Integrating Design Tools: An Object-Oriented Approach

by
Robert Amor
Lindsay Groves
Mike Donn

Victoria University of Wellington
PO Box 600
Wellington
New Zealand

Abstract

The problems of integrating design tools from the many disciplines in architectural design are well known. Many approaches to solving this problem have been proposed, and much research has gone into attempts at solutions. In this paper we present the approach used at Victoria University in the ICAtect project.

ICAtect integrates several design tools commonly used in architectural design, through the use of an object-oriented model of a building based upon the frames and worlds systems developed from research into Artificial Intelligence. The building model developed is structured so as to capture all the information required by a selection of design tools, as well as to provide a level of intelligence capable of ensuring only valid designs are described in the model.

There were many problems encountered when integrating the various design tools with the building model in ICAtect. The solutions to these problems are discussed, along with the value of the object-oriented approach in overcoming these problems.

1. 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 tools for design analysis and building performance prediction as possible. Numerous CAD systems, simulation programs, and, more recently, expert systems are being developed to assist the team with this analysis.

Each of these tools requires a detailed description of the building or structure being modelled. Preparing the data describing a building for use in a design analysis tool can take several days. Even if the design tool is a computer program, constructing the description of the building, in a language understood by the design tool, takes a similar length of time. When the same building is to be analysed by several different tools, similar base data has to be prepared for each tool, entailing considerable duplication of effort and many opportunities for error. If a building is to be described for several design tools, updating the design or testing the performance changes resulting from design modifications requires changes to be made in all the descriptions of the building for all the design tools. This duplication of data leads to possible inconsistencies of building versions between tools, which increases the likelihood of erroneous design decisions being made on the basis of design tool advice.

Each design tool is generally designed to perform one particular type of analysis or simulation. Few are designed with a view to 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. The exception to this is in CAD systems, where it has been realised that some sort of exchange protocol is needed to allow data to be moved from one CAD system to another. This has led to several different protocols being defined by the major vendors. Only recently has an ISO standard been defined for the exchange of data between CAD systems (IGES 1986).

From the problems outlined above it was concluded that there was a need for a system that would allow a user to automatically transfer common building data between different design tools, i.e. translate between different building description languages. This system should also enable the user to create building descriptions for different design tools without having to learn the command syntax and idiosyncrasies of each tool. Writing translators to transfer data between all design tools would require approximately n^2 translators (actually $n(n-1)$), where n is the number of design tools. The huge number of translators required as the number of design tools increases can be reduced by creating a common building model (CBM) to hold all the information on a building. With this common building model as the mid-point between translators, the number of translators required reduces to $2n$.

This is the approach that has been taken with ICAtect (see Figure 1 for the structure of ICAtect). To hold the information on the building being designed there is a central representation of the building called the common building model. This common building model represents all the information that may be required by any of the design tools integrated into ICAtect. To move information between the various design tools and the common building model a two stage mapping process has been defined. At the centre of this mapping, for each design tool, is a model of the data described using the design tool’s building description language. This intermediate representation models all the objects and attributes found in the tool’s data files including any defaults of the attributes, allowable ranges, methods for calculation, etc. This greatly simplifies the mapping process by making one stage a direct translation between the tool’s data file and the intermediate representation, while the second stage becomes a purely manipulative process between the two different representations.

To allow the designer to use the system, a separate user interface is provided. It is designed to ease data entry to the common building model, and to present the output of the analytical tools.

2. The benefits of integration

A system comprised of multiple integrated design tools allows the possibility that the user interface functions as a standard interface to all of the tools in the system. This has the advantage that users need to learn only one method of entering information or conversing with the system, instead of the many complicated languages which need to be mastered to enter information to a range of different design tools, which is the case today. The advantage to the users is that a single interface to many different performance analysis tools makes it easier to use these tools in design. A single consistent language and thought process is used to access all the tools, reducing the distraction from the process of designing a building. This system also allows the designer to devote more time to the actual design as it removes the need to re-enter information about the building each time a new design tool is used to analyse its performance.

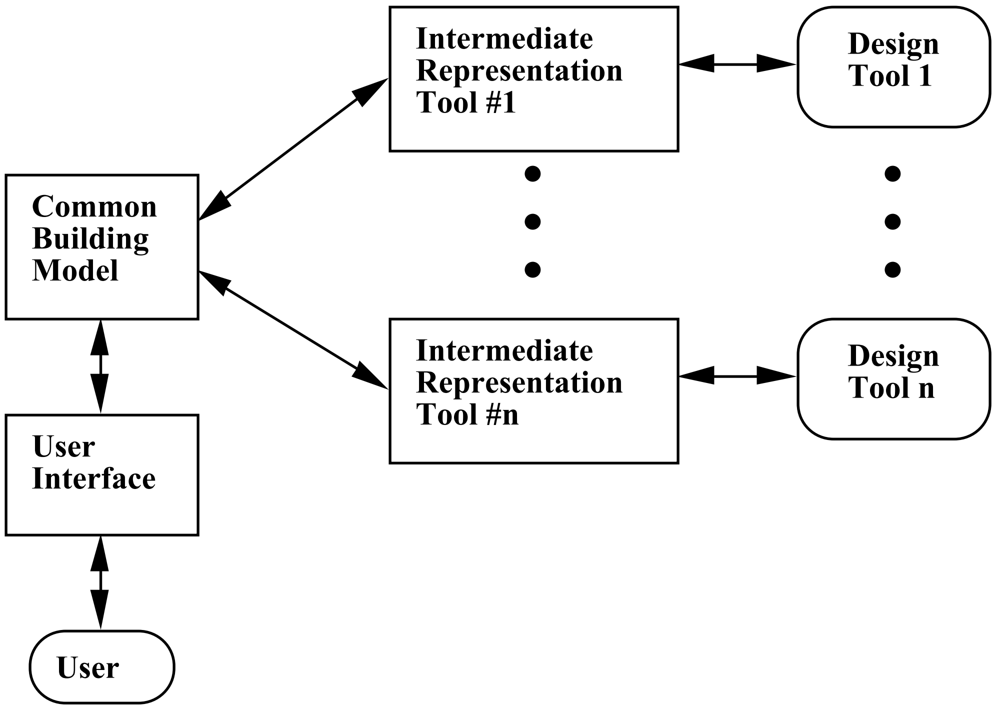
Intermediate Representation Tool #1
Design Tool 1
Common Building Model
Intermediate Representation Tool #n
Design Tool n
User Interface
User

Figure 1. ICAtect's structure

The integrated approach offers the architect a way of developing better designs because it offers a means by which the design performance can be checked quickly by several different design analysis tools and expert systems. The use of an object-oriented model of a building allows the data entry for each design tool to be checked as it is entered. This can be used to catch simple placement errors in the design, by specifying requirements of connections between objects, e.g., doors must be placed inside a wall, windows must be placed inside a wall or door, etc. It can also pick up gale force ventilation rates or a grossly over-sized heating plant.

An integrated system helps with the consistency of the building description data. As there is only one central model of the building being designed it is not possible to have inconsistent or outdated models of the building encoded in building descriptions for several design tools. All changes happen to one model, and all analyses are derived from the same source of data.

A well designed integrated system will be an open-ended solution, not locking the user into one or two packages by any particular company. In the ICAtect system it is a relatively easy matter to add another design tool to the system by specifying the mapping between the design tools data file and the common building model in ICAtect, as described later in this paper. This type of solution allows the designer to keep using design tools he or she is familiar with and understands.

3. Problems encountered during integration

There are several major problems that are encountered when considering the integration of design tools into a single system. The quest for solutions to these problems are central to the research undertaken on the ICAtect system, and their solutions define the working system. The problems that were considered can be summarised under three headings:

3.1 The construction of a common building model

This is a problem in two parts, one being what to represent in such a model of a building, the other being how to represent the building model. The solutions selected for each of these parts of the problem greatly influence the final form and performance of the integrated system. For example, the information represented in the model determines what types of design tools can be integrated into the system, and the way the information is structured determines the ease of integrating new design tools into the system. Also the method of representing the information determines how that information can be accessed by the system; the type of query needed, and the speed and efficiency of access are all affected.

The representation method determines the types of information that can be represented in the system. For example, there are many types of information that can be represented in an object-oriented data base that cannot easily be represent in a standard relational data base. These types of information are: the notion of classes and inheritance of information between related classes; the representation of formulae describing how to calculate values for an attribute; the representation of code constraints as found in local and national building codes and standards.

The selection of a representation method can even affect the building design process by driving the way in which the system runs. The goal with ICAtect was to allow designers to be free to work at any level they chose, and not to force them to work in a pre-defined manner.

3.2 Interfacing the common building model to existing design tools

The problem is that all relevant information must be transferred between the design tool’s data file and the common building model, and vice-versa. It is also important that any tool-specific data can be requested from the user before the tool is invoked. The ideal solution to this problem is for a system that could take the description of a design tool’s data files and automatically create the mappings required to transfer data between it and the common building model. This can be difficult as many of the existing design tools were created before much research on parsing had been undertaken. As a result the design tools’ building description languages have been structured in ways that do not allow standard parsing techniques to be used.

3.3 Interfacing with the user

It is widely accepted that most design tools in use today, especially CAD systems, work at too low a level of descriptive power to be truly usable by designers. However, new methods of interaction with the designer are slowly being tested and adopted by some of the so-called 2nd generation CAD systems. The most common trend is to design the CAD system around an object based system (Sonata 1988; Vervoert 1987). This is the approach being used in ICAtect. It is also a major goal to ensure that the user interface will allow designers to work at any level they choose, keeping away from structured design and guided conversations in the system. This extends to allowing users to define multiple views of their building design, and to allow them to easily design and test many alterations to a single building without major effort.

4. The Common Building Model

The design criteria of ICAtect's common building model were:

- that it should be capable of modelling the objects, relationships and information used in any of the design tools examined.
- that it should be easily modifiable to allow the addition of classes of objects without major modifications to its structure.
- that the data captured in the model should only exist in one place, eliminating redundancy in the model.

On top of these considerations of the model structure, there were the considerations of the design process, which can be affected by the method by which this information is represented.

Consideration of the design process placed a number of constraints on the nature of the common building model. The most severe constraint was provided by the goal of allowing 'design freedom': designers must be free to work at any level of detail they choose; they must be free to explore the performance impact of variations in their early concept design; and they must be able to test modifications to this design as a means of developing the best possible design.

To help the decision of which representation to use for the building model, several possible categories of information used to describe a building were examined. These categories were drawn from an examination of the different design tools and from general consideration of what categories could exist (see Amor et al. 1990b). The resulting list of categories provided a way of ranking representation methods by determining which categories of information could be represented. After evaluating the capabilities of the various representations (see Amor et al. 1990b), it was clear that the only two contenders for representing buildings were: object-oriented data base systems and frames systems. The frames approach was chosen, primarily because of the availability of software (the frames system is fairly easily implemented in Prolog) and the freedom for expansion or modification offered by a system implemented on site. It was also noted that a frames system developed on site would be very flexible and easily modifiable to handle extra modelling capabilities as needed. This proved useful when modelling design alternatives (see below).

4.1 Structuring the common building model

Defining the structure of the common building model was potentially a very complicated task. It was greatly simplified by creating a database of the attributes and objects used in many of the design tools that exist at the School of Architecture. This database now contains information on 17 design tools spanning the range of structural, thermal and lighting design, as well as information on weather file formats and CAD exchange representations. It contains over 2,600 entries describing the attributes defining the building model in each of these design tools. The database proved very useful for analysing the attributes required to represent an object in the common building model for many different types of applications.

The major problem was one of defining the objects required in the common building model, and the relationships that could exist between the different objects. From our analysis of the description languages for buildings, and from charts of objects and relationships defined in the design tools, it was apparent that a building should be described in a hierarchical manner. The abstraction hierarchy developed splits objects into five levels (see Figure 2), much like the abstraction hierarchy in RATAS (Björk 1989). The five parts of the abstraction hierarchy are:

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 floor, etc.

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

Sub-system level: As the system level represents very high level concepts this level is used to break a system into its major functionally distinct components. This allows the grouping of objects that have common functions. Thus far, it has been used mainly in the description of the HVAC system, which is broken up into plant, distribution, and terminal sub-systems, and then further divided to represent all the major functions of the components in an HVAC system.

Object level: This level represents each physical object in a building. All major components in a building 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, condensors, and cooling coils are parts of a chiller.

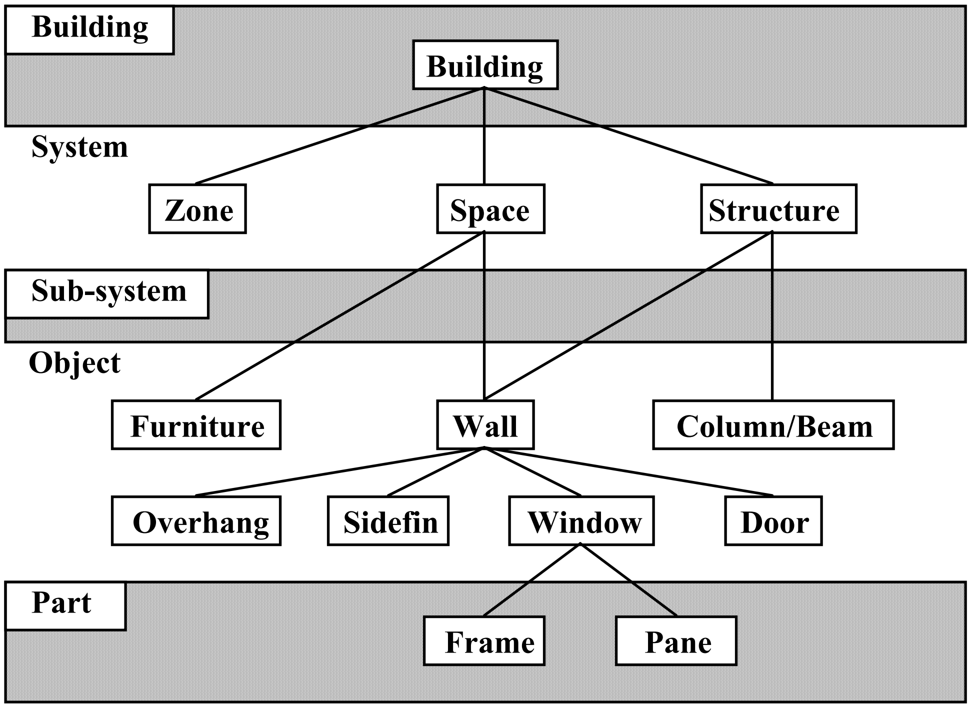

Figure 2. Abstraction hierarchy

4.2 The frames system

Frames systems (Minsky 1975; Winston 1977) have evolved from the field of artificial intelligence as a method of embodying the types of knowledge humans have in their minds, and the methods they use to manipulate it. As this is a very object-oriented concept, the power and representation capabilities of frames systems are very similar to object-oriented DBMS systems. Frames systems can represent the following types of information: structured objects and their attributes; relationships between objects; classes of objects and the inheritance of information between them; formulae and in most cases whole programs inside an object.

The major reason for developing the frames system in-house was that commercial frames systems suffer from restrictions on the types of facets on attributes and relationships between objects that are allowed. These constraints are coded by the developers, who aim their product for a set market and are less concerned with the wider applicability of a more open system. Another major drawback for the purposes of this project was their scarcity in the market, and the price of those that exist. However, as noted above, frames systems are relatively easy to implement in the languages used in artificial intelligence work, and most researchers end up implementing a system that suits their particular needs, as was the case in ICAtect.

In keeping with the generalised concept of frames systems (Winston 1977), ICAtect's frames system is conceptualised in three levels. These are:

• Frame	Holds all attributes needed to define an object.
• Attributes	Attributes of an object (and their values).
• Facet	Properties of an attribute (and their values).

The frames system has been split into two semantically different segments: classes and instances (see Figure 3 for the Prolog representation of classes and instances). Classes are further subdivided into two conceptually different segments: one for the common building model; and one for the design tools. Due to the nature of the information held in a description of a design tool, some of the relationships and facets used are different from those in the common building model. This information is mainly concerned with the format of attributes in a design tool's data file, and constraints on the number of allowable objects of any one type.

Class structure

```
class(class_name, data_model, tool_type).
slot(slot_name, class_name, relationship, facets, data_model,
tool_type).
```

Instance structure

```
frame(frame_name, data_model, instance_number).
slot(slot_name, frame_name, relationship, facets).
```

Figure 3. Class and instance structure

Classes are used to describe the structure of the model or design tool. They include all objects that can appear in the model or design tool, all the attributes that can be associated with the objects, and facets of the attributes describing defaults, constraints, units, etc. Instances hold specific information on an individual building, they hold the actual values that describe the objects in a building. The instances make use of the class definitions to determine what attributes can be attached to what objects, whether a value is valid and what objects it can be attached to.

As ICAtect has to manipulate the information on one or more buildings in the common building model and several different design tools at the same time, it was important that the frames system could handle many representations simultaneously. As the frames system is implemented in Prolog, which keys everything to a single global database, it was necessary to make explicit the tool to which a piece of information belongs (*data_model* in Figure 3). To this extent the frames system keys all the information in Prolog by the design tool name or the name 'Common Building Model'. To cope with the possibility of many buildings being handled at any one time all the information is also keyed by a unique number denoting the building being modelled (*instance_number* in Figure 3). The design tools offer the added complication that many of them can accept the description of a building in different formats, in most cases using either imperial or metric units. To handle this, all of the class information about a design tool is keyed with a number identifying which format of the building model is being described (*tool_type* in Figure 3).

It may be noticed in Figure 3 that each frame is described in two stages: first, the class or frame predicate detailing the object name and the design tool it belongs to; and second, the slot predicate describing the properties of a single attribute for the particular design tool. This two part structure was derived from an earlier attempt to describe a frame as one predicate, with all attribute descriptions in a large list inside the frame predicate (much like the facets are in the slot predicate). The single predicate frames system created very large data structures which involved a large amount of searching and list manipulation to extract the required information about an attribute. Breaking the frame into two predicates makes Prolog do all the work and speeds up searches considerably. As Prolog indexes each predicate in its data base, searching for a named predicate is very fast and requires no work on the programmer's part. This also reduced the amount of code for the frames system to about half of what was required previously.

As can be seen in Figure 3, all attributes are classified by a relationship type. This relationship type distinguishes the function of the attribute in the description of the object to which it belongs. This allows the selection of attributes of an object which perform a certain task, for example to find all attributes which describe the connection to another object. This classification also gives the frames system some assistance in deciding how to handle the values for the attribute. There are nine types of relationships defined in ICAtect. They are fully described in Amor et al. (1990a).

The *facet* section of the attribute's definition is a list defining all the facets applicable to the attribute. There are nine types of facets defined in ICAtect, they were determined by what facets would be needed to describe the properties of attributes in such a system. These properties can be used by the frames system to determine default values, the type of data expected, restrictions on the attribute, where the value was derived, a description of the function of the attribute, etc. The full list of facets is described in detail in Amor et al. (1990b).

4.3 Augmenting frames: Worlds

The frames system is quite capable of modelling buildings and their components. However, as a standard frames system is not capable of representing design alternatives, which is one of the major considerations for ICAtect's building representation, the frames system has been augmented with a worlds system that allows this. This system is based upon the concept of worlds as described in KEE (1988). Worlds in KEE are used to explore alternative choices in an analysis and to maintain derived results. KEE incorporates a truth maintenance system for tracking beliefs and their presence in various worlds, a facility not deemed necessary for ICAtect. The addition of worlds to the frames system demonstrates the flexibility of the frames system in allowing modifications to suit a particular purpose.

The worlds system allows the designer to take the model of a building constructed in the frames system and examine various modifications to this base building without destroying the base building description, and without having to duplicate any of the data in the base building. This concept is very similar to the layers systems in CAD packages, where a layer is like a sheet of clear plastic on which the designer draws a certain object. If the sheet is laid over the rest of the CAD drawing the object is in the building, and the designer has a different view of the building without having duplicated the rest of the CAD drawing. The worlds system is different from the layers system in that all modifications, or worlds, are recorded in a hierarchical tree. This structure allows the designer to explore many different branches of thought, and with the hierarchical tree recording all the detours, provides a method of changing quickly from one view to another.

The worlds system works by taking the concept of inheritance one step further than the inheritance of attributes between objects, to the inheritance of whole sets of objects, in this case comprising a whole building. The worlds system is based upon a hierarchical tree of worlds, with each world inheriting all the modifications made further up the tree right back to the root world which is the original description of the building being examined.

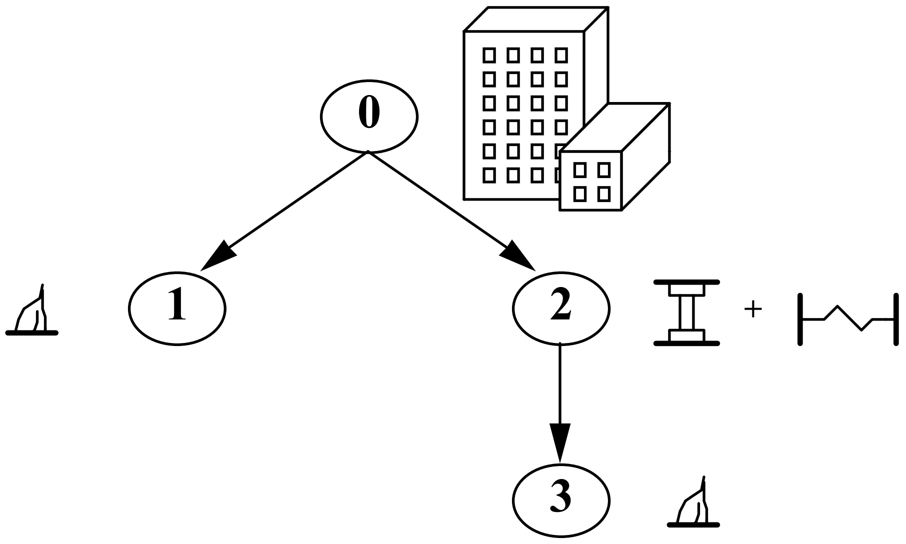

Figure 4. Worlds

The example in Figure 4 shows four worlds. The root world (world 0), describes the whole building which, for the purposes of this example, has a coal fired boiler and single glazed windows. After analysing the base building, the designer decides to examine the effects of two changes. First, the coal boiler is replaced with a gas fired boiler (world 1). Second, all exterior windows are changed to double glazing and the coal fired boiler is replaced with an electric boiler (world 2). After further analysis of these two modifications the designer decides to test a gas fired boiler instead of the electric boiler in world 2 (world 3). Worlds 1 through 3 demonstrate the power of the worlds system. With very little effort the designer is able to construct a new building, derived from the existing design, without duplicating any objects and adding only the changed systems. The designer now has three variations of the base building, all with the detail of the base building without having to duplicate any of the base building data. The designer can move between the different variations just by changing the world number he or she is working on.

The actual implementation of the worlds system required remarkably little effort. Representing an attribute’s value for the different worlds required only a small modification to the value facet of the frames structure. The notation used for labelling worlds is that the root world (base building) is number zero, and all other worlds are numbered sequentially as they are created. Keeping track of which world a value relates to is achieved by changing the value of the value facet to a list of values, each denoted by a world number. In this way any number of worlds can be accommodated.

```
e.g., in frames   facet(value, 12.5)
      with worlds facet(value, [world(0, 12.5), world(1, 8.7)])
```

The frames system was modified to handle worlds by the addition of a *world_number* parameter to all operators which deal with instance frames. This parameter is passed through to the low level operators for extracting, deleting or adding a value to a facet. At this level the operators had to be re-written separately to handle a value facet, which is indexed by world numbers.

The hierarchical tree of worlds is represented in a simple structure, representing the world number and its parent world for each building model. Figure 5 shows the world hierarchical tree for Figure 4. Using this structure when looking for a value for an attribute, the frames system will look for a value for the current world, for example world 3. If there is no value for that world it will look up the hierarchy, in this case world 2. If there is no value for that world it will look at the next world up in the hierarchy, and so on, until it reaches the root world. If there is still no value, or possibility of a default value, in the root world then it will fail.

```
world_tree(1, 0, “COMMON BUILDING MODEL”, 1, 1).
world_tree(2, 0, “COMMON BUILDING MODEL”, 1, 1).
world_tree(3, 2, “COMMON BUILDING MODEL”, 1, 1).
```

Figure 5. Worlds hierarchy representation

There are only two inconveniences with this worlds representation, both involved with removing objects. The first occurs when it is necessary to remove a whole world from the world hierarchy. While it easy to restructure the hierarchical tree of worlds to accommodate this removal, if there are any worlds inheriting data from the world being removed it is necessary to find every value asserted for the world being removed and make it a value for the world or worlds under the one being removed, while checking that the value of the world being removed has not been superseded by a value in one of the lower worlds. This change would have to be propagated down through every world right down the hierarchical tree from the one being removed. For this reason worlds are not allowed to be removed in ICAtect. The second occurs when the designer wants to modify the base building by removing an object. The worlds notation has no mechanism for signifying the removal of an object. In this case the value of the parts relationship, which specifies the objects which exist as part of a higher level object, must be duplicated for the current world but omitting the reference to the object to be removed.

4.4 Summary of model

The final model developed in ICAtect contains 93 objects, enabling representation of information for a building and its structure, thermal properties and lighting systems as required by the design tools at the School of Architecture.

The use of a frames system augmented with the worlds concept provided a powerful yet flexible method to represent the information required by a wide range of design tools. The use of this system to represent the building model in each of the design tools linked to ICAtect has allowed much information about the design tools to be encapsulated inside the ICAtect system. This information proves invaluable when integrating the design tools as described below.

Structuring the building model through the use of the abstraction hierarchy has given us a system which is very modular, and proves to be easily extendible for new application areas as they need to be added to ICAtect.

5. Integrating design tools

Having developed the common building model to capture building designs, it was necessary to develop an interface between this and each of the tools used. For each tool, ICAtect needs to be able to generate the input to the tool in order to run it. ICAtect also needs 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, ICAtect needs to be able to read these and construct the relevant parts of the building description from them.

To construct the data file required to drive a particular tool, ICAtect has to be able to extract the necessary information for a building from the common building model and format it in the appropriate way. It is also necessary for the user to be able to add other parameters required by the tool, such as the length of time over which to run the 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, a parser is needed that understands the structure of the file being read. ICAtect then needs to be able to display the results in an appropriate form and make suitable entries in the knowledge base.

The mapping process between the description of a building in the knowledge base and a data file produced by, or prepared as input for, a specific tool, is performed in two stages. The link between the two stages is the frames representation of the information in the tool's data file, giving the structure in Figure 6.

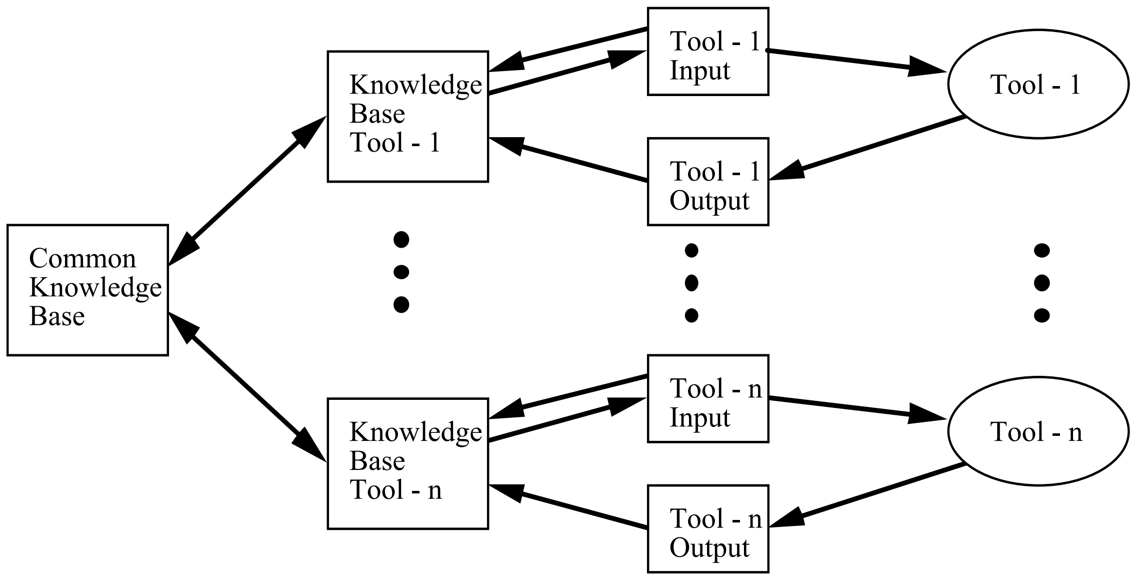

Figure 6. Detailed structure of ICAtect system

The advantage of splitting the mapping into two stages in this way is that the distinct types of mapping are separated. The mapping between the knowledge base and the frames 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 frames representation and the data file for a particular tool is only concerned with the information needed by the tool and not with any other aspects of the complete knowledge base.

Splitting the mapping into two stages with an intermediate knowledge base for each design tool defines the attributes needed by a specific tool. This is of enormous benefit when the system has to determine what extra information is required by the design tool to be used. From the intermediate representation, it is possible to ascertain what attributes are needed by the design tool, where they come from in the common building model, and whether they are actually present in the common building model or have to be requested from the user.

The initial approach to implementing these interface mappings has been to write specific programs to perform the mappings required for each tool used. These programs, like the frames system described earlier, are written in Prolog. With example programs which perform a mapping between a certain type of data file and the common building model, it should be easy to incorporate tools of the same type. Ultimately, these interfaces could be generated automatically from descriptions of the input and output formats and the knowledge base.

5.1 Mapping between the CBM and an intermediate representation

Mapping between the CBM and the intermediate representation for a particular tool involves the manipulation of structures of objects and the combining or splitting of attributes. From the work to date this appears to be the hardest part of the mapping process.

Attributes needed by a tool but not in the common building model are asked for at this stage of the mapping process. These values are only required when moving data from the CBM to a tool’s intermediate representation, as the tool must have all the data required to perform its analysis. These values are not requested when moving data to the CBM as the CBM can hold much data which relates to different fields of design, many of which will not be needed by the designer. The strategy is to only ask for information from the user when it is absolutely necessary.

When information is mapped between the CBM and an intermediate representation it is necessary to keep track (an audit trail) of what model in the CBM is mapped to what intermediate representation, and vice-versa. This gives the designer some information on where the information is coming from, and allows results from a particular analysis to be incorporated back into the original model.

There are several different types of translations that can occur when mapping objects and attributes. These are: a one to one mapping; many to one mapping; and a one to many mapping of both attributes and objects. Mixtures of these types of mapping can prove difficult to handle in a consistent notation, e.g., when several objects have some attributes which map into the same object in the new representation. At present a mapping between two objects is represented by a set of mappings between attributes. A mapping between attributes can be a straight one to one mapping, or it can map a set of attributes to a set of attributes by a user defined procedure.

The generalised form of the ‘map’ predicate is shown in Figure 7. There is a ‘map’ predicate defined for each set of attributes that has to be transfered between two objects in different representations. This predicate defines the mapping between one object in a design tool and one object in the CBM which is of a certain type, defined by *Mapping_Type*. This can either define that a new object is created, or the attributes are added to an existing object, or that the mapping denotes an object in a design tool which will be incorporated into an object in the CBM through the reference of another object in a design tool.

```
map(Tool_Object, CBM_Object, Mapping_Type, [
        ...
        All mappings between these objects
        ...
], Data_Model, Tool_Type).
```

Figure 7. General form of the 'map' predicate

To incorporate a mapping between the CBM and a tool's representation requires a set of mappings to be defined for each object in the tool's model. Creating these mappings consists of analysing each object in a tool's model and deciding where each attribute's data belongs in the CBM. A mapping procedure must be selected for each attribute to be mapped between representations. If it is a straight-forward mapping then it is likely that there already exists a procedure which will handle what is required. However, for mappings which require tool-specific knowledge, it will be necessary to hand-code mapping procedures to perform the task required.

5.2 Mapping between a tool's intermediate representation and a tool's data file

Mapping between the intermediate representation and a tool's data file involves parsing and pretty-printing the data files used by the various tools. Much of the parsing work can be implemented with the standard parsing techniques used in compilers. It was hoped that it would be possible to develop a standard method of parsing these data files into their intermediate representation. However, due to the range of data file types and their complexity, this does not seem feasible. The best approach seems to be to develop standard methods of parsing particular types of files, and provide as many services as possible to facilitate the addition of new tools into the system. To this extent a range of parsers and pretty-printers have been developed that can handle the majority of data file types that exist.

6. Current work

The discussion above describes ICAtect as at the time of writing. As was stated at the start of this paper, the development of ICAtect is part of an on-going programme of research. Further work is currently being done in the following areas:

6.1 Object-Oriented user interface

This is the final component required to make ICAtect truly usable. Work is underway investigating the addition of an object-oriented user interface based upon the objects defined in the common building model. This interface will allow users to perform the complete design of a building through ICAtect, and to analyse the results in a friendlier environment than the current text-based system offers.

6.2 Extending the design tools integrated

The number and types of design tools incorporated into ICAtect is being expanded. This will, with the graphical user interface, enable an evaluation of the usability of this type of system in a real working environment. It is envisaged that there will be five design tools from the major areas of structural, thermal and lighting design integrated into ICAtect before the system is evaluated.

7. Conclusion

The prototype ICAtect system developed provides proof that the integrated design tool environment that was visualised can actually be constructed. It shows that it is possible to develop a model of a building capable of holding the information required by a variety of design tools, and demonstrates that by using an object-oriented common building model it is possible to construct a system that can move information between several disparate design tools. The object-oriented intermediate representation of a tool's data file greatly simplifies the mapping between tools, and also simplifies the addition of new tools.

The ICAtect system demonstrates that the use of an object-oriented representation, that of a frames system augmented with the worlds concept, provides a powerful yet flexible representation method for use in representing knowledge about a building. The building model constructed has proved capable of holding the majority of information needed by a range of design tools from disparate areas of design. The worlds system developed allows the designer to examine many different options during the design of a building, with an ease that is not possible with the state of the art design tools.

The technique of providing example methods for interfacing to the ICAtect system seems the best way to approach the problem of interfacing to design tool’s data files, given that a generalised interfacing system seems to be out of contention. The methods examined to date, cover all the data file types that exist at the School of Architecture. This means that the addition of a new tool only requires modifying the method that matches the new data file type, not writing a parser from scratch.

References

Amor, R., Groves, L. and Donn, M. (1990a). Integrating Design Tools for Building Design, *ASHRAE Transactions*, Volume **96**, Part **2**, pp. 501-507.

Amor, R., Groves, L. and Donn, M. (1990b). An Augmented Frame Representation for Building Designs, Proceedings of the 4th New Zealand Expert Systems Conference, Palmerston North, New Zealand.

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

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

KEE (Knowledge Engineering Environment) (1988). *KEE Software Development System User’s manual*, Unisys, USA.

Minsky, A. (1975). A framework for representing knowledge, *The psychology of computer vision*, McGraw-Hill, pp 211-277.

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

Vervoert, C. (1987). Intergraph’s Objects Based Architectural Design and Production System, *NBS Data*, Vol. 24, September, pp. 39-59.

Winston, P. H. (1977). *Artificial Intelligence*, Addison Wesley.