AN INTERACTING MULTIDISCIPLINARY BUILDING PRODUCT MODEL

by

G. N. Kodagoda and W. P. S. Dias

ABSTRACT

This paper presents a practical product model based on an object-oriented design, where a primitive-composite scheme is used to ensure flexibility and reduce complexity. Hierarchies of architectural and structural elements are proposed, as well as those of primitives (or properties). These primitives (both global and local) can be attached dynamically to the element classes, thus creating composites. Geometry however is implemented directly in the building elements. Wherever possible local geometry was used, while semantic relations were used to define relationships between instances of building elements. A software package called PROMOD was developed based on this product model. Data entered into the product model was successfully exported to AutoCAD to generate 3D views, and also to PROKON, to carry out structural analysis. In addition, it is shown that the semantic data stored within the model is sufficient to answer a varied range of cross-disciplinary queries.

1. INTRODUCTION

A building is a structure that is built up of solids and spaces. It has a high space to solid ratio. The information relating to a building includes the attributes and functions of the solids and spaces, and the relationships between solids and spaces.

Several multidisciplinary professionals are involved in the design, construction and maintenance of a building. Each discipline has a different perspective of the same building. These differences include how the elements are aggregated, how a particular element is viewed, and the prominence given to certain elements (Dias 1996).

For example during the design of a building a structural engineer would simplify a building as an aggregation of columns, beams, slabs, walls etc. The columns and beams would be simplified as linear elements during a structural analysis. Architects on the other hand would view columns and beams as solid three dimensional objects. The architect would also prefer to idealise a building as an aggregation of spaces.

Computers are now used in most areas in the building industry (Debney 1999). Architects use CAD software like AutoCAD to generate drawings. There are also special versions of CAD meant for the building industry such as AEC AutoCAD (Sheperd 1996), which make this process easier. In addition to the generation of traditional 2D plans of the buildings in paper, architects can generate 3D computer models of buildings. It is also possible to create virtual walkthroughs of these 3D model buildings making visualisation easier.

Structural engineers use structural analysis and design software such as PROKON, STAAD III and CADDS. Some of this new software allows the engineer to model the entire structure in 3D, analyse the structure and then to carry out the design. Quantity surveying software is also available for quantity surveyors to create bills of quantities. Services engineers make use of special software that can be used to design services such as heating and air-conditioning of a building.

Although computers have been used in the building industry for well over thirty years as indicated above, the primary means of communication of building information between different disciplines is still largely through traditional 2D paper drawings. One of the main problems with such traditional systems is the difficulty in the exchange of data between the software of the different disciplines. This is mainly due to the fact that each of these software only tries to capture data that is relevant for that discipline. For example in structural analysis software, only the data related to an idealised structural form such as a frame can be entered.

Eng. G.N. Kodagoda, BSc(Eng), graduated in 1995 from the University of Moratuwa's Department of Civil Engineering, where he is currently working as Systems Analyst and also pursuing an MPhil. Degree.

Eng. (Prof.) W.P.S. Dias, BSc Eng (Hons) PhD, DIC, MIStructE, FIE(SL), who is a Professor of Civil Engineering and Chairman, Engineering Research Unit at the University of Moratuwa.

Modifications to the building are also difficult to carry out because each of the different software used needs to be updated separately. For example if a structural engineer insists that an additional column needs to be inserted, the architect needs to modify his drawing and the quantity surveyor would need to update his software. Due to these reasons it is easy for inconsistencies to occur.

A product model overcomes this limitation by capturing the multidisciplinary data of a building into a single data structure (Bjork 1989, Eastman 1992). Product models have been implemented in relational databases, hypermedia software and object oriented databases (Bjork and Pentilla 1991). Most product models have been based on an object oriented hierarchy.

At present multidisciplinary software is not integrated in general. The product models available today are for narrow domains. It is unlikely that a single product model would be developed in the immediate future that is capable of handling all aspects of analysis, design, construction and maintenance of a general building structure. This research has shown that not all data needs to be stored in the product model in order to generate a model to represent a multidisciplinary viewpoint. For example the geometry of the product model in this research was modelled as 2D elements, but sufficient data was captured to generate a 3D representation in AutoCAD.

2. OBJECTIVES

This paper describes the following.

(i) A product model based on an object oriented primitive-composite approach.

(ii) Some of the representation issues encountered in modelling.

(iii) The prototype software PROMOD, developed in Delphi.

(iv) Querying of the product model and its interaction with structural analysis and 3D drawing software.

3. THE PROPOSED PRODUCT MODEL

3.1. OBJECT-ORIENTEDNESS

The proposed model is based on an object oriented approach. The major advantage in this methodology is the ease for modelling real life entities and their complex relationships. The model is based on a primitive-composite approach, somewhat similar to that of Howard et al. (1992). The object hierarchy is based on single inheritance.

A class is defined as a blueprint of an object. The beam class for example contains the attributes and behaviour of a beam. The *Depth, Length, Width* are some examples of the geometrical attributes of a beam. These attributes define the properties of that particular class. Methods are used to describe the behaviour of a class. *Calc_Volume* is an example of a method in the beam class that is used to calculate the volume of a beam. Attributes are similar to variables, which are used to store data values, and methods are similar to subroutines. While classes are the blueprints of an entity, instances of these classes are the actual objects. For example *beam1, beam2* and *beam3* are instances of the beam class.

When we define new classes there is no need to define them from scratch, as we can derive new classes based on existing classes. Inheritance provides a classical mechanism to achieve this through a sub class - super class relationships (*is*-a type of relationship). Each sub class, for example a beam, would derive all the attributes and behaviour of its super class, for example a "structural" (see Figure 1). We could then add the unique properties and behaviour of a beam in the beam class.

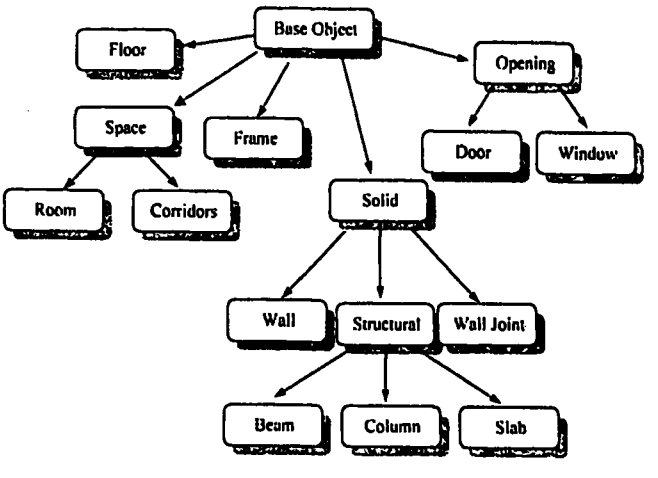

Figure 1 - Building Class Hiearachy

If a sub class is defined with respect to several super classes then we would have multiple inheritance. Although in the real world many objects belong to several classes, it has been found that by avoiding multiple inheritance, the complexity of the object hierarchies can be reduced. Due to this, many new object oriented programming languages, including Java, support only single inheritance.

Composition is another way in which classes can be related together. Here an attribute is used to refer to another class. Composition is useful in a situation where a descendent class doesn't strictly follow an *is*-a type of relationship with the super class (Venners 1998). Using

a mixture of both inheritance and composition, complex entities could be assembled.

3.2. PRIMITIVE-COMPOSITE APPROACH

Howard et al. (1992) proposed a primitive-composite approach to create complex composite objects. A primitive class is defined as a class that represents a single concept such as materials, forces, finishes etc. A primitive classification hierarchy contains primitive classes of one particular concept in increasing specialization, for example geometrical properties and topological properties. There can be several primitive hierarchies as needed for the different disciplines. Any complex entity is built up by combining several primitive entities together. For example a wall entity could be built up from several architectural, structural and services primitive classes. The main advantage of developing a model based on composite scheme is the ability to develop complex and diverse entities. It also reduces the complexity of the objects themselves.

The model proposed in this paper is based on two main class hierarchies: the building class hierarchy (see Figure 1) and the primitive class hierarchy (see Figure 2). The building class hierarchy contains the classes that constitute the building entities such as walls, beams, columns, etc. Each building entity class contains attributes that describe its dimensions and geometry, and stores relations with other objects and methods to perform computations and to save and retrieve itself.

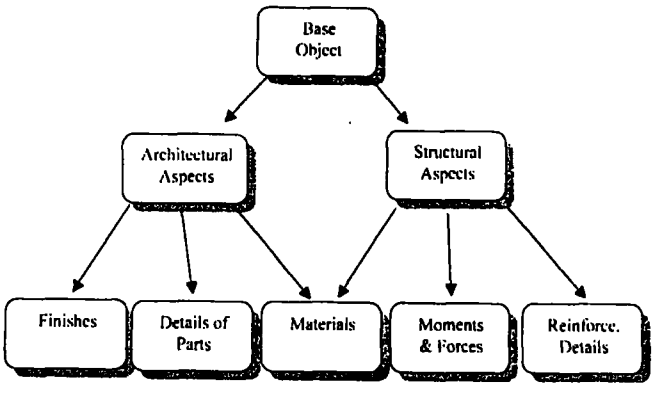

Figure 2 - Primitive Class Hierarchy

The primitive class hierarchy contains primitive classes, which capture various types of multidisciplinary data such as finishes, material properties, forces etc. Each primitive class has attributes to store data specific to the primitive and methods to save and retrieve itself.

A building entity class contains only the bare basic attributes to describe its geometry and dimensions. To store multidisciplinary data in the building entity one has to attach the relevant primitive classes such as finishes, forces etc. to create a composite building entity (see Figure 3). This can be done dynamically as and when needed.

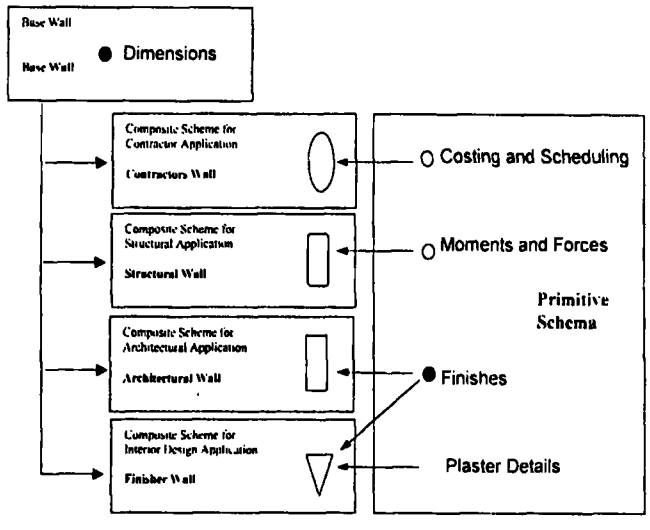

Figure 3 - Primitive Composite Scheme Proposed

The main difference between the implementation of the primitive-composite approach proposed here and that of Howard et al. (1992) is that their composite classes are created through multiple inheritance from the relevant primitive classes. They also consider geometry and dimensions as primitives. In the proposed model the geometry and dimensions are directly implemented in the building object hierarchy, recognizing that these attributes are not multidisciplinary in nature but rather act as the binding mechanism of the different disciplines.

The types of primitives used in this research were predefined composite primitives. However it is possible to make use of basic primitive schemas to dynamically create (in addition to attaching) composite primitives. This becomes important if the product model needs to expand dynamically to cater to a previously unforeseen application domain. The advantage of predefining the composite primitives is that in addition to the increased efficiency, there is also the ease of attaching behaviour to these entities.

Certain simplifications were made in developing the model so that it would enable a feasible prototype to be developed. Only architectural and structural aspects of a building were considered. The buildings considered were of rectilinear type having flat roofs. This simplification isn't a major limitation considering that most office buildings, hospitals etc. are rectilinear by nature.

3.3. RELATIONS AND ENTITIES

The following relations were used in the proposed model.

Components: A list of elements which are the components of the element considered. For example the components of a wall would be its openings such as door and window elements.

Component_of: A list of elements that the element is considered to belongs to. For example a door is a component of the wall element it is contained in.

Parts: A list of similar type of elements that the element considered can be sub divided into. For example *wall1* could be made up of sub-walls *wall1a, wall1b, wall1c* etc. Note that the sub-elements are of the same type.

Part_of: The element that the element considered is part of. For example *wall1a* is part of *wall1*.

Connected: A list of elements a given element is physically connected to. For example *beam1* could be connected to *column3, column4,* and *beam2*. Here there is no distinction made whether the element connected is similar or different to the element being considered.

In the proposed model the main building components are considered to be the floors of the building. Elements such as walls, rooms, etc. are components of their respective floors. Elements like doors and windows are components of the wall they are contained in. Some elements such as beams and columns are components of both their respective floors and frames.

The wall element acts as a container to openings such as doors and windows. One wall can consist of several sub-walls. The boundaries of these sub-walls are columns or other walls that intersect them. A room is defined as an enclosed space surrounded by walls (Bjork 1992). In each floor the spaces evolve after detailing the wall elements.

Structural elements are defined using a centre-line representation. A frame object acts as a container of structural elements. Beams and columns are defined with respect to a frame element. A frame is defined as a vertical plane of fixed length. The column and beam elements are defined with respect to the frame element.

4. ISSUES IN REPRESENTATION

4.1. GLOBAL VS. LOCAL COORDINATES

Since the proposed product model must generate the geometry of the building in 3D, all elements must be described using 3D geometry. The geometry of the elements can be described either by using global or local co-ordinates.

In the proposed model, wherever possible local geometry was used. The main advantage of this approach is its flexibility. For example if the coordinates of a door or window in a wall are defined using local geometry with respect to the wall, there would be no need to make alterations to the coordinates if the wall was shifted. The global coordinates of the door or window could be computed easily. Another advantage is that it is easier to define coordinates in local terms.

Each floor has a reference point, which has a 3D global coordinate. The walls and frames in a given floor are defined with respect to this reference point. The doors and windows in a wall are defined with reference to the respective wall. The x, y (i.e. horizontal) coordinates of structural elements are defined with respect to the frame. The z (i.e. vertical) value is defined with reference to the floor.

Figure 4 shows the geometrical relationships between the different building elements. By using this approach, any change in the coordinates of a major element would not necessitate a change in the coordinates of a minor element. Also, because there is only one path to a given element from another for geometrical definition, inconsistencies due to data redundancy are avoided. There are of course many other logical relationships (e.g. *connected*) between these elements. It should be noted that although rooms are containing elements, they are in fact built up from their contained walls.

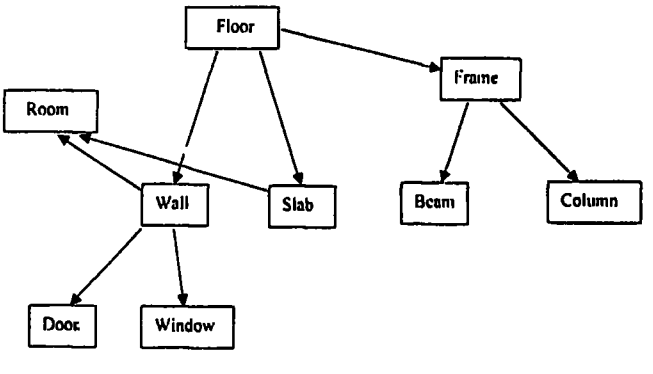

Figure 4 - Geometrical relationships between different elements

4.2. LINE VS. SOLID REPRESENTATION

One of the objectives of this research was to capture sufficient geometrical information to generate a 3D model of a building. There are several techniques available to represent three dimension models. These include wire frame representations, surface modelling and solid modeling.

A wire-frame model is the simplest 3D representation. This provides a skeletal description of an object. A surface model extends a wire-frame model with additional data about the outer surfaces. Both these techniques cannot be used to capture the solidity of an object. Solid modelling can be used to capture this. Different techniques have been developed to represent solids, among them boundary representation (brep) and construction solid geometry (CSG) are prominent. A solid modelling representation allows one to build up complex solid models by performing set operations on primitive solid models. These set operations such as union, intersection and difference could be used to combine elements of a building to generate a single model. It is also possible to perform computations like calculation of volume, and derive other mechanical properties such as centre of gravity and moment of inertia of these solid models (Saeed 1994).

In the prototype that was developed only two-dimensional graphics are handled directly by the product model itself. AutoCAD was used to visualize the product model in three dimensions. The representation chosen was solid modelling. In order to generate the three-dimensional objects from AutoCAD it was necessary to store necessary parameters for each object. A beam or column could be represented as a line along the centre line. The additional parameters would include attributes such as depth and width in the case of a beam.

A three dimensional model of a beam (or column) could be generated by storing the coordinates of the ends of the idealized centre-lines of the beam (or column), in addition to the depth, width and length. Due to the centre line approach adopted it would be necessary to carry out certain corrections in beam column connections. This could be accomplished by making use of the beam-column connected relationship (see Figure 5).

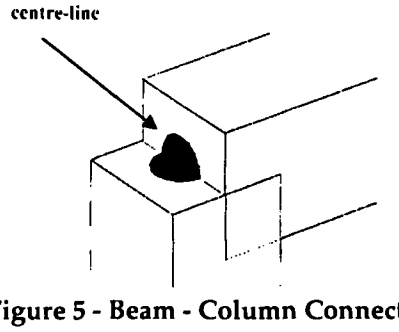

Figure 5 - Beam - Column Connection

A wall is also represented using its centre line representation along the level of the floor. Additional properties needed include length, width and height. There are corrections to be performed when it is converted to a three-dimensional drawing. This includes corrections for wall to column and wall to wall connections. The wall to column and wall to wall connection can be established through the *connected* relationship.

5. IMPLEMENTATION

5.1. DEVELOPMENT TOOL

In order to develop the prototype of the model, the requirements were that the design tools used to implement the model be capable of object oriented modelling, interacting with other software, handling graphics, providing a Rapid Application Development (RAD) environment, and storing data in an object oriented database.

Some of the tools considered were object oriented modelling shells, CAD software and general purpose programming languages. Delphi was chosen for developing the prototype since it satisfied most of the requirements. The software developed is called PROMOD.

Delphi, unlike C++ and Java, is a proprietary language. Delphi was developed as the successor to Turbo Pascal; it has a similar RAD environment as Microsoft's Visual Basic. Although Delphi has not become as popular as Microsoft's Visual Basic, it has established itself especially within the engineering community. The new Windows 95 version of PROKON (structural analysis and design software) was developed using Delphi. Delphi is based on a dialect of Object Pascal. It has object-oriented support based on single inheritance. Unlike Visual Basic it has a true native compiler and produces fast, smaller executables. The performance of software developed in Delphi is comparable to C++.

5.2. PRIMITIVE CLASSES

The primitive classes were defined based on an instance hierarchy as shown in Figure 2. Each primitive class has its own save and retrieve methods. If computations are necessary within the primitive class, these could also be incorporated as methods. The implementation was carried out in such a way that primitive components (instances of the primitive classes) could be added to the building components dynamically. This was achieved by storing all primitive components as a list in the respective building components.

A primitive component could be declared locally or globally. Global primitive components are stored under a global list called *primitive_list* (see Figure 6). Several

building components could be attached with one single primitive component. For example if all external walls of a building have the same architectural finish, we can instantiate the primitive finish class and create one global primitive finish component. The wall components can now be attached with the global primitive component. In the above example (see Figure 6) the two primitives *Finish-1* and *Material-3* are global primitives and are stored physically in a list called *primitive_list*. The building components *Wall-1* and *Wall-5* share the *Finish-1* primitive. The *Beam-2* and *Column-4* components share the *Material-3* primitive.

A local primitive component is defined within the building component. It is meant to be used only by that building component. Usually the moments and forces of structural components would vary from member to member. Due to this each structural member would need a separate moment and forces primitive attached to it.

There can be situations when multiple instances of the same primitive could be attached to a building component. For example an external wall would have two types of finishes, for the outside and inside. For each building element, the type of primitives attached and the number of instances attached for each type is kept track of. Each primitive class has a unique class identifier which is an integer starting from zero. The primitive component types that are attached to a building component are kept track of by the *PrimitiveBts* property. The number of instances attached for each type is maintained in a list called *PrimitiveIns*. When new primitives are attached to the building component, it checks the *PrimitiveBts* property to find whether the primitive is already declared and update the number of instances attached as appropriate.

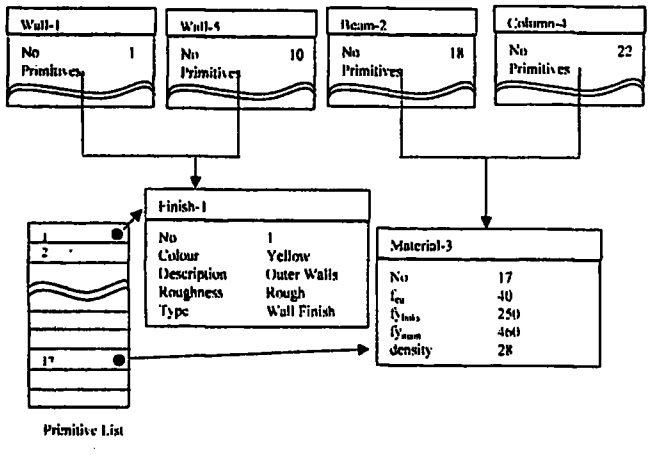

Figure 6 - Representation of Global Primitives

The *PrimitiveStatus* property, which is also defined under each building component, is used to indicate the global/local status of the primitives attached. The implementation of this is similar to the *PrimitiveBts* property. For an attached primitive the bit value unity indicates that it is a global primitive, while the value zero means that it is local.

5.3. BUILDING CLASSES

The building class hierarchy is shown in Figure 1. The functionality of a typical building component (instance of the building class hierarchy) is implemented in the building component base object. The relationships components, *component_of, parts, part_of,* and *connected* are declared as lists. These attributes are used to establish relationships between instances of the building classes. Most relations are implemented as a two-way relationship. For example if *beam1* is connected to *column2*, the *connected* property of *beam1* would contain the *ldno* of *column2*. Similarly the *connected* property of *column2* would contain the *ldno of beam1*.

Some relationships such as *parts* and *part_of* work together. For example *if wall1* is made up of *wall1a, wall1b* and *wall1c*, the *wall1's* parts property would contain the *ldno* of *wall1a, wall1b* and *wall1c*. On the other hand, in each of *wall1a, wall1b* and *wall1c* the *part_of* property would contain the *ldno* of *wall1*. There are additional relationships which are derived by the product model. Table 1 summarizes some of the above ideas for a wall element.

The methods include those to display, get reference point, store, retrieve, get details of components attached, get the primitives attached to it, get data about the space the component occupies, store its geometrical data into a neutral file format used to generate a 3D drawing, etc.

Table 1
Description of Wall Elements

Relationship	Details
Component of	rooms the wall belongs to
Components	doors and windows defined within the wall
Connected	other walls it is connected to
Part of	larger wall this wall belongs to
Parts	sub-walls, of defined
Floor	Floor which the walls are attached to

A building is defined as an aggregate of instances of the building class hierarchy. The *EntityType* property contains the name of the building class a given building component belongs to. All building components defined are stored in a global list called building components. Each instance is uniquely identified by a unique integer called *Idno*. This *Idno* is used in establishing relationships between instances. The relational attributes *components, component_of, parts, part_of* and *connected* contain links to other instances via this *Idno*. The storage of local primitive components are carried out by the respective building components. At present, the above data is entered into a spreadsheet (see Figure 7), although a graphical user interface is being currently developed for entering the building component data.

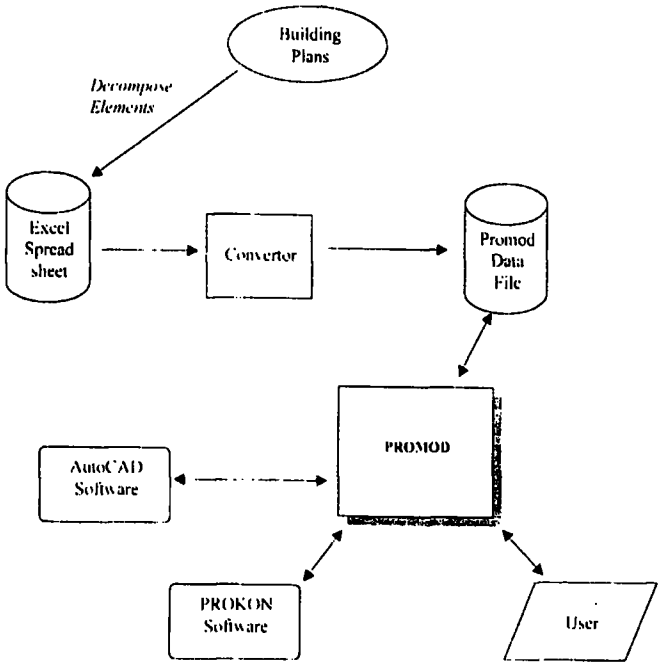

Figure 7 - Entering Product Model data to PROMOD

6. INTERACTION AND QUERYING

6.1. INTEGRATING WITH AUTOCAD

AutoCAD 2000 can act as an ActiveX server. This means that using a programming environment like Visual Basic or Delphi a program can control an AutoCAD session. The product model initially creates a neutral data file containing geometrical data in 3D of all building elements. Then it runs a driver program written in Visual Basic which reads the neutral file and generates a three dimensional model of the building in AutoCAD. Each building element type is created in a separate layer enabling the user to then turn on/off any undesired details. This 3D model is generated making use of solid modeling. Visual Basic was used in the development of the driver because of the better support provided. Figure 8 shows a 2D view of a sample building and Figure 9 shows one of its 3D views (walls only; no columns, beams or openings).

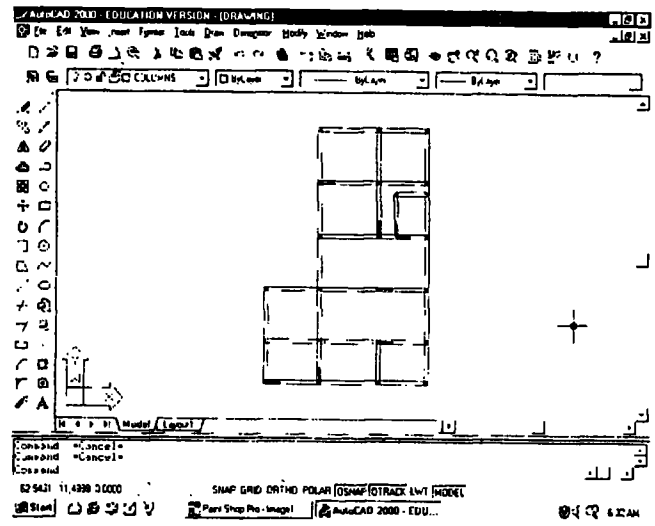

Figure 8 - 2D view of sample building in AutoCAD

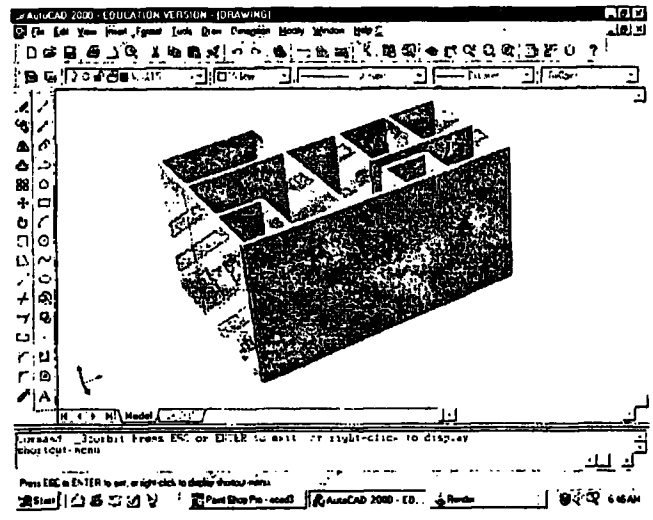

Figure 9 - 3D View of walls in sample building

6.2. INTEGRATING WITH PROKON

A DOS version of PROKON was used during the development of the prototype. The user selects the frame to be analysed from the prototype. An algorithm determines the nodes and elements in the frame. The values of A (cross sectional area) and I (second moment of area) for the elements are calculated by the methods of the respective structural objects. This data is then converted into an input file format which can be read by PROKON. After PROKON is run, the output file which is also in

ASCII format is read back, and the product model is updated with the forces and moments generated. Figure 10 shows a PROKON output for one frame of the sample building.

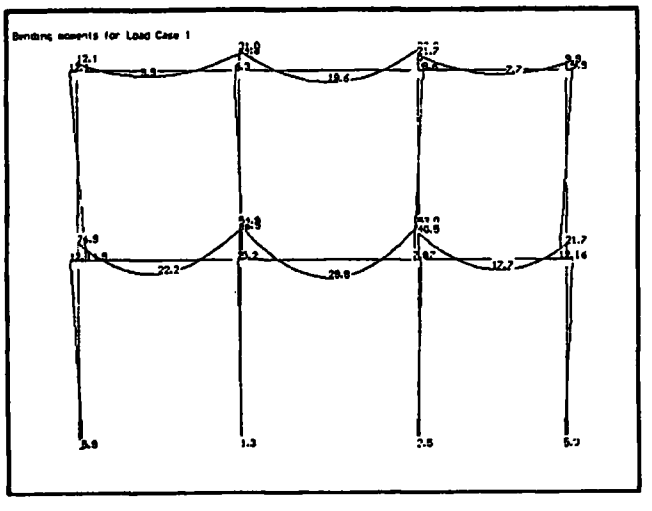

Figure 10 - Bending Moment Diagram of sample building frame

6.3. QUERYING OF PRODUCT MODEL

Some examples of how PROMOD can be expanded for handling user queries are presented below.

Finding ratio of a room's openings to area - By using the *components* relationship of a room we can determine the walls that make up that room. By making use of a general algorithm to calculate an area of a polygon the area of the room can be determined. For each wall the openings can be determined by making use of the *components* relationship of that wall. The area of each opening can be found and aggregated and the ratio of the openings to the room area can be found.

Finding an adjacent room - By using the *components* relationship of a room we can determine the walls that make up that room. For each wall we can check if the *component_of* relationship contains another room. All such rooms would be adjacent to the original room.

Structural elements that support a room - By using the *components* relationship of a room we can determine the slab that supports the room. The *connected* relationship of the slab can be used to determine the beams that support the slab. For each beam the *connected* relationship can be used to determine the columns that support the beams.

7. DISCUSSION AND CONCLUSIONS

An object-oriented approach was seen to be eminently suitable, if not indispensable for a building product model, especially because there is a clear mapping between the object classes and building entities. Furthermore, building elements (especially structural ones) can have a large number of calculations associated with them, and these can be expressed in methods that are encapsulated in the classes.

The proposed conceptual model and its implementation demonstrated the flexibility of the primitive-composite approach that uses two separate hierarchies, one for building elements and the other for primitives. It also allows complex entities to be easily and dynamically built up out of simple ones. The possibility for primitives to be declared as global (in addition to local) entities can also reduce the amount of data entry.

The floor and frame were the main architectural and structural building components respectively, constituting horizontal and vertical planes respectively. It was found more convenient to define the walls (solid skeleton) of a building with respect to a floor and to allow the spaces (e.g. rooms) to emerge from them. The beams and columns were defined with respect to a conceptual frame (i.e. a plane); this caused the emergence of the actual structural frame.

The semantic relations *components, component_of, parts, part_of* and *connected* were found to be sufficient for all the functions of this model. These relationships reduce the complexity of the modelling, enable entities such as rooms to emerge, and allow the model to make inferences and to function as a rich database.

Geometry was implemented directly in the building elements rather than as a primitive. Since all building elements were represented as lines, and subsequently fleshed out according to their cross sectional dimensions, corrections had to be made for overlapping etc. at joints between elements. This drawback was considered negligible compared to the complexities of representing the elements as solids.

The use of local geometry (e.g. for contained elements such as doors and windows with respect to the containing wall) greatly facilitated data entry. It was found that a maximum of four entries was sufficient for all local geometry.

Integration with AutoCAD for 3D visualization and with PROKON for structural analysis was demonstrated.

Future work on the model will make graphical data entry possible. This may require a different hierarchical structuring of building entities, as the emphasis will be on graphical properties rather than on structural or architectural ones.

8. ACKNOWLEDGEMENTS

The authors wish to thank the National Science Foundation (NSF), formerly the Natural Resources, Energy and Science Authority (NARESA) of Sri Lanka for financial support through research grant RG/95/E03. They also wish to thank the Civil Engineering Department of the University of Moratuwa, Sri Lanka for infrastructural and other assistance required for the execution and continuation of this research.

9. REFERENCES

Bjork, B.C. (1989), *"Basic structure of a proposed building product model"*, Computer Aided Design, Vol 21, No 2, pp. 71-78.

Bjork, B.C. (1992), *"A conceptual model of spaces, space boundaries and enclosing structures"*, Automation in Construction, Vol 1, No 3, pp. 193-214.

Bjork, B.C. and Penttila, H. (1991) *"Building product modelling using relational databases, hypermedia software and CAD systems"*, Microcomputers in Civil Engineering. Vol. 6, No. 4, pp. 267-279.

Debney, P.M. (1999), *"CAD - today is only just the beginning"*, The Structural Engineer, Volume 77, No 3.

Dias, W.P.S. (1996) *"Multi-disciplinary product modelling of buildings"*, ASCE Journal of Computing in Civil Engineering, Vol 10, No 1. pp 78-86.

Eastman, C.M. (1992) *"A data model analysis of modularity and extensibility in building databases"*, Building and Environment, Vol 27, No 2, pp. 135-148.

Howard, H.C, Abdalla, J.A. and Phan, D.H.P. (1992), *"Primitive-composite approach for structural data modelling"*, Journal of Computing in Civil Engineering, Vol 6, No 1, January, 1992. pp. 19-39

Saeed, M. (1994), *"Shared understanding in synchronous collaborative design"*, PhD Thesis, University of Sydney.

Sheperd, R.B. (1996), *"Mastering AutoCAD AEC"*, Addison Wesley Longman Limited.

Venners, B. (1998), *"Inheritance versus composition"*, Java World Magazine, November 1998, www.javaone.com.