

# Object-Oriented Integration of Product and Simulation Models in the Building Industry, using Object-Oriented Databases

*PhD student Johnny N. Holst  
Norwegian University of Science and Technology (NTNU),  
Department of Energy and Process Technology, Trondheim, Norway*

## **Abstract**

This paper describes a product modeling technique that can be used in development of computer-aided decision making tool for the building industry. The paper starts with an introduction to modeling and a brief description of the differences between relational database management system and object-oriented database system, and follows with an extensive presentation and discussion of the object-oriented modeling techniques.

## **Introduction**

The global demand for more sustainable development has resulted in an increasing number of new technologies and design strategies aimed at improving buildings with respect to a variety of performance considerations, such as energy, comfort, cost, aesthetics, environmental impact, etc. As the number of design and technological options increases, so does the complexity and the cost of deciding which combination that is most useful for a given design. Informed decisions require the management of huge amounts of information about the combinations and their performance. Manual management methods are almost impossible to use. As a result, most design decisions are only partially informed, resulting in missed opportunities and often unaccounted, undesired effects [1].

The rapid advances in information technologies give opportunities for the development of computer-based tools that may significantly improve decision-making and facilitate the design process. Successful implementation, however, requires comprehensive understanding of the design process and efficient data flow between existing design and simulation program solutions. This development is driven by the increasing demand for fast, convenient and fault-tolerant data exchange and data sharing among usually isolated program systems within the planning processes.

The planning process of modern buildings is characterized by the cooperation of many involved specialists, very often not affiliated to the same company. The required information has to be exchanged in a standardized and generally accepted way to reduce the costs and efforts spent to coordinate the project partners and to transfer data amongst them.

Currently, this exchange is normally performed by using print-out documents and two-dimensional drawings. The loss of information due to data exchange using drawings can result in misinterpretations and inconsistencies, influencing the quality of the whole planning and construction process. Other considerations are the requirements for authenticity, identification, safety of access and legacy aspects of the cooperation.

Electronic data exchange and data sharing are two orthogonal methods to address the requirements of the information technology based integration to the concurrent and distributed nature of the design process of buildings. Data exchange addresses the need for delivery of

approved revisions of final design results. Data sharing supports for all involved parties, the permanent access even to intermediate iterations within the design process. The design process demands support for both technical approaches.

### **Requirements for integrated approach**

Usually, a number of application programs are utilized to support the engineering tasks during different design stages. Different aspects of detailing and different in-depth support for highly specialized subtasks have led to highly diversified and sophisticated CAD and simulation data models. These data models are developed independently by specialist software vendors. Therefore, the external representation of this data is not straightforward map able among different applications and is not necessarily compatible.

Nevertheless, in order to improve the quality of the design of buildings, to reduce the cycle time for the planning process, to meet the requirements of demanding customers, and to reduce the costs of planning significantly, information of different involved partners in the planning process has to be integrated. The internationalization of the market has led to the internationalization of the planning process itself. Support for data exchange scenarios and concurrent engineering approaches will be essential for the economic success of the planning organizations.

Data exchange or sharing requires additional interpretation and partial re-entry of data. Particularly, non-geometric data are frequently loosed and cause unnecessary project coordination efforts. The common product model itself does not necessarily exist explicitly in its own. Proprietary solutions rely on the availability of the software and are usually not open for user driven extensions. Although, the file format itself defines standard ways for extending the data model, this approach is not seamlessly inter-operable and the vendor is in the position to re-define the standard on his solely decision.

### **Standards for product modeling**

To overcome the dependency on proprietary solutions standardization offers significant advantages. Already in 1984 the International Standard Organization (ISO) started to establish the Standard for the Exchange of Product Model Data ISO 10303 commonly known as STEP [2]. The standard itself comprises a formal modeling language (EXPRESS), application programmer's interface for STEP database access and application protocols [3]. The API methods define bindings of database access functions to standard programming languages, e.g. C, C++, Java, and CORBA IDL. The application protocols are developed utilizing integrated application independent resources. The relevant underlying generic resources like Part 42 (geometric and topological representation), provide profound support for modeling of complex building geometries.

Other standard boards, like the Object Management Group (OMG) which is dedicated to develop the inter-operability middle-ware standard CORBA for standardized client-server applications, are just about to develop accompanying specifications, for instance the CAD-Enabler [4]. The approach to define business and engineering object interfaces reflects the need for harmonized interface models for inter-operable and distributed program solutions. The nature of the OMG specifications is focused on the operational aspects rather than the data model centric approach followed by STEP. Therefore, both standard developments fit neatly together. They cover different aspects, and have complementary focus on the data exchange and data sharing problems described above.

Alternatively, solutions to the integration problem are offered by the industry alliance for inter-operability (IAI) [5]. The organization is a non-profit association of software vendors

mainly influenced by Autodesk Inc. and developments committed to AutoCAD applications. The member board is specifying so-called industry foundation classes (IFC) for commonly used construction industry objects, their geometry and attributes, including non-graphical information. IFC addresses both the data modeling requirements by specifying an EXPRESS model and the computational aspects by defining an IDL interface and an object-oriented API. The approach borrows from the results of the previously described STEP development. The implementation of the specified models is the responsibility of the software vendors.

### **Integration by common data model**

It is clear from the investigation of the different steps and phases in the building design process that not one single model is powerful enough to reflect all the necessary facets and possibilities for the design of a building. The complexity of the modeling approach may be reduced by use of several sub-models. By splitting the complete task into appropriate subtasks a system of complementary models can be defined. Based on inter-operable interfaces the sub-models can be combined to fulfill the requirements of an integrated approach supporting seamless data exchange and sharing.

The sub-models can be stored in a relational database management system or an object-oriented database system. The whole design process is characterized by many steps of iterations between the sub-models. It is important to store all iteration steps so that all the history of the design process can be reproduced. The database system may be extended by middle-ware technology to serve as a data warehouse.

### ***Use of Object-Oriented Databases versus Relational Databases***

Since the sub-models can use either a relational database management system or an object-oriented database system, it is necessary to look at both systems to find out which system is favorable to use. In the following text, the underlying information model differences between RDBMSs and ODBMSs will be examined in order to understand their differences [4, 6, 7].

#### **Relationships**

In a relational system, to represent a relationship between two pieces information (tuples, rows), the user must first create a secondary data structure (foreign key), and store the same value of that foreign key in each structure. Then, at runtime, to determine which item is connected to which, the RDBMS must search through all items in the table, comparing foreign keys, until it discovers two that match. This search-and compare, called join, is slow, and gets slower as tables grow in size. Although the join is quite flexible, it is slow and is the weak point of relational systems.

To create a relationship in an ODBMS the user simply declares it. The ODBMS then automatically generates the relationship and all that it requires, including operations to dynamically add and remove instances of many-to-many relationships. Referential integrity, such a difficult proposition in RDBMSs, usually requiring users to write their own stored procedures, falls out transparently and automatically. Importantly, the traversal of the relationship from one object to the other is direct, without any need for join or search-and-compare. This can literally be orders of magnitude faster, and, unlike the RDBMS approach, scales with size. The more relationships, the more benefit is gained from an ODBMS.

#### **Varying-sized structures**

Another typical problem area in RDBMS is varying-sized structures (e.g. time-series data or history data). Since the RDBMS supports only fixed-size tables, extra structures need to be added, resulting in extra complexity and lower performance. In order to represent such

varying sized structures the user must break them into some combination of fixed size structures, and manually manage the links between them. This requires extra work, creates extra opportunity for error or consistency loss, and makes access to these structures much slower. Instead, in an ODBMS, there is a primitive to support varying-sized structures. This provides an easy, direct interface for the user, who no longer needs to break such structures down. Also, it is understood all the way to the storage manager level, for efficient access, allocation, recovery, concurrency, etc.

### **User-extensible structures**

A similar situation arises when a user wishes to alter a few rows in a table. Suppose, for example, that two new fields are to be added to 3 rows. The RDBMS user is left with two choices. He can enlarge all rows, wasting space. Or, he can create a separate structure for those new columns; add foreign keys to all rows to indicate which have these extra columns. This still adds some (though less) overhead to all rows of the table, but also now adds a new table, and slow joins between the two. The ODBMS user has no such problem. Instead, the ODBMS manages the changes directly and efficiently. The user simply declares the changes as a subtype (certain instances of the original type are different), and the ODBMS manages the storage directly from that, allocating extra space just for those instances that need it, with no extra foreign key or join overhead.

### **Flexibility and efficiency of structures**

Flexibility can be critical to many applications, in order to vary structures for one use or another, or vary them over time as the system is extended. The RDBMS structures are static, fixed, with hard rectangular boundaries, providing little flexibility. Changes to structures or additions typically are quite difficult and require major changes. With an ODBMS, the user may freely define any data structure, any shape. Moreover, at any time, such structures may be modified into any other shape, including automatic migration of preexisting instances of the old shape. Any new structures can always be added. Such structures in the ODBMS may be very simple, with just a few fields, or may be very complex. Using the varray (varying-sized array) primitive, an object can have dynamically changing shape. In fact, by combining multiple such varrays into a single object, very complex objects can be created.

ODBMS structures can also include composite objects, or objects that are composed of multiple other (component) objects. Any network of objects, connected together by relationships, can be treated as a single (composite) object. Not only may it be addressed structurally as a unit, but operations may propagate along the relationships to all components of the composite. In this way, object may be created out of objects, and so on, to any number of levels of depth. Since the relationships are typed, a single component object may also participate in a different composite, allowing any number of dimensions of breadth. Thus, arbitrary complexity may be modeled. More importantly, additional capability may always be added. Users may always add new composites that perhaps thread through some of the old composites' objects as well as adding some new ones, without limit. There is no complexity barrier or relational wall in complexity. In an attempt to allow storage of complex structures, RDBMSs have begun to add BLOBs, or Binary Large Objects. Unfortunately, to the DBMS, the BLOB is a black box; i.e., the DBMS does not know or understand what is inside the BLOB, as the ODBMSs do for objects. In fact, storing information in a BLOB is really no different than storing them in a flat file, and linking that to the DBMS. Flat files can be useful, but with a BLOB the DBMS cannot support any of its functionality internally to the BLOB, including concurrencies, recovering, versioning, etc. It's all left to the application. Similarly, in an attempt to support user-defined operations, some RDBMSs now support stored procedures. On certain DBMS events (e.g., update, delete), such procedures will be invoked

and executed on the server. This is much like executing methods in an ODBMS, except that ODBMS methods may apply to any events (not just certain ones, as in RDBMS); may execute anywhere (not just on server); and may be associated with any structures or abstractions. Also, many RDBMS features may not work with them, because they're not tables. Certainly, any robustness, limited scalability, etc., would not apply since they're newly added outside the RDBMS table-based engine.

Very similar to stored procedures are Data Blades or Data Cartridges. These are pre-built procedures to go with BLOBs. They add the ability to do something useful with the BLOBs. In this way, they're similar to class libraries in an ODBMS, except that the ODBMS class libraries may be written by users (data blades typically require writing code that inserts into the RDBMS engine); and may have any associated structures and operations. All these efforts (BLOBs, stored procedures, and data blades) stem from a recognition by RDBMS vendors that they lack what users need; viz., arbitrary structure, arbitrary operations, and arbitrary abstractions. Unfortunately, they take only a small step in this direction, providing only limited structures, predefined operations and classes. If the RDBMS were modified enough to generalize BLOBs to any object structure, and stored procedures to any object method, and data blades to any object class, it would require rebuilding the core DBMS engine to support all these, instead of just tables, and the result would be an ODBMS.

Data often changes over time, and tracking those changes can be an important role of the DBMS. The RDBMS user must create secondary structures and manually copy data, label it, and track it. The ODBMS user may simply turn on versioning, and the system will automatically keep history. This is possible for two reasons. First, the ODBMS understands the application-level objects, so it can version at the level of those objects, as desired. In the RDBMS, the data for an application entity is scattered through different tables and rows so there is no natural unit for versioning. Second, the ODBMS includes the concept of identity, which allows it to track the state of an object even as its values change. The history of the state of an object, of the value of all its attributes, is linear versioning. Branching versioning, in addition, allows users to create alternative states of objects, with the ODBMS automatically tracking the genealogy. In addition to supporting arbitrary structures and relationships, objects also support operations. While the RDBMS is limited to predefined operations (select, project, join), the ODBMS allows the user to define any operations he wishes at any and all levels of objects. This allows users to work at various levels. Some can work at the primitive level, as in an RDBMS, building the foundation objects, which others can then use without having to worry about how they're implemented internally. Progressively higher levels address different users, all the way to end users, who might deal with objects such as manufacturing processes, satellites, customers, products, and operations such as measure reject rate, adjust satellite orbit, profile customer's buying history, generate bill of materials, etc. As new primitives are added, the high level users immediately can access them without changing their interface.

### **Encapsulation of operations for integrity, quality, and cost reduction**

Encapsulating objects with such operations provides several benefits, including integrity, ability to manage complexity, dramatically lower maintenance cost, and higher quality. Integrity results from embedding desired rules into these operations at each level, so that all users automatically benefit from them. As we saw in the section above, RDBMSs allow (indeed, force) users to work at the primitive level, so they might violate or break higher level structures, or change primitive values without making corresponding changes to other, related primitives. The encapsulated operations prevent such integrity violations. The same remains true with GUI tools, because they, too, get the benefit of the higher-level objects with built-in operations.

Operations allow managing greater complexity by burying it inside lower structures, and presenting unified more abstract, user-level interfaces. Encapsulation can dramatically reduce maintenance cost by hiding the internals of an object from the rest of the system. Typically, large systems become complexity-limited because any change to one part of the system affects all others. This spaghetti-code tangle grows until it becomes impossible, in practice, to make more changes without breaking something else somewhere. Instead, with an ODBMS, changing the internals of the object is guaranteed not to affect any users of the object, because they can only access the object through its well-defined external interface, its operations. In addition, new sub-types of objects can be added, extending functionality, without affecting higher level objects, applications, or users. For example, a graphical drawing program might include a high-level function (and user interface menu or button) to redraw all objects. It would implement this by invoking the draw operation of all objects. Different objects might have very different implementations (e.g., circles vs. rectangles), but the high-level routine doesn't know or care, it just invokes draw. If a new sub-type of drawing object is added (e.g., trapezoid), the high-level redraw function simply continues to work, and the user gets the benefit of the new objects. Similarly, if a new, improved algorithm is implemented for one of the drawing object subtypes (e.g., a better circle drawing routine for certain display types), the high level redraw code continues to work as-is, unchanged.

Finally, objects encapsulated with their operations can improve quality. The old approach to structuring software, as embodied in an RDBMS, creates a set of shared data structures (the RDBMS) which are operated upon by multiple algorithms (code, programs). If one algorithm desires a change to some data structure, all other algorithms must be examined and changed for the new structure. Similarly, if an algorithm from last year's project turns out to be the one you wish to use this year, you must still copy it over and go through and edit it for the new project's data structures. Code reuse requires changing it, creating a new piece of software with new bugs that must be eradicated, and creating a new maintenance problem. Instead of building on past work, software developers are continually restarting from scratch. With ODBMS objects, however, the code and the data it uses are combined, so changes can be made to them together, without breaking or affecting the other objects. If an object from last year's project does what we need for this year's project, we can simply use it as-is, without copying and changing. It's already tested and debugged and working, so we gain higher quality by building on the work of the past. Even if the old object isn't exactly what we want, we can define a new subtype for that object (inheritance), specifying only the difference (delta) between the new and old. This gives the flexibility to allow reuse in practice, reduces the required new programming (and debugging and potential quality problems) to the much smaller delta, and reduces the maintenance by keeping the same, main object for both the new and the old system.

### **Complex queries**

Last, we'll address complexity. Some might look at all this information modeling capability, including versioned objects, composites, subtypes, etc., and ask what price is paid for this increased complexity. In general, no price is paid. In fact, there is a reduction in complexity as seen by the user. The question is really backwards. The complexity lies not in the DBMS, but in the application, in the problem it's trying to solve. If it is simple, then the data structures and operations in the ODBMS will be simple. If the application's problem is inherently complex, the more sophisticated modeling capabilities of the ODBMS allow that complexity to be captured by the ODBMS itself, so it can help. Instead of forcing the application programmer himself to break that complexity down to primitives, the ODBMS allows him to use higher level modeling structures to directly capture the complexity, and then manages it

for the user based on the structure and operations the user has defined. In short, the complexity is in the application, not the DBMS.

The same is true for queries. The RDBMS query system might look simpler, closed, easier to predict, and it itself is. However, the application's desired query is not. It must be translated from the natural, high-level application structures and operations down to the primitive RDBMS structures and operations, and the complexity is all in that translation. Instead, in an ODBMS, the high level structures and operations can be used directly in the query, and the ODBMS query engine executes the query itself, with no need for the user to translate to lower-level primitives. True, the resulting ODBMS query and query system is more complex, but that is because of the nature of the query. It was still true for the RDBMS-based complex application query. The only difference is who handles the complexity, the user or the DBMS.

### **Conclusion**

As mention in the text above, ODBMS give more flexibility and efficiency when dealing with complex, varying-sized and user-extensible structures. Complex relationships can easy be declared and managed. The complexity of a building model and its underlying product models implies that ODBMs would be the best choice when designing product models.

### ***Product models***

In this paper the term “product” is used to refer to building components and systems. While the primary focus of this paper is related to the building industry, I believe that these theories and techniques also will be of value to other industries as well.

### **Modeling theory**

In general we understand product modeling as the representation of a product in terms of parameters that reflect it's descriptive and performance characteristics [8]. Descriptive parameters, such as geometry, color, etc., are defined herein as those controlled by the designer (decision-maker). Performance parameters, such as comfort levels, energy requirements, etc., are defined as those that the designer uses to judge the appropriateness of the product. Context parameters are those used to describe the environment within which the product is assumed and evaluated. The values of performance parameters may depend not only on the values of descriptive parameters, but context ones as well. Modeling based on these parameters facilitates communication and supports testing applications of new and existing products [1].

Based on the above definitions, most activities in building design are forms of modeling. Currently, the most common models used in the building industry are drawings, such as plans, sections, elevations, isometrics, perspectives, etc., as well as physical scale models. These models adequately support the evaluation of spatial layout and aesthetic appeal and are usually complimented by mathematical models that address other aspects, such as structural, energy and economic performance.

### **Computer-based models**

Advances in computer applications over the last few decades have resulted in the gradual replacement of manual modeling with computer-simulation models. While computer-based models have been developed for a large variety of building performance considerations, computer-aided drafting models have been the most widely used in the building industry. Most others tend to be used mainly for research purposes, modeling performance aspects such as comfort, energy, and economics. Some of these models are able to address not only the building design needs, but construction and operational requirements as well [9].

Computer-aided drafting was originally developed to serve the needs of electronic circuitry design and typically generated very complex two-dimensional drawings. The same types of algorithms were later adapted for general drafting applications, including architectural and engineering drawings of buildings, their components and systems. The main limitation of the widely used drawing-based models is the distance between the very abstract, two-dimensional representations and the actual products the drawings suggest for construction. The major advancement in computer-graphics that facilitates the representation of three dimensional solids and the tools to create and visualize these objects brings us one step closer to representing the actual components of construction. Parallel to the developments in computer graphics, a large number of computer-based models, or simulations, are being developed by building researchers that address various aspects of building performance, such as comfort, energy, economics, etc. The development of such models over the past twenty years has been broad, with various levels of success in modeling capabilities and prediction accuracy. While most models were originally implemented on mainframe and mini computers, those that are still under development have shifted their development onto powerful workstations and personal computers. Developed primarily for research purposes only, most of these applications tend to be difficult to use. They require an extensive description of the building and its context and they provide output in the form of alphanumeric tables that are cumbersome to review and interpret. Research efforts in computer applications in the building industry during the last decade have focused on developing new models that will combine the capabilities of a large variety of existing models. These new models will provide for more cost-effective performance prediction of multiple design alternatives.

### **Object-Oriented Modeling Techniques (OOMT)**

Depending on the performance aspects being addressed (e.g., energy, esthetics, cost, etc.) design and simulation tools use different building modeling representations [10]. A thermal simulation program, such as EnergyPlus, uses a building description in terms of thermal barriers with properties such as U-value, solar transmittance, etc., required for heat transfer computations [11]. A lighting and rendering simulation program, such as RADIANCE, uses a building description in terms of geometric solids, such as cones, spheres, etc., with properties such as texture, visible reflectance, etc., required for illuminance and luminance computations [12]. Even if simulation programs such as EnergyPlus and RADIANCE were easy to use, building designers would have to describe each alternative design multiple times, in terms of thermal barriers, geometric solids, etc.

To address this problem, the use of a single, object-oriented building representation, which allows building designers to describe the building in terms of real world objects such as spaces, walls, windows, etc. would be necessary [13]. The program system should automatically “translate” these objects into thermal barriers, geometric solids, etc., as required by the simulation tools linked to the system, thus relieving building designers from the modeling complexities associated with each simulation tool. The system should use a *generic* object-oriented representation of the building. It should be *generic* in that it is not focused towards any specific domain or application, but instead models the actual physical components of the building. An object-oriented representation is based on the notion of *objects* (e.g., windows) that have *parameters* (e.g., U-value, visible transmittance), *relations* (e.g., composed of) to other *objects* (e.g., frame, glazing), and *methods* (e.g., display) which describe their behavior. The advantage of an object-oriented approach is that each object has not only a description of itself, but also an explicit behavior built into it which enables it to manage its own actions. For example, a Wall object contains a Construction object (which defines its materials), and two Surface objects (one for each side) describing the characteristics of the final finish (Figure 1). If the room on one side of an interior wall is

deleted, the Wall object can automatically change its Construction and exterior Surface objects to match the requirements for an exterior wall. This kind of interaction is possible because each object has knowledge about itself and its actions.

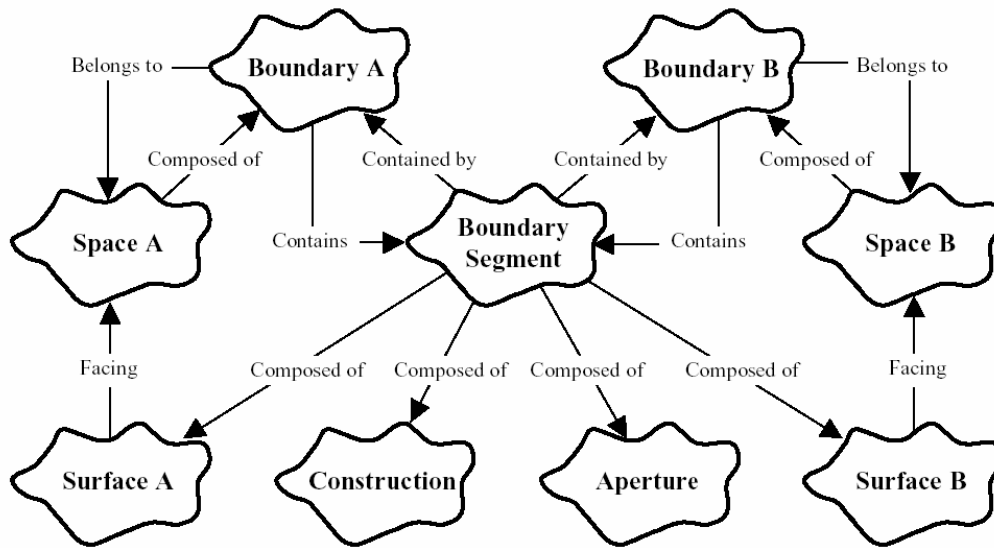


Figure 1. A small window into the object-oriented building model showing the network resulting from the relations among objects [14].

The object-oriented representation should be designed to allow flexible creation of objects, so that it can be extended to address the modeling needs of simulation tools to be linked in the future. It is based on a so-called “meta-schema,” which considers *relations* and *attributes*, as well as the attribute *values*, as objects themselves [15]. By modeling each attribute as an object, an attribute becomes more than a simple value holder. As an object, the attribute knows its name, its units, its display options (e.g., as a numeric value, as a 2-D graph, etc.) and its dependencies on other objects (e.g., simulation programs that may use it as input or output). All of this information is used to facilitate the effective communication between the system and the various simulation and visualization tools linked to it, through the API.

A model that meets the demands as described earlier could for instances consist of tree databases and various applications that operate on them.

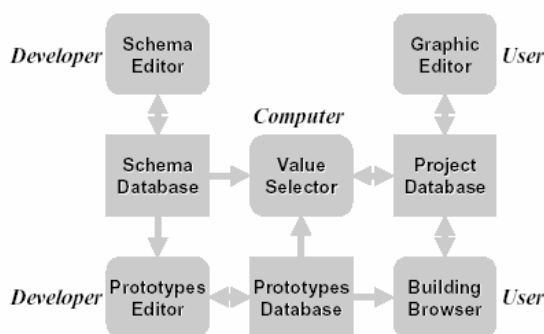
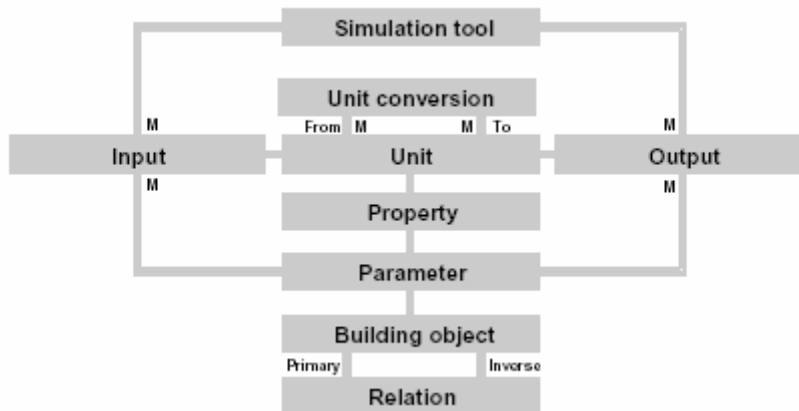


Figure 2. Schematic diagram showing the three main databases and the main processes that operates on them [14].

The Schema Database is a data dictionary where definitions for Building Object Types (e.g. space, wall), Properties (e.g., height, U-value), Units (e.g., ft., cm., degrees), Relations (e.g., has, faces) and Simulation Tools (e.g. EnergyPlus, RADIANCE, etc.) are stored. Parameters

of building components and systems are then defined as links between Object Types and Properties (e.g., space height, wall U-value). Each parameter is also linked to the simulation tools that use it as input or output along with the associated type of units (Figure 3).

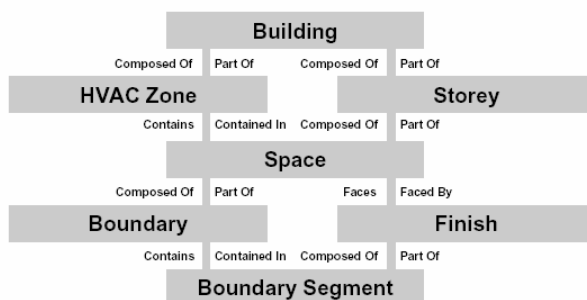


**Figure 3.** Schematic diagram of the meta-schema showing the main objects and their relationships [14].

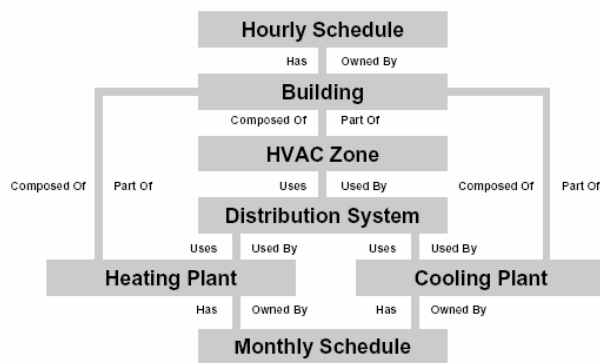
The Prototypes Database is used to store Libraries of Building Object Type Instances (or Prototypes). Each Prototype is created with its own list of parameters as defined in the Schema Database, and each parameter is assigned a Value from some Source or Data Reference. The Prototypes Database is the main source of building components and systems available to the user for the description of the building.

The Project Database is used to store the Building Object Type Instances that is created at run-time by the system. Staying with our “generic” approach, classes for different building objects are not defined. We rather defined classes for Run-time Building Object Type Instances, Run-time Parameter Instances, and Run-time Value Instances along with five derived classes to handle integer, real, string, real array, image, and multi-media data types.

The building model is a network schema with five types of relations used to link the various building components and systems among themselves as well as with the objects used to define the building context (Figure 4 and 5). All relations are defined as pairs of primary and inverse expressions as follows:



**Figure 4.** Partial view of the building model focusing on the schema that relates the building to the spaces its boundaries [14].



**Figure 5.** Partial view of the building model focusing on the schema that relates the building to the HVAC system [14].

**Composed Of/Part Of:** An object may be an assembly that is composed of one or more parts. When an assembly is deleted, then all of its parts are also deleted. Each part is part of one and only one assembly. Deletion of a part has no effect on the existence of its assembly.

**Contains/Contained In:** An object may be a container, which means it may contain one or more contents. The deletion of a content has no effect on the existence of its container. Each content may be contained in one or more containers. When a container is deleted, only those contents are deleted that do not have either a part of or a contained in relationship to any other container.

**Has/Owned By:** An object may be an owner, which means it may have one and only one feature of a particular object type. Deletion of a feature has no effect on the existence of its owner. Each feature is owned by one and only one owner. When an owner is deleted, then all of its features are also deleted.

**Uses/Used By:** An object may be a client, using one or more *servers*. Client deletion has no effect on the existence of a server. Each server is used by one or more clients. Server deletion has no effect on the existence of a client but it does eliminate the service that was provided.

**Faces/Faced By:** This is a special relation that is used to address spaces and their boundaries. A boundary's finish faces one and only one space. Boundary deletion has no effect on the existence of the space that it faces. Each space is faced by one or more boundary finishes. When a space is deleted, then boundaries whose finishes do not face other spaces are deleted, while the ones whose finishes face other spaces may switch to a different instance (e.g., from interior to exterior wall).

The data exchange between the different databases and applications has to be controlled by an object-oriented database management system (OODBMS) [6], which is also required for consistency of all databases. On the one hand it is possible to control the access and the manipulation of data and on the other hand changes of one model are exposed to other planning partners. This storage method prevents a loss of information because the complete object model is represented within the database without any modifications.

The OODBMS also offers a version management, so that all steps of the planning process can be reproduced at any time. Thus, it is possible to obtain information on the responsibility of one of the partners if problems occur during the construction or the utilization of the building.

### Current status and future directions

This paper has presented a research prototype for the complete integration of several object-oriented product models into an object-oriented building model using the persistent storage

management system of an Object-Oriented Database Management System. The system presented has been used in to prototype system and are still under developing [14, 16].

Distributed computing and multi-user collaborative design over the Internet is a major part of the long-term vision. Both data and computational processes can be distributed over local and wide area networks, while specialized additions such as multi-video conferencing and white/transparent board capabilities can be used for effective communication among multiple building design, construction and operation participants.

## References

- [1]. Papamichael K. Designers and information overload: A new approach. 1996;.
- [2]. STEP. ISO-STEP,ISO-TC184/SC4 Industrial Automation Systems, Product Data Representation and Exchange. *Draft International Standard* 1992;.
- [3]. Plokker W, Soethout LL, De Vries P, Rombouts W. EXPRESS/STEP interface kit for COMBINE. *TNO B-93-0095 Report, Delft* 1993;.
- [4]. OMG Manufacturing Domain Revision Task Force. Request for Information CAD-Enabler. <http://www.omg.org/doc/dtc/99-02-02> 1999;.
- [5]. IAI. *IFC 2x - Model Implementation Guide* . 2001;.
- [6]. Objectivity Inc. Objectivity/C++ Programmer's Guide Release 6.0. 2000;.
- [7]. Cattell RGG. The Object Database Standard: ODMG 2.0. . Morgan Kaufmann Publishers; 1997.
- [8]. Dubois AM, Flynn J, Verhoef MHG, Augenbroe GLM. Conceptual Modelling approaches in the COMBINE Project. *First ECPPN Conference, Dresden* 1994;.
- [9]. Augenbroe G. Integrated Building Performance evaluation in the early design stages. *Building and Environment* 1992;**27**(2):149-61.
- [10]. Usher JM. A STEP-Based Object-Oriented Product Model for Process Planning. *Computers & Industrial Engineering* 1996;**31**:185-8.
- [11]. Drury B C, Linda K L, Frederick C W, W F B, Y JH, Curtis O P et al. EnergyPlus: creating a new-generation building energy simulation program. *Energy and Buildings* 2000;**33**(4):443-57.
- [12]. Ward G. The RADIANCE 2.3 Imaging System. *Lawrence Berkely Laboratory, Berkely, California* 1993;.
- [13]. Papamichael K. Application of information technologies in building design decisions. *Building Research & Information* 1998;**V.27**(1):20-34.
- [14]. Papamichael K, Chauvet H, LaPorta J. Automated integration of multiple simulation tools. *LBNL-40591* 1999;.
- [15]. Augenbroe G. COMBINE 1 Final Report. EU DG XII JOULE Report. . ; 1993.
- [16]. Clarke JA, Hand JW, Mac Randal DF, Strachan PA. The development of an Intelligent, Integrated Building Design System within the European COMBINE Project. *ESRU, Energy System Division* 1994;.