A DYNAMIC PRODUCT MODEL
A base for distributed applications

RICHARD JUNGE
Faculty for Architecture, Professorship for CAAD
Technical University Munich

RASSO STEINMANN
Faculty for Civil Engineering, Professorship for Bauinformatik
Fachhochschule Munich

KLAAUS BEETZ
Dipl. Math.
Nemetschek Programmsystem GmbH, Munich

Abstract

The project work described in this paper is a part of the ESPRIT VEGA Project. It is related to two companion papers issued in this conference proceedings. ‘Product Data Model for Interoperability in an Distributed Environment’ (Junge and Liebich. 1997, this volume) and ‘The VEGA Platform’ (Junge, Koethe, Schulz, Zarli, Bakkeren. 1997, this volume) are describing the technological basis for an application modeled to capture and convert the working environment of architects and building engineers, in short: the building design team, to an computer environment. The ESPRIT projects are increasingly forced into “public and private risk funding and sharing policy. This part of VEGA is explicitly directed to exploitation of the EU funded project. This can be reached by a stepwise (small steps) transition from research to commercial implementation.

This paper is demonstrating the path VEGA project partners are heading. A major orientation point is an implementation based on the following technologies:

• use of object oriented design and implementation
• use of product data modeling
• dynamic schema evolution
• use of related standards as STEP, EXPRESS, IAI’s IFC, OMG’s CORBA, WFMCS Interfaces
• use of latest software engineering technologies as OO, COM/DCOM
• use of latest communication technologies based on Internet, Intranet, etc.
1. Introduction

Several research initiatives have covered the application of product modeling for the building industry. These efforts have proven the applicability and usefulness of product modeling theoretically. However, only little or no experience have been made with introducing it into commercial implementations. Furthermore, the use of information technology in the architectural and engineering domain is characterized by a large number of applications for different realms in a scattered industry. Interoperability between these applications is not supported, it is reduced to data exchange by file exchange using whatever format.

Product modeling is an acknowledged method of describing the specific definitions of very complex products, such as ships, airplanes or in -our domain- buildings. Buildings are, although of normally very simple geometry, extremely complex structures. This is not the limited view of a ‘building freak’. Buildings even when very simple are one-of-a-kind but consist of a multiplicity of components. These components are planned and constructed with the aid of many different applications. Each of these applications has its own data structure and generally the applications are incompatible with each other. Moreover, a common product model can not be derived from the specific data structure of all these applications.

Today’s modern communication technologies (Internet, Intranet, ISDN etc.) offer many possibilities for computer-based concurrent engineering. In order to use these technologies effectively there has be a holistic and unified product model providing the underlying semantic. The definition of product models is the goal of different initiatives, like STEP, IAI etc. is a critical opinion is that most members of these initiatives consider product modeling merely as an established concept with the sole, though important, purpose of semantic based data exchange. But why not go one step further and use product modeling also for application internal, rather than just inter-applicative requirements (Junge and Liebich 1995).

The vision of this project is an application environment which supports a building project during a whole life-cycle as a project server. This project server is able to store all relevant data of a building during its life time. It can be accessed by other applications using standards like AP225, IFC or others to come. It is using the Internet or some other network as transport medium. Furthermore, access through runtime using CORBA or COM/DCOM is a base functionality, thus enabling different client applications to behave as if their own internal data-structure were based on the same product model as the server’s. This means that every application supporting such a standard can work with the project server and can retrieve stored data or add new data. The project server acts as a turntable for all relevant data, taking data security and data integrity issues into account.
The key is in information sharing in an distributed environment and by no means in data exchange between applications. This seems to be a fundamental difference to many existing product modeling approaches, where data exchange is the standard goal.

Another vision is that of applications based on an integration platform for applications tailored for the building industry linked together at the same time distributed through the VEGA Platform under development in the ESPRIT project VEGA.

2. The base technology

Although the project described in this paper in general is clearly positioned in the area of architecture, building engineering and construction it’s first phase has been mostly occupied with the development of base technologies needed for the envisaged software solutions. The project does not claim to have invented the concepts of these underlying technologies. The achievements rather are on one hand bringing this number of concepts together to form an integral complete set and on the other hand to further develop this set of concepts to a very mature and stable state, which finally will allow to build commercial software solutions.

2.1 OVERVIEW

The core technology for the software application part of the project is pure product modeling technology as such. The advances made lie in a solution to overcome the normally rather static then flexible structures of product models. This kind of product model has named DPM Kernel (Dynamic Product Model Kernel). This name should indicate that while the application is directed to fulfill requirements for its use in the
building domain it generally is a generic solution from an IT point of view. The DPM Kernel consists of the following four basic components:

- **Instances** — are the concrete entities as stored in the data base.
- **Patterns** — are describing the properties of the instances. The properties may be attributes, methods or the ability to be part of an association.
- **Methods** — are describing the behavior of a pattern’s instance, e.g. when the value of an attribute or an associated object changes.
- **Filters** — are enabling the user to request and find any information stored in the model.

In order to realize how these four parts are helping to gain the high flexibility, openness and extensibility of the loaded schema, that mark the main benefits of the approach, it is important to understand the generic base technology of the implementation. This technology allows object oriented modeling of the knowledge of a specific domain, such as architecture, as well as Dynamic Schema Evolution (DSE). Dynamic Schema Evolution means that the once modeled part of the world is not kept in a static and determined schema. DSE is modification, creation and deletion of schema information at runtime.

The DPM is a meta model and because of that the components of the DPM Kernel are not ‘hardcoded’ classes of the entities of a specific domain. There is for example no class ‘wall’ or a class ‘room’ in the DPM. The traditional ‘hardcoded’ implementation strategy would result in the implementation of a class together with its attributes as data members and additionally its methods for every entity of a domain. As a consequence, the entities could only be changed in a new version of the application (Figure 2). Thus the whole model is static and can not be expanded or modified after loading it to the DPM Kernel (see Section 2.6).

<table>
<thead>
<tr>
<th>Schema</th>
<th>Harcoded</th>
<th>Generic</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENTITY building_object;</td>
<td>class building_object</td>
<td></td>
</tr>
<tr>
<td>name: STRING;</td>
<td>{</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>char*name;</td>
<td></td>
</tr>
</tbody>
</table>
| END ENTITY | ... | }

Figure 2: Generic vs. hardcoded approach

But the fact that the project didn’t stick to the traditional ‘hardcoded’ concepts doesn’t mean the paradigm is a non object-oriented one. In contrary, all important principles of the object oriented paradigm (Stoustrup 1992), such as abstraction, inheritance,
information hiding or polymorphism are also fully available in the DPM Kernel (see Section 2.3).

The main constructs of the DPM Kernel are the patterns. Patterns are equivalent to classes in the traditional ‘hardcoded’ concept. They are providing the functionality to have attributes, methods and to be part of associations. Such a pattern gets its specific meaning of being, e.g. a wall, by a process called Dynamic Data Typing (DDT). Dynamic Data Typing is a functionality provided by the DPM-Schema-Loader (see Section 2.6).

2.2 DPM INSTANCES

DPM Instances are similar to C++ objects in the traditional hardcoded approach. The big difference to the hardcoded approach is that particularly, their internal structure, described by their attributes can be modified at runtime according to their pattern information. In the DPM Kernel four kinds of instances are available:

- objects — represent any concrete entity of a specific domain e.g. walls, windows, rooms in the architecture domain.
- attributes — store values (can be of various types, string, integer etc.) describing the internal state of an instance.
- associations — represent relationships among objects and store information about the relationships.
- taxonomies — manage a collection of subcategories as well as a collection of objects or attributes (targets of the arrangement) belonging directly to this category.

2.3 DPM PATTERN

Patterns are similar to C++ classes objects in the traditional hardcoded approach. They are dynamic class representations describing the properties and the behavior of their instances. The properties may be attributes, methods or the ability to be part of an association.

But in contrary to the ‘hardcoded approach in the DPM approach an attribute or a method can be added to or deleted from a pattern at runtime, even if instances of this pattern exist. A pattern may be derived from another one, in the sense of the object oriented paradigm [2], and thus they may form hierarchies. According to the inheritance principle of the object oriented paradigm, attribute inheriting and overwriting and method inheriting and overwriting is supported.

Every pattern has its name which is unique throughout the model and its attributes and methods have names unique in the name space of the pattern.

Like the instances, the DPM Kernel distinguishes four kinds of patterns:
• object patterns — describe any entity of a specific domain e.g. walls, windows, rooms in the AEC domain
• attribute patterns — describe the attributes of an object pattern by specifying e.g. the data type, default value, unit etc.
• association patterns — serve for modeling relationships among objects
• taxonomy patterns — enabling instances to be structured in any hierarchical order, e.g. the topology ordering of a building in floors or in any other functional decomposition

2.3.1 The DPM Object Pattern

DPM object patterns are described by generic attribute patterns and methods. Moreover they can have relationships to other objects, so called associations. One can distinguish between one_to_one, many_to_one, one_to_many and many_to_many associations. In the example in Figure 3 the object pattern “Wall” is described by a set of attributes. It has a self referenced association describing the connection of two or more walls.

As shown in Figure 3 the geometry is an attribute among others just like the material. This is a major difference to traditional CAD systems, where an object can only exist by virtue of one favored geometrical representation. All other descriptions, especially all non-geometric attributes are regarded as peripheral ones, having only a supplementary nature.

In the DPM approach it is possible to have more than one geometrical attribute, e.g. a set of parameters, describing the geometry, a polyhedra as result of the geometrical parameter set, a polyhedral, including the recesses and openings, thus taking the information from associations into account.

![Figure 3: Example of the DPM object pattern wall](image)

2.3.2 The DPM attribute pattern

In the DPM-Kernel two kinds of attribute patterns are distinguished, simple attributes and complex attributes. A simple attribute pattern has a common data type like string,
real, date, currency, etc. and the corresponding DPM-attribute instance holds the actual value. Whereas a complex attribute pattern has reference to other attribute pattern as data type. These referenced patterns can be simple and complex attribute patterns. The corresponding attribute instance has a reference to the other attribute instance as a value. Thus, attributes are built that include a whole tree structure with simple attributes at the leaves of the tree. In the example in Figure 4 the complex attribute “Material” consists of two simple attributes, “Name” and “Description” and another complex attribute, called “Physical properties”, which itself contains the two simple attributes, “Moment of inertia” and “Thermal conductivity”.

Figure 4: Example of a complex attribute pattern

2.3.3 The DPM association pattern

The association pattern has the same abilities as a common DPM pattern itself. This means the associations can be provided with attributes and methods and can be part of another association, as well. This ability may be used to describe the relationship of two objects and to specify the behavior of the objects, whenever an associated object is changed. For example, the “connected_with” association in Figure 3 has some attributes and rules, describing the contact of two walls. Moreover, it has some methods calculating the resulting polyhedra of the two walls, according to the attributes and rules that describe the contact behavior.

The ability to be part of another association can be used to model dependencies between associations. For example, in an early stage of the design the association “connect_with_rooms” determines that two rooms should have a connection. The association “connect_rooms” describes the concrete physical connection of the rooms in a later stage of the design. These two associations might be associated with another association which thereby expresses that the first association requires an association of the second kind.
2.3.4 The DPM taxonomy pattern

What applies to the associations holds for taxonomy patterns too. They are common DPM patterns and can be provided with attributes and methods. A taxonomy pattern’s purpose is to describe the hierarchical arrangement of a set of DPM instances by categorizing them into a tree. A taxonomy pattern manages a collection of subcategories as well as a collection of DPM instances (targets of the arrangement) belonging immediately to this category. A taxonomy forms a tree which consists of taxonomy nodes and leaves. A taxonomy node refers to a collection of other nodes, to a leave node. A taxonomy leave contains only a collection of DPM instances.

The DPM Kernel allows the simultaneous use of more than one taxonomy. If for example STEP AP225 is loaded into the DPM, the building is structured in building sections or levels accordingly. Additionally, the DPM Kernel offers the use of any other taxonomy, which structure the building in any other way, e.g. in a functional hierarchy structuring the rooms in departments or functional units. Consequently, the methods can be used to implement rules that determine the behavior of the taxonomy node or leave, e.g. the taxonomy node ‘building’ only refers to nodes of a type ‘building_section’ or a taxonomy leave should only contain rooms.

2.4 THE DPM METHODS

The methods of a DPM pattern have to be implemented as a class in the sense of a name space. All the classes with the implemented methods of all DPM patterns are available for the DPM Schema Loader in dynamic link libraries.

A common DPM pattern provides several hooks at which the methods can be attached as callbacks. These hooks define the execution time of the methods, e.g. on creation, on deletion, when an attribute is updated.

An attribute may be provided with a domain rule. This domain rule is called by every assignment of a value. This domain rule can contain any functionality, e.g. it can prove that the value of the attribute “length” is always greater than or equal to zero.

Besides these predefined methods, every DPM pattern can contain other methods, which may be called within the hooks methods or by application code.

2.5 THE DPM FILTERS

Filters are similar to database queries. They allow a set of DPM Kernel objects to be selected that satisfy one or more conditions.

The DPM Kernel provides three classes of filters:

• object filters query a set of objects specified by a condition on the attribute set
• association filters query a set of objects associated with a specific object by an association and which satisfy some condition of the attribute set of the object or of the association.
• taxonomy filters query a set of objects classified under some taxonomy node and which satisfy some condition of the attributes of the object.

Furthermore these three can be regarded as ‘atomic’ filters which can be combined to compound filters using set theory operators (union, intersection, difference and complement). To speed up performance, objects and attributes can be supplied with indices that improve the execution time of filters, sometimes by several magnitudes.

2.6 THE DPM SCHEMA LOADER

The DPM Schema Loader fills the abstract data structures of the DPM-Kernel with the knowledge of a specific domain. Hereby the knowledge of a specific domain is formulated as an EXPRESS schema. This schema is loaded in our application and the methods that implement the functionality of the schema have to be available in a dynamic link library.

The DPM-Schema Loader first parses the EXPRESS schema, then a process is started, called Dynamic Data Typing (DDT). This process of dynamic data typing transforms a common DPM pattern into a specific pattern according to the EXPRESS schema, e.g. transforms the EXPRESS entity “Wall” with all its attributes to a wall pattern and adds the functionality to the wall pattern. The same occurs with associations which are described in the EXPRESS schema, they were also converted to DPM associations with all its attributes and the methods were added to the pattern. After the schema loading instances to the patterns could be created or loaded from an physical file.

2.7 CONCLUSIONS

With these four basic components in place the semantics of any specific domain can be described and stored. Moreover, description of the domain is not only a static one. It is dynamic in two respects.

First, methods are used to describe the behavior of the instances and associations are used to propagate the changes of an instance to the associated instances. Secondly, with the generic approach the model can be extended even at runtime.

The base technology described above is an abstract technology for modeling the knowledge of any specific domain. The problem is reduced to a appropriate formalized description method for the knowledge of the domain. This is the place for the product modeling technology itself.
A specific schema of the building domain is loaded in the application as it currently is, which includes the construction and the semantics of a building. But, it has to be emphasized that in principle any schema formulated in EXPRESS can be loaded. To get real instances for the model, standards from STEP and IAI are used. In the current version the AP225 protocol is used to read in a whole building from any CAD system, which supports the AP225. An IFC interface is also existing as a first prototype.

The next step will be to distribute the information stored in and managed by the DPM Kernel over wide area networks, so that architects and engineers can work concurrently on the building model and have access to it regardless where they are working. We see the forthcoming COAST Platform from the VEGA project to be the appropriate way to do this. Therefore the DPM Kernel provides also an API, through which it can communicate with the COAST-platform.

3. The application

One can discuss the value, the potential extensions of functionality that DSE could have to end user' applications against the dangers of creating chaotic situations in an end user environment and the dangers DSE could have in information exchange. Without any doubts, however, are the advantages it opens in application development. The DPM constitutes a powerful and flexible basis for application development on top of a product data models. Together with foundations described in ‘Product Data Model for Interoperability in an Distributed Environment’ (Junge and Liebich. this volume) and 'The VEGA Platform’ (Junge, Koethe, Schulz, Zarli, Bakkeren. this volume) it
provides possibilities for new application software in the architectural domain. Only a few of those possibilities can be touched in the following.

The DPM itself is a neutral implementation method of any Express schema totally independent of any specific domain. What is needed to make it’s use specific and useful to the architecture and building engineering domain is a core model of the kind described in ‘Product Data Model for Interoperability in an Distributed Environment’. It follows a strategy to provide a core of semantics to allow domain models to be ‘plugged in’ to it. This core, in the project called ‘BPM Kernel’ (Building Product Model Kernel) provides the necessary semantic for communication between these domain models. Without such a level of commonly agreed semantics a technique like DPM would be useless in an implemented environment. The dynamically under runtime created extensions of a product model schema would not make any meaning to anybody besides the creator himself.

The applications using the DPM, as well as the VEGA platform together with the described product data model, currently under investigation are pointing in different directions, which are:

- CAAD kernel
- Coordinator Workstation
- A tool set (the current prototype implementation)

3.1. A CAAD KERNEL

Current CAD systems are ‘geometric’ systems not object oriented systems. The centerpiece of all these systems is the ability to create and manipulate geometrical objects. The only way to provide some meaning to these objects is to simply label these geometrical objects with names and in some cases also with attributes. A label, as for instance, wall or beam does not make more difference to the computer as that these two strings differ from each other. The objects themselves still remain to be pure geometrical objects. An architect or engineer however, when designing does not think in geometrical objects but in objects of his design task.
A computer could become a design assistant for architects and engineers only when he would know not only about drafting objects but rather about design objects and their meaning in the architects professional world. Traditional CAD Systems are the assistants of the draftsman. A new CAAD generation should become the assistants of the architects. For such a new CAAD environment it is the first prerequisite that the system would behave like a design assistant and not as a drafting tool. In (Junge 1995) and (Junge 1997) the author has described some functionality’s to be fulfilled by such a new CAAD system:

- **Continuous refinement.**
  The design process is often called to go from the coarse to the fine or from vagueness to precision. This is a process of continuous refinement starting from the first conceptual ideas, where only vague representations of the designed objects are possible to be made to more and more precise definitions of the design. This clearly is a top down approach, but exactly this is in contradiction to the bottom up approach of today’s CAD systems.

  ![Diagram of continuous refinement](image)

  **Figure 7** Aspect of continuous refinement

- **Extendibility.**
  The world of building elements seems to be infinite. There are so many that no one could bring them into one product model schema. In a schema only definition of classes down to a certain level can and should be defined. This however makes mechanisms necessary to extend definitions provided by a model. Design objects clearly have the tendency to be multifunctional and to be used for purposes they
• originally where not meant for. This is happening during design but more often the ‘use phase’, in facility management. Mechanism that allow such changes in the schema are necessary. See also the many papers Eastman (Eastman, C. 1995) is describing his motivation for the EDM development.

• Multiple geometric representations
During the process of a gradual definition of the design, the representation and presentation of it’s form is having many expressions. For example it starts with simple symbols in early phases and goes in a stepwise manner finally to very precisely defined ones in shop drawings in late phases. The representation and presentation are not only dependent from phases of the design process but also from those engineering disciplines for which they are meant to be used. Consequently there are many valid representations of a design object coexisting at the same phase or time. For a certain task in structural analysis, for example, it may be appropriate to have only the center line of gravity and not the maybe the 3D volume definition of a girder. The representation of design objects is dependent from phases and disciplines, from the context in which they are used.

Figure 8 Aspect of multiple geometric representations

• Continous specification.
An architects thinking, especially in earlier phases of the design, is oriented to form and function, which seems to be merely geometry oriented, but in reality there is a second, parallel stream in the design work. This stream is dealing with all ‘non Gestalt’, the non geometric, properties of the design object. These properties are, although often regarded less important and less sophisticated, compared with the Gestalt properties, almost the majority. Not only the ‘Gestalt’ but also these non geometric properties are defined in a stepwise manner during the design process.
All these specifications are made during the drawing process but they are not stored in the paper drawing neither are they in the CAD data base. They are almost lost, unretrievable. This information that has to be reconstructed later for the BoQ for example and that is done by people who almost where not involved in the design itself.

- Dynamic shift of focus.
  During the design task the designer's interest is constantly shifting between various aspects of the design objects specification. So for instance at one moment he might be working one the functions to be fulfilled in a specific room and the relating spatial needs and adjacencies. In the next moment he might shift to the architectural appearance of the spatial arrangement, the surfaces, or finishes of the space boundaries, their structure color, etc.; as an interior design aspect. The next focus might shift from the bounding wall as a spatial aspect, to the wall core as a constructive element itself. A question is what are the implications if these four aspects being objects of their own, existing in separate systems as they in today’s CAD.

- Distribution and communication.
  Design is done by a design team, which for the purpose of a coordinated or integrated design has to have a very intense communication. All design tasks of the design team members are highly interrelated and dependent on each other. The basis for this communication between humans is a to a certain degree common understanding on a very high semantical level. That is the basis for to use a slogan, for ‘integration by communication’. A design computation environment, able to assist the design team, has to provide a means to communicate on such a high semantical level in an software-integrated manner.

3.2. A COORDINATOR WORKSTATION

The distributed environment envisaged to become reality by the VEGA Platform lets arise some questions to be answered before it can be successfully implemented in design offices. How can such a distributed environment be managed, be made transparent for the user? What are the functions needed? It seems to be an approach appropriate to be followed again is a recollection of processes and functions as in today’s design offices. In today’s practice a very important role in project work is on coordination. In larger projects a specific role is that of coordinating all participants, may it be in the home office itself, or with all engineering offices participating in the project. The tasks that such a coordinator has on his agenda have to be translated to this new computer tool. Candidate functions of the Coordinator Workstation are:

- Distribution and collection of models or model parts.
- Control of communication, network and workflow.
- Coordination of design tasks.
- Conflict detection and management.
These functions are having two diverse aspects. Today both are fulfilled more or less perfect in an environment depicted in figure 9. First is the aspect of specific engineering discipline ones. These are among others:

- Technical coordination of design tasks.
- Tracking and backtracking of design tasks and results and their state.
- Management of human resources assigned to the project tasks.

The second aspect is on information technology issues. This is a wide area spanning from tools for:

- Visualization of the technical processes:
  - Schema/model browser
  - Model viewer
  - Network viewer
  - Conflict detector

- Work flow management:
  - Work flow modeler
  - Work flow controller

- Communication flow management:
  - Translation, conversion
  - Dividing and assigning of selected schema and model parts
    - Distributor
    - Delta storage

3.3. PROTOTYPE IMPLEMENTATION

The prototype applications which are implemented today upon the BPM/ DPM-Kernel contain, among other features, three main parts. These parts are giving access to the stored building for the user:
• The Project Explorer
• The 3D-Window
• The Spreadsheet Window

3.3.1 The Project Explorer

The Project Explorer is making the DPM-Taxonomies provided by the BPM/ DPM-Kernel visible for the user. Apart from the functionality described in section 2.3.4 the Project Explorer allows different kinds of documents to be attached to every taxonomy node. These documents are also DPM-instances. Because of this, they have the same functionality as a common DPM-instance, e.g. they can have attributes. In the prototype version the following kinds of documents are available:

• 3D-View documents include a 3D view of the whole building or a part of it, with a specific camera position.
• Report documents include the result sets of filters in a spreadsheet, which may be designed by the user.

The User can browse through the whole building taxonomy as well as view and modify the attributes of the building elements at the click of a button. Moreover, it is possible to import the same building in different versions in the same project. Executing the same reports on the different versions will show the consequences of the changes done in the CAD application.

Figure 6: Example of the Project Explorer with a building taxonomy
3.3.2. The 3D-Graphic-Window

The 3D-Graphic-Window visualizes the geometric attributes. This means that the client gets a 3D-image of the whole building or of a part of it. It allows an animated navigation through the building. Identifying and picking of one or more elements is possible. However, the attributes of identified elements can be viewed and modified. Furthermore, it is possible to highlight all elements that are the results of a filter execution or that are identified in the project explorer. The attributes of these highlighted elements can also be viewed and modified.

![3D-Window with the whole building (perspective view) and 3D-Window with one floor (bottom view)](image)

Figure 7: 3D-Window with the whole building (perspective view) and 3D-Window with one floor (bottom view)

3.3.3. The Spreadsheet-Window

The Spreadsheet-Window shows the results of filters in a very comfortable way. The spreadsheet is fully EXCEL-compatible and offers all abilities like formatting, calculation etc., that are known from EXCEL. In the design window the user can define the whole layout of a report specifying which attributes it should include, defining how attribute values are sorted and so on. All defined filters can be stored in a catalog categorizing them under specific groups e.g. “All room filters”. So a previously defined filter can easily be used in another project.
### References

Structure of a Product Database supporting Model Evolution. CIB Workshop proceedings, Stanford University.

Product modelling for applications: Model for next generation CAAD, Computing in Civil and Building Engineering, Pahl & Werner (eds), Balkema, Rotterdam

Integration by Communication, 1st International Symposium on Building System’s Automation-Integration. Conference proceedings, Univ. of Wisconsin, Madison

Aspects of New CAAD Environments. CIB Workshop proceedings, Stanford University.

Junge, R (1997)  
Building Product Model for Architectural Design Computing, Kluwer. in print

Stroustrup, B.. (1992)  
The C++ programming language, Addison-Wesley Publishing Company, Inc.