Knowledge Representation for Integrating Design Tools

ROBERT AMOR†, LINDSAY GROVES† and MIKE DONN‡

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

Abstract This paper examines the need to provide a common interface to expert systems, simulation tools and CAD systems for architectural design. We observe that a large amount of information is duplicated in describing buildings to several different tools. We describe a frame based representation for building designs, and an interfacing mechanism to move information between various tools. We also examine the use of a sophisticated graphics front end to the system, and the ability to learn from runs that are performed by the different tools.

1. INTRODUCTION

In the architecture field there exist numerous CAD systems, simulation programs and, more recently, expert systems to aid in the design and evaluation of buildings. These tools include programs to check the loads on beams and walls, simulation of the thermal properties of a building over a period of time and expert systems to check that a building or structure conforms to existing building codes (AutoCAD, VersaCAD, DOE, SUNCODE, MicroSTRAN, Lumen, Fenestra, Hosking 1987).

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

Many of these packages perform very little or no checking of the parameters which are entered for the building, so incorrect parameters will usually not be detected. There is usually no means of viewing a graphical representation of the described building, which would help identify geometric errors such as misplacement of windows, doors etc.

Each tool is generally designed to perform one particular type of analysis or

simulation. They are not designed with a view to interfacing to 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 realised 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). Unfortunately this standard only deals with geometrical data and so is already woefully behind the times in terms of the types of systems being used in the workforce.

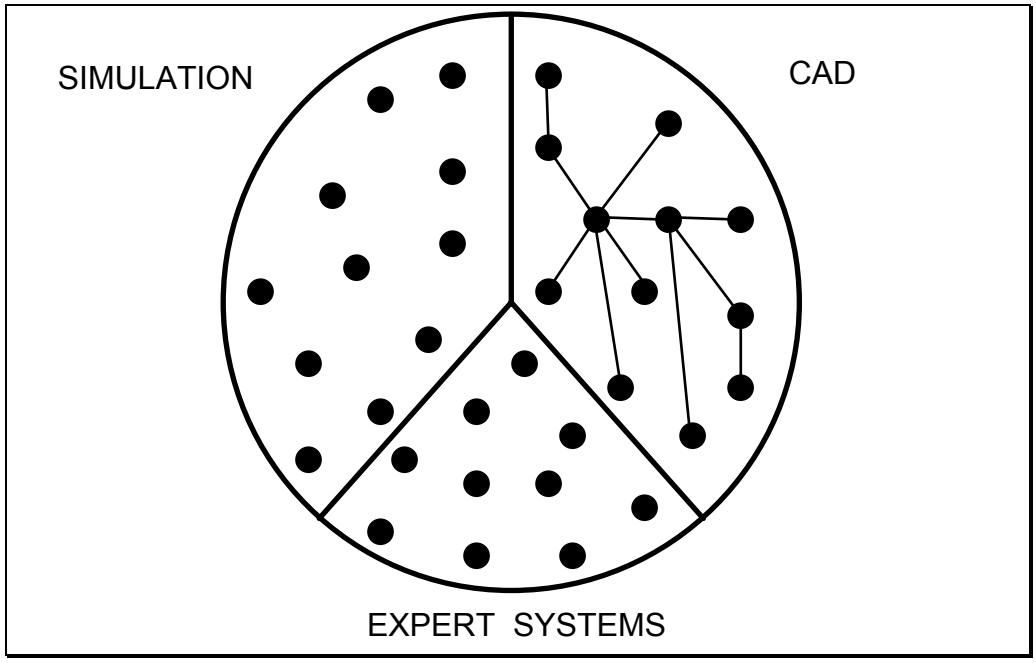


Figure 1. Design tools: The current situation

In the remainder of this paper we discuss the design of a system that will allow a single description of a building design to be used to provide descriptions of the building to several different tools. After discussing the general approach (Section 2), we discuss the framework we have developed for representing building designs (Section 3) and the way in which this system interfaces to the various tools (Section 4). We then discuss the role of a graphical interface in such a system (Section 5) and briefly mention the possibility of running the system in a distributed processing environment and the use of the system as a learning tool.

2. SCENARIO

The problems outlined above suggest that there is a need for a system which 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 their 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 aim to show that such a system can be created.

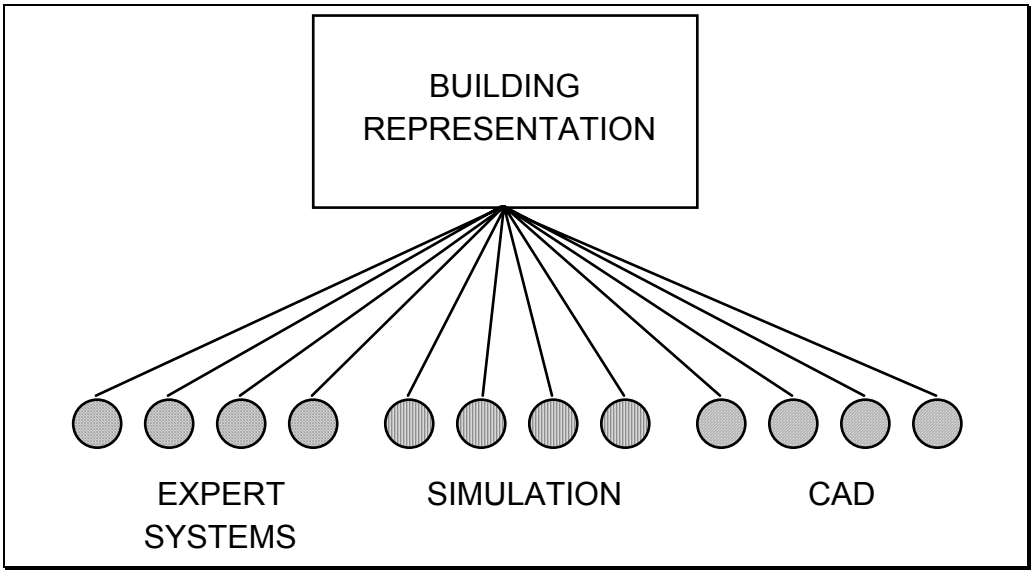


Figure 2. Design tools: The future

A number of projects with similar aims have been initiated recently in Europe, where Finland (Björk 1988, Björk 1989a,b,c,d, Enkovaara 1988 and Hannus 1987), France (Rubinstein 1987, CSTB 1986, CSTB 1988) and England (Clarke 1989) have major industry and state cooperation on the development of systems for the interchange of information between design tools throughout the architectural design process. These projects are intended to enable all design tools used in the country to communicate with each other, so as to avoid redundancy in the information and make the transfer of information between the disciplines in the area easier and less error prone.

These integration programmes are combined with proposals to create databases of design information and codes which are easily accessible by the whole architectural design community. This development has been aided by the spread of compact disk technology, which allows databases containing vast amounts of information to be made available to practitioners at a reasonable price. These systems are being developed with an emphasis on producing the documentation for a building project, rather than facilitating access to a variety of design tools.

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, which can accommodate all of the information required by the tools we are considering. 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.

3. BUILDING REPRESENTATION

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 cover a range of areas, including structural analysis, lighting design, thermal design and code checking. Specifically:

- Thermal simulation programs that have building description languages of varying degrees of complexity (DOE and SUNCODE).
- A structural design package for PC's (MicroSTRAN 3-D).
- Lighting design packages for PC's (Lumen and Fenestra).
- A New Zealand fire code expert system (Hosking 1987).

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 1987).

The large number of objects that compromise a building and its systems, and the large number of attributes needed to describe some of these objects, made it clear that we were undertaking a large and complex task. To keep the project manageable, we have decided to restrict our attention to the building's structure with structural components, heating, ventilation and air conditioning (HVAC) systems and lighting.

As all these tools are all concerned with some aspect of architectural design, there is a significant amount of overlap in the knowledge they need to describe the basic properties of a building. Different tools, however, may view a building from different perspectives. For example, some systems view a building as a set of zones with certain dimensions, which may or may not have surrounding walls (e.g. DOE or SUNCODE), while other systems take a more geometrical approach and view a building as a set of points, with different points joined together to form a surface or volume (e.g. ESP). These systems represent the same types of buildings, but are looking at the building from two different views.

The representation we were looking for had 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.

Non-redundant: To minimise redundancy in the representation, any data captured in the description should only exist in one place.

3.1 Abstraction Hierarchy

From the description languages for the tools we studied, 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, subsystems etc. 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 provide for defaults.

We have defined an abstraction hierarchy which breaks the description of a building into five levels: building, system, subsystem, object and part (see Figure 3).

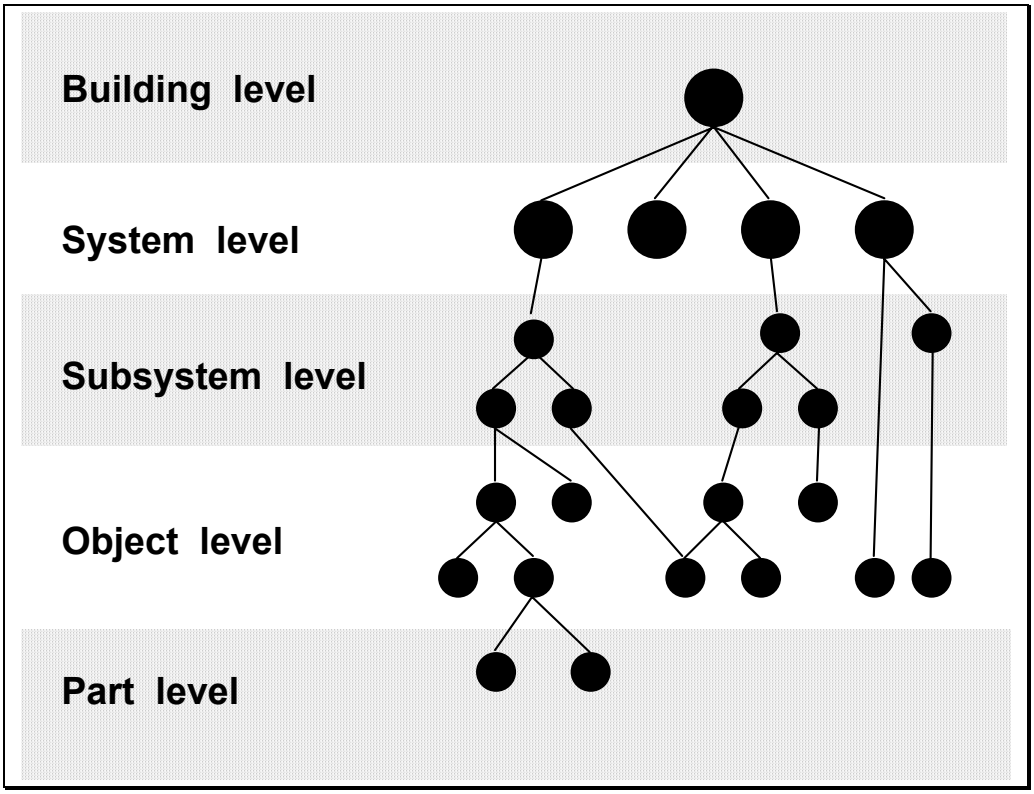


Figure 3. Abstraction hierarchy

Building level: This is the top level of the structure. There is only ever 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 a building up into its main functionally distinct systems. For our purposes these are: the spaces in a building, the structures, HVAC system, lighting system, lifts, power, communications, etc.

Subsystem level: This level is used to break a system into its major functional components. We have used this level mainly in our description of the HVAC system which 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 at this level 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 structure is very similar to the one proposed in the RATAS project (Björk 1989a,c and Enkovaara 1988), although they group different sets of objects into their hierarchies due to the nature of their system.

3.2 Frame representation

The abstraction hierarchy described above could be implemented in a number of ways, for example using a relational database or an object oriented programming language such as Smalltalk. 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 4).

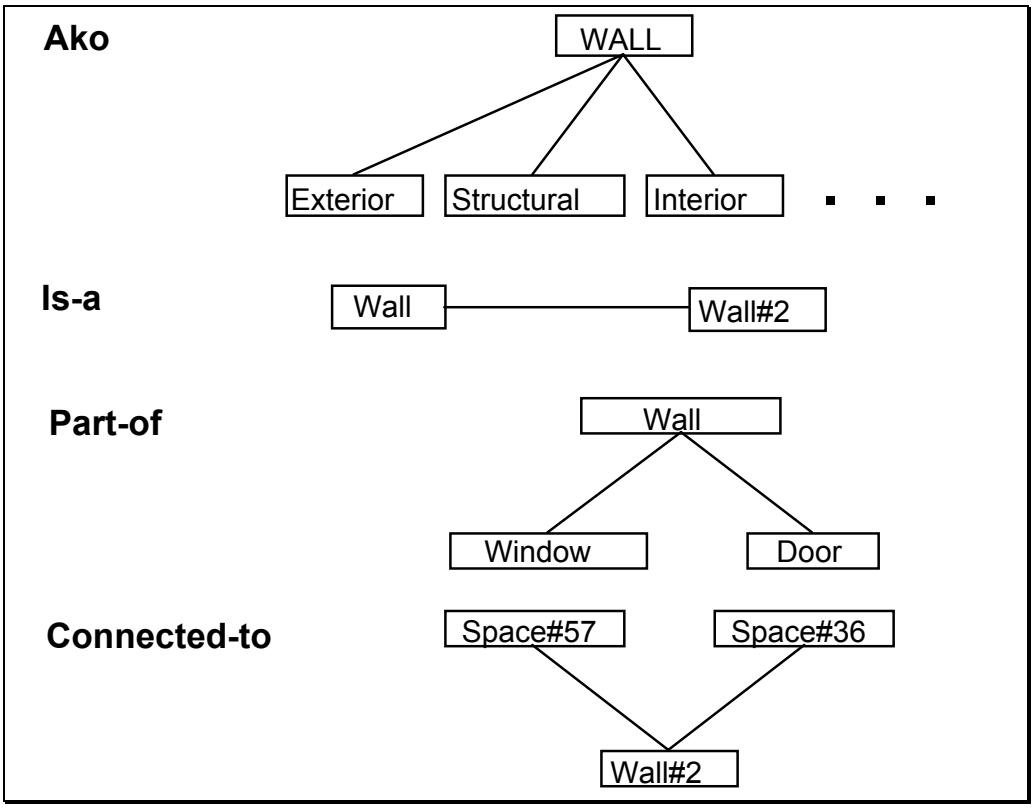


Figure 4. Frame relationships

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.

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 the Appendix.

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.

3.3 Attributes

For each object in the hierarchy described above, we need to specify the attributes required to describe that object. The set of attributes must be sufficient to describe the object in all tools interfaced to the system.

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 to use the most detailed attributes for a component and provide methods for calculating general attributes where appropriate. This mechanism is also used for unit conversion, since the same attribute may be represented using different units in different tools, depending on their country of origin.

In designing our building representation, we have carefully considered the structures and attributes used in the tools listed above, and structured the *is-a* hierarchy to maximise the use of inherited attributes. The attributes describing the input and output for these tools number several thousand. Through the use of the abstraction hierarchy and inherited attributes, we have developed a representation with just over one thousand attributes.

4. INTERFACING MECHANISMS

Given the common representation for building designs discussed above, we need to be able to construct an interface between this and each of the tools we wish to use. For each tool, we need to be able to construct 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. It is important that interfaces for new tools can be added easily, since this will ultimately determine how successful our system is in practice.

4.1 Previous work

The problem of interfacing between tools has been studied by Lamb (1987). Lamb's solution to this problem is based on an "Interface Description Language" (IDL) and consists of three parts:

(i) A notation to describe programs in a system and the data structures they communicate with.

(ii) A program organisation imposed on program components of the system for permitting them to communicate.

(iii) A translator that analyses the description of the system, written in IDL, and generates fragments that plug into the program organisation.

IDL represents the data in a system as typed, attributed, directed graphs. For

each tool in an IDL system, IDL requires a reader to map a common exchange representation to the tool's internal representation, and a writer to map the internal representation to a common exchange representation. The IDL system has been used mainly to construct compiler environments. In these systems there is a separate IDL program to move data between any two tools that need to communicate with each other. While IDL does not deal with source files, which are used in the tools for this project, the ideas are still applicable.

There have been a number of implementations of packages using IDL which integrate specific tools in the manner described above (Goos 1983 and Intermetrics 1986). While these systems only facilitate the transfer of data between tools, there are other systems which increase the functionality of packages via integration of tools. For example, interfacing an expert system to a simulation program (Levary 1988) or to a database system (Alzobaidie 1987).

The IDL approach provides some interesting ideas about how to tackle the problem of interfacing tools, but is limited by the fact that it only deals with internal representations of data to be transferred between tools. It does not deal with data files or requesting data from the user.

4.2 Our interfacing mechanism

To construct the data file required to drive a particular tool, our 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 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 which 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 file and a 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.

Our initial approach to implementing these interface mappings will be to write specific programs to perform the mappings required for each tool that 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 analysing 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.

5. GRAPHICAL 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 modelled and to enter information that has not previously been provided.

Most commercial CAD systems, especially those in use on PC's (e.g. AutoCAD, VersaCAD), require the designer to draw a building with low level graphical objects such as lines, points and 3-D faces. There seems to be a move away from this type of system to ones which operate with higher level objects such as walls, windows and doors that constitute a building (e.g. Sonata). The development of these higher level CAD systems seems to be based on the same philosophy as the design of high level programming languages, namely that the functionality associated with each command in these programs is much greater than in the lower level programs. This is achieved in two ways: firstly, by letting designers create a building design using the objects that actually constitute a building and, secondly, by giving the system some knowledge about the types of objects it is manipulating and the ways these objects can fit together.

For a system to draw at an object level there has to be a break from the traditional general purpose CAD systems. These systems are often very large and complicated, due to the need to service many fields of interest. The newer systems are designed for a specific area of CAD and provide only the tools needed for that area of design, creating a faster and leaner system. This seems to have the effect of increasing the usability of the systems, making it easier

for a practitioner in that field to understand the system and to enter information into it.

We intend to follow the work in this area (Kharrufa 1988, Bijl 1985, Szalapaj 1984, Balachandra 1987 and 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 to give the user a feel for the locality and use of the object.

6. 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.

6.1 Distributed processing

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

- an FEM simulation 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.

6.2 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 generalise 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 you make to help lower the heating kW while still maintaining the same comfort level in the building.

7. 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 to move 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, that will be independent of the method used by the tools the data will be used in. For tools that do not offer a method to visualise the building being modelled, this system will offer a means of visualising the building. In time it will act as an intelligent assistant to the user, by being able to suggest changes to a design to meet a certain design criterion.

REFERENCES

Alzobaidie, A. and Grimson, J. B. (1987). Expert systems and database systems: how can they serve each other?, *Expert Systems*, 4(1), February, pp. 30-37.

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

Balachandran, M. and Gero, J. S. (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, Bo-Christer (1988). *RATAS - an Object-Oriented Conceptual Building Model*, Technical Research Centre of Finland, 16p.

Björk, Bo-Christer, Penttilä, H., Saarinen, H., Moisio, J., Finne, C. (1989a). A Prototype Building Product Model Using a Relational Database, *ARECDAO 89 Symposium*, Barcelona, Spain, 13-14 April, 17p.

Björk, Bo-Christer (1989b). Product Models of Buildings and Their Relevance to Building Simulation, *Building Simulation 89 Conference*, Vancouver, Canada, 23-24 June, 6p.

Björk, Bo-Christer and Penttilä, Hannu (1989c). 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, Bo-Christer (1989d). Basic structure of a proposed building product model, *Computer-Aided Design*, 21(2), March, pp. 71-78.

Clarke, J. A., Rutherford, J. H. and MacRandal, D. (1989). *An Intelligent Front-End for Computer-Aided Building Design*, University of Strathclyde, Scotland, 7p.

CSTB (1986). *Application de Langages Orientes Objets à une Maquette de Batiment en Thermique et Calcul de Structure*, Centre Scientifique et Technique du Batiment CSTB, Convention n85.61.521 avec la Direction de la Construction.

CSTB (1988). *FARTEC Faciliter l'Accès aux Règles TEChniques*, Centre Scientifique et Technique du Batiment CSTB, Comité d'Orientation, 1er Décembre, 38p.

DOE-2.1C (1981). Los Almos National Laboratory, Los Almos, NM.

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

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.

Goos, G., Wulf, W. A., Evans, A. and Butler, K. J. (1983). DIANA: An intermediate language for Ada, *Lecture Notes in Computer Science 161*, Springer Verlag, New York

Hannus, Matti (1987). Object Oriented Modelling In AEC Applications, *Proc. NBS/DATA Nordic Seminar "Neste generations datasystem i byggebransjen"*, 24 September, pp. 100-115.

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.

IGES (1986). *Initial Graphics Exchange Specification 3.0*, National Bureau of Standards, Gaithersburg, MD 20899, USA.

Intermetrics, Inc. (1986). *Compiler retargeting tools user's guide* , IR-MA-623, Intermetrics, 733 Concord Ave, Cambridge, Mass. 02138, May.

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

Lamb, David A. (1987). IDL: Sharing Intermediate Representations, *ACM Transactions on Programming Languages and Systems*, 9(3), July, pp. 297-318.

Levary, Reuven R. and Lin, Chi Y. (1988). Hybrid Expert Simulation System (HESS), *Expert Systems*, 5(2), May, pp. 120-129.

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.

Rubinstein, Michel (1987). Advanced Computer Systems in the French Building Industry, *NBS DATA*, 24, September, pp. 12-38.

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

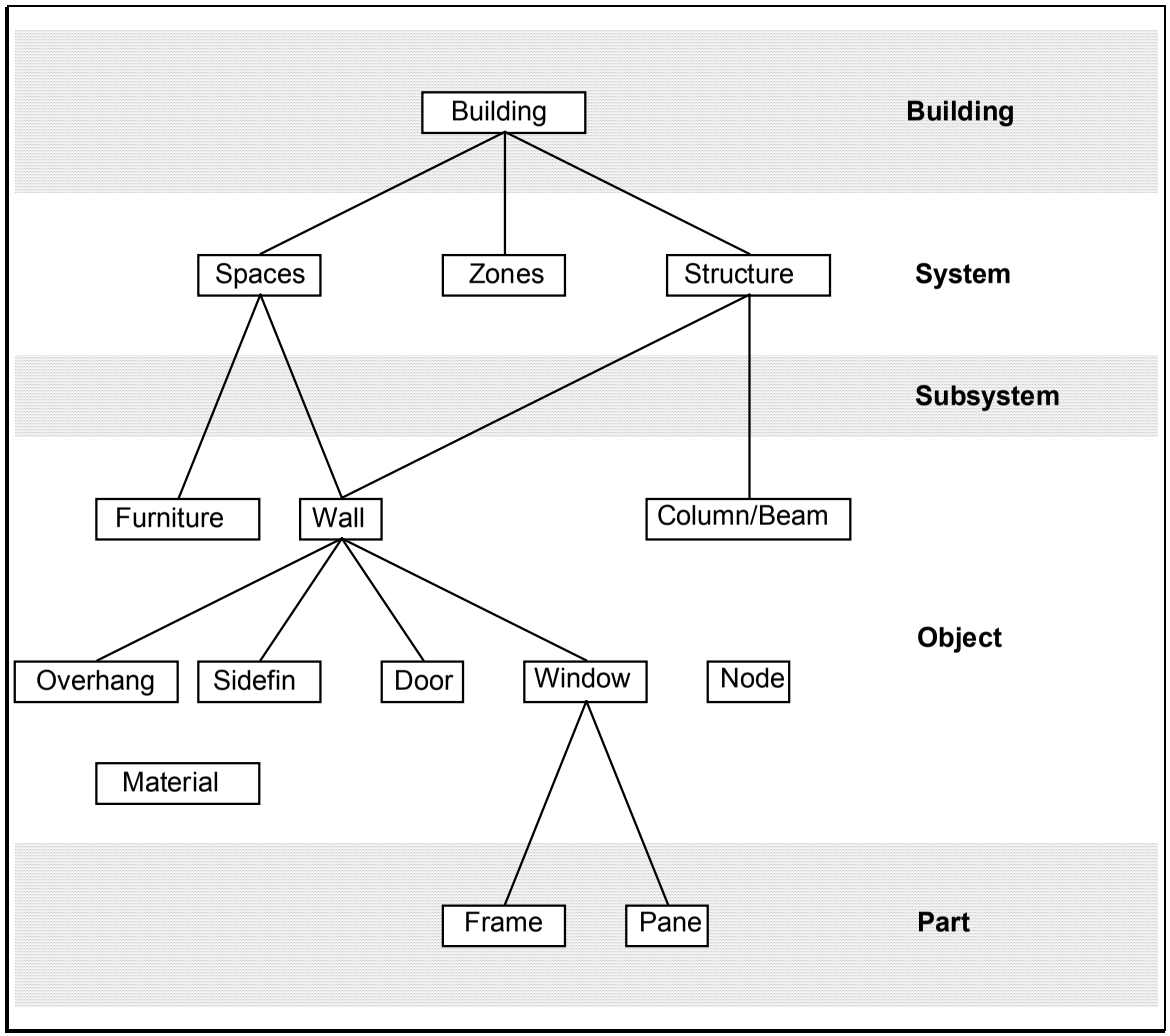
SUNCODE (1981). Ecotope Group, 2812 E. Madison, Seattle, WA 98112.

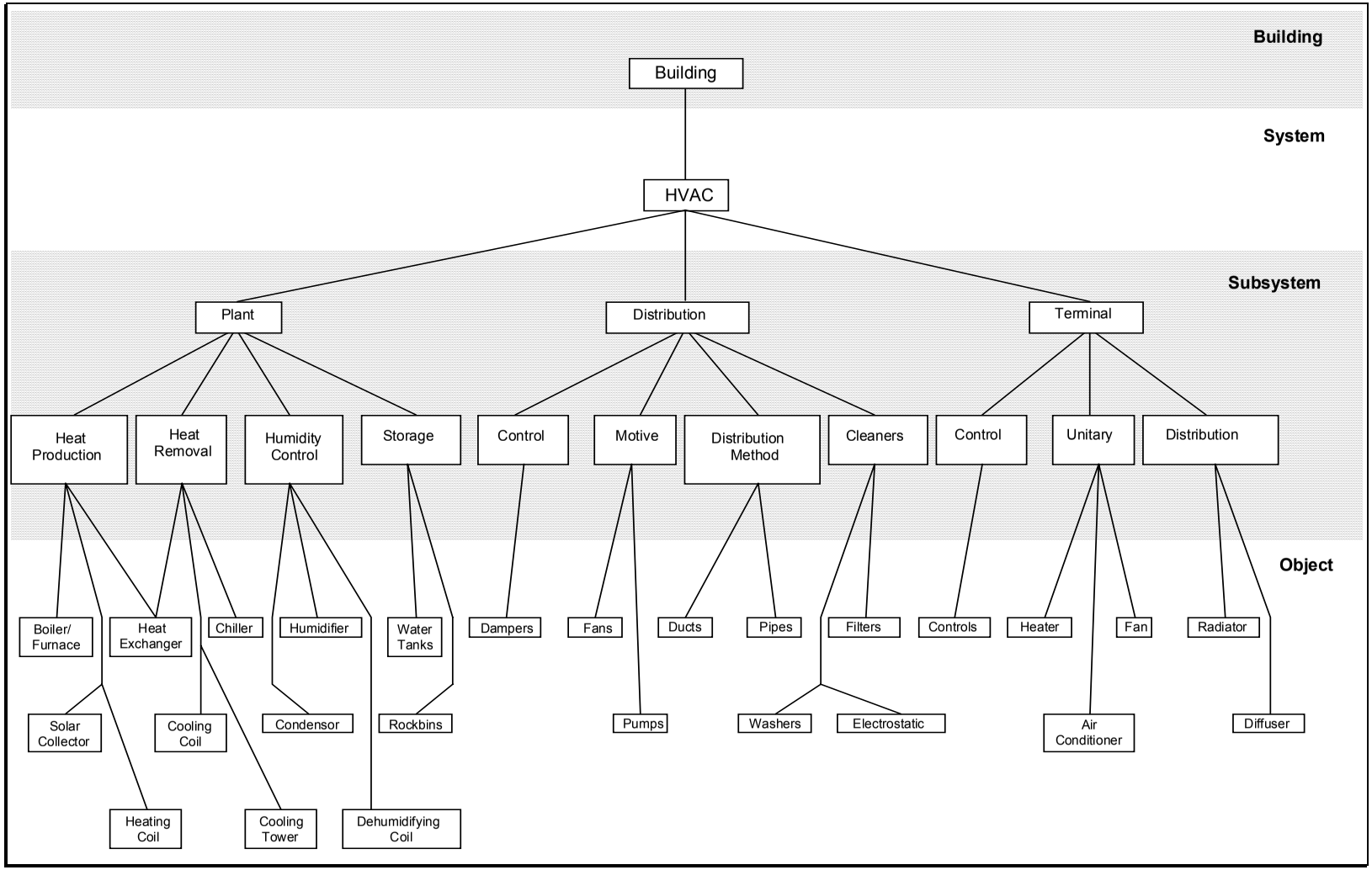
Szalapaj, P. J. and Bijl, A. (1984). Knowing Where to Draw the Line, *Knowledge Engineering in Computer-Aided Design*, September, pp. 147-169.

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

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

APPENDIX. Sample portions of the part-of relation


Building
Building
Spaces
Zones
Structure
System
Subsystem
Furniture
Wall
Column/Beam
Object
Overhang
Sidefin
Door
Window
Node
Material
Frame
Pane
Part


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