Integrating Design Tools for Building Design

ROBERT W. AMOR, LINDSAY J. GROVES and MIKE R. DONN

[†]Department of Computer Science, [‡]School of Architecture
Victoria University of Wellington
Wellington, New Zealand

Abstract This paper examines the methods being devised at Victoria University to provide an intelligent, knowledge based interface to expert systems, simulation packages and CAD systems for architectural design. Currently a large amount of information is duplicated in describing a building to different design tools. To enable designers to use different design tools to evaluate a building design, without having to duplicate so much of their effort, we propose a system that uses a common building model to represent their knowledge of a building. On top of this building model we provide an interfacing mechanism to move information between the different design tools and the common building model.

KEYWORDS: Computer, Research, Systems (Knowledge based system, knowledge representation, user interface, integrated design)

ABSTRACT

This paper examines methods being devised to provide an intelligent, knowledge-based interface to expert systems, simulation packages, and CAD systems for architectural design. Currently, a large amount of information is duplicated in describing a building for different design tools. To enable designers to use different design tools to evaluate a building design, without having to duplicate so much effort, we propose a system that uses a common building model to represent their knowledge of a building. On top of this building model, we provide an interfacing mechanism to move information between the different design tools and the common building model.

INTRODUCTION

In a world that expects its buildings to be one-off designs with increasing performance and reliability, the building design team is under great pressure to use as many of the available design analysis tools as possible. Numerous CAD systems, simulation programs, and, more recently, expert systems are being marketed to meet this demand.

Each of these tools requires a detailed description of the building or structure being modeled. Constructing a description of a building for entry into a simulation program from existing plans can take several days. When the same building is to be analyzed by several different expert systems or simulators, similar base data have to be entered into each tool, so there is considerable duplication of effort and many opportunities for errors.

Each tool is generally designed to perform one particular type of analysis or simulation. Few are designed with a view toward interfacing with other tools. This leads to a situation where many tools exist in isolation and there is little or no transfer of information between tools in different areas (see Figure 1). The exception to this is in CAD systems, where it is now realized that some sort of exchange protocol is needed to allow data to be moved from one system to another. This has lead to several different protocols from the major vendors, and only recently has an ISO standard been defined for the exchange of data between CAD systems (IGES 1986).

R. W. Amor is a Masters Candidate, and L. J. Groves is a Senior Lecturer, Computer Science Department, Victoria University, Wellington; M. R. Donn is a Senior Lecturer, School of Architecture, Victoria University, Wellington.

Further problems arise from the "black box" nature of many analytical programs. While it is relatively simple to encapsulate the mathematical expertise of a structural engineer in a program to calculate the load on a beam, it is far from simple to encode their expertise in the application of the mathematics. This means that a naive user requires assistance, not only to interpret the results, but also to enter the initial building data accurately.

The problems outlined above suggest that there is a need for a system that allows a user to create building descriptions for different tools without having to learn the command syntax and other idiosyncrasies of each tool. The user should be able to transfer relevant data about a building from one system to another, instead of having to duplicate the effort. This situation is illustrated in Figure 2, where all tools can exchange information through a common building model. These goals seem to be achievable with the current state of technology and we intend to show that such a system can be created.

Internationally there has been considerable interest in projects of this nature, and we have gained from the experiences of Björk (1989b), Clarke et al. (1989), Howard and Rehak (1986), Levary and Lin (1988), Rehak et al. (1984), and Selkowitz et al. (1986).

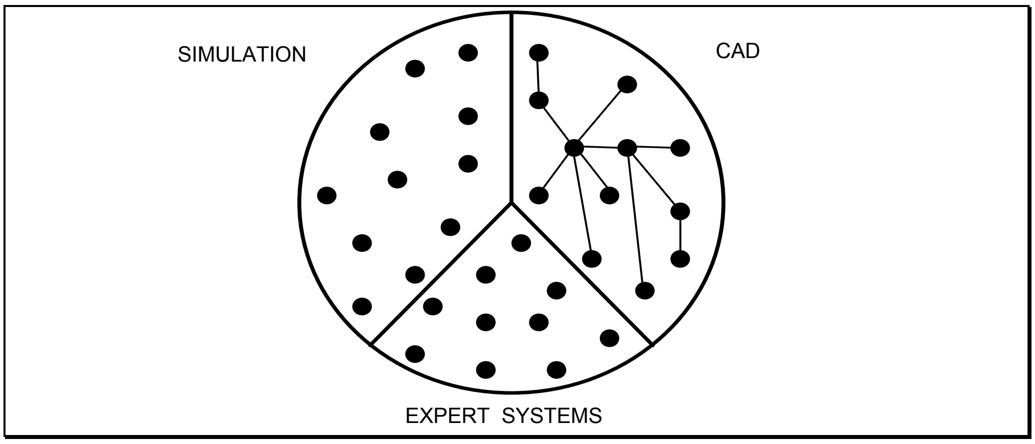


Figure 1. Design tools: The current situation

The system we are developing will allow a single description of a building design to be constructed and used to provide the necessary data for a variety of design tools. To do this, we need a framework for representing a building design that can accommodate all the information required by the tools we are considering, and a method to represent this information. We also need a mechanism for interfacing the various tools, allowing the data required by each tool to be extracted from the central knowledge base and allowing data produced by each tool to be added. Finally, we need an interface between the system and the user.

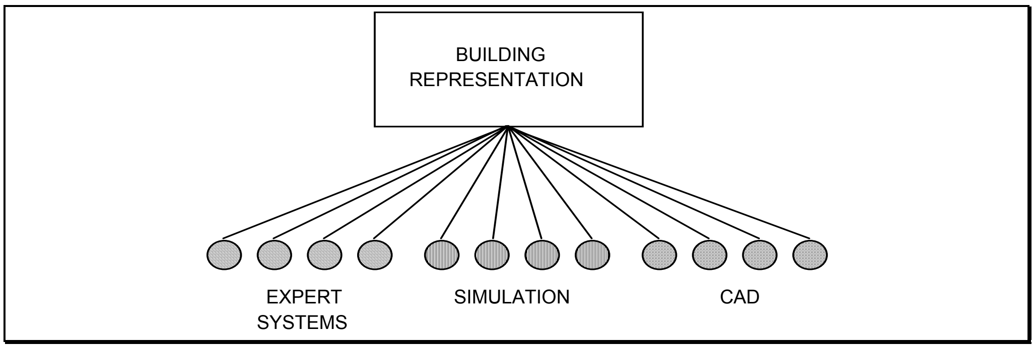


Figure 2. Design tools: The future

DESIGNING A COMMON REPRESENTATION FOR BUILDING DESIGNS

In devising a framework for representing building designs, we have closely examined the description languages for a range of tools to determine the kinds of information they require. These tools were selected to cover a broad spectrum of building design issues, including structural analysis, lighting design, thermal design, and code checking (see Appendix A).

We have also studied the draft product data exchange standard, integrating the STEP and PDES standards, which is being considered by ISO as a basis for an ISO standard (Wilson and Kennicott 1987).

From our analysis of the data base of attributes, and the structuring of objects in the tools, we determined that the building representation would have to satisfy three major criteria. It should be:

Comprehensive: The representation should be able to represent the types of information required by a reasonable range of the design tools.

Modifiable: The representation should be structured in such a way that new kinds of information required by new tools can be added without major modifications to the structure. Such additions should not require major changes to previously defined interfaces.

Nonredundant: To minimize redundancy in the representation, any data captured in the description should only exist in one place.

For the current project, we have restricted our attention to the building's structure; its heating, ventilating, and air-conditioning (HVAC) systems; and its lighting. While this decision has limited the range of tools that will be able to operate off this building representation, the building representation created will be easily extendable to encapsulate the knowledge needed for new design areas.

Evaluation of the Tool's Attributes

To determine the range of objects and attributes needed to represent a building and to ensure that the building representation we choose will encapsulate all the data needed for the different design tools being considered, we mapped out the objects and the relationships between objects used in these tools. These charts allow us to compare the structures used by different tools to describe similar objects.

To keep track of the massive number of attributes involved in the range of tools examined, we established a data base of all attributes in the tools. For each attribute describing an object, there is one record in the data base. This record contains:

-- the name of the tool of which this attribute is part

-- the object in the tool of which the attribute is part

-- the attribute's units

-- default values for the attribute

-- range of possible values for the attribute

-- type of the attribute (e.g., real, integer, string)

-- length of the attribute when written in the data file

-- format of the attribute when written in the data file

-- constraint on the cardinality of the object (i.e., number of objects allowed)

-- a description of the attribute and its use in representing the object.

This data base has become invaluable for assimilating the range of attributes used by design tools to describe an object. The data base can be queried to find all attributes needed by a range of tools to describe a particular object, such as a door. The result of the query lists the various ranges, defaults, and units on the attributes. With this information, it is a simple task to choose the general attributes needed to describe an object and to define reasonable default values and ranges for these attributes.

The description retained in the data base of an attribute proves especially useful when one might want to find all attributes that relate to a specific object. The description can identify all attributes of a specific object, even when different tools use different names or classes to describe that object. The description also helps distinguish between attributes that seem to have the same meaning but are subtly different or to identify common attributes between classes of objects. This analysis helps identify the classes and subclasses of objects that are likely to be needed in the final structure of the building representation.

Examining the data base has highlighted the problems that occur where different tools view a building from different perspectives. For example, walls are often represented in different fashions (see Figure 3). One system may represent a wall as being the area enclosed by joining a set of points in 3-dimensional space (Figure 3a); another package may represent a wall as having a bottom left corner at some point in 3-dimensional space and a height and length specified as an offset from this point and at a certain angle from north (Figure 3b); and another may represent a wall by approximating it to a column with a large width (Figure 3c). These systems all represent the same type of wall but are representing it in very different manners. Our representation of classes must be flexible enough to handle many representations.

The data base of attributes has helped ensure that the building representation conforms to the three criteria specified above. We can guarantee a comprehensive representation, as we define objects and their attributes in our building representation from a conglomeration of all the attributes held in the data base. The objects and classes that we choose are easily modifiable, as they have been selected as generalizations of the classes used by as wide a range of tools as possible. With the data base, it is also possible to identify which objects require similar information and to minimize redundancy by selecting classes to integrate similar objects.

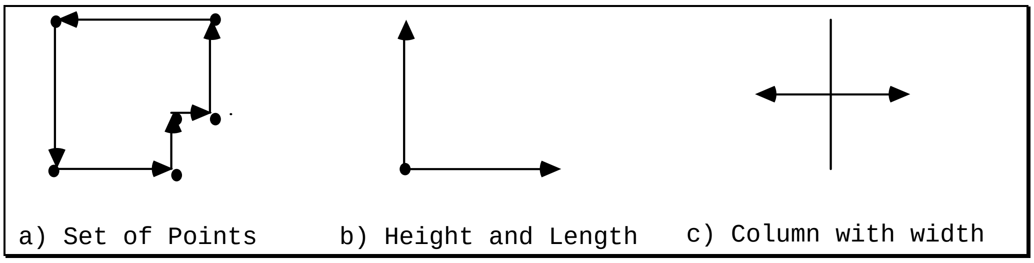


Figure 3. Wall representations

Abstraction Hierarchy

From the description languages for the tools we studied and the data base of attributes of the objects, it was apparent that we should describe a building in a hierarchical manner, as a collection of systems, each consisting of a number of subsystems, etc. This hierarchy should provide classes of objects, corresponding to the systems and subsystems in buildings, each of which is described by a set of attributes. It was also clear that this hierarchical representation should allow inheritance of values from higher levels in the structure to lower levels and should provide for defaults.

We have defined an abstraction hierarchy that breaks the description of a building into five levels: building, system, subsystem, object, and part (see Figure 4).

Building level: This is the top level of the data structure. There is only one building object for each building. It contains only building-specific data, such as the location, orientation, number of floors and gross area, height, etc.

System level: This level breaks up a building into its main functionally distinct systems. For our purposes these are: the spaces in a building, the structure, HVAC system, lighting system, lifts, power, communications, etc.

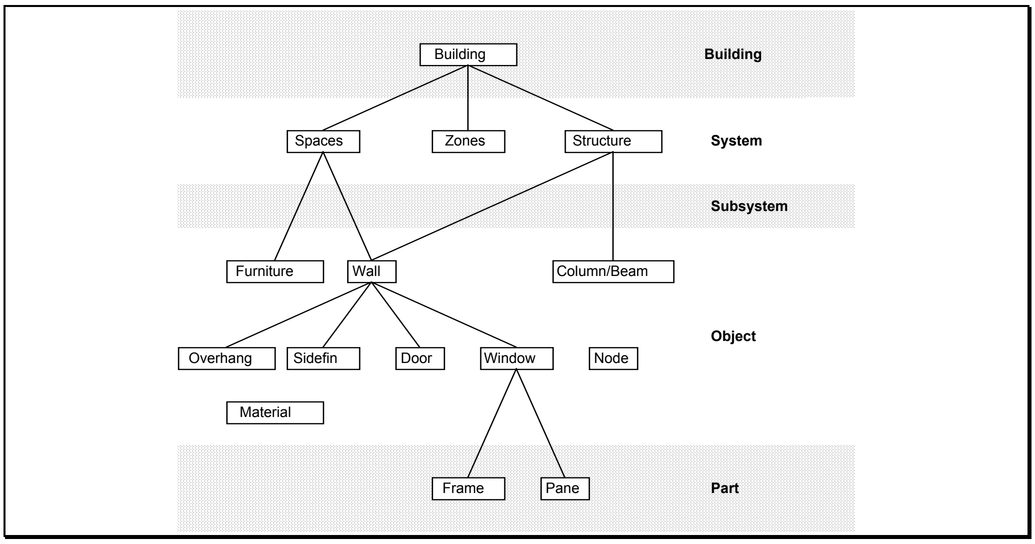


Figure 4. Abstraction hierarchy

Subsystem level: This level is used to break a system into its major functional components. Thus far, we have used this level mainly in our description of the HVAC system that is broken up into plant, distribution, and terminal. These are further subdivided to represent all the major functions of the components in an HVAC system. For example, the plant subsystem is further divided into heat production, heat removal, humidity control, and storage.

Object level: This level represents each physical object in its own generic class. All definable objects are listed on this level. The types of objects are walls, doors, windows, columns, boilers, chillers, fans, diffusers etc.

Part level: This level is used to represent the constituent parts of components at the object level. For instance, a window at the object level is represented as a frame and panes at the part level. The same applies for some plant equipment, e.g., fans, condensers, and cooling coils are parts of a chiller.

This hierarchy turns out to be very similar to the one proposed for the RATAS project in Finland (Björk 1988; Björk and Penttilä 1989a; Björk 1989b; Enkovaara et al. 1988; Hannus 1987), although their hierarchy represents different objects at different levels due to the goals of their project.

REPRESENTING BUILDING KNOWLEDGE IN A FRAME

The abstraction hierarchy described above could be implemented in a number of ways, for example, using a relational data base or an object-oriented programming language. Given the need for inheritance and defaults and the need for flexibility, we have decided to use a frame system, implemented in

Prolog. We use four types of relations to describe the relationships between objects: *ako*, *is-a*, *part-of*, and *connected-to* (see Figure 5).

Ako: This relation describes the hierarchical structure among classes of objects. Each class in the *ako* chain inherits the attributes of all classes above it and can introduce additional attributes. This is useful where common attributes are shared by several classes of objects. A good example of this hierarchy arises in the description of walls, where the classes further down the hierarchy from the generic wall are distinct wall types (i.e., exterior wall, interior wall, structural wall, trombe wall, underground wall, and partition), all of which inherit the general attributes of a wall, such as position, height, width, tilt, surface coefficient, and solar absorptance fraction.

Is-a: This relation describes the relationship between classes of objects and instances of those classes. An instance of a class inherits the attributes of that class. The values of inherited attributes are also inherited, except where these are explicitly overridden.

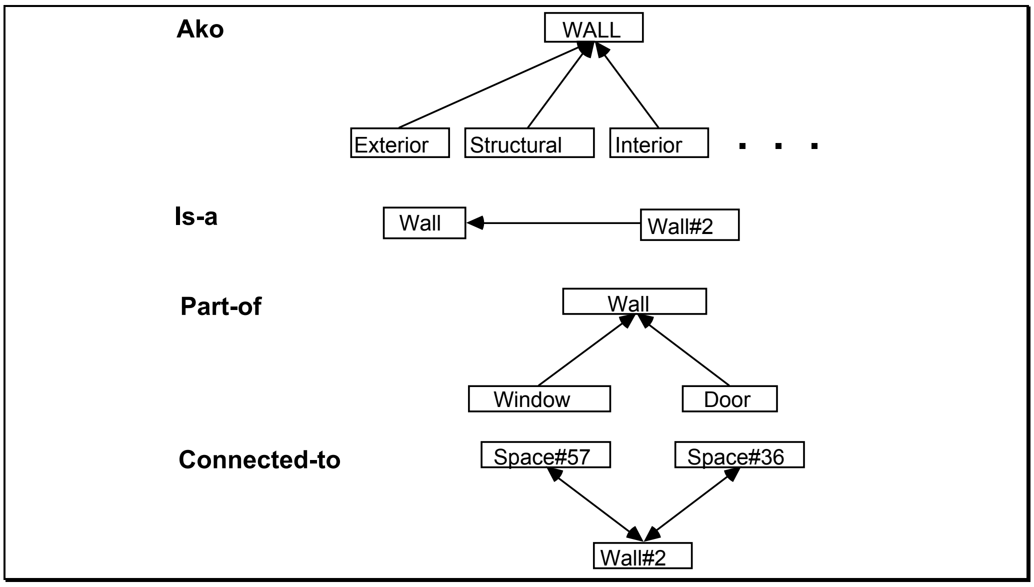


Figure 5. Frame relationships

Part-of: This relation describes the relationship between an object (or class of objects) and its components. These links correspond to the links in the abstraction hierarchy. For example, a window and a door are *part-of* a wall. A portion of the *part-of* relation is shown in Appendix B.

Connected-to: This relation describes connections between objects at the same level in the abstraction hierarchy. For example, a wall is *connected-to* two spaces, one of which may be the exterior.

In keeping with the frame systems semantics, we conceptualize our frame system in five levels. These are:

- Frame: Holds all attributes needed to define an object.
- Attribute: Name of an attribute of the object.
- Attribute value: Contains the value for the attribute.
- Facet: Name of a property of the attribute.
- Facet value: Contains the value for the facet.

To simplify the implementation of the frames system from a programming point of view and to improve the speed of the system when looking through inheritance paths, we have split our frame level into three separate segments. These are:

Class frames: The definition of all classes needed for the common building representation.

Tool frames: The definition of objects available inside each tool. The attributes in these frames have additional facets over the class frame attributes to describe the formats and lengths of the attributes in the tools data file.

Instances: Instances of either of the frames defined above.

Frame Attributes

For each object in the abstraction hierarchy, we have specified the attributes required to describe that object. With the use of the data base of attributes in tools, we have guaranteed that the set of attributes used is sufficient to describe the objects in all tools that we have examined.

In several cases, we found that different tools aimed at the same area of design describe components of a building at different levels of detail. For example, a material may be described just by its thermal resistance or by detailing its thickness, conductivity, density, and specific heat. In such cases, we have used the most detailed attributes for a component and provided methods for calculating general attributes where appropriate.

In designing our building representation, we have carefully considered the structures and attributes used in the tools defined in our data base and structured the *is-a* hierarchy to maximize the use of inherited attributes.

To simplify the implementation of the system, we define nine separate sets of attributes and relationships that can hold in our frame system and split all attributes among these levels. In addition to the relationships previously defined for the generic frames system, we have defined these levels:

Object-Cardinality: Defines the number of this object that can exist in the system. Used mainly in tool's frames to record restrictions on objects posed by the maximum number of objects the tool can handle, i.e., maximum number of walls allowed in a particular design tool is 40.

Comment: Description of the particular class or object being represented. Used to provide help to the user about the classes that are defined. When used with instances of an object, it can describe the general properties of the instance, i.e., aluminium sliding door with twin inset glass panels.

Values: The attributes relating specifically to this class or object, These attributes do not contain relationships between objects, as defined previously in the frames system, i.e., door height, width.

Parts: These are the constituent parts of this object, i.e., the parts of WALL#29 are DOOR#15, WINDOW#2, and WINDOW#17.

Subclasses: A list of return pointers to the subclasses defined below this class. This is the inverse of the **Ako** relationship, i.e., the subclasses of the class wall might be trombe wall, exterior wall, interior wall, etc.

Facets on Attributes

Attached to each attribute defined in the building model and the tool's classes are the facets that describe the properties of the attribute. These properties can be used to determine the values or restrictions on the attribute. The major facets defined for our frame system are:

Default: Contains the default value for the attribute. Is used to determine the value of the attribute if no value is supplied and the user requests the attribute's value, i.e., the default value for the attribute height for the class doors might be 2.2.

Units: Specifies the units in which the attribute is represented. For attributes in the common building representation, these are SI units. For attributes in the tool's frames, the unit is whatever unit the tool uses to represent the attribute in its representation. The mapping system provides automatic conversion of values between the units in the common building representation and the tool's representation.

Range: A list defining the range of values between which an attribute can lie or the set of values an attribute can take. The type of range that is defined is dependent on the attribute type.

Length: The number of characters an attribute can occupy in the tool's data file. This facet also specifies the format of numbers, in terms of the number of digits before and after the decimal point. This is used only by tool's frames and specifies the format under which an attribute must be written. In many cases, it also restricts the range of values or accuracy of numbers represented, i.e., for the attribute height of the class door, the format for one simulation program might be 5.2 (5 digits before the decimal point and 2 after).

Attribute-Cardinality: The number of values this attribute can take. This facet doubles as a required facet by specifying the minimum cardinality to be greater than zero. Mainly used in tool's frames to specify restrictions on numbers imposed by the tool and how much data it can handle.

INTERFACING MECHANISMS

Given the common representation for building designs discussed above, we need to be able to construct an interface between it and each of the tools we wish to use. For each tool, we need to be able to generate the input to the tool in order to run it. We also need to be able to read the output from the tool in order to interpret the results and add information obtained to the knowledge base. Finally, if there are existing data files for a particular tool, we need to be able to read these and construct the relevant parts of the building description from them. This gives the interfacing system a structure as in Figure 6. It is important that interfaces for new tools can be added easily, since this will ultimately determine how successful our system is in practice.

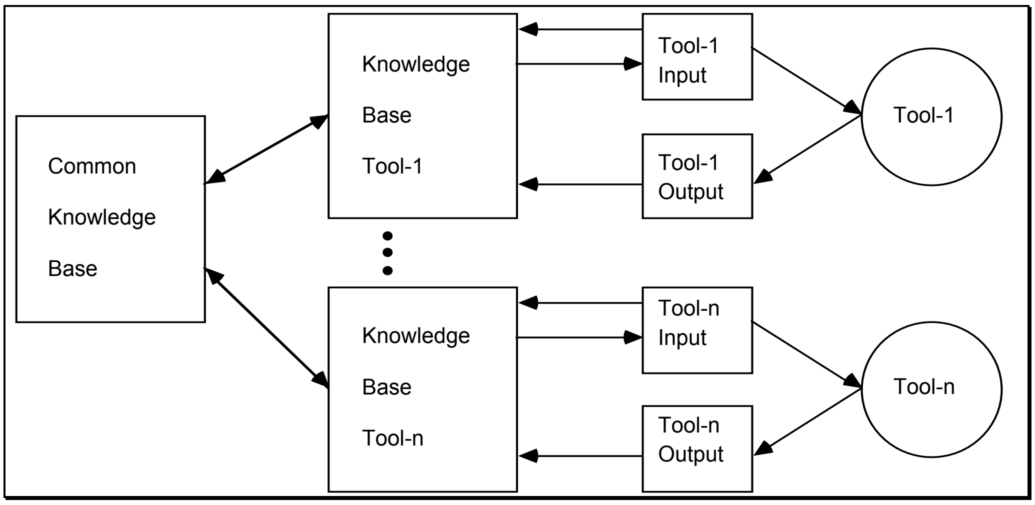


Figure 6. Mapping representation

The Interfacing Mechanism

To construct the data file required to drive a particular tool, the system needs to be able to extract the necessary information from the knowledge base and format it in the appropriate way. This can be viewed as a report-writing task, so it can clearly be done in a straightforward manner. It is also necessary for the user to be able to add other parameters required by the tool, such as the length of simulation, where to find weather data, or what output options are required. To read the output from a tool, or a data file for a tool, we need a parser that understands the structure of the file being read. This then needs to be able to display the results in an appropriate form and make suitable entries in the knowledge base.

To facilitate this process, we perform the mapping between the description of a building in the knowledge base and a data file produced by, or prepared as input for, a specific tool in two stages. The link between the two stages is a frame representation of the information in the data file. This representation mimics the structure of the tool's building description but also incorporates the assumptions about ranges and defaults that the tool makes about the building data.

To construct a data file to run a tool, the user indicates via a simple interactive dialogue what information is required. The relevant information is extracted from the knowledge base and combined with other parameters supplied by the user to form a description of the required file. When this is complete, the file is written in the format required by the particular tool.

To read a file produced by a tool, or an input file prepared for a tool, the process is reversed. We first parse the file and construct a frame representation of it. We then add this information to the knowledge base.

The advantage of splitting the mapping into two stages like this is that it separates the distinct types of mapping. The mapping between the knowledge base and the frame representation for a particular file is only concerned with the relationship between two knowledge bases, not with the syntactic format of the description language for a particular tool. The mapping between the frame representation for a particular tool and a data file for a particular tool is only concerned with the information that tool needs and not with any other aspects of the complete knowledge base.

The splitting of the mapping into two stages also has the added bonus of defining the attributes needed by a specific tool. This is of enormous benefit to the system when we need to determine what extra information is required by the tool we wish to use. From the intermediate representation, we can ascertain what attributes are needed, where they come from in the common building model, and if they are actually present.

Our initial approach to implementing these interface mappings will be to write specific programs to perform the mappings required for each tool we wish to use. These programs, like the frame system described earlier, will be written in Prolog. Ultimately, we would like these interfaces to be generated automatically from descriptions of the input and output formats and the knowledge base. For constructing data files for a tool, we expect to use techniques similar to those used in implementing fourth generation languages. For analyzing the output from a tool (or an existing data file), we expect to use parsing techniques based on a formalism such as definite clause grammars or attribute grammars.

USER INTERFACE

On top of the system described so far, we intend to build an object-oriented graphical interface. This will allow a description of a new building to be entered in a more intuitive manner than offered by current CAD packages. It will also allow the user to view the building being modeled and to enter information that has not previously been provided.

For a system to draw at an object level, there has to be a break from the traditional, general purpose CAD systems. Allowing a system to draw at an object level is achieved in two ways: first, by letting designers create a building design using the objects that actually constitute a building, second, by giving the system some knowledge about the types of objects it is manipulating and the ways these objects fit together.

We intend to follow the work in this area (Kharrufa et al. 1988; Bijl 1985; Szalapaj and Bijl 1984; Balachandran and Gero 1987; Barth 1986) and design a system based on the hierarchy defined in our building representation. This should give a very effective and fast way to enter information about a building, and provide an easy way to ask for missing information from the user.

To this extent, our graphical interface will only allow the user to design with objects that are defined in our abstraction hierarchy. The system will use default values for most attributes of the objects being placed, based on the type of building being designed and the position of the object in the building, e.g., internal or external walls. The user will always see the frame for the object being pointed to or defined and will be able to change the attributes or the defaults for an object at any time. When the system needs a value for an attribute that hasn't been defined, it is envisaged that the user will be queried for the value by displaying the object and its surrounds and then prompting for the value. This should help give the user a feel for the locality and use of the object.

The user will also be able to design a building from scratch with this system. One of our goals in the design of the user interface is to make sure the designer is free to design the building in the manner they deem best. To this extent, our system will allow the user to design with objects from any level of the hierarchy. We try not to impose a design methodology on the designer. This also means that objects can be placed without specifying a large number of their attributes, leaving the attributes to default, or filling in values for attributes at a later stage as necessary.

As results from different analyses of a building are stored in the same structure as the building representation, this allows them to be examined in the same way as one would examine the values for any object in the building. To make the results more useful to the designer, the user interface will allow the designer to perform simple statistical analysis of the output and to generate plots of the data from a simulation. The designer will also be able to compare and contrast the results of analyses of different buildings in this manner.

FUTURE WORK

There are many ways in which the kind of system we are developing could be extended. We will briefly mention two particularly interesting directions for further development.

Distributed Processing

This type of system is begging to be distributed. It should be possible for the various design and analysis tools to run on different machines, all being controlled by our building description system. For example, a user might concurrently run:

- a finite element analysis of the building structure on a super computer,
- a thermal simulation on a local mainframe, and
- an analysis of the output of a code-checking expert system on a desktop PC.

The changes to our system to enable it to handle a distributed environment would be fairly minor and would increase the usability of the system to a PC user, giving them access to much more power than afforded by the design tools now available on a PC.

Learning

An interesting extension to this system would be a learning component that would sit in the background and examine the input and the results from the different tools that are used in the design process. This learner would be set up to examine a limited range of attributes in the representation. For different design decisions in the input, it would look at the differing outputs from the tools and try to generalize from the results.

This learning system could be an intelligent helper for the user, who could ask the system for some form of help in the design process. From what the learner had observed from cause-and-effect studies, it would offer some guesses at the type of changes to the system that would generate the result required, e.g., what suggestions can be made to help lower the heating kW while still maintaining the same comfort level in the building.

CONCLUSION

The system we are developing will allow the designer to examine many different options during the design of a building, with an ease that is not possible with the state of design tools at the current time.

This system will serve several important functions. It will provide an interface between tools and allow movement of information back and forth between the tools and the building representation. It will reduce the effort required to use a new tool, as the user will not be required to learn the syntax and idiosyncrasies of the new tool.

The system will provide a uniform method for entering data for an object, which will be independent of the method used by the tools in which the data will be used. For tools that do not offer a method to visualize the building being modeled, this system will offer a means of visualizing the building. In time it will act as an intelligent assistant to the user by being able to suggest changes to a design that would help meet a certain design goal.

REFERENCES

Balachandran, M., and J. S. Gero 1987. "A model for knowledge-based graphical interfaces." Artificial intelligence developments and applications, pp. 147-163.

Barth, P. S. 1986. "An object-oriented approach to graphical interfaces." ACM Transactions on Graphics, 5(2), April, pp. 142-172.

Bijl, A. 1986. "Logic modelling in computer-aided design." Planning and Design, 13(2), April, pp. 233-242.

Björk, B-C. 1988. RATAS - An object-oriented conceptual building model, Technical Research Centre of Finland, 16p.

Björk, B-C., and H. Penttilä 1989a. "A scenario for the development and implementation of a building product model standard." Current research and development in integrated design, Construction and Facility Management, CIFE, Stanford University, California, 28-29 March, 18p.

Björk, B-C. 1989b. "Basic structure of a proposed building product model." Computer-Aided Design, 21(2), March, pp. 71-78.

Clarke, J. A., J. H. Rutherford, and D. MacRandal 1989. An intelligent front-end for computer-aided building design University of Strathclyde, Scotland.

Enkovaara, E., M. Salmi, and A. Sarja 1988. RATAS Project - Computer-aided design for construction Building Information Institute, Helsinki, Finland, 62p.

Hannus, M. 1987. "Object oriented modelling in AEC applications." Proc. NBS/DATA Nordic Seminar "Neste generations datasystem i byggebransjen", 24 September, pp. 100-115.

Howard, H. C., and D. R. Rehak 1986. "Expert systems and CAD databases." Knowledge engineering and computer modelling in CAD, September, pp.236-248.

IGES. 1986. Initial graphics exchange specification 3.0 National Bureau of Standards, Gaithersburg, MD 20899, USA.

Kharrufa, S., A. Saffo H. Aldabbagh, and W. Mahmood 1988. "Developing CAD techniques for preliminary architectural design." Computer-Aided Design, 20(10), December, pp. 581-588.

Levary, R., and C. Lin 1988. "Hybrid expert simulation system (HESS)." Expert Systems, 5(2), May, pp. 120-129.

Rehak, D. R., H. C. Howard, and D. Sriram 1984. "Architecture of an integrated knowledge based environment for structural engineering applications." Knowledge engineering in computer-aided design, IFIP WG5.2 Conf., Budapest, Hungary, September, pp. 89-117.

Selkowitz, S. E., K. M. Papamichael, and G. M. Wilde 1986. "A concept for an advanced computer-based building envelope design tool." International Daylighting Conference, November, Long Beach, CA, November, pp.496-502.

Szalapaj, P. J., and A. Bijl 1984. "Knowing where to draw the line." Knowledge Engineering in Computer-Aided Design, September, pp. 147-169.

Wilson, P. R., and P. R. Kennicott 1987. STEP/PDES testing draft St. Louis edition ISO TC 184/SC4/WG1 Doc. N 165, September.

APPENDIX A. Design Tools examined

AutoCAD (1988). Autodesk, Inc, Sausalito, CA 94965.

DOE-2.1C (1981). Building Energy Simulation Group, Lawrence Berkeley Laboratories, University of California, Berkeley, California 94720.

ESP (1982). Environmental Systems Performance, ABACUS, University of Strathclyde, 131 Rottenrow, Glasgow G4 ONG, Scotland.

Fenestra (1986). A Personal Computer Aid for Window Design, School of Architecture, Victoria University of Wellington, Wellington.

Hosking, J. G., Mugridge, W. B. and Buis, M. (1987). "FireCode: a case study in the application of expert-systems techniques to a design code", Planning and Design, 14(3), July, pp. 267-280.

Lumen Micro (1987). Interior Lighting Analysis System, Lighting Technologies, 3060 Walnut St, Suite 209, Boulder, Colorado 80301.

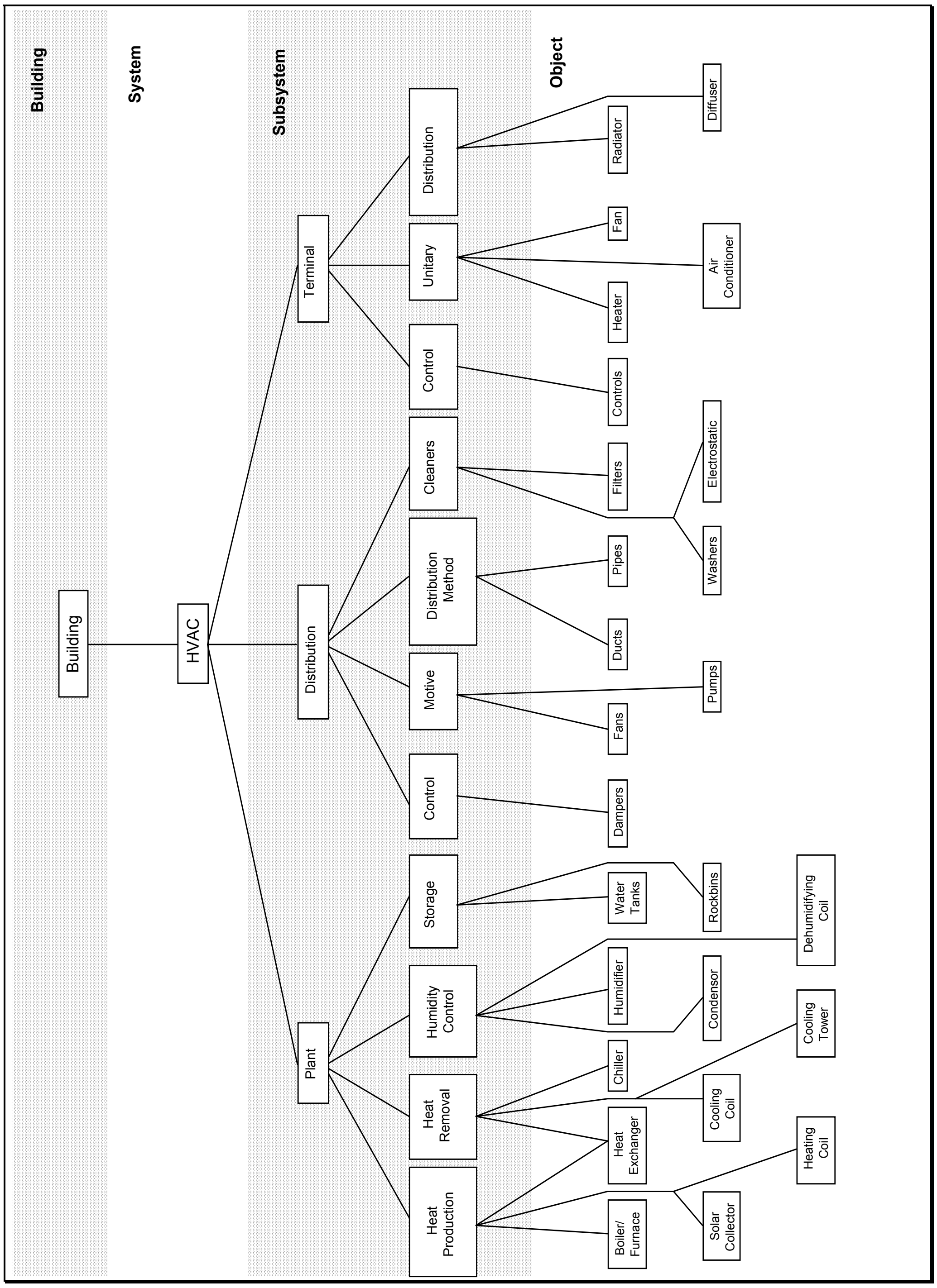
MicroSTRAN (1987). Engineering Systems Pty Ltd, 27 Linden Ave, Pymble, NSW 2073, Australia.

Sonata (1988). t^2 Solutions Limited, The Teamwork Centre, Prince Edward Street, Berkhamsted, Hertfordshire HP4 3AY, England.

SUNCODE (1981). Ecotope Group, 2812 E. Madison, Seattle, WA 98112.

VersaCAD (1988). Versacad corporation, 2124 Main St, Huntington Beach, CA 94965.

APPENDIX B. Sample portion of the part-of relation


Building
System
Subsystem
Object
Building
HVAC
Terminal
Distribution
Plant
Distribution
Unitary
Control
Cleaners
Distribution Method
Motive
Control
Storage
Humidity Control
Heat Removal
Heat Production
Diffuser
Radiator
Fan
Air Conditioner
Heater
Controls
Electrostatic
Filters
Washers
Pipes
Ducts
Pumps
Fans
Dampers
Rockbins
Water Tanks
Dehumidifying Coil
Humidifier
Condensor
Chiller
Cooling Tower
Cooling Coil
Heat Exchanger
Heating Coil
Solar Collector
Boiler/ Furnace