MODELLING STATIC AND DYNAMIC KNOWLEDGE DURING DESIGN

Robin Drogemuller[1]

ABSTRACT

The ISO STEP standards for product data modelling and exchange are based on the use of Express. Express has acknowledged limitations. Modifications have been suggested to extend Express (Express-E) into a more object-oriented language and to add rules as a means of encoding knowledge. The EDM group have presented solutions to many of these limitations for the modelling of buildings.

This paper discusses BDeCS (Building Design and Construction System), an interactive system that supports building design and construction. BDeCS uses ideas from Express, EDM and the various knowledge representation formalisms developed in AI research. The initial development of BDeCS concentrated on the representation of knowledge within the system. The user interface was then developed around methods of interacting with the knowledge in as "natural" a way as possible. Limitations of both knowledge representation formalisms from the AI field and traditional software development have been identified.

The needs for extensibility and sharing of information among all of the members of the building design and construction team have lead to a software architecture that is unique. Methods of organising teams of people to work concurrently on the one project, while maintaining consistency of the data have also been developed.

Introduction

The use of computers to support design, rather than drafting, requires developments in three main areas. The first is in the modelling of information in the design domain. The second area is in modelling and supporting the expertise of the designer. The third area is in providing interfaces to computer systems that allow the designer to interact in a natural way with the computer system by hiding the detail of the information being manipulated.

The way that information is modelled during the design stage is critical to the ease with which computer based tools can assist in the design process (Eastman, 1975) and provides the basis for the product models used during manufacture, maintenance and disposal of a product.

---

1 Head, Division of Construction Management
James Cook University of North Queensland, Townsville Qld 4811, Australia
Robin.Drogemuller@jcu.edu.au

Recent developments in product modelling are based on object oriented approaches. Terminology varies widely, so the following definitions will be used:

Class : a group of objects with common properties and behaviour.

Instance : a single object that is a member of a class.

Where the terminology used within a particular implementation varies from this, these terms will be placed in braces after the alternative term.

Modelling Static and Dynamic Information

The information modelled in a computer system to assist design has a number of basic properties. Eastman (1994) lists them as:

Aggregation: of attributes and relations which describe one object;

Generalisation/Specialisation: into a classification hierarchy;

Composition: of component objects into a composite object (a type of aggregation); and

Classification: of an instance as a type of an object.

Aggregation is the means by which individual classes are derived, while Generalisation/Specialisation provides the basis for development of a hierarchy of classes. One of the problems with using classification systems within the Construction Industry (ie SfB) is that building information does not follow a simple hierarchical structure. A single wall panel can be an internal or external wall, load bearing or non-load bearing and may be one structural unit or part of a larger structural unit such as a shear wall running through several stories. The use of multiple inheritance provides a method for handling this complexity and also allows different disciplines within the construction industry to use specialised classification systems.

When considering the dynamic aspects of modelling there is a distinct separation between the modelling needs of domain knowledge - information about buildings for example - and modelling a particular instance or class of instances. For example, the general class 'door' has a definition that is stable. While defining the characteristics common to all doors is difficult, it is simple to decide whether an instance is a door. Once the class 'door' is defined it can stay within the schema for our model. There is no time dependence on the class 'door'. If an instance 'door_1' of 'door' is defined the time at which the door was created and (possibly) deleted must be stored. This information is required for two reasons - firstly, it is necessary to ask "What was the state of the model at time T?" and secondly, the user must be able to step backwards through the design space as part of the normal exploratory nature of design. Variation also exists in the constraints imposed on a building. The total cost, for example, may be fixed but there may be considerable latitude in the costs of particular elements.

There are no instances in a building project which can be guaranteed to last for the full life of the project. The design process may commence with a known site for which a cost effective use must be found or it may start with a perceived need for a built facility for which an appropriate site must be found. There is no guarantee that either the site or the building type or both will not change during the design process. If we can not

guarantee the continued existence of the two highest level instances within a project, then the component instances are no more stable. Even the client may change if a development project is sold before completion.

Problems with Current Technology

Computer Aided Drafting (CADr) systems are not readily able to support the life cycle of buildings since a large proportion of the decisions are made before any graphical design is started (Ferry & Brandon, 1991). Hence significant amounts of information need to be stored before there are any graphical elements to which information can be attached.

The standard relational data modelling techniques do not work well in design/ manufacture disciplines. The problems with using relational databases lie in three areas - the representational, the theoretical and the implementational.

Representational problems occur when the data is incomplete (Reiter, 1982). When a building project is first proposed, we know that it will have a roof and walls (in most cases). However, we do not necessarily know what construction technologies and materials will be used. For example, the value of the attribute material, for the roof cladding may be unknown until well into the design process. While designing an architect may want to use either terra cotta or concrete tiles for a roof, but not metal deck. Relational databases have problems representing such a list of possible choices or exclusions which are a subset of the allowed values.

Theoretical problems are caused by basic assumptions made in most relational systems (Reiter, 1982). Domain closure assumes that the only individuals are those that exist in the database. It is assumed that individuals with distinct names are distinct. The closed world assumption implies that the only possible instances of a relation are those implicit in the database. The RM/T relational model (Codd, 1979) provided a separate identifier for each individual that allowed two individuals with the same name (ie John Smith) to exist within the database. The scope of this identifier was over all instances within the database.

The complexity of data models for most design/manufacture domains imposes implementation problems. The complexity of the information that must be modelled means that capturing the full semantics of the data and relations is extremely difficult. The lack of commonality between different projects increases the difficulty of developing re-useable data models . Adding to the problems caused by complexity is the dynamic nature of much of the information. This dynamism means that the relational database model is unsuitable for most design systems (Eastman, 1994). This is due to the need to change the definitions of the relations from which the tables of data are structured and to then rebuild the tables. This is not an exercise to be performed several times a day as alternatives are tested.

Object-oriented database systems (OODB) are under development to overcome some of these problems. Since OODBs are not yet a mature technology individual systems must

be evaluated to ensure that they allow the schema to be modified at run-time to allow for restructuring of information as decisions are made.

A major constraint with many types of information systems is the issue of completeness/incompleteness. When is a model of a building complete? What information is required to completely define a building element? A further constraint on the development of industry knowledge bases is the need to localise knowledge. Different materials, techniques and regulations apply in various localities. Single skin masonry construction, for example, is acceptable in northern Australia, where cyclones are common, but is not accepted in other non-cyclonic areas.

Object-based approaches

STEP (ISO, 1994a) is the major international product modelling effort. It is supported by ISO and involves many people from a multitude of countries. STEP started in 1984 as an initiative to allow vendor independent exchange of all types of design knowledge, including graphics. The aim of STEP is to support the exchange of information throughout the building lifecycle. STEP does not specify how information is stored within a system. The only concern is the interfaces between systems.

The core of STEP is the Express product modelling language (ISO, 1994b). Schema are defined in Express and then exchanged using text files based on the exchange file syntax (ISO,1994c). The sharing of databases is also supported by a Standard Data Access Interface (SDAI) (ISO, 1994d).

Express is built on object-based concepts. Possible worlds are defined in terms of entities (classes). Each entity has attributes (properties) which can be explicit, optional, derived from other attributes or inverse. The attributes are typed and cardinalities can be set. Local and global constraints can be applied using rules. Functions and procedures can also be defined. A product is defined by instances with values assigned to the attributes. Within the parts of STEP, relations between instances are coded as attributes rather than by modelling relationships explicitly.

Explicit attributes are of little use when modelling the entire lifecycle of buildings. An explicit attribute giving a unique key with global scope is required for identification. In most circumstances all other attributes should be optional since it is difficult to specify unique attributes that will be applicable throughout the life of an instance.

Constraints are either undefined or satisfied. Partial integrity can be modelled using 'optional' attributes in conjunction with the EXISTS and NVL (null value) functions. Rule can be used to define the requirements for a particular stage of the design process but this is rather inelegant.

The underlying philosophy is that the whole world of discourse can be modelled in a universally applicable way. There are however calls within the SDAI that allow the extension of a schema and instances in a dynamic manner.

Most of the proposals for improving Express will add more object-oriented capabilities and will move it more towards being a programming language rather than a modelling language. The CSTB have developed rule-based extensions to Express than allow for more flexibility in knowledge representation.

EDM (Engineering Data Model) (Eastman, 1992) was developed at UCLA to address problems identified in architecture design. It has a long heritage, stretching back into the 1970's. EDM addresses the problems of storing and manipulating information within CAD systems and consequently has a wider scope than STEP.

The basic structure of information within EDM is very similar to STEP. Functional Entities (FE) correspond to STEP Entities The major differences within EDM are:

There is no explicit difference between instances and Function Entities (classes);

The scope of constraints is restricted to the FE and its component FEs.

The lack of distinction between instances and classes is an attempt to handle the dynamics of change within the design process. This provides a great deal of flexibility but leads to problems with separating instances from FEs (objects). Semantic nets (Mac Randal, 1988) have similar problems. Alternative methods of handling instances and classes are under consideration for the next version of EDM.

Restricting the scope of constraints within EDM is a significant difference from STEP. EDM prevents a model specifying that two walls are parallel within the constraints attached to the walls. The constraint that forces the walls to be parallel has to be part of the FE that defines the room. This forces the constraint to be placed in a position that is semantically more meaningful.

Approaches from Artificial Intelligence

The two most common representations for modelling information developed within AI research are rules (Buchanan & Shortliffe, 1984) and frames (Minsky, 1975).

Rules are single statements which specify what should happen of certain conditions are met:

IF rain_noise is a concern THEN roof_cladding becomes tiles OR roof_cladding becomes metal_deck_with_insulation.

Rules are useful in clearly defined sub-problems but become clumsy and difficult to maintain when large numbers of factors are interrelated.

Frames are very similar to objects in that they aggregate the properties of an object. Constraints and dæmons can be attached to attributes. However frames do not normally allow these constraints and dæmons to be inherited in the way that methods are inherited within object-oriented systems such as Smalltalk or C++ .

While neural nets (Coyne, 1990) can assist in design their 'black box' nature means that they are not particularly interesting when the representation of design information is the major interest.

User Interfaces

The developers of Computer Aided Drafting (CADr) systems are faced with a dilemma. Do they provide a general system where operations by users add generic information (points, lines, etc) or do they build in 'intelligent' support where the CADr system automatically converts geometrical constructs into the relevant building elements? The users of CADr systems appear to want both - systems which provide semantic support, making them discipline specific, while also wanting to exchange information between disciplines without loss of information.

BDeCS

BDeCS (Building Design & Construction System) is a modular system to assist in the design of buildings. BDeCS is intended to support building designers without replacing them. BDeCS concentrates on the representation and manipulation of information and provides decision support, leaving the selection of alternatives to the designer. Since flexibility in coding, extensive pattern matching and rapid Prototyping are required BDECS is largely written in Prolog (Clockson & Mellish, 1987). The Flex knowledge representation system (LPA, 1992) provides frames, rules, constraints and a forward chaining inference engine. The provision of backward chaining inference in Prolog and forward chaining in Flex covers the design and diagnostic aspects of reasoning required in the Analysis/Synthesis/Appraisal loop of design. Flex has been extended by adding the concepts of type and cardinality for attribute values to match the capabilities of Express (ISO, 1994b). Adding these concepts has made the development of the user interface easier since this meta-knowledge is used to allow the automatic generation of dialog boxes.

A user interacts with BDeCS through a typical CAD interface. There are a series of graphics screens to display different images of the current state of the building, a text area to provide textual feedback to the user in an unobtrusive manner, on-screen icons to allow common commands to be selected easily and pull-down menus for the less common commands. Large amounts of text, such as cost breakdowns and reports on thermal performance, can be displayed in separate text windows.

Augenbröe (1992) suggested an integration framework for building product models

Life cycle axis - stages in the life of a building

Actor/task axis - flow of information, jurisdiction amongst procurement team

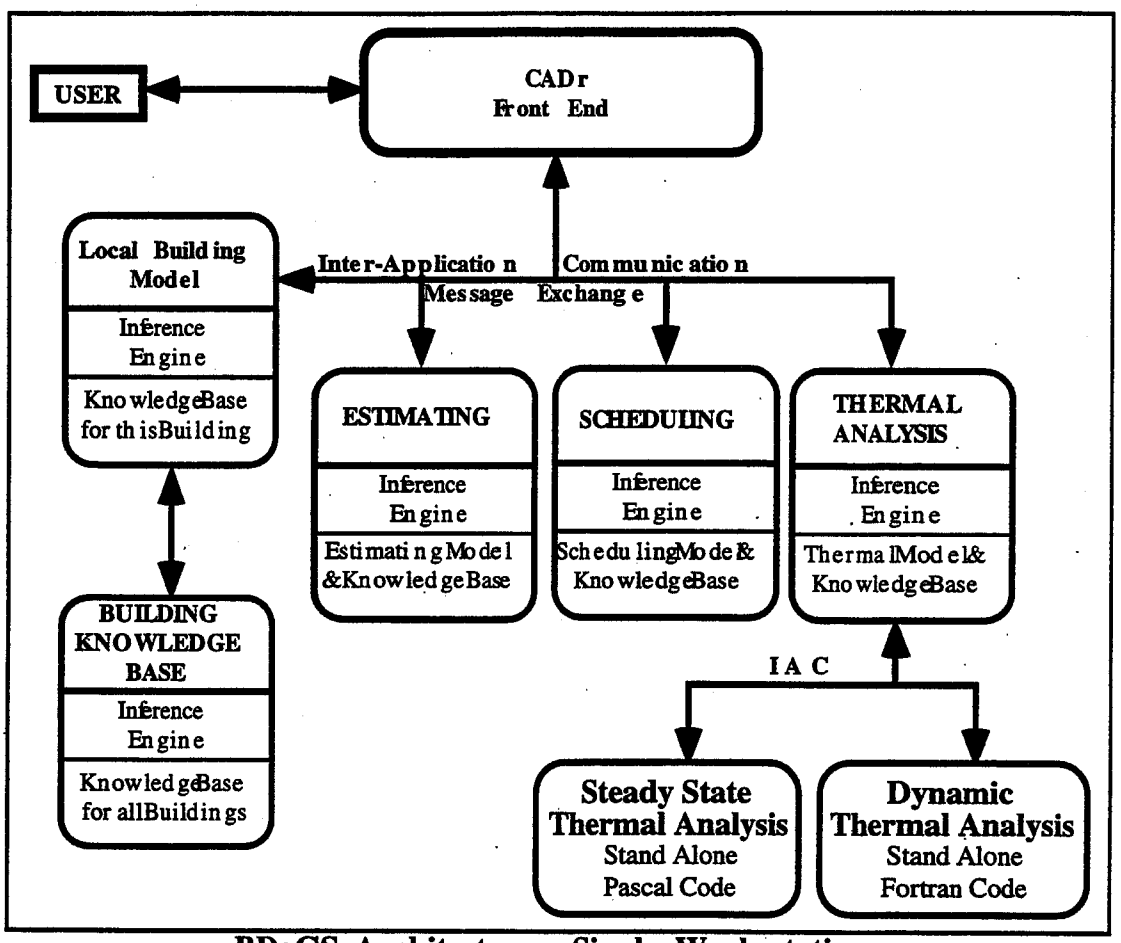
Enterprise axis - relating company-wide administration & organisational tasks

IT axis - actual implementation in hardware and software

To which we add two extra axes:

Building usage axis - user oriented aspects

Construction technology axis - choice of available construction systems



BDeCS Architecture : Single Work station
Figure 1

IT Axis
BDeCS consists of separate components that communicate using 'messaging'. Each message consists of a command that modifies the current building model. The expected current state is part of the message to allow automatic detection when two people want to modify the same element in different sessions. The user's identification is stored for audit purposes.

```
command(<Object>, <Current State>, <[Command, Parameters]>, UserID)
```

The components at a single workstation consist of a CAD user interface, a Project Database and various support programs that maintain their own internal databases (Figure 1). Each support program provides a single function or report required by the user. This component architecture allows the gradual extension of BDeCS capabilities in a modular manner. The messaging is currently implemented using AppleEvents (Apple Computer, 1993) but is portable to other platforms. Many of the components of BDeCS are currently written in Prolog due to the ease in prototyping and compactness of code but could be written in any computer language.

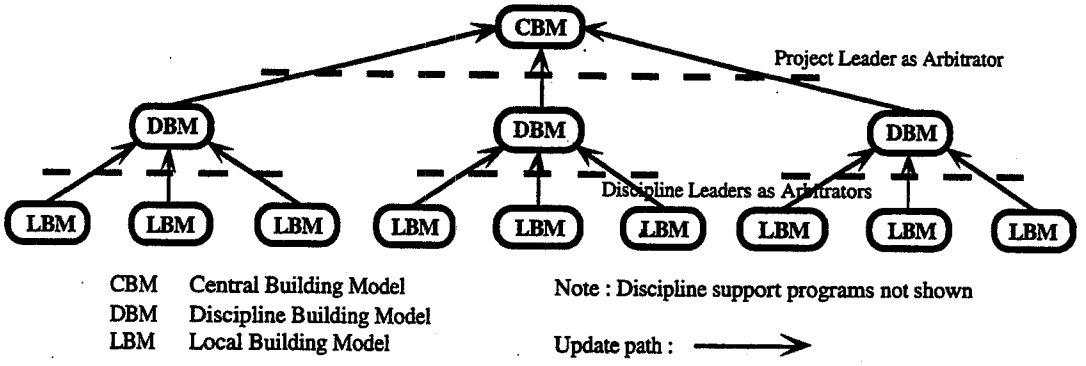


Figure 2 :Workgroup Structure

Life Cycle
BDeCS currently supports the development of design briefs, sketch design and limited aspects of detailed design. When developing the brief, the user is presented with a series of dialogs which allow the requirements to be selected. Provision has been made for all of the classes of buildings defined under the Building Code of Australia. The information displayed in the dialogs consists of lists of predefined items from the range of possible choices with default values already selected. The range of possible choices can be extended by clicking on a button and defining the new choice. The new item becomes a permanent part of the knowledge base.

Actor/Task and Enterprise Axes
These sectors are supported by providing a distributed architecture (Figure 2). The structure of BDeCS places the central building model (CBM) at the root of a tree with LBMs in surrounding nodes. The CBM stores the current 'approved' state of the design. The CBM maintains a complete model of the building project in a knowledge base, the available computer resources (either knowledge based or procedural) and the people who are playing particular roles in the development of the project.

When a task is complete the user uploads the changes required to building model from their workstation to the superior node where the changes can be checked for consistency with other concurrent work. After checking, the changes can then be passed up the tree for more checking and eventual merging with the CBM. The CBM makes the changes to its internal model and then broadcasts a message to all of the other LBMs.

Building Usage Axis
The types of activities which occur within a building place significant constraints on the choice of structural system, external and internal walling, servicing, etc. The classifications of building types within the Building Code of Australia provided a useful starting point. This information is encoding within BDeCS. Once the user selects a particular class(es) of building, knowledge bases containing the relevant knowledge are attached.

Construction Technology Axis
The choice of the various construction technologies is dependant on the use, location, etc of the building. No restrictions have been placed on the choice of technologies for any building type or location. Assistance is provided by using defaults in selecting the most commonly used options, but these can be easily overridden or new technologies added. If there are reasons for not selecting a particular choice the reasons are displayed but the final choice rests with the user.

Eastman (1993) lists capabilities required of systems where alternatives are being assessed:

Version control: guarantees data consistency. Allows alternatives to be developed in parallel;

Integrity management: ensuring that all constraints are satisfied;

Concurrency: allows several operators to work on the model simultaneously;

Extensibility: allows the building model to be changed as technologies and systems are selected in the design stages.

and breaks representations into:

State description: defines the geometry and material properties;

Behavioural description: describes the interaction with the surroundings.

BDeCS supports version control by maintaining one definitive copy of the building model in the CBM. Alternatives can be developed as clones of the CBM at a particular point and then refusing all updates until a particular alternative has been selected.

Integrity management is handled by the constraint system built in to *flex* . The management of concurrency and extensibility have been described above.

The CBM and LBMs contain the state descriptions of the building. The behavioural descriptions are embedded in the support nodes at each workstation (figure 1)

Geometry
The methods for handling geometry in commercially available CAD systems vary widely. Some systems provide basic geometrical primitives, such as point, line, arc, etc, while others provide parametric relationships and constraints. BDeCS only uses the geometric primitives in the graphical interface. All parameters and constraints are stored within the building model. This allows a uniform treatment of all constraints without making an artificial distinction between geometric and non-geometric constraints. It also allows constraints and parameters to be applied or overridden at several levels of granularity. For example, the standard roof slope can be specified for the whole roof and then overridden for a particular roof plane.

When defining the geometry of many of the elements within a building the three dimensional plane is the defining element on which the geometry is based. Most other elements can be set out with respect to the relevant planes. This is partially implemented within BDeCS and will be fully implemented within the near future.

Conclusion

The continued development of CAD systems is dependent on developments in product modelling technology if the full life cycle of buildings is to be supported. There are many issues which need to be solved before proposals such as STEP and EDM fulfil these product modelling needs. Two key issues are the representation of knowledge within a model and the standardisation of industry models. Other issues include the modularisation of knowledge, sharing of knowledge and the coordination of concurrent work by groups of people on the one model.

Solutions and partial solutions to some of these problems have been implemented in BDeCS. A merging of knowledge representation formalisms from both product modelling and the AI field has proved useful. The ability to encapsulate discipline specific information in one program which can then provide support to the designer significantly reduces the size and complexity of code. The ability to extend building product models while designing is necessary, but introduces some complex issues regarding the exchange of information.

References

Apple Computer, 1993, *Inside Macintosh : Interapplication Communication*, Addison-Wesley

Augenbröe, G., 1992, Integrated Building Performance Evaluation in the Early Design Stages, *Building and Environment*, 27 (2), pp 149 - 161

Buchanan, B.G. & Shortliffe, E.H., 1984, *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*, Addison-Wesley

Clockson, W.F. & Mellish, C.S., 1987, *Programming in Prolog* , 3rd Ed, Springer-Verlag

Codd, E.F., 1979, Extending the Database Relational Model to Capture More Meaning, *ACM Transactions on Database Systems*, 4 (4), December 1979, pp 397-434

Coyne, R.D., 1990, Design Reasoning Without Explanations, *AI Magazine*, Winter 1990, pp 72 - 80

Eastman, C.M., A Data Model Analysis of Modularity and Extensibility in Building Databases, *Building and Environment*, 27 (2), pp 135 - 148

Eastman, C.M., 1994, A Data Model for Design Knowledge, in Carrara, G. & Kalay, Y.E. (eds), *Knowledge-Based Computer-Aided Architectural Design*, Elsevier

Ferry D.J. & Brandon, P.S., 1991, *Cost Planning of Buildings* , BSP Professional Books

ISO, 1994a, ISO 10303-1 Industrial Automation Systems and Integration - Product Data Representation and Exchange - *Part 1: Overview and Fundamental Principles*, ISO

ISO, 1994b, ISO 10303-11 Industrial Automation Systems and Integration - Product Data Representation and Exchange - *Part 11: Description Methods: The EXPRESS Language Reference Manual*, ISO

ISO, 1994c, ISO 10303-21 Industrial Automation Systems and Integration - Product Data Representation and Exchange - *Part 21: Implementation Methods: Clear Text Encoding of the Exchange Structure*, ISO

ISO, 1994d, ISO 10303-22 Industrial Automation Systems and Integration - Product Data Representation and Exchange - *Part 22 Implementation Methods: Standard Data Access Interface Specification*, ISO

LPA, *flex* Expert System Toolkit Version 1.2 (Updated), LPA

Mac Randal, D., 1988, Semantic Networks, in Ringland, G.A. & Duce, D.A. (eds) *Approaches to Knowledge Representation: An Introduction* , John Wiley & Sons

Minsky, M., 1975, A Framework for Representing Knowledge, in Winston, P.H. (ed) *The Psychology of Human Vision* , McGraw-Hill

Reiter, R., 1984, Towards a Logical Reconstruction of Relational Database Theory, in M. L. Brodie *et al* (eds), *On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases and Programming Languages* , Springer-Verlag