A Framework for the Integration of Design Tools

Robert W. Amor and Dr John G. Hosking
Department of Computer Science, University of Auckland, Auckland

ABSTRACT
The development of a system capable of integrating a range of design tools from a particular domain poses many problems. A framework for integrating design tools provides a structured approach to the problem and allows individual parts to be tackled independently. In this paper we detail a particular framework for integration and describe its individual components. A method for modelling and implementing each component is described showing how such an integrated system can be realised.

1. Introduction

The ability to utilise a set of disparate design tools in an integrated system is problematic for any domain. It is particularly difficult in the fields of architecture and engineering where integrated systems have long been touted. The majority of the claims made for computerisation and integration in these fields have never been achieved. For architects there are few, if any, tools which are useful for initial or sketch design and in many architectural firms the only use of computers during the design process is for detailing the building plans with a draughting tool (CAD). In the engineering domain there are similar problems. Even with a computerised plan of the building it is unlikely that building information can be extracted from this plan for use in other design tools. Many practitioners re-enter building information into their design tools by hand, taking measurements off the printed plans. Despite the existence of a wide range of excellent design tools, such tools are seldom used because of the difficulty and time required to enter building information in order to invoke the tool.

Examining the state of computing in these domains it is clear that the majority of the problems lie with the conceptual models of a building. In a wide range of CAD tools there is no underlying model of a building. That a plan defines a building can only be ascertained by human perception. To the computer a plan is a set of 3D graphical entities. Until computers can mimic the human ability to perceive building structure from a plan, it is impossible to extract useful building information from a traditional CAD system. There is also a problem with the wide range of design tools that model buildings. Each of these design tools has a model of a building most suited to the type of computation it performs on the building, and in most cases this is a minimal model to ease the burden of data entry for the user. As can be imagined, this means that models of buildings for tools such as thermal simulation, structural simulation, lighting analysis, fire-code compliance, quantity surveying, etc. have very different structures. As a result, describing a building for use with two different design tools is like trying to explain exactly the same thing in two totally different languages.

One other important problem is that of consistency and coordination. In a building project there are often several design professionals responsible for different parts of the design. A project manager has the task of coordinating the various designers, and in a manual design practice a range of procedures are used to ensure that the design is kept consistent between all designers. In a computerised design environment it has become no easier to perform this task, and in many cases much more difficult, as the number of plan portions which can be generated with computerised tools is much greater than with manual design.

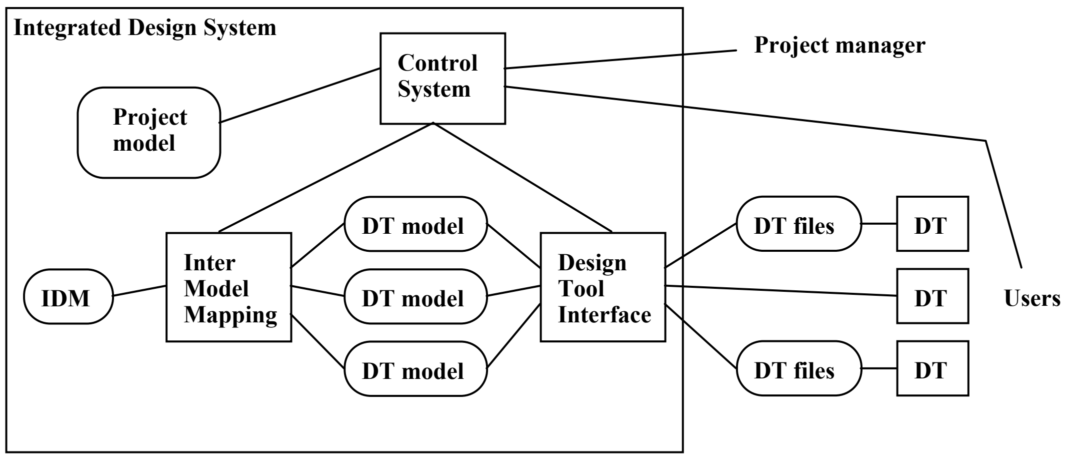


Fig 1. Structure of an integrated design system

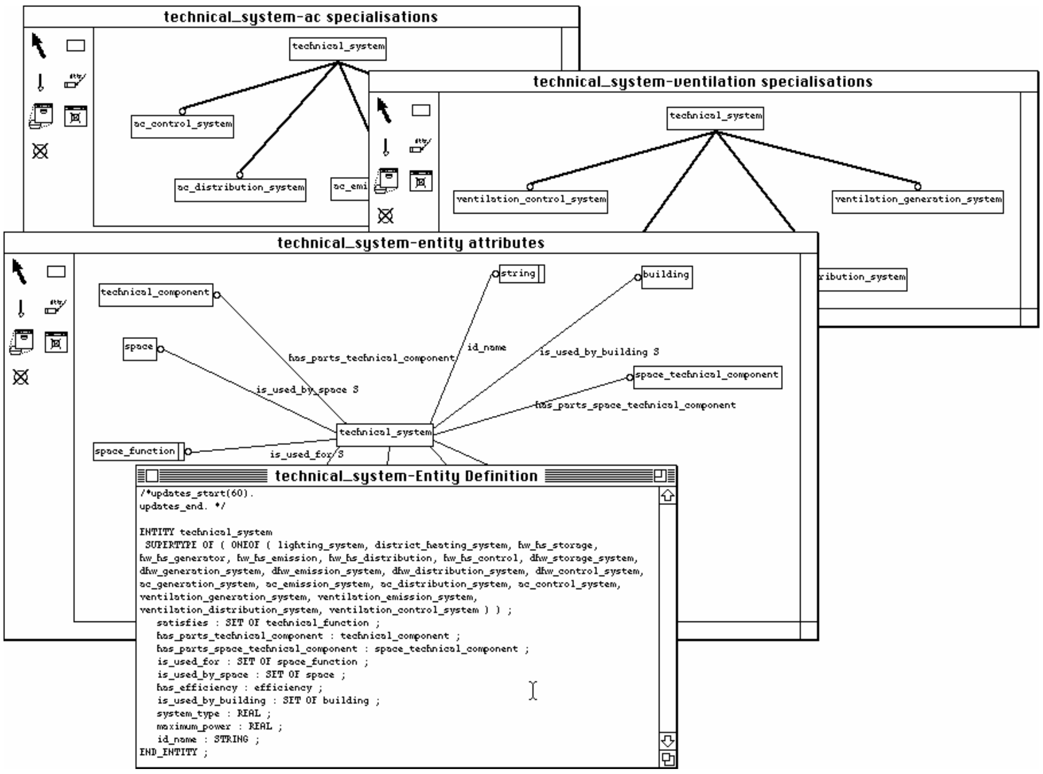
To help solve these problems many research groups [1, 2, 3] have explored integrated design systems with structures similar to that in Fig. 1. In such a system a control system draws upon the project model to determine the next set of allowable actions by users of the system. Users utilise their design tools (DT) to accomplish certain design tasks, the results of which are passed into the integrated design system and merged with data stored in the integrated data model (IDM). Conflicts between the data from various users are detected and the affected users negotiate compromises to their problems. The control system monitors the problems and resolutions, and, dependent upon the results, determines the next set of available design tasks for the attached users. The remainder of the paper details the functionality of each of the components in Fig. 1 along with the authors' approaches to developing modelling tools required to define each component and methods of implementation.

2. Modules in the Integrated Design System

The integrated design system shown in Fig. 1 can be implemented in a variety of ways, from a monolithic system with all modules tightly coupled, through to a completely modular system with links to relational or object-oriented databases and where many of the modules operate as independent systems utilising a brokering architecture to communicate data between modules. However the system is implemented, the same major conceptual sub-systems exist in the integrated design system. In this section each of these sub-systems is detailed to give some idea of the scope of the task each must handle and to offer, the authors' approach to modelling and implementing each module.

IDM

This is the very general model of the domain which must cover the information requirements of the designers and design tools which interact with the system. The model required for buildings is probably the largest and most complex information model developed in the world to date. Recently several collaborative projects have worked on general models of buildings based on the ISO STEP fundamentals for restricted aspects of buildings [4, 1, 2]. The development of the IDM requires the input of domain experts for all the sub-domains being modelled as well as the help of modelling experts to structure the model. The IDM remains as a fairly static model, though the introduction of new design tools, construction techniques and building constructs may require modifications to the model.


technical_system-ac specialisations
technical_system
ac_control_system
ac_distribution_system
ac_emi
technical_system-ventilation specialisations
technical_system
ventilation_control_system
ventilation_generation_system
ribution_system
technical_system-entity attributes
technical_component
space
string
building
space_technical_component
space_function
technical_system
has_parts_technical_component
id_name
is_used_by_building S
is_used_by_space S
has_parts_space_technical_component
is_used_for S
technical_system-Entity Definition

```
/*updates_start(60).
updates_end. */

ENTITY technical_system
 SUPERTYPE OF ( ONEOF ( lighting_system, district_heating_system, hw_hs_storage,
hw_hs_generator, hw_hs_emission, hw_hs_distribution, hw_hs_control, dhw_storage_system,
dhw_generation_system, dhw_emission_system, dhw_distribution_system, dhw_control_system,
ac_generation_system, ac_emission_system, ac_distribution_system, ac_control_system,
ventilation_generation_system, ventilation_emission_system,
ventilation_distribution_system, ventilation_control_system ) ) ;
   satisfies : SET OF technical_function ;
   has_parts_technical_component : technical_component ;
   has_parts_space_technical_component : space_technical_component ;
   is_used_for : SET OF space_function ;
   is_used_by_space : SET OF space ;
   has_efficiency : efficiency ;
   is_used_by_building : SET OF building ;
   system_type : REAL ;
   maximum_power : REAL ;
   id_name : STRING ;
END_ENTITY ;
```

Fig. 2. Modelling the IDM in a multi-view environment

The development of the IDM is an arduous process requiring collaboration and communication between the various domain experts as well as the integrators of the design tools. Computerised modelling systems currently in use provide little high level modelling support to the developers of the model. Often these systems can provide only one view of the entities being modelled using only one modelling paradigm, or very limited consistency maintenance between multiple views of the same entity. To provide a full modelling environment for the specification of the IDM the authors have specialised the MViews and SPE modelling environment [5] to provide a modelling environment for the EXPRESS lexical and EXPRESS-G graphical modelling languages [6] (see Fig. 2 for several graphical and textual views of a *technical_system* entity). This environment, EPE [7], allows multiple textual and graphical views of an entity to be specified. Overlapping information in each of these views is kept consistent through a canonical form of the model which is used to propagate all changes to one entity to all views which reference it. The modelling environment automatically makes modifications to affected graphical views and allows both automatic and user assisted modifications to the textual views. EPE maintains a central log of all modifications to all entities which can be annotated to provide development documentation. Navigation between any view referencing a particular entity is supported to allow easy access to the various design level views of an entity.

Working as a companion to EPE is a similar system for the manipulation of instances of the model. This tool, Cerno [8], allows multiple views of the instance data to be visualised in a format specified through a display language. Cerno provides for instance debugging and modification in multiple views, again with consistency propagated through to all views.

Project model

This is a project specific model defining the designers who will be working on a project, their roles in the project and the design functions required to fulfil those roles. The project model can also be used to define the flow of control in a project, specifying hand-over points between various designers and sequences of specification that must be followed to help guarantee an optimal design at the termination of the design process. As the design process is not a fully understood process it can not always be completely described at the start of a project. Therefore, it is likely that during the course of the design, new designers may need to be involved in the design or new design tools used for difficult aspects of the design. To enable this flexibility the project model must be able to be modified during the project to be able to take into account changes in personnel and new flows of control.

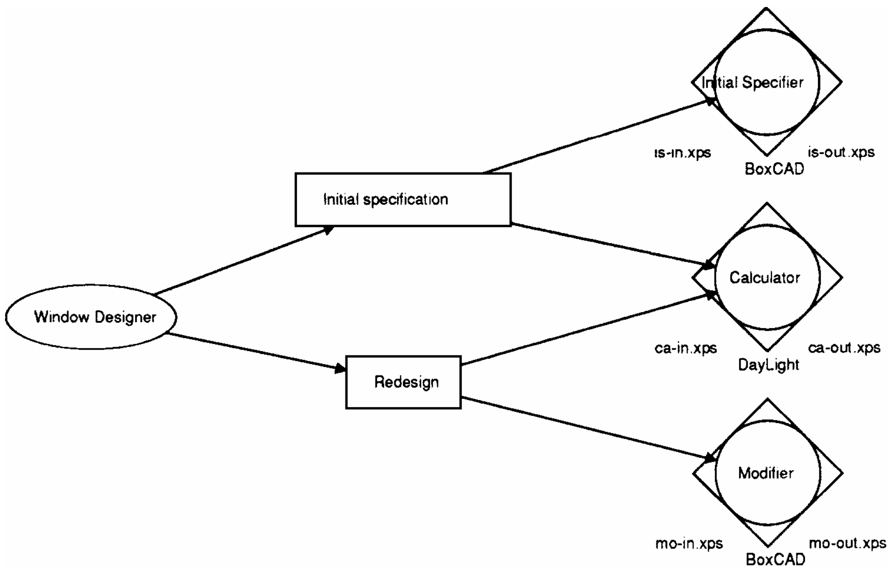


Fig. 3. Specifying actors, design roles, and design functions in a project

A graphical formalism to specify the designers, their design roles, and the design functions required to perform that role, was developed in the project (see Fig. 3 for a small specification). In this formalism the links between designers and their design roles are made explicit, as well as the link between design role and the design functions they require. In this formalism each design function is connected to a particular design tool and the schema representing the input and output to the design tool are specified.

The design functions are used in the specification of the flow of control in the project through another graphical formalism drawn from ideas in Petri-Nets [9]. Flow of control is depicted as moving from one design function to another through a set of transition points at which time the flow of control can branch to various design functions (see Fig 4. for a simple flow of control specification). This specification allows for aggregate and global functions (not shown in the figure) as well as looping transitions (shown as a double bar). Through the use of many layers of flow of control specification the whole flow of control for a portion of the design task, called a project window, can be specified. With this specification a portion of the control for a design task can be simulated.

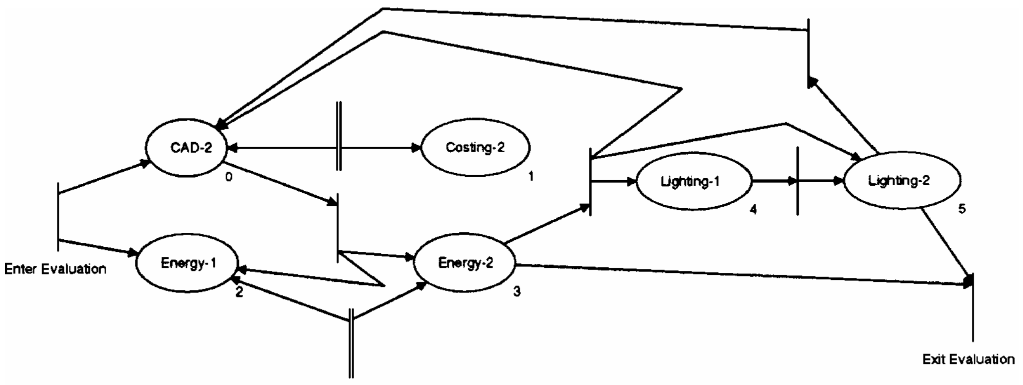


Fig. 4. Specifying flow of control

DT models

Each of these models defines the structures and data requirements of a design tool. The type of design tool and its capabilities will determine the number of models that are required for each tool. In most cases two models are used, one to specify the data input to the design tool, the second to specify the data output from the design tool. The creation of these models is a task undertaken by modellers who have great familiarity with the particular design tool being modelled. This is because the model must capture not only the data structures explicitly defined by the tool and the data constraints specified by the design tool (eg. ranges on attribute values, cardinality constraints on lists, etc.), it must also contain the constraints defined implicitly by the design tool (eg. that all walls are vertical). A fully specified DT model allows the system to determine whether there is enough data to create the input file for the DT to execute and to determine whether a semantically correct description of the building can be created from the data available. The DT model remains a static model throughout the use of the same version of the DT.

The specification of design tool models is performed with the same tools as used for the IDM, ie. the EPE environment for specifying EXPRESS models and the Cerno tool for examining and modifying the instances of input and output data from the design tools. The views of a model can optionally be shown in Snart form, or translated from EXPRESS into Snart, using SPE notation, as shown in Fig. 5.

Inter model mapping

This module performs the translation of data between the IDM and the DT models. It must determine whether it is possible to create a consistent model from the IDM for any one of the DT models based upon the constraints in that DT model, and it must ensure that the IDM remains consistent upon update from a design tool's output. It must detect conflicts between various sets of data and must be able to invoke processes to enable negotiation over conflicting data from different design tools and users.

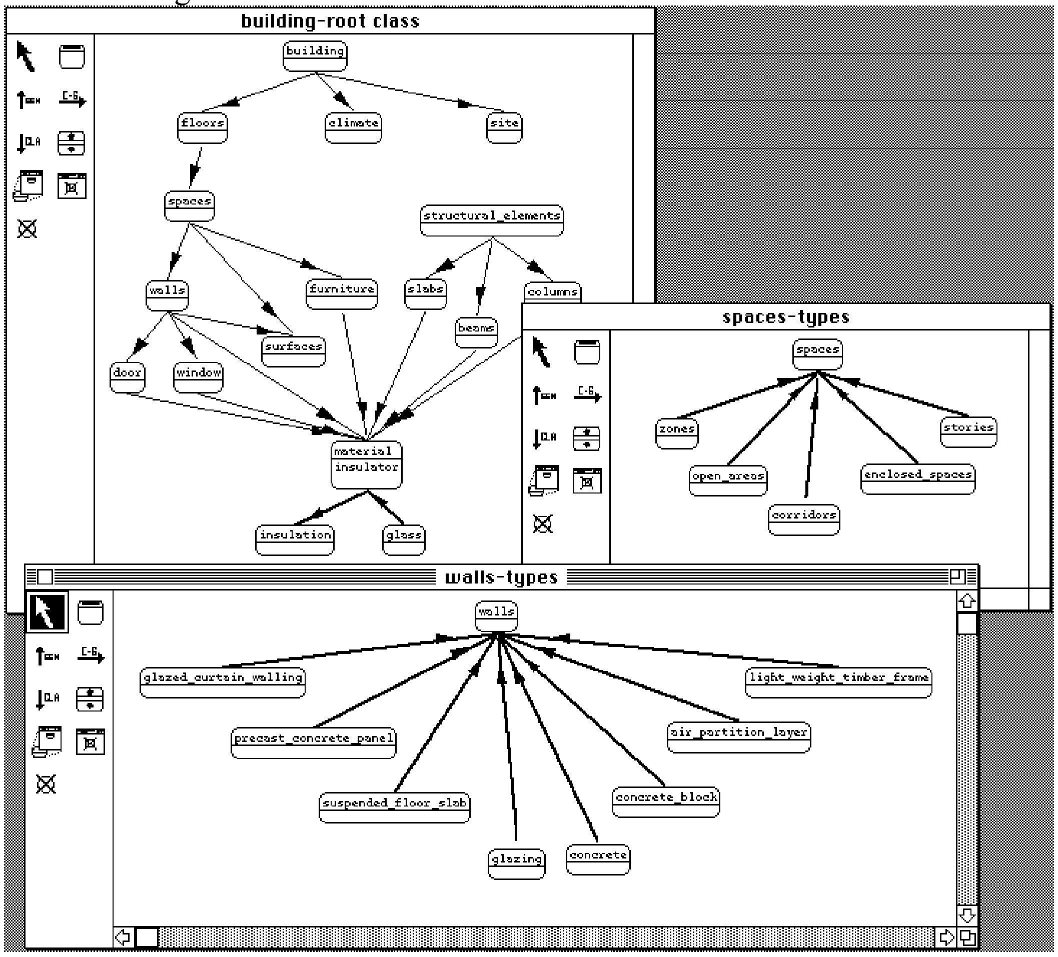


Fig. 5. Modelling a users view of a building

To specify the correspondences between the IDM and the various design tools it is attached to, a new declarative mapping language was developed. The view mapping language, VML [10], has both a lexical (see Fig. 6 for a simple mapping between classes) and a graphical (see Fig. 7 for the equivalent graphical form) notation for specifying a mapping. Mappings are specified through a set of *inter_class* definitions (as in Fig. 6) which specify: the classes involved with the mapping; the conditions under which the mapping may take place (invariants); and the attribute mappings that can be applied. Mappings are specified through the use of equations, which can be used in both directions, functions and procedural code. Where procedural code is used two mappings must be specified to be able to move data between both views.

```
inter_class([trombe_wall],[trombe_wall, trombe_type],
      invariants(trombe_wall.trombe_type = name),
      equivalences(area = height * width,
                  glazing = glazing,
                  vent_area = sum(vents=>(height * width)),
                  trombe_type = trombe_type,
                  perf_ratio = perf_ratio)
      ).
```

Fig. 6. Mapping specification in VML

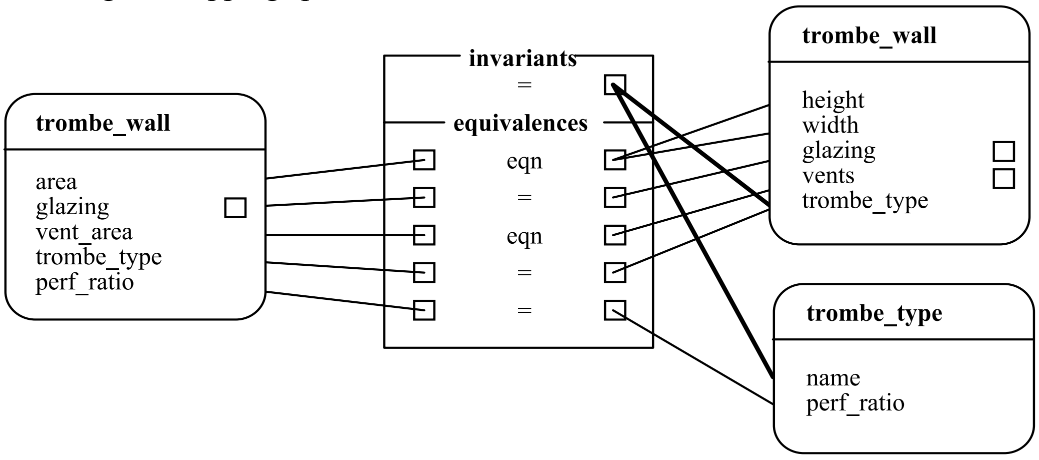


Fig. 7. Graphical specification of a VML mapping

A mapping system has also been developed which can connect two views through the use of a VML specification. This mapping system uses a transaction based approach to receive notifications from attached views of a set of modifications to be mapped to the other view. The mapping system takes the set of modifications and uses the *inter_class* specifications to determine whether to create new objects, delete objects, or to calculate modifications to object values in the other view.

Design tool interface

This module provides the connection between the design tools utilised in the system and their data models. For an off-line DT this module must be able to create the input files required for the DT (from the DT model in the system). It must then be able to invoke the DT to perform its work and finally retrieve the output from the DT into the DT model in the system. For interactive DT's it must perform the same task but driven by the demands of the design tool. Traditionally this module is implemented by hand-coding the parsing and pretty-printing for each new design tool. In future integrated design systems, and for some classes of

DT's, it is expected that this module will be realised by modelling the interaction required by each DT (for example the data file structures) and using this model to drive the interaction with the DT.

Control system

This directs both the inter-model mapping and design tool interface modules to perform their tasks as required by the designers or directed by the project manager. This module simulates the flow of control defined in the project model, determining what design functions are able to be performed at any particular time. It also interfaces with designers to let them specify the tasks that they wish to perform next and interfaces with the project manager for decisions upon which designers should be involved in new phases of the project. This system directs the inter-model mapping module to map data between different models, or to ascertain whether it is possible to perform the mapping. It also directs the design tool interface to invoke a design tool with appropriate data and to collect results upon termination.

The interface the control system provides to the project controller is similar to that shown for the specification of flow of control (see Fig. 4). It is enhanced through the annotation of place markers to show the current state of the project as well as calculations of the possible design functions that can be invoked at that particular time.

3. Conclusions

This paper describes a framework which enables the integration of a range of disparate design tools from a single domain. The various modules necessary to implement tasks such as flow of control, and mapping of data between design tools and the central model are specified. Where modelling tasks are necessary in the specification of the integrated system, high level modelling tools were created and modelling formalisms developed where existing formalisms were not suitable for the task.

An implementation of all the modules in the described framework gives a system which allows multiple users to work concurrently on a single design, yet still maintain data consistency between each other. In contrast to other workers' efforts, which require hand-coding to tie all the pieces together, the implemented system highlights the ability to create a working system through high-level modelling of data models, correspondences between data models, and the required flow of control in a project.

Acknowledgment

The first author acknowledges the support of the University of Auckland Research Committee for their travel grant and the University of Auckland for their PhD scholarship.

References

[1] Augenbroe G, COMBINE: A Joint European Project Towards Integrated Building Design Systems, *Symposium on Building Systems Automation - Integration, A/E/C Systems 92*, Dallas, Texas, USA, 10-12 June, In Building Systems Automation-Integration, August, 1993, University of Wisconsin-Madison, (1992) 731-744.

[2] Böhms HM and Storer G, Architecture, methodology and Tools for computer integrated LArge Scale engineering (ATLAS) - Methodology for Open Systems Integration, ESPRIT 7280, Technical report, TNO, Delft, The Netherlands, (1993).

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

[4] ISO/TC184, Part 1: Overview and fundamental principles in Industrial automation systems and integration - Product data representation and exchange, Draft International Standard, ISO DIS 10303-1, ISO-IEC, Geneva, (1993).

[5] Grundy JC, Hosking JG, Fenwick S, and Mugridge WB, Chapter 11, *Visual Object-Oriented Programming*, Burnett M, Goldberg A, Lewis T Eds, Manning/Prentice-Hall, (1994).

[6] ISO/TC184, Part 11: The EXPRESS Language Reference Manual in Industrial automation systems and integration - Product data representation and exchange, Draft International Standard, ISO-IEC, Geneva, Switzerland, ISO DIS 10303-11, August, (1992).

[7] Amor R, Augenbroe G, Hosking J, Rombouts W and Grundy J, Directions in Modelling Environments, accepted for publication in *Automation in Construction*, (1995).

[8] Fenwick S, Hosking JG and Mugridge WB, Cerno-II: A program visualisation system, University of Auckland, Department of Computer Science, Report No 87, February, (1994).

[9] Jensen K, Coloured Petri Nets: A High Level Language for System Design and Analysis, Advances in Petri Nets 1990, *Lecture Notes in Computer Science*, (1990).

[10] Amor R, A Mapping Language for Views, Department of Computer Science, University of Auckland, Internal Report, (1994) 30p.