ISSUES IN THE DEVELOPMENT AND IMPLEMENTATION OF A BUILDING PROJECT MODEL FOR AN AUTOMATED BUILDING SYSTEM

R. Sacks

ABSTRACT: While other generalized building project models have been designed to support computer-based integration between various construction applications, it is proposed that the project model of an Automated System must be specifically designed for the purpose. The aim of an Automated, Computer Integrated Building Realization System is to automatically generate all of the information required for the design, planning and execution of a building project. The project model forms the foundation of the system, and must therefore include all of the relevant information about the facility and the resources required through the various realization stages. This paper describes a project model designed and implemented specifically for this purpose and details some of the considerations in its development. The model has been tested for life-cycle applicability in a prototype interface of an Automated System. Its completeness at the final stage has also been validated through description of an existing 10 story building.

Keywords: Building Project Model, Computer Integrated Construction, Automated Building System.

INTRODUCTION

Recent efforts to promote Computer Integrated Construction have focused mainly on the subject of Project Model based communication and information sharing (e.g. COMBINE (Augenbroe 1994), STEP (ISO 10303 1994), CIFEWORLD Khedro et al 1994), ATLAS models (Tolman et al 1994)). However, CIC research should also consider the possibility of comprehensively automated building processes. Such systems would employ various information technologies, such as Knowledge Based Expert System techniques, Data Bases, network communications and others, to provide maximum automated assistance to their users. In particular, Object Oriented Project Modeling can provide the foundation for them. The issues described in this paper relate to the development of a Building Project Model specifically designed to serve the needs of an Automated Building System (Warszawski 1990), (Sacks and Warszawski 1997).

The Building Project Model (BPM) outlined here, like the Automated Building System Prototype used to test it, is specifically limited to the design and construction of orthogonal, multi-story buildings with planar (single level), repetitive floors only. The range of building technologies is not limited. The model includes the schema of the principal classes and class relationships (links), hierarchies of classes for each of the principal classes, and many of the methods required for their operation. It also covers resource, activity and product data, hence the term Project Model, as opposed to Product Model.

The concept of an Automated Building Realization System (Warszawski 1990), is based on the proposition that it is possible to advance through the various stages of conception, design and construction of a building project in a fully computerized process. The controlling factor is that after each of the design stages, and intermittently during the construction phases, the project is presented to the owner/developer (or their professional consultants) for approval, change or rejection. Construction may be fully automated (robotics) or conventional construction controlled by the system, or both.

Program modules function collectively at each stage to advance the project (figure 1). These software ‘black boxes’ contain Knowledge-Based Expert System Modules and Data Bases and include rules, procedures, algorithms and data. The programs also have access to external data bases containing all the information relating to the technological, business and physical environment (product catalogs, design codes, town planning requirements, building regulations, economic and financial data and physical site data).

Various components of the automated system have been investigated at the Israel National Building Research Institute. These include preliminary design (Wiesel and Warszawski 1993), detailed design of prefabricated structures (Retik and Warszawski 1994), scheduling (Warszawski and Shaked 1994), planning of robot work (Warszawski et al, 1995) and robot construction (Navon, 1995).

1 Rafael Sacks, Doctoral Candidate, Faculty of Civil Eng., Technion Israel Institute of Technology, Haifa 32000, Israel. (e-mail: cvsacks@techunix.technion.ac.il)
It is widely accepted that Object-Oriented technologies are most appropriate for Project Model development (Bjork 1994). This is the approach adopted in this work. Object-Oriented terminology, (such as object, class, inheritance and instance) is used as defined by Rumbaugh et al (1991).

Examination of the proposed automated system concepts led to the definition of a number of specific starting principles in addition to the basic object-oriented capabilities required. These include (Warszawski and Sacks 1995):

1. The structure of the project model must directly facilitate the addition of detail through the project life-cycle in a way which conforms to the process stages defined for the automated system. The classes defined in the Project Model should enable storage of all of the information at the end of each stage concisely, without requiring instantiation of objects whose attributes might be largely unused at that stage. This is distinct from the situation of design of a generic Project Model, where no clear cut ‘stopping points’ can be defined. Its major space and product classes should have parallel activity classes defined (this reflects the objectives of minimum interference and interdependence between work teams during construction). A particular advantage of the automated system as conceived is that it adopts a top down approach to both design and planning. As such, construction activity instances are not the product of an ‘expert system’ which analyses the building as designed and deduces activities, but are instead instantiated at design time, in accordance with the technologies required for the construction of each assembly (called work assemblies).

2. It must support the storage of libraries of Intelligent Parametric Template (IPT) style solutions at all levels. The use of IPTs is an integral part of the automated system - they provide the primary mechanism for the synthesis, evaluation, selection and layout of design solutions. Their implementation requires that the Project Model enable definition of their Work Assemblies, Elements, Activities and Resources, storing rule sets in an inheritance tree, and provide for multiple inheritance of building element classes. The software modules which perform design synthesis at various stages may use parametric templates, case-based solutions or standard library items. Each of these should be fully expressible in terms of the project model classes.

2. The model need cater only for the type of building contemplated for design by the automated system - rectangular multi-story buildings. This enables the model to incorporate only simple geometric objects.
BUILDING PROJECT MODEL

At the outset, a schema was proposed for the project model. The schema is based on:
a) the conclusions reached in previous modeling efforts as expressed in the extensive literature on the subject, particularly regarding integration of product and process models (Sanvido 1992, Fischer and Froese 1996), expression of form, function and behavior (Phan and Howard, 1994), the level of aggregation and decomposition (Ekholm 1994),
b) special consideration for the rectangular shaped, multi-story building type, in particular work aimed at developing tools for automated scheduling of high-rise buildings (Warszawski and Shaked 1994),
c) the special requirements imposed by the automated nature of the system, as stated above (Warszawski and Sacks 95, Sacks and Warszawski 1997).

The problems related to maintenance of data integrity make simplicity of the model a key goal. Therefore, the number of classes and relationships in the schema was reduced to those presented in Figure 2. The model must describe the project in terms of its Spaces, its Functional and Physical Systems, and its Activities: thus the schema has three principal axes - the building’s spaces, the assemblies and parts of which the building is composed, and the activities through which it is built. Each axis has three levels of decomposition.

The first axis defines the building spaces. Its object classes are:
- Building - Defined by its name, shape, layout on site, functional system requirements, building assemblies and primary spaces.
- Primary Space - A geometrically distinct sub division of the building. The primary space is also defined for purposes of construction planning as the location in which an activity is performed. For the rectangular shaped multi-story buildings dealt with here, a primary space is a building floor. It is defined by its name, function, height, work assemblies and its layout of secondary spaces.
- Secondary Space - A geometrically distinct sub division of a building floor with performance requirements a part of a floor, such as a room. It is defined by its name, location on the floor, function, size, minimum clear height and performance requirements.

The second axis has the building ‘product’ object classes:
- Functional System - an abstract definition of a complete system, defined by its performance requirements. Exterior Enclosure, Water Supply, Vertical Structural Support are examples of functional systems. A system is implemented by one or more Work Assemblies.
- Building Assembly - a collection of Work Assemblies which fulfill a common function for the whole building, such as ‘Structure’.
- Work Assembly - a collection of Elements which are assembled in one space by one activity in order to fulfill a required function or functions. Work assemblies are differentiated by the work type and materials (i.e. the technology), required to build them. They are defined by name, technology, the functional system(s) which they implement, the spaces in which they are installed, their elements, and the activities which install them.
- Element - a basic building part, such as a wall, beam, column, flooring section etc. Apart from its name (identity), description and association to a Work Assembly, its attributes differ according to its nature, and are defined within an inheritance hierarchy of different element types.

The third axis describes the construction process object classes:
- Task - a collection of activities which together build a Building Assembly, usually performed by a distinctive organizational unit (such as a specific work team or a sub-contractor).
- Activity - represents the installation of a specific Work Assembly in a Primary Space. It contains a collection of Basic Activities which enable calculation of its required resources, cost and duration. The Activity is also the key object for use in project scheduling. It is defined by its name, description, the work assembly it installs start dates (early and late), duration and dependencies on other activities.
- Basic Activity - defined as installing an Element. The basic activity does not have dates associated with it as no scheduling is performed at this level, but rather concentrates the resources required and associates them to the activities.
- Resource - a quantity of material, a component, labor or equipment required by a basic activity.
Figure 2. Project Model Schema (informal representation).
(Sacks and Warszawski 1997)

For a more thorough description of the schema elements and the Automated Building System, see (Sacks and Warszawski 1997). The discussion which follows outlines the considerations involved in implementation of this schema, and the issues behind each consideration.

BUILDING PROJECT MODEL DEVELOPMENT ISSUES

Defining Project Data

The first stage of the Automated System process is to state the owner’s requirements. These include general requirements, specification of the site for the building(s) and space requirements. Each space requirement should include a minimum floor area and required performance ranges for various behavioral attributes, such as temperature range, acoustic levels, etc. Three objects are introduced to store this information: the Project (including target start and completion dates, budget and desired cash flow, etc.), the Site (location, size, topography, municipal services, climate, soil profile, etc.) and the Space Requirement.

The Site class has geometry attributes and associated methods which are not relevant for the Project class. A site could also be replaced with an alternative site for comparison of the different design alternatives which would arise. The Space Requirement class may have inherited classes with additional functional performance attributes. For these reasons, the Space Requirement and Site classes are separated from the Project class rather than included as additional attributes of it.
These objects are sufficient for storage of all of the information required by the automated system in order to compose the design brief output.

Defining and Maintaining Consistency of the Building Geometry and Topology

Layout design for the building and its floors is performed and the results stored in instances of the Building, Primary Space and Secondary Space classes. As proposed by Eastman and Siabiris (1995), the building spaces are decomposed into lower level spaces. However, this is not done recursively as in their model - only three specific levels are defined, and each is functionally distinct from the other. Multi-story buildings as considered here lend themselves to division into floors and secondary spaces on the floors (rooms) - this is sufficient, as both design and construction planning do not require sub-division of spaces beyond the level of the room. Each of these can be an orthogonal polygon, defined by a closed sequence of node points. They also have a number of area related methods in common. It is therefore convenient if each inherits from a common primitive shape class - the AC_FACE class. The Primary Space class (usually a building floor) also has a ‘height’ attribute, and the height of the whole building can be calculated as the sum of the floor heights.

Previous work in defining Building Product Models has addressed the problem of whether to describe the basic geometry of a building in terms of its spaces (rooms), its space boundaries, its enclosing structures (walls) or combinations of them (Bjork 1992, Eastman and Siabiris 1995). Bjork (1992) proposed a schema including all three, and added a topological relation between the Space and the generic Opening class - Opening `serves [1:2]` Space. In the model proposed here, physical walls and other partitions (the ‘enclosing structure’ of Bjork’s schema), doors and windows are considered technical solutions to functional system requirements for space separation and space access, and are therefore not instantiated at this early design stage. However, spaces and the openings for access between them must be defined, and must be shown on layout drawings for owner approval; the definition of Space Separator and Space Access primitive objects resolves these problems.

The Space Separator is linked to two space objects which it separates, and the Space Access is positioned along the length of a Space Separator. At later stages of the design process, the physical position of partition elements is defined through their ‘has_position’ links to Space Separators, and, similarly, opening elements such as doors and windows are positioned by relationships to Space Access objects (figure 3).

![Figure 3. Secondary Space, Space Separator and Space Access objects](image)

This scheme maintains both geometrical consistency (as the opening - door or window - element is defined as positioned by the Space access, which in turn is positioned along the Space Separator), and also provides the topological information required (separates the spaces).

Topological relationships must be maintained within the model in order to enable the function of various procedures and in order to support graphic editing tools. Of course, such relationships must remain consistent throughout the life of any instance of the model. Redundancy of information in the model can often lead to consistency being compromised. The OOAMS model (Watanabe and Watanabe 1991) included a complete set of node, axis and line objects to define the geometry of the model, and each building object derived its position and its topological relation to other objects by relationship to one or more of these. These objects and their instances are quite distinct from the actual building product objects. The resulting network of links is complex and therefore difficult to maintain.
Other models, such as the Component-based building representation (Harfmann and Chen, 1993) store all of the geometric and topological information directly within the building component objects and relationships. While this may be sufficient for communication of static product information, it cannot support incremental growth of instance models. This is because the topological relationship between any two elements may need to be stored before the second product element is instanced. For example, at the Layout design stage, the presence of doors for access between spaces must be defined even though neither the wall nor the door elements have been instanced. The Space Separator and Space Access classes explained above highlight the way in which this capability is provided in the proposed BPM.

In contrast to the latter cases discussed, the BPM proposed here does use topology primitives; however, unlike the OOAMS approach, a minimum of topology primitives have been used, and each has direct semantic meaning.

The decomposition of the building into floors allows aggregation of the physical parts of the building according to the floors in addition to aggregation according to the type of work required for their installation. A collection of building elements installed by a particular work team in a distinct activity on a particular floor is called a Work Assembly. This feature of the Project Model structure enables association of activities to spaces, which is important for scheduling the construction with minimum conflicts. This reflects a key premise adopted in the definition of the automated system, i.e. that construction activities should be instanced top-down, simultaneously with the placing of the Work Assemblies and their elements (rather than being instanced by a knowledge module which would aggregate basic activities at a later stage).

However, such an arrangement does not account for building spaces which have vertical continuity (such as elevator shafts, stair wells, utility shafts and structural columns). It has been suggested (Warszawski and Shaked 1994) that such spaces be defined as a specialized type of Primary Space. However, since such vertical spaces are in general not built as a whole, but rather floor by floor, the solution selected was to introduce a geometric primitive class, Shaft (figure 4). A shaft (or its children) is fixed in place geometrically through a number of floors, and positions the secondary space or element on the floor itself. Thus shafts are not actual building spaces.

Similarly, the position of a column could be stored either within a column attribute or in a separate, linked topology primitive. For the type of building considered, structural considerations dictate that columns be placed above columns on the floor immediately below, or at least within a small distance from them. A vertical Column Axis topology primitive is defined which extends through the full height of the building. The Column Axis inherits the attributes of an AC_POINT (i.e. X and Y values). Each column on each floor may have different shape, reinforcement, finish etc., but its position can be fixed as either on the axis or at an offset from it (figure 5). The offset is set and controlled by the structural knowledge module which eventually places them. The Column element has a ‘has_position’ link to the Column Axis, and includes attributes named x_offset and y_offset.

![Figure 4 Building, Floor, Shaft and Secondary Space.](image)
Storing Design Process Information

A number of attempts have been made to describe building design in forms which lend themselves to computerization, algorithmically (Oxman 1991, Sause & Powell 1990, Tsakalias 1994). One general approach has been to see the process as the statement of functional requirements and the selection of a technical solution to fulfill those requirements (GARM, Gielingh 1988). This is the approach adopted here for the purpose of selecting work assemblies. It is implemented through a family of Functional System Requirement classes (note however, that this method is not used for preliminary and detailed design of Work Assemblies - this is dealt with in the next section).

Once spaces exist the functional systems needed to create the desired environment can be listed. Also, specific requirements of their functional systems can be defined. Such information is stored in instances of objects which inherit from the Functional System Requirements (FSR) abstract class (figure 6). An FSR can collectively represent the requirements for an entire building floor or for a particular room or set of rooms on the floor. In certain cases, it can express a requirement for the whole building (e.g. a ‘Lateral Load Resistance’ system). For this reason, an FSR is defined as serving a Space class, from which the Building, Primary Space and Secondary Space inherit. Also, the AC_FACE geometric attributes and methods are now inherited by the Building, Primary Space and Secondary Space classes through the Space class.

A problem arises in situations where a functional system is required to perform to different levels within its scope. For example, a certain section of a floor slab may be required to support higher live loads than other sections; a particular wall within the space partition work assembly of a floor may be required to provide higher acoustic insulation than the others. Three strategies are possible to account for this need:

1. In keeping with the Functional Unit - Technical Solution strategy of the FSR to Work Assembly relationship, the adoption of a Functional Element was proposed. This approach allows each element of a Work Assembly to have local requirements expressed to override those stated in the FSR. However, this would require both a large number of additional classes in the schema and many instances in any particular model, many of which would be redundant. Also, a local load requirement may not be geometrically coincident with an Element. Thus this approach was rejected.

2. A set of Load classes could be defined. These could be attached to FSRs whenever required. The preliminary and detailed design expert methods of the Work Assemblies could then identify such Load instances and use them for design. This requires that the Loads have geometric or topologic information associated with them.

3. View the specific functional requirements in the FSR attributes as nominal values for purpose of preliminary design. Work assembly methods will, in any case, perform analyses as part of detailed design, and can therefore retrieve all of the load requirements from the spaces (in effect, from the space-
requirement instances) they serve (the loads may also vary depending on the nature of the Work Assembly itself). This is the preferred strategy.

The adoption of option 3 in the above discussion is an example of a general dilemma which faces designers of project models: whether to prefer more explicit storage of data or more processing of data (Fischer & Froese 1996). An important reason for favoring the processing option rather than the storage one wherever feasible, is to avoid the lack of data integrity which can arise due to its storage in more than one place.

The ability to backtrack is an important feature of the automated system. This is because the operators’ ability to guide the process is limited to specific review points (at the conclusion of each stage). In a more interactive process this would not be as critical, since user input is being provided continuously. It was therefore necessary to maintain previous versions. Also, in order to allow evaluation of results against goals, it is important to record design intent. A possible solution to this was the inclusion of class attributes whose values would be ‘collected’ as the system run progressed. For example, the area of a Space could be stored in separate attributes for Proposed Area (brief proposal), Designed Area (layout design) and Actual Area (after detailing columns and walls). This is problematic as it requires storage of numerous values (possibly as arrays) for many attributes, which complicates the model. The solution adopted therefore, is to simply store the current state of the instanced Project Model at each stage of the progress of the Automated System. It is recognized that this does not allow recording of the reasons why decisions were taken.

In order to select a Work Assembly to fulfill a Functional System Requirement, at the start of preliminary design, a technology must first be selected. For example, a structural floor might be built using Steel, Reinforced Concrete, Steel Frame and Precast Concrete slab sections, Post-Tensioned Concrete or other methods. In earlier versions of the Project Model, a Technology class was included for this purpose. However, in building the Work Assembly selection expert for the prototype Automated Building System, it became clear that this information need not be stored, but may be restricted to the scope of the knowledge module. To this end, the attribute ‘possible_technologies’ was added to the FSR class. Once the technology has been selected, the known Work Assemblies utilizing the technology can be compared and the optimum assembly selected.

Inheritance Hierarchy Classification

The ISO Technical Report 14177 “Classification of Information in the Construction Industry” (Ekholm 1996) defines a building element foremostly by its characteristic function “without regard to the type of technical solution or form of construction”. However, the BPM must support a design decision making process in which a work assembly is selected for a functional system requirement by first establishing the technology to be employed and then selecting the work assembly itself. In addition, the basic goal of coherent construction planning gave rise to the definition of the Work Assembly class as being executed by one activity. Thus while the definition by function is maintained in the BPM, the technical solution and form of construction cannot be disregarded. The solution in the BPM is to provide three Work Assembly classification layers - according to different functional systems, different technologies and then individual work assemblies (figure 7). It should be noted that there is a fundamental difference between the method of classification at the technology level and that at the Work Assembly level. The technologies are distinguished by construction methods, materials and behavior; the Work Assemblies are classified within the technologies according to object-oriented implementation considerations of commonality of attributes and/or methods.
Aggregation

In some cases, at the end of preliminary design, a situation may arise where more than one element is placed in one position by one or more Work Assemblies. This can happen, for instance, when a structural wall coincides with an architectural partition. Such occurrences must be identified and the elements rationalized into single, dual-function elements. This is done by linking a single unified element, whose class inherits from each of the two distinct element classes (multiple inheritance), to both Work Assemblies: hence the m:n cardinality for the Work Assembly - Element link.

Two alternative interpretations are possible for the Building Assembly as shown in the initial Project Model proposal. The first option is to view a Building Assembly as equivalent to a Work Assembly except for the fact that it serves the entire Building, while a Work Assembly serves a Primary Space. In this way, an Assembly for resistance of lateral loads on the building could be considered a Building Assembly. The second option is to consider the Building Assembly as an aggregation of Work Assemblies which function together to serve a building-wide function. For example, the building structure would be composed of all of the structural Work Assemblies. This is useful for purposes of analysis, rationalization of elements of Work Assemblies, and is adopted in the model. A corollary of this view is that it must be possible to associate Work Assemblies to any level of Space, not only to Primary Spaces. This is done by linking the Functional System Requirement directly to the Space class.

Decomposition - Component Objects

For most of the assemblies in a building, decomposition to the level of Element is sufficient. However, certain elements contain parts which must be detailed further because they are supplied separately and assembled on site. For example, reinforced concrete elements contain reinforcing bars, which must be detailed individually or in groups; a door element may refer to installation of a particular combination of different door, frame and fittings components. Further decomposition of the Element into Components was therefore proposed. In fact, models have been proposed in which the component is the central modeling starting point (Howard and Phan 1992). At the same time, such components are also a resource and are consumed in the basic activities which create the element. For this reason, in early versions of the BPM schema the Component class was shown as inheriting from the Material Resource class.

It is proposed however, that for the automated system Project Model, the direct Element - Component link is unnecessary. In the top-down approach of the Intelligent Parametric Templates (described in the next section) Activities are instanced at the preliminary design stage, in direct relation to the Work Assemblies they build. At the detailed design stage, both the Elements of the Work Assembly and the Basic Activities which build them are detailed. Thus Component instances are created only at the detailed design stage, and in fact provide detailing for the basic activities to which they are related. For example, a basic activity for installation of a door element requires the detailed information of which door component to install. Thus it is sufficient to link Component
instances to the model through the Basic Activity instances. This greatly simplifies both the model and the automated system (for example, the cost assessment knowledge module’s Element class methods can accumulate both the material and work costs of an element with relative ease). Therefore, this approach is adopted in the BPM. An example of a door element and its three components is shown in Figure 8.

A similar approach is adopted by Tah (1996). The Task and Unit_of_Work in this model are parallel to the Activity and Basic Activity in the BPM, and here too the component is not attached to the Structural Member (Element), but rather to the Unit_of_Work (Basic Activity).

Figure 8. A Door Element instance with its Basic Activities and Components

The link between the Basic Activity and the Material Resource must include a value for the quantity consumed. In the IRMA model (Froese 1994), the solution adopted was to insert a Resource Use class between the Basic Activity and Material Resource. A better solution would be to provide a ‘quantity’ attribute for the link. However, since link attributes cannot be implemented directly using the current platform, the relationship is ‘objectified’ and a Resource Use class is defined (this is discussed fully in the section on Implementation Tools and shown in Fig 16).

Once the detailed design has been approved, a project schedule is proposed by the Construction Planning Knowledge Module. Since no scheduling is done at the Basic Activity level, attributes such as ‘start_date’ and ‘finish_date’ are only defined for the Activity class. Similarly, Work Resources (such as Labor or Equipment) are linked only to Activity instances.

Repeated Object Instances

Three different levels of repetition of building object instances have been identified (Rivard et al 1995). These are 1) Same - reference is made to the same building element, 2) Similar - two (or more) distinct elements exist but they are similar, and 3) Identical - two (or more) identical elements exist. In the first case, a model must rationalize the two (or more) references into one object instance (this is done in the BPM by rationalizing elements after preliminary design of Work Assemblies). In the second case, each separate element must be maintained as such, although they may be linked to common primitives. In the last case, each separate object instance is identical to the next apart from its identity. One useful way of expressing this would be to create a generic ‘Copy’ class, with a ‘copy-of’ link to the first element. Alternatively, the identical elements could be stored in an ‘array’ structure. In the current BPM, no attempt has been made to address this issue - each individual element is stored without regard to its similarity to other elements.

Howard et al (1992) proposed a method of building product data modeling using a ‘primitive - composite’ approach. Rivard et al (1995) adopted this principle and defined building envelope elements as objects of classes which were composed, through object-oriented inheritance, of form, function and behavior primitive classes.
Definitions done in this way use a small number of primitive classes, but require definition of a multitude of composite classes. For example, a square column and a circular column must each have separate classes. An alternative is to implement the primitive-composite approach using object links rather than inheritance. A column instance will have a ‘has_shape’ link to a shape primitive. This allows great flexibility at run time. In particular, it enables an automated system to instantiate an element and defer its detailing (e.g. shape selection and sizing) to later stages. The BPM uses this strategy for geometric positioning, cross section detailing, reference to material resources and for associating materials to elements.

The full central schema of the Project Model obtained at the conclusion of the implementation is shown in Figure 9 (the axis shown at the right of the figure is separated from the main schema only for sake of clarity).

**Figure 9. Main part of BPM Schema (AutoLISP++ representation)**

**SUPPORT FOR AUTOMATED DESIGN**

As stated above, various attempts have been made to automate building design. They have used a variety of approaches:

1) Expert Systems have been implemented for building systems, such as HI-RISE, (Maher and Fenves 1985), KTISMA (Tsakalias 1994), and for individual building parts such as EIDOCC (Sacks and Buyukozturk 1986). They focus on encapsulating human expert knowledge in design systems, in particular through processing of heuristic rules.

2) Case-Based Reasoning draws on previous design cases as starting points for development of new designs. Systems such as ARCHIE (Domeshke and Kolodner 1992), function by locating appropriate cases
(Retrieval), using the knowledge contained in the case (Reuse), adapting the case to suit the current design requirements (Revise) and storage of the new case (Retain) (Aamodt and Plaza 1996).

3) Algorithmic approaches such as the Multi-Level Selection and Development paradigm (Sause and Powell, 1990).

In some expert systems and in the Multi-Level Selection and Development paradigm, the design is formulated step by step - decisions taken at any junction are independent of the decisions to be made at lower levels of the design. In the case of preliminary and detailed design of systems for the type of buildings considered here, the authors suggest instead that the design solutions should not be amorphous combinations of a variety of elements arrived at through step by step reasoning, but should rather be adopted as complete system solutions. In this case, only one key decision need be made for each functional system - once the technical solution is chosen, its prescribed elements and their required activities can be placed and then detailed. This approach also ensures that the design arrived at will be cohesive and constructible. To this end, the use of ‘Intelligent Parametric Templates’ (IPTs) is introduced for preliminary and detailed design of Work Assemblies (Sacks and Warszawski 1997).

Each IPT is a complete technical solution and includes
1. the object class definitions of the Work Assembly, its Elements and Activities,
2. all of the specialized class methods required for designing and detailing itself, and
3. knowledge-based rule sets required by the system modules which manipulate the IPT.

The rules and methods prescribe the way in which values for parametric slots of the Work Assembly are filled (e.g. the depth of a concrete slab might initially be set according to depth/span ratio rules, and then enlarged by a method which checks punching shear). The expert system type rules and routines are used particularly at the conceptual and general design stages, for feasibility checking, evaluation and layout. It should be noted that rule sets are inherited in a similar fashion to methods. Not only does this top-down approach make automated design feasible; it also allows new technical solutions to be added to the library of IPT’s without any change to the system itself or to the previously defined solutions.

An example of the use of IPTs is that of selection, design and detailing of a structural slab work assembly to support the floors of a building. Each possible type of slab is defined and stored in a hierarchy of slabs, as in Fig 10. The slab solution IPT class is a specialization of the Gravity Load Support Work Assembly class family. The IPT’s elements are the slab itself and the beams and columns which support it. The activity which builds the slab is also part of the IPT. The rule sets include rules for assessing column layouts, checking the suitability of the slab to a given design situation, evaluating a possible configurations and preliminary design of the slab. Since definition of all of a new IPT’s object classes is done through specialization of classes already present in the BPM, only those rules, methods and element classes which are unique for the new IPT need be defined. For example, a rule for feasibility checking for a general class of concrete slabs states that:

“If the minimum overall slab depth required is less than the maximum clear height available, the slab is not feasible”.

This rule is applicable to all of the children of this slab class, and need not be redefined for child classes. A child class, such as that of a Flat Slab Reinforced Concrete IPT, might include a rule such as

“If the slab spans in two directions, the minimum depth required is 1/30 of the larger span.”

One consequence of this structure is that none of the rules are comparative (i.e. they concern only the IPT they refer to and do not allow comparison with other possible solutions). This is distinct from traditional Rule-Based Expert systems, where rules such as “if condition A exists then solution B is feasible, else solution C is feasible” are used. Therefore, comparisons between possible solutions must be made indirectly, using criteria appropriate for each case, such as cost, resources required, etc.
The IPT approach draws on the Case-Based Reasoning approach in that it is based on retrieval, selection and reuse of complete solutions. However, the IPTs are stored as templates without parameter values, unlike cases or ‘prototypes’ (Rosenman et al, 1991). The knowledge required for manipulation of an IPT is stored within the IPT, unlike the CBR approach where general selection and adaptation rules are applied to any of the cases retrieved. Also, adaptation of IPTs to specific design situations is limited to parametric adaptation only - they do not support substitution of elements or topological changes.

It is important to note that implementation of this solution is dependent on the existence of a well defined object-oriented project model. Addition of a new IPT requires definition of a new Work Assembly type class, new Element and Activity type classes (if necessary), and writing the specialized methods and rule sets for its implementation. Since attributes, methods and rule sets can be inherited, this is in general a reasonably brief task when compared to the large time investment required to add a new type of technical solution to the Rule Base of an Expert System (Watson, 1996).

A knowledge module of the Automated Building System which implements the IPT design strategy is currently being developed. The module performs preliminary and detailed structural design for rectangular shaped multi-story buildings.

**DEMONSTRATION PROTOTYPE OF AUTOMATED BUILDING SYSTEM FOR TESTING BPM**

A simple prototype of the proposed ABS interfaces has been built in order to demonstrate the process and to test the BPM. As can be seen from the system architecture presented in Figure 2, many standard program modules, knowledge modules and data bases are required for a fully functioning system. The prototype includes a minimum number of demonstration modules, which mimic the functioning of the proposed system by creating object instances and providing values for their attributes directly, rather than using fully fledged Knowledge Modules. It is run from a central control panel (Fig 11), from which the operator can instruct the system to advance a step, go back a step, browse the current project data instances as they exist in memory, or produce on demand, automatically, any of the drawings or reports which are listed for the current stage. The controller can also change the project data through the Edit option - this opens various editing dialog screens and drawings as appropriate to the current stage.
The prototype functions through the following stages:

1. Statement of owner requirements and initial feasibility check.
2. Brief proposal.
   a) Build full list of space requirements.
   b) Detail functional performance levels for each type of space requirement.
3. Conceptual design.
   a) Layout of the building on the site, building height and shape.
   b) Floor layouts.
4. General design.
   a) Detail Functional System Requirements.
   b) Work Assembly selection.
   c) Preliminary Design (synthesis).
5. Detailed Design.

An example of a typical system run is shown in the series of figures 12 to 14. The project owner requires 1000 m² net area of office space and 400 m² net area for shops. The maximum building footprint allowed on the site has dimensions of 39m x 17m. Fig 13 shows the building instance with its attributes and links directly, using the Instance Browser. Examples of automatic system outputs are shown in Figures 12 and 14. It must be emphasized that the prototype in its current form does not perform ‘best’, ‘intelligent’ or even ‘good’ design: it is intended purely as a test bed for the Building Project Model.
Figure 13. Building Instance after Preliminary Layout Design

Figure 14. Detailed Architectural Floor Plan.
BUILDING REPRESENTATION EXPERIMENT

In order to test the viability of storing all of the information for a full scale project using the proposed model, a ten storey office block under construction in Jerusalem was modeled. The experiment was static in that the project was modeled as it existed at the end of the detailed design phase. The building selected for the experiment has a skeleton of reinforced concrete walls, columns, beams and ribbed slabs. Interior walls are of dry wall construction. Exterior walls are of poured in-situ concrete with a natural stone facing layer. Only the architectural and structural systems were modeled fully. The architectural plan of a typical floor, the structural layout of a typical floor and one elevation are shown in Figure 15.

The project was instanced by running a ‘script’ style program which consisted mainly of calls to constructor methods of the instance classes (the constructor methods are part of the BPM class library). The constructor methods were called with all the relevant object attribute values as arguments. For example, the building instance was created and linked to the project instance using the following code segment:

```
(setq a_building (building             ; call the ‘building’ constructor
                   "Turim Street Office Block"        ; building name
                   '("High")                          ; standard of finishes
                   '(0.0 0.0 0.0) '(31.35 0.0 0.0)    ; building corner points
                   '(31.35 16.30 0.0) '(0.0 16.30 0.0))
))
(o-o #project "contains" a_building)   ; establish the link to the project
```

![Figure 15(a). South Elevation of Test Building](image1)

![Figure 15(b). Structural layout of Typical Floor](image2)
The full model required a file of 838 kB. Two software tools were used to examine the fully instanced model: the AutoLISP++ Instance Browser (see next section), and the ABS output generating module. Figure 16 shows a screen from the Instance Browser; the drawings shown in Figure 15 above were all produced automatically by the ABS output module on the basis of the project information stored according to the BPM model schema.
SYSTEM DEVELOPMENT TOOLS

Ten alternative combinations of software packages were assessed for compatibility to serve as an implementation platform for the Building Project Model prototype. The range of products covered included Object-Oriented Data Base Management Systems, Expert Systems Development Tools, Object-Oriented Program Development Environments and CAD Based Development Environments. Specific minimum requirements for the platform were set out in Warszawski and Sacks (1995). Thus each tool was evaluated in light of the following basic requirements:

- Object-Oriented programming.
- Object persistence.
- Dynamic Typing.
- Multiple Inheritance.
- Rule processing capabilities.
- Ability to interface with relational data bases and text/hypertext information sources.
- Ability to be incorporated with/compiled for/interfaced with AutoCAD® 13.

Of these, the two highest scoring system combinations were selected for more detailed inspection - Jspace™ (Jacobus Technology 1995) and AutoLISP++ Objected Oriented Interpreter. Jspace is a commercially available application development environment designed specifically for object-oriented AEC system development, and runs on Unix platforms. AutoLISP++ is an implementation of a CLOS (Common Lisp Object System) (Tello 1989) within the AutoCAD® AutoLISP development environment. AutoLISP++ was built within the framework of this research.

The basic BPM schema was implemented using each of the two abovementioned platforms. At this stage, additional criteria for evaluation were set. These concerned costs of use, specifically in terms of time investment on the part of the developer. The criteria are:

- Ease of definition of object schema (is there a graphical schema editor ?)
- Ease of implementing changes in schema during development (length of the compile-link-debug cycle)
- Time investment required for learning the system.
- Time investment required for application development (user friendliness, provision of library routines).
- Degree of openness to incorporation of external programs or user-defined routines, and support for programming languages.
- Cost
- Hardware platform requirements.

Although Jspace™ includes object structures which are well designed for supporting Project Model development, the absence of a graphic schema editor and the relative length of the compile-link-debug cycle proved to be disadvantages for prototyping. The environment selected was the AutoLISP++ Objected Oriented Interpreter. The system includes a Graphical Schema Editor, an Instance Browser and a built-in rule processing engine. The interpreter extends basic AutoLISP by adding the functions and syntactical constructs required for defining classes and class links, calling member functions, retrieving instance attribute values, etc. Class methods can call regular AutoLISP functions, AutoLISP++ functions or ADS C++ functions. The schema editor allows the developer to build the class hierarchies through dialog boxes, and maintains the topology of the schema drawing. At any stage, the schema can be automatically written to a text file in a form which can be run by the interpreter. The schema drawing and the text file are comparable to the ISO 10303 (1994) based EXPRESS-G and EXPRESS formats; in fact, the AutoLISP++ schema editor could be programmed to output an EXPRESS file format. Using the instance browser, the developer can view the attribute values for any of the existing object instances at any stage, and can also follow the links from instance to instance. The rule processor functions using forward chaining on rule sets which are expressed in an IF-THEN-ELSE form using standard LISP syntax. Rules can also call class ‘rule-methods’ which allows a rule to incorporate different nuances for different objects.

The experience gained through implementation of both the Project Model and the prototype Automated Building System has confirmed that system prototyping can be done relatively quickly. The system is sufficiently flexible to allow changes to the schema to be made with ease. Both the schema editor and instance browser proved to be indispensable development tools.

Application programs run relatively slowly by comparison with those developed in comparable environments, such as JSpace™, although the response time is still tolerable for research purposes. Software processors are available which can convert interpreted AutoLISP code into compiled routines which run within the AutoCAD® ADS system - use of such a tool would enhance the running speed significantly. Another drawback is the lack of the possibility to define link attributes. This is overcome by ‘objectifying’ object links for which attributes are required. For example, a Basic Activity ‘consumes’ a certain quantity of a Material Resource (e.g. Pouring Concrete for a Column Element consumes $n$ cubic meters of Concrete Material). The simplest way to represent
this link would be as in Fig 17a - with the \textit{quantity} value assigned as part of the ‘consumes’ link. In the current version, the Resource Use class must be introduced, as shown in Fig 17b.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{fig17a}
\caption{a) Link with Attribute}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{fig17b}
\caption{b) Objectified link.}
\end{figure}

\section*{CONCLUSIONS}

The Building Project Model schema presented has been developed to fully support the description of a building project throughout its life-cycle as required by an automated building system. A prototype Automated Building System interface was built and run in order to test the model’s expression of the project data as it is incrementally added through the different project stages. The information for an existing building at its detailed design stage was also expressed using the model data structure. These two experiments have shown that the model is sufficiently stable and mature to facilitate further CIC research and Automated Building System development.

The model in its present form is designed for buildings with multiple floors of uniform, orthogonal shape only. Only a small number of representative classes have been defined and implemented for each main schema class hierarchy. The multi-story and repetitive nature of the buildings covered is inherent to the model’s design. On the other hand, the limitations on floor shape and class scope have been imposed purely due to limited research resources, and should not affect the basic applicability of the model.

The next stage of the work will focus on the implementation of a module for preliminary and detailed structural design, which will take the form of a collection of Intelligent Parametric Templates, including Work Assembly and Element methods, and associated data bases. This will enable examination of the underlying premise of the Automated System, i.e. that it is possible to advance through the building process in a stage by stage manner using the Building Project Model schema as the basis for storing the project data.

\section*{Acknowledgment}

This paper is based on parts of the author’s D.Sc. thesis, which was supervised by Professor Abraham Warszawski and Professor Uri Kirsch at the Technion I.I.T. Their assistance is acknowledged and greatly appreciated.

The author wishes to thank Eitan Ronel, of Ronel Architects Ltd. in Jerusalem, for providing drawings and other construction documents describing the test building, Jacobus Technology Inc., for providing the Jspace software, and Autodesk Israel Ltd. for providing AutoCAD 13 software.
REFERENCES


Jacobus Technology Inc.,1995, Jspace™...an environment for objects, Jspace Users Manuals, Jacobus Technology, 7901 Beechcraft Avenue, Gaithersburg Maryland 20879.


