Multi-Disciplinary Views for Integrated and Concurrent Design

R W AMOR and J G HOSKING[*]

ABSTRACT

The definition and implementation of user views is likely to be a major factor in the success of the ISO-STEP standard for computer-based representation of building components and their inter-connection. The development of a method for describing user views to a particular model is also likely to increase the usage of existing design tools. Currently, the arcane languages and the detailed knowledge required of the physics and terminology of the specific domains of many of these tools limit their use by designers. This paper addresses these issues by describing a system that can present information from a base computer model of a building to a given user. The language and level of detail of the system are directed at the needs and understanding of the user. This system allows multiple concurrent views to the base model, each view tailored to a particular discipline (*eg*, architect, structural engineer, services engineer, *etc*) and further tailored to meet the specific needs of the particular user in terms of understanding of the various disciplines and the level of information required. Used with a system that integrates various design tools through a computer based building model, this system will offer users information from a range of design tools at a level that they can comprehend.

Key Words
multi discipline views; product modelling; user interaction

INTRODUCTION

The emerging standard ISO 10303, colloquially known as STEP (STandard for the Exchange of Product model data, STEP 1991), is being developed for the exchange of product data in the fields of manufacturing, architecture, engineering, construction and electronics. While it is likely to be many years before a full representation of buildings can be supported by this standard, it is almost guaranteed success through the support of major commercial and governmental organisations such as Boeing, the US Army and the US Navy.

The STEP standard will become the preferred method for transferring building information between various CAD systems, simulation tools and knowledge based systems. It will also make available technical information

[*] *Department of Computer Science, University of Auckland, Private Bag 92019, Auckland, New Zealand.*

about products in a form that can be accessed by the design tools. This will help make these tools more attractive to use in the design of a building. Such design tools have been available to designers for many years, but there have been several obstacles to their acceptance and use. Amongst these are:

- The duplication of effort in describing the same building to multiple design tools. To gain information about varied aspects of the building design the same data must be input into each tool in its own specific format.
- The language used to describe a building to any tool. This is often arcane and in many cases unreadable to a human operator. This leads to long learning curves, and many opportunities for errors in describing the building to the tool.
- The level of expertise required to describe a building to the design tools. A high level of expertise in the specific area, and a good knowledge of the physics and mechanical aspects of the components involved, is often assumed.

While the STEP standard will help overcome some of the problems detailed above, it will do little to change the interface to design tools, apart from providing a view of a building suitable for computer based transfer of building data.

The problem of interacting with users in a manner with which they feel comfortable has long been recognised. For example, in 1977, Kay writing about trends in computing noted that:

'...20th-century physics assigns equal importance to a phenomenon and its context, since observers with different vantage points perceive the world differently. In an observer language, activities are replaced by "viewpoints" that become attached to one another to form correspondences between concepts.' (p239, Kay 1977).

Only recently, however, has the emphasis of the conversation with a design tool shifted from requirements necessitated by the structure of the program and the language in which it was written, to something closer to the views of the user. Unfortunately, most of the tools which incorporate such user-friendly interfaces are aimed directly at one market in the industry, *eg*, architects, and so do not address the needs of other users.

To understand what we envisage when talking about views we describe what we believe constitutes a user view. There are several components to a view, many of them technical ones relating to the technology required to support integration and concurrency. However, from a users viewpoint we describe a view as being able to:

- Provide the information required by the user in the form they request, request information that the user will know, only request information that is needed, and use other sources to find other required information, for example other users in the system, defaults for specific objects, or information from external databases.

- Work at the same level of comprehension as the user. This involves tailoring concepts to the user's level of comprehension so as not to overwhelm the user.
- Follow the methods of design with which the user is familiar.

In this paper we look at what is required to establish multiple views to a data model such as STEP. This is discussed with reference to available methods for providing views and the current research being undertaken in the field. Later sections develop an initial method for providing user views and demonstrate views to a building model. As STEP is not yet in a form which is useable for such a project, we use the model provided in the ICAtect system described later in the paper.

METHODS OF PROVIDING VIEWS

Existing technology and current research provide some methods of describing user views to a given base model. In this section we briefly examine some of these methods.

Database Views

Databases have long been the main method used to store data about objects in a particular domain. A database, whether in relational form, network form, *etc*, provides a model of a particular domain. In the field of databases a mechanism is provided to specify a view of the base data for users and external systems. This gives the user access to a database through one or more levels of abstraction from the physical implementation of the database.

From Ullman (1982) we define a conceptual database as an abstraction of some real world domain. A view to a database is defined as an abstract model of a portion of this conceptual database (see Figure 1). This view is defined through some subscheme data manipulation language, which in practice is usually a constrained version of the schema definition language used to define the conceptual database. In some senses a view is just a conceptual database, and at the same level of abstraction. There are senses in which it is more abstract, in that it can be constructible from the conceptual database (through formulae and aggregate functions), but not actually present in it.

Database views provide for data manipulation on the conceptual database, including updates, additions and deletions. However, there are very severe constraints on the types of views that these operations are allowed to act upon. In most systems, such as SQL (van der Lans, 1988), the restrictions on updateable views limit the cases almost to the point where there is a one-to-one correspondence between the view and its base relation.

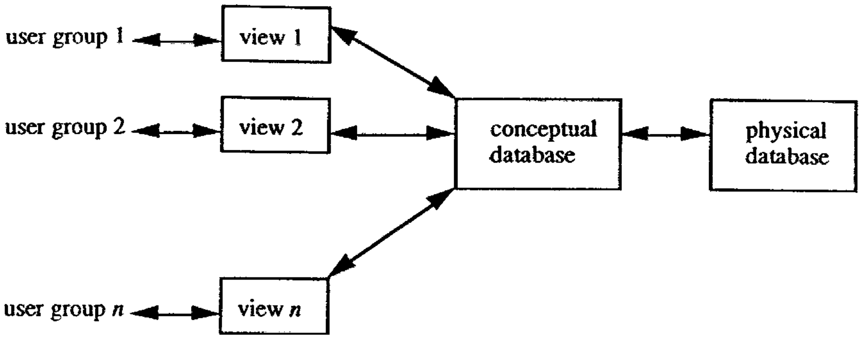


Figure 1. Views of a Database (Ullman, 1982)

One-to-one mappings between base relations and views do not provide for particularly friendly or useable views. The data presented to and requested from the user will be very much a reflection of the base relations, which is a computer based view of a domain, serving the needs of many groups. This is unlikely to be at the level at which an individual user would wish to work.

The research area of database schema integration (Batini *et al*, 1986) provides insights to the problems in mapping between the various perspectives of similar objects when integrating databases from various organisations. These mappings can be considered analogous to the mappings that will be required between a user view and a building model.

Views in Engineering and Architecture

Many researchers throughout the world are working on the problem of defining and maintaining views as part of their systems. Their work falls into two categories, the first being similar to the database views in that the mappings are essentially one-to-one, and the second being at a higher level through the use of functional abstractions.

The first approach (Luiten and Tolman 1992; STEP 1991) specifies a sub-set of the global model to be visible in a particular view. This approach relies on the central model containing all the objects and abstractions that a user would wish to use. It is compatible with current database technology as the mapping to a view is always one-to-one, and the database system can use its locking mechanisms to control access to the data and to maintain consistency of updates even when multiple users have overlapping views.

The second approach (Bowen and Bahler, 1992; Wong *et al*, 1992; Clarke *et al*, 1989;) specifies functional abstractions of the global model which are visible in a particular view. This approach relies upon the ability to define a mapping from the global model to a specific view and to enable data to be converted in both directions as needed. This makes the approach more powerful than the model sub-sets, and in fact subsumes that approach. The systems that take this approach require application-specific code to handle mappings between objects in the various views, and, in most cases, require specially tailored knowledge bases or code attached to the global objects, to maintain the conceptual views for each user.

The ICAtect Project

An example of the type of project described above is the ICAtect project (Amor *et al*, 1990; Amor, 1990). ICAtect has been developed over the last four years to examine a method of design tool integration, specifically for preliminary architectural design. The aim is to make quality information available to architects from these design tools as they examine variations in their building design to satisfy the design requirements.

The core of the current ICAtect system is a model of a building capable of holding all information required by a range of design tools useful to architects in the preliminary design stage (see Figure 2 for the structure of ICAtect). This common building model (CBM) was created from an analysis and amalgamation of the classes and attributes used by various design tools to describe a building for their simulation purposes (in a similar manner to the database schema integration methods described in Batini *et al*, 1986).

To allow data to pass between ICAtect and the design tools, a mechanism for moving common building data, mainly geometric, is necessary. This is achieved by providing a mapping of data between every design tool required, and the CBM, enabling ICAtect to move its description of a building from one design tool to the next as needed. This is similar to the view methods provided by functional abstraction systems, as described in the previous section.

To allow user interaction with ICAtect an interface is provided that is structured to be easy and intuitive to use. It provides one language to describe a building to any design tool; and, through the use of constraints on classes and attributes in the CBM, validates the consistency of the building design as information is entered.

ICAtect is structured to allow the user to analyse a building design at an early stage when very little of the building information has been specified. This is done by providing much of the detailed information required by the design tools as defaults. Once the system knows the type of building being constructed and its general locality, the defaults can be used to provide other unstated information.

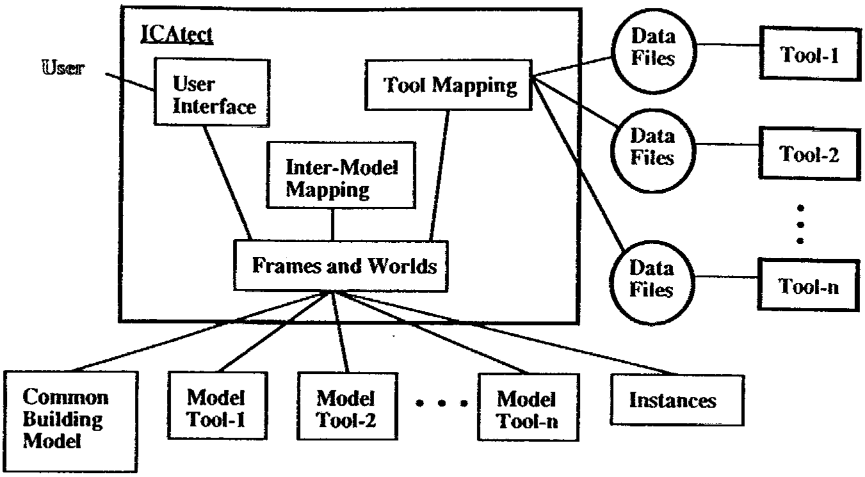


Figure 2. The Structure of ICAtect

A New Direction for ICAtect

Analysis of the prototype ICAtect system and further work on a graphical interface to ICAtect (Dearden, 1991) highlighted some of the deficiencies of the present model and system. Having constructed a CBM from various simulation tool models, access to the resulting system still requires the user to think and work at the same semantic level as the various simulation tools, *ie*, the user must still describe the building design in a language similar to those of simulation tools.

To address this problem it is necessary to generalise the structure of ICAtect's user interface subsystem to make it capable of handling multiple views of the building model. This structure will allow views of the database to be tailored not just for different classes of design tools, such as simulation and knowledge-based systems, but also for the different classes of practitioner in the building profession such as architects, engineers, developers, *etc.* Thus, an architect is presented with a quite different view of the building from a structural engineer's. This ensures that users of each class are addressed in a language they understand, and are presented with the view of the building most relevant to them. The amount of similar work undertaken in the areas of quantity surveying, thermal engineering and structural engineering (Karstila *et al*, 1991; Luiten and Tolman, 1992; Wong *et al*, 1992; Clarke *et al*, 1989) demonstrates quite clearly that this project has greater scope than the field of preliminary architectural design for which it was first envisaged.

To find some help to the problem of describing multiple user views we look to research in the field of software engineering. Many aspects of software engineering mirror those in building design, making the application of software engineering techniques a viable approach to solving building industry problems. In particular, the work on programming environments is directly applicable to the provision of views of a building model as described below.

Visual Programming Environments

As diagrams are useful in the software lifecycle to help define, explain and understand concepts that are difficult to represent in text, it is natural that researchers look at methods of incorporating diagrams into the software development and maintenance cycle. This has lead to the increasingly popular field of visual programming (Ambler and Burnett, 1989).

Grundy and Hosking (1992), working in the area of a visual programming environment for object-oriented languages, have developed the MViews framework. Some of the major benefits of their system are:

- Multiple graphical and textual views of the program can be defined, with any view being able to share information with other views.
- The use of diagram views can focus a programmer's concentration on particular aspects of the program, yet the programming of details is still supported in a textual view.
- Consistency is maintained throughout all the views, whichever view the information is modified in.

The structure of the MViews visual programming environment is defined at three levels. The base level contains the complete definition of the program. Above this are subset views which, as their name suggests, define a subset of the program. These subsets are free to overlap, and common information can be accessed and modified in any of the subsets. At the top level are the display views which describe how some piece of code can be displayed and manipulated on a screen (see Figure 3). Several display views can be used on the same subset view, providing a different rendering of the same information in each view.

From an analysis of the MViews environment it is clear that it is sophisticated enough to provide the basic support required for describing views to a given model and generalisable to domains other than that of object-oriented programming. The structure of its solution for a programming environment with its definition of a base level is certainly applicable to the field of integrated and standardised building models.

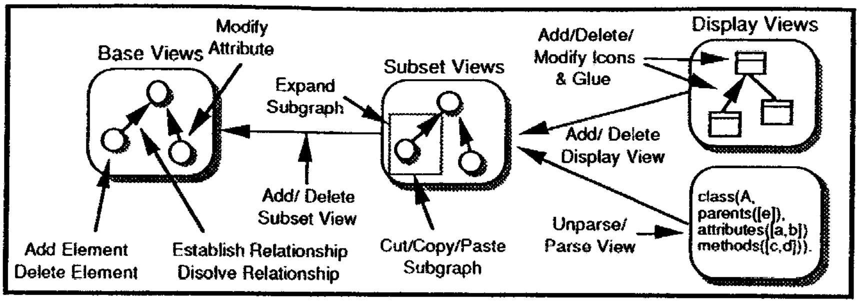


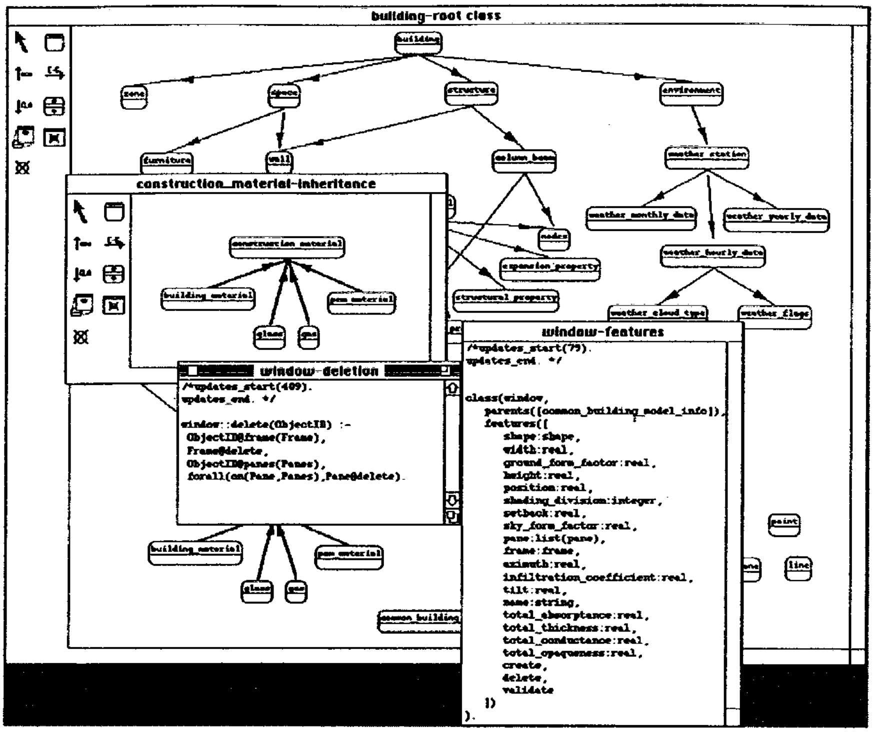
Figure 3. View Management in the MViews System (Grundy and Hosking, 1992)

VIEWS THROUGH MVIEWS

Given the requirements for a new interface to ICAtect, and the basic support offered by the MViews system, this appears to be the most fruitful research direction. In addition to being able to display differing views for differing purposes in ICAtect, it is also imperative to maintain the consistency between the various views, so that modifications to any view are propagated appropriately to other views. The MViews visual programming environment provides the requisite support for multiple, overlapping, editable views of a program, with a built-in consistency management system which propagates modifications from view to view automatically via a common model of the program. The challenge is to extend the method of view definition to be able to define and manage views that reflect the needs of users other than programmers.

Given the similar definition of the CBM in ICAtect, a frames system, and the object-oriented language supported in MViews it was a straight forward task to re-implement the CBM inside MViews. This provides a system for managing the CBM in terms of making updates to the model, checking connections between classes and editing the definition of the classes in textual views, all in a unified and consistently maintained environment.

Figure 4 shows a snapshot of the CBM being manipulated in the MViews system. The main graphical window shows the base graphical view of the CBM. This view details all the classes in the CBM and the *part-of* relations between the different classes. The small graphical window on the left shows the inheritance relations between material classes in the CBM (notice that all the classes in this window are previously referenced in the base graphical window). The textual window on the right details the attributes and methods of the class window and the textual window on the left provides a second textual view of the class window detailing its deletion method.



window-deletion

```
/*updates_start(409).
updates_end. */

window::delete(ObjectID) :-
 ObjectID@frame(Frame),
 Frame@delete,
 ObjectID@panes(Panes),
 forall(on(Pane,Panes),Pane@delete).
```

window-features

```
/*updates_start(75).
updates_end. */


class(window,
   parents([common_building_model_info]),
   features([
      shape:shape,
      width:real,
      ground_form_factor:real,
      height:real,
      position:real,
      shading_division:integer,
      setback:real,
      sky_form_factor:real,
      pane:list(pane),
      frame:frame,
      azimuth:real,
      infiltration_coefficient:real,
      tilt:real,
      name:string,
      total_absorptance:real,
      total_thickness:real,
      total_conductance:real,
      total_opaqueness:real,
      create,
      delete,
      validate
   ])
).
```

Figure 4. The CBM in MViews

The MViews system provides many functions to help manipulate the CBM. The icons to the left of the graphical windows provide access to certain functions on classes, a pointer tool for selecting and manipulating classes is at the top left, and the tool for adding classes at the top right. When placing a class the user defines a class name and, optionally, other pertinent information. If the class exists in the base then a connection will be established between the existing definition and the graphic icon, or else a new class definition is entered in the base. In a similar manner the next two rows of icons allow the user to define generalisation, client-server, and classification links between classes, and to define features of a class. The last two rows of icons provide for view creation, hiding classes, links and views, and deleting the class, link or view. Each class icon has several regions sensitive to mouse clicks, which can be used to: bring up the list of views for the class; automatically display the class text view; show the feature views; show the feature text; show the class features; and show all class features.

To ensure the results of the work on user views are generalisable to the tools that are integrated in the ICAtect system we have also created a view for one of the more difficult tools integrated into ICAtect. This view (see Figure 5) describes the model of a building used by the design tool SUNCODE (1981). This model was difficult to integrate into ICAtect as it is based purely on the need of a thermal analysis through paths of heat flow. Hence it has little notion of room geometry and location as required by most other design tools and the CBM.

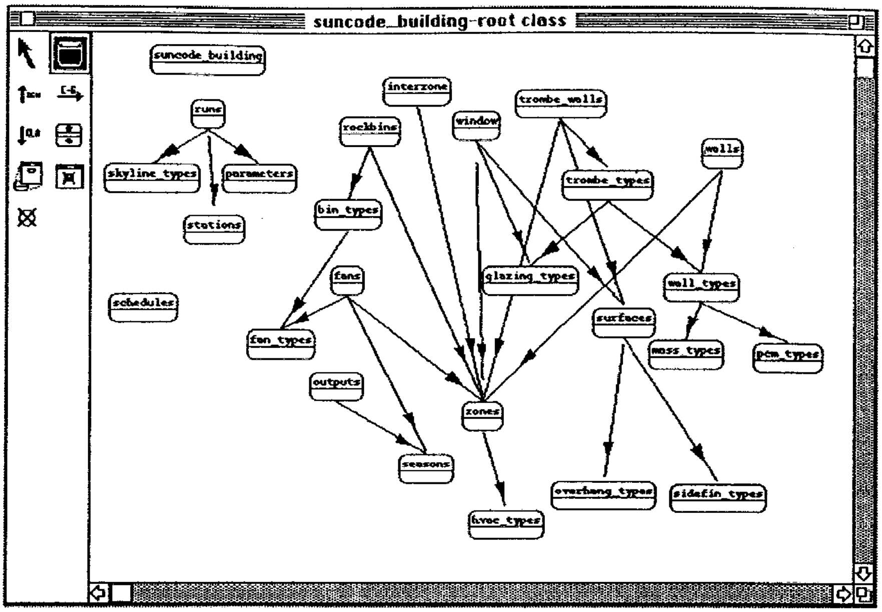


Figure 5. SUNCODE Building Model

The MViews system is now being extended from views of the class definitions to provide the same services for the instances of classes created in the ICAtect system. At this point we have defined the first user view to the model, that of the programmer/implementor who will need to deal with instances of classes in their raw form, but who also needs to see the overview of classes and the connections between them in the system.

From the definition of the first user view (the programmer/implementor) of the model the next step is the definition of user views for various practitioners. Through interviews with a graduate architect and a building scientist we have developed two views of a building which are directly applicable to these two users. These views capture the classes, and relationships between classes, that these two users think of when working with a building. These two views have been described in the MViews system with the same types of display views as used for the CBM (see Figure 6 for the model of a user view). This allows us to manipulate the structure of the views and to query and examine the information stored in the views.

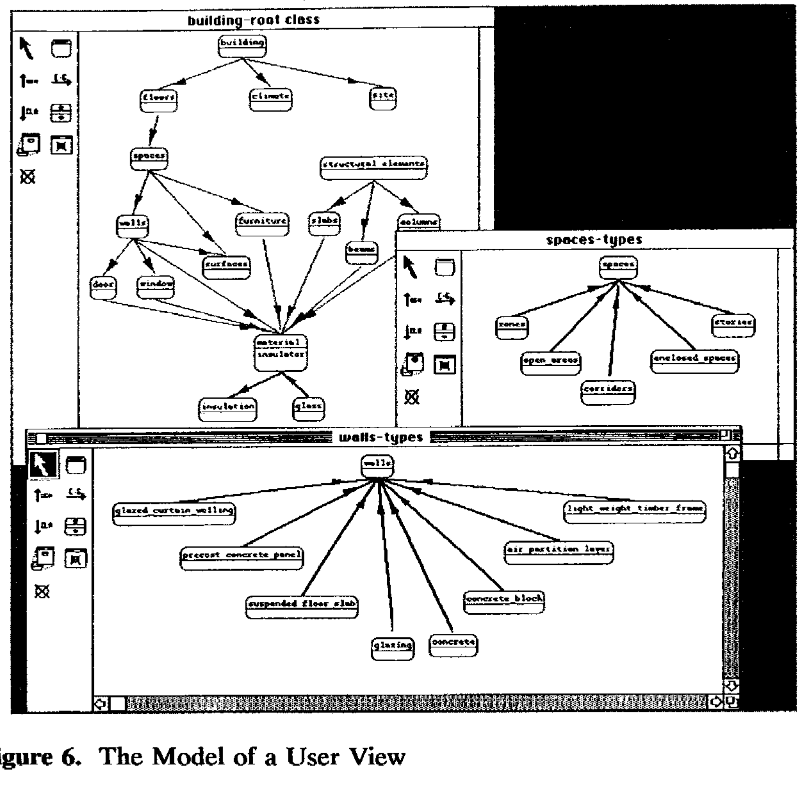


Figure 6. The Model of a User View

With this set of views defined in MViews we now need to analyse the mappings required between the views and the CBM. From the results of this analysis we will know exactly what types of mappings will need to be supported for a particular type of view and will then work on the method for defining the requisite mappings. The definition of the mappings will be developed along with the method to transfer data back and forth between the views and the CBM.

CONCLUSIONS

With the evolution of very large integrated, and increasingly standardised, building models for the building industry it is becoming more and more important to provide user definable and modifiable interfaces. Many of the technologies required to provide multiple user views have already been developed and there are many groups working towards this goal. However, little of the work has concentrated on a formalised schema for the definition of user views of a base model.

The use of the available research in visual programming environments provides us with the starting point in providing user definable views. The MViews system provides a wonderful programmers interface to a common building model, users views of a building and a design tool's view of a building. With the definition of these views further work is being undertaken to analyse the required mappings between the views and the CBM and on developing methods to describe and manage these mappings in the MViews environment. When completed the system will finally provide user views tailored to user needs.

Acknowledgment

The financial assistance provided by an Auckland University Graduate Scholarship is gratefully acknowledged.

References

Ambler, A and Burnett, M (1989), Influence of Visual Technology on the Evolution of Language Environments, IEEE Computer, 22, pp 9-22.

Amor, R, Groves, L and Donn, M (1990), Integrating Design Tools: An Object-Oriented Approach, Building Systems Automation-Integration, First International Symposium, June 2-8, University of Wisconsin, Madison, Wisconsin, USA.

Amor, R (1990), ICAtect: Integrating Design Tools for Preliminary Architectural Design, MSc thesis, Dept. of Computer Science, Victoria University of Wellington, Wellington, New Zealand.

Batini, C, Lenzerini, M and Navathe, S B (1986), A Comparative Analysis of Methodologies for Database Schema Integration, ACM Computing Surveys, 18(4), December, pp 323-364.

Bowen, J and Bahler, D (1992), Negotiation in Concurrent Engineering, Technical report TR-92-10, North Carolina State University, USA.

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

Dearden, R (1991), Object-oriented CAD system, Honours report, Department of Computer Science, Victoria University of Wellington, Wellington, New Zealand.

Grundy, J and Hosking, J G (1992), The MViews framework for constructing multi-view editing environments, Proceedings New Zealand Computer Science Research Students' Conference, Hamilton, 28-30 October.

Hosking, J G (1990), A PlanEntry Package for Thermal Design, BRANZ contract No. 85-024, Technical Report No. 7, Department of Computer Science, Auckland University, Auckland, New Zealand.

Hosking, J G, Mugridge, W B and Hamer, J (1991), An architecture for code of practice conformance systems, in Kahkonen and Björk (eds) Computers and Building Regulations, VTT Symposium 125, VTT Espoo, Finland, pp 171-180.

Kay, A C (1977), Microelectronics and the Personal Computer, Scientific American, September, pp 230-244.

Karstila, K, Björk, B C and Hannus, M (1991), A conceptual framework for design and construction information, Building Systems Automation-Integration, First International Symposium, June 2-8, University of Wisconsin, Madison, Wisconsin, USA.

Luiten, B and Tolman, F (1992), Computer Aided DfC (Design for Construction) in the Building and Construction Industries, CIB W78, Computer and Building Standards Workshop, Montreal, 11-15 May.

STEP (1991), Part 1: Overview and Fundamental Principles, Draft N14, ISO TC 184/SC4/WG6.

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

Ullman, D U (1982), Principles of Database Systems, Computer Science Press.

van der Lans, R F (1988), Introduction to SQL, Addison-Wesley Publishing Company.

Wong, A, Sriram, D and Logcher, R (1992), SHARED: An Information Model for Cooperative Product Development, submitted to IEEE Computer.

---

c. 1993, Management of Information Technology for Construction, K. Mathur et al (Eds), World Scientific Publishing Co., Singapore.