design.

3 DESIGN KNOWLEDGE SERVERS

A *design knowledge server*, DesKs, is a networked server that manages and distributes design knowledge, which is created and accessed using client applications. The main rationale for the system is to support collaborative design, but it will also serve other purposes such as creating repositories for public design information. For an extended presentation of DesKs, see (van Leeuwen and Fridqvist 2002).

DesKs are based on the Feature Based Modelling (FBM) framework, developed by Jos van Leeuwen at the Eindhoven University of Technology (TU/e) (van Leeuwen 1999). The DesKs as well as the FBM framework is sorted under the VR-DIS research pro-

gramme of the Design Systems group at the TU/e (de Vries et al, 2001).

DesKs servers are envisioned to create a network where information is distributed and can be searched. The FBM framework supports a layered structure of information, allowing type definitions residing in one node of the network to reference type definitions in other nodes. This way, both new types and specialisations of previously published ones can be defined and published. This allows building product information for specific products to be created as specialisations of generic types that define the general properties of the products.

It is our conclusion that a DesKs server could function as the "building product information gateway" to serve designers with product information that has been described by Augenbroe (1998).

The DesKs technology supports both modelling of generic properties and searching for products that carry them. It also provides a programming interface to allow applications that handle IFC and PLIB data to become parts of a DesKs server. This makes the DesKs a plausible candidate for creating Augenbroe's building product information gateway.

3.1 *Publishing product information using DesKs*

Using the DesKs technology to publish product information has a number of advantages for product suppliers. General advantages of online publications apply, such as the ability to provide the most up-to-date information the moment it is requested. There is no delay in the process of publication and communication like there is with, for example, printed documentation. This ensures up-to-date information about the product itself, but also offers the possibility to include accurate information about availability, pricing, delivery times, logistics, conditions, etc.

In addition to these advantages, DesKs provide the opportunity to have a direct interaction between suppliers and designers. Clients connect to the supplier and communicate design problems through formal models. Suppliers can provide design solutions that represent a product or a range of products. The solutions can include information for the particular design case, such as alternative design details, instructions for construction and assembly, case-specific information concerning costs, delivery, etc. The designer can use this information directly in his design model, and communicate it with his project-partners. The DesKs technology is generally based on reference, thus the supplier data will remain within the supplier's responsibility.

4 THE FEATURE BASED MODELLING FRAMEWORK

The following part is a brief explanation of some important aspects of the feature based modelling (FBM) framework; for an in-depth description, the reader is directed to other accounts of the FBM framework, in particular (van Leeuwen 1999).

The FBM framework supports design by allowing the user to model both generic concepts and individuals. This makes the FBM framework a foundation for the kind of flexible tool that has been seen as necessary by many researchers, and which is also exemplified by other systems such as EDM2 (Eastman & Jeng 1999), the BAS·CAAD system (Ekholm & Fridqvist 1998) and the SOFA system (Galle 1994).

4.1 *Property-oriented modelling*

The FBM framework is a property-oriented approach to modelling design information (van Leeuwen et al 2001). Property-oriented information systems are characterised by their focus on the properties of the things to be modelled, e.g. 'fire resistance', 'mass', or 'colour'. This distinguishes these systems from class-oriented systems, which base modelling on classes of things, such as 'wall', 'window', or 'floor slab'. Property oriented systems, like FBM, allow objects under design to be specified incrementally, i.e. the designer can add specifications as they become known through the design process, since partially defined models are allowed.

4.2 *Complex features are defined through components*

In the FBM framework, generic concepts are modelled through *feature types*, and particular individuals through *feature instances*. Thus, a model consists of a collection of interrelated feature instances, which refer to generic concepts modelled as feature types.

Features that carry higher-level meaning are created by combining lower-level ones in a structured manner. Complex features combine other features, both at the type level and at the instance level, through *components* that connect the complex feature to other features. Components are *named*, and can have one of three *role types* that define the relation between the higher-level feature and the lower one, i.e. decomposition, specification or association. Components thus define what roles the connected features play in the context of the complex feature.

At the type level components additionally include *cardinality* information, which defines the lower and upper limits of the number of instances that, at the

instance level, should be related through one single component.

4.3 *Subtype-supertype hierarchies*

Complex feature types may be arranged in subtype-supertype hierarchies. A subtype inherits all components of its supertype, but it may re-define them to become more specific. Similarly, a subtype inherits the constraint assignments of its supertype (see below).

4.4 *Constraints*

Constraint feature types define constraints on a generic level. They include a list of *typed parameters*, and an *expression* that defines the actual constraint.

Constraints serve two different but related functions: at the type level, when used in a complex feature type definition, they form part of the semantic content of the complex feature type. At the instance level, constraints are also used to evaluate the model; the result is either *fulfilled* or *not fulfilled*.

Constraints can be used to model a thing's *structure* at the type level. The ability to define the structure is necessary to model functional properties, since these are part of the structure. The structure of a thing can be parted in the internal structure and the external structure, where the internal structure is the complex of relations among the parts of the thing, and the external structure is the complex of relations between the thing and its environment (Ekholm & Fridqvist 1996).

To highlight how constraints are used to define the structure of things, we will use the following example, illustrated in Figure 1. In this case, we want to define a type that would be instantiated as to the right of the figure, having two parts where part *A* has an *a*-relation to part *B*. The corresponding type is illustrated to the left.

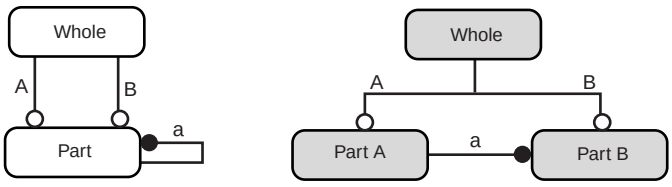

Figure 1 Modelling structure in FBM; type level to the left and instance level to the right. (FBM graphical notation.)

The reader should note that the feature type *Part* to the left only defines that the component *a* should be filled by some instance of type *Part*, not what particular instance. Apparently, just adding the two components *A* and *B* to the type *Whole* is not sufficient to ensure the desired outcome. The necessary additional piece of information can be contributed by adding a constraint to feature type *Whole*, to constrain the sub-component *a* of component *A* to be identical with component *B*.

Apparently, using constraints to define structure requires constraint expressions to be able to access the interiors of components. Support for this has been included in the implementation.

5 FEATURE TYPE RECOGNITION

Feature Type Recognition (FTR) is a technique made possible by the FBM framework. Through FBM, a data item that represents an individual can be recognized as an instance of a type, by comparing the components of the individual with those of the type.

While a feature instance is created based on a specific feature type, to provide the needed flexibility of modelling the user is free to change the components and to add new ones. Thus, after a while a feature instance may no longer be a true instance of the original type. As a result, the higher-level meaning of the instance may have become hidden, i.e. it is not explicit, but only implicit in terms of the combination of its lower-level concepts as defined by the components. To make the meaning explicit, a higher-level concept needs to be found that represents the actual meaning. This is achieved through feature type recognition (FTR).

The FTR function can be initiated either by the (human) user, or by some application. For the user, FTR can clarify the meaning or interpretation of a model that has undergone many changes, as said above. It can also suggest further development of the design by showing various ways to make a general model more specific, or vice-versa. This functionality is utilised in the application part of the research project that runs in 2002.

For applications, FTR can support analysis of models by interpreting model features according to type libraries defined for the purpose of analysis, and thus making the model accessible to the analysis software. This functionality might also be used to translate models between different modelling schemas, to support data exchange.

5.1 *How FTR is implemented*

FTR finds the types that best cover the present status of a feature instance by comparing the components of the instance with the components of the types. The FTR algorithm has two phases. The first is to find applicable candidates, i.e. feature types that might be an appropriate type for the instance. The second phase is to select one or several of the candidates.

To find the candidates, the procedure illustrated in Figure 2 is followed:

1 Inspect the instance's all components.
2 Collect the *types* of all component fillers (2.1) as well as the fillers' *supertypes* (2.2).
3 For each type *t* from step 2, collect the types where *t* is a component. The result is the list of candidate types.
4 For each of the types from 3, add the subtypes to the candidate list.

A component will have one or more *fillers*, i.e. feature instances that represent a part or a property of the main feature instance. To be a candidate, a type must primarily have a component in common with the instance, i.e. the components should have identical names and role types. Secondly, the types of the fillers of the components must match. If both of these criteria match, the type is added to the candidate list.

The second criterion might introduce an element of recursion, since the correct types of the instance's fillers need to be known before the evaluation can be made. Currently, however, no typing of the fillers is done. The main reason for this is that currently the user needs to make the final selection of the type, and it would be too confusing to repetitively have to select the type of subcomponents. To be efficient, recursion needs automatic type selection.

To aid the user, the candidate list is ordered according to applicability. The ordering is based on several criteria, such as:

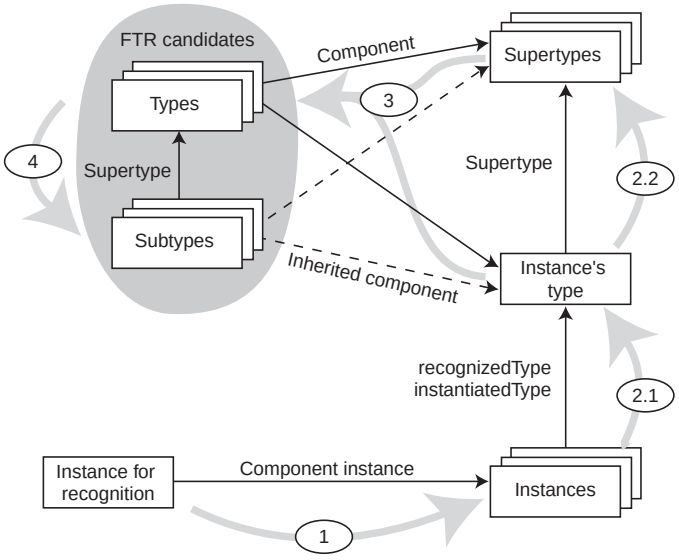

Figure 2 The algorithm for finding feature type candidates. Legend: Rectangles represent feature types and instances. Thin straight arrows represent relations defined by FBM. Thick arrows represent how the FTR process traverses the feature data.

1. How well does the candidate's set of components match the actual components of the instance?
2. How well do the candidate's components match the instance's component fillers?

For criterion 1, there might be an exact match of components between the candidate type and the instance, or the type may have more or less components than the instance. In case a type has components that the instance lacks, the type represents a *specialisation* of the instance. Choosing such a type implies adding the missing components to the instance, and thus adds information to the model.

The problem of how to automatically select types is studied in the second year of the research project (2002), and the subject is not ready for conclusions at the time of writing. Nevertheless, it can be said that the outcome of an automatic FTR process depends on what weights are given to the different matching criteria, and the problem consists of determining what impact different weights will have. Plausibly, different settings would be preferable for different purposes. What is already clear is that an automatic FTR process should not add information to a model. Thus, an automatic process will not consider candidates that by the manual selection might be presented to the user as suggestions to further specify the design model.

6 STANDARDS FOR PRODUCT INFORMATION

To allow product information searches, standards need to be developed for the purpose. We have studied two standards that are relevant in the context of building product information.

6.1 *PLIB – ISO 13584*

The ISO 13584 standard, called PLIB, is a standard for parts libraries, i.e. catalogues that organise information about standard components. This section makes a brief presentation of the PLIB standard. The information on PLIB was obtained from (Pierra 1997) and (Pierra et al 2000).

A PLIB parts library consists of two parts:

– a classification tree where component families and technical properties are identified and connected;
– a set of templates that describe successively each component family and each technical property.

Thus, the technical properties are closely connected to the component families, and vice versa. The close connection seems to make it difficult to define generic properties that are common for many otherwise dissimilar products. The following quotation confirms this: "Unfortunately, the world is not a tree, and it is impossible to associate properties to the class hierarchy in such a way that every common

property is only defined once, at some level of a hierarchy, and that it applies to all the sub-classes. The class where a property is defined being part of the identification of a property, two properties defined in two different classes would be two different properties." (Pierra 1997)

However, the PLIB standard is a generic framework, where different professions will define their own product hierarchies. It would thus in principle be possible to create several separate product libraries to cover different functional aspects for one range of products. Unfortunately, since there is no mechanism to make the correspondences between such libraries explicit, this approach would be quite obscure and difficult to maintain.

6.2 *Industry Foundation Classes - IFC*

The main current effort to define a building product modelling schema is the Industry Foundation Classes, IFC. The rationale for developing the IFC has been to provide "a basis for project information sharing in the building industry" (IAI website). Thus, the IFC is primarily a vehicle for communication, not a system on which to build design software.

IFC 2x is semantically a single hierarchy of classes that model objects recognised in buildings. These classes generally represent building components as viewed at the time of construction, i.e. when the functional design is complete. However, the classes are quite generic, and may not be sufficient to describe the differences of products well enough to support product libraries. In addition, IFC 2x is not designed to offer extendibility, e.g. for new products or construction methods.

6.3 *Conclusions*

Both PLIB and IFC 2x are based on single hierarchies of classes, and none implements modelling generic properties and functions. This makes it hard to include this kind of information to product information, and consequently to search for products based on generic properties or functions.

This, however, would not be a problem for the intended use of PLIB, since an engineer would very well know that he needs, e.g., a screw. His problem is rather to find the screw that best fits his purposes. DesKs addresses a different problem, where the solution of a design problem is less well known. Where PLIB assists in finding a specific product among a great number of nearly similar ones, the DesKs approach excels in finding products with a set of *generic properties*, i.e. properties shared by many different categories of things.

The DesKs core can be used by different applications. Thus it is feasible to create an interface for the PLIB standard that would allow DesKs users to search PLIBs on a more generic level. Similarly, an interface could be created for IFC 2x. Such interfaces would be accompanied by collections of feature types that semantically relate PLIB component families or IFC classes to each other and to other feature types. Such collections would also contain feature types to make the correspondences between the different classes explicit.

7 MODELLING PRODUCTS TO SUPPORT PRODUCT FINDING

To serve as an example for evaluation and demonstration, a small product database will be implemented in the DesKs application. Since this work belongs to the second phase of the project, conclusive results are not expected until later this year (2002). Nevertheless, some requirements that product finding puts on FTR are briefly presented here to highlight some of the complexities of implementing FTR in the context of feature based modelling. In particular, to successfully implement FTR does not only involve writing the FTR program code, but also to study how to use the FBM framework for modelling.

The practicability of the FBM framework, and thus of the DesKs technology, depends on well-defined systems of feature types to represent classes of things and properties that are useful for the different actors in the building process. Particularly to support product finding, the functional properties of a product need to be modelled in addition to other properties such as shape, material, colour etc. To define these feature types, however, is a task for international and professional organisations.

The FBM framework supports the "rich product semantics that provide complete product models with embedded links to relevant codes and regulations, specifications, geometry, assembly instructions, etc." that will be the foundation of electronic product catalogues (Jain and Augenbroe 2000). Such catalogues will, according to these authors, "offer added capabilities such as: dynamic updates; sophisticated search capabilities based on performance criteria, availability; multiple information sources; customisation based on user/firm preference. "

7.1 *Product finding for architectural design*

Architectural design differs qualitatively from much engineering design in one aspect, namely the importance of human values. While many engineering tasks can be rather precisely defined in terms of technical requirements, architectural tasks involve people and people's emotions to a high degree.

Since the latter cannot be as precisely defined as engineering requirements, the architect needs to have the possibility to be ambiguous during much of the design process. The ambiguousness can be understood as focusing on only one or a few of the properties that a component may have, for instance only the visual enclosing properties of a wall. For a product finding system to be fully useful to architectural designers, it needs to address this problem and support product finding for generic properties.

Research on design shows that designers evolve the design solution in concert with their understanding of the design problem (Gedenryd 1998). In other words, they actually begin by creating provisional solutions that they subsequently analyse and gradually refine. The ability to create provisional design solutions relies on the designer's knowledge of available options. The better his knowledge, the more likely it is that he makes good choices early on. A suitable product information catalogue could help the designer to learn about his options. Conclusively, the catalogue needs to be based on generic properties also for this purpose.

Because they are class-centred, neither PLIB nor IFC 2x are well suited to fill the roles we have drawn in these paragraphs. Instead, a property-oriented approach, such as the FBM framework, is needed. By allowing searching for products by generic properties, the DesKs technology could provide a powerful tool for designers.

8 CONCLUSIONS

To have knowledge of available components is important for designers to make good choices at early phases of design. Computer-based systems to assist finding product information might leverage the designer's abilities in this respect. However, to be useful in early stages these systems need to be searchable by generic properties, and not only by classes of products.

Standardisation efforts, such as PLIB and IFC 2x, are based on classes of products, and do not readily support searches on generic properties. Thus, these approaches are aimed towards searches at later design stages, but not directly useful for the early stages the DesKs approach addresses.

REFERENCES

Augenbroe, G. (1998) Building Product Information Technology, Executive white paper, Construction Research Center, Georgia Institute of Technology, 1998.

Coyne, R. , Lee, J., Duncan, D., and Ofluoglu, S. (2001) Applying web-based product libraries. Automation in Construction 10 (2001) pp. 549-559.

Eastman, C. and Jeng, T-S. (1999) A database supporting evolutionary product modelling development for design, in: Automation in Construction 8 (1999) pp. 305-323.

Ekholm, A. and Fridqvist, S. (1996) Basic Object Structure for Computer Aided Modelling in Building Design in: Turk, Ž. (ed) Proceedings of the CIB-W78 International Conference "Construction on the Information Highway", University of Ljubljana, Slovenia, pp 197-206.

Ekholm, A. and Fridqvist, S. (1998) A Dynamic Information System for Design Applied to the Construction Context in: Björk, B.C. and Jägbeck, A. (eds) Proceedings of the CIB W78 workshop "The Life-cycle of Construction IT", Royal Institute of Technology, Stockholm, 1998, pp.219-232.

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

Gedenryd, H. (1998) How designers work - making sense of authentic cognitive activities. Dissertation. Lund University Cognitive Studies 75, Lund University, Lund.

Galle, P. (1994) Specifying objects as functions of attributes: Towards a data model for design. Invited paper for the 7th international conference on systems research, informatics and cybernetics, Aug 15-21, 1994, Baden-Baden, German

IFC 2x website:
http://www.iai-ev.de/spezifikation/Ifc2x/index.htm

IAI website: http://iaiweb.lbl.gov/.

Jain, S. and Augenbroe, G. (2000) The Role of Electronic Product Data Catalogues in Design Management, presented at the CIB W96 Conference "Design Management in the Architectural and Engineering Office", May 19-20, 2000, Atlanta, Georgia, USA.

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., Hendricx, A., and Fridqvist, S. (2001) Towards Dynamic Information Modelling in Architectural Design. In: Proceedings of the CIB-W78 International Conference "IT in Construction in Africa" 2001. CSIR, Division of Building and Construction Technology, pp 19.1-14.

van Leeuwen, J.P. and Fridqvist, S. (2002) On the Management of Sharing Design Knowledge, in: Proceedings of the CIB W78 conference "Distributing Knowledge in Building", June 12-14 2002, Aarhus, Denmark.

Pierra, G. (1997) Intelligent electronic component catalogues for engineering and manufacturing In: Proc. of the Internat. Symp. on Global Engineering Networking GEN'97, Antwerp, Belgium, April 23-24.

Pierra, G. , Potier, J.C, and Sardet, E. (2000) From digital libraries to electronic catalogues for engineering and manufacturing. Downloaded from: http://www.plib.ensma.fr/plib/english/publications/publication.asp.

de Vries, B, Achten, H, Coomans, M.K.D, Dijkstra, J, Fridqvist, S, Jessurun, J, van Leeuwen, J.P, Orzechowski, M, Saarloos, D, Segers, N, Tan, A. (2001). The VR-DIS Research Programme, Design Systems group. In: Proceedings of the Computer Aided Architectural Design Futures Conference 2001, 8-11 July 2001, Eindhoven University of Technology, The Netherlands, pp 795-808.