ICAtect-II: A Framework for the Integration of Building Design Tools

Robert Amor, John G. Hosking, and Warwick B. Mugridge
Department of Computer Science
University of Auckland

ABSTRACT

The development of a system capable of integrating a range of building design tools poses many challenges. Our framework for integrating design tools provides a structured approach, allowing individual parts to be tackled independently. In this paper we detail the framework and its individual components. A method for modelling and implementing each component is described, showing how such an integrated system can be realised. To illustrate, a system developed using the framework and which integrates several design tools is described.

1. Introduction

It can be difficult to effectively use a set of disparate design tools in architecture and engineering due to a lack of integration. Even with a computerised plan of a building, it is unlikely that, using current practice, building information can be extracted from it for use in other design tools. Practitioners are typically forced to re-enter building information into their design tools by hand, taking measurements off the printed plans. This means that despite the existence of a wide range of technically excellent design tools, such tools are seldom used because of the difficulty and time required to enter the requisite building information. In addition, changes to the design have to be transferred individually to each of the tools, increasing the possibility of inconsistencies between them. As a result, the majority of claims made for computerisation and integration in these fields have never been achieved.

These incompatibility problems have a number of causes, but result primarily from the conceptual building models involved. Many tools lack soundly based conceptual models. Others utilise building models oriented heavily to the types of computation they perform. In most cases, this is a minimal model, in order to ease the burden of data entry for the user. Hence models of buildings for tools such as thermal simulation, structural simulation, lighting analysis, fire-code compliance, quantity surveying, etc. have very different structures, and hence very different data entry requirements.

Another 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. 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.

In this paper we present ICAtect-II, an approach, together with a set of tools, that permit the development of integrated building design environments which alow building data to be shared between multiple tools, and multiple building design professionals, alleviating the problems noted above. Section 2 introduces the integration approach we have taken. Sections 3 to 6 describe tools and methods we have developed for implementing various components of an integrated design system. Section 7 descibes an example design system developed using ICAtect-II. Section 8 describes related work, and Section 9 presents conclusions.

2. An Integrated Data Model (IDM) Approach

ICAtect-II's approach to solving the problem of data sharing is to have a central Integrated Data Model (IDM) describing all of the relevant data needed for a building. Instances of this model then correspond to individual building designs. To apply a design tool to a building design, an appropriate translation of the data is made from the IDM instance into the form of data that the tool requires (see Fig. 1). Data generated or modified by the tool is also translated back into changes to the IDM instance.

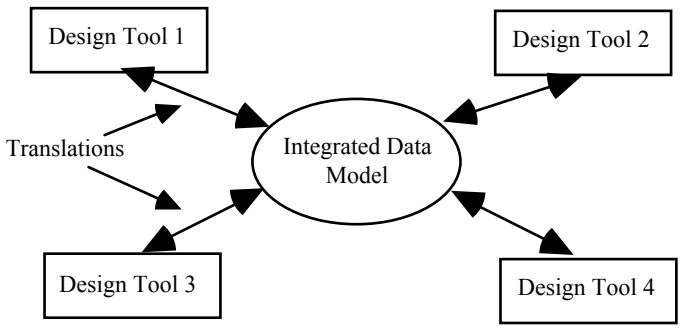

Figure 1 General IDM/Tool/Translation approach

This approach is a common one in integrated building design research (eg [1], [2]). Our variant, based on our earlier ICAtect work [3, 4], is a refinement of the approach of Figure 1, as illustrated in Figure 2. In our approach a control system draws upon a 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.

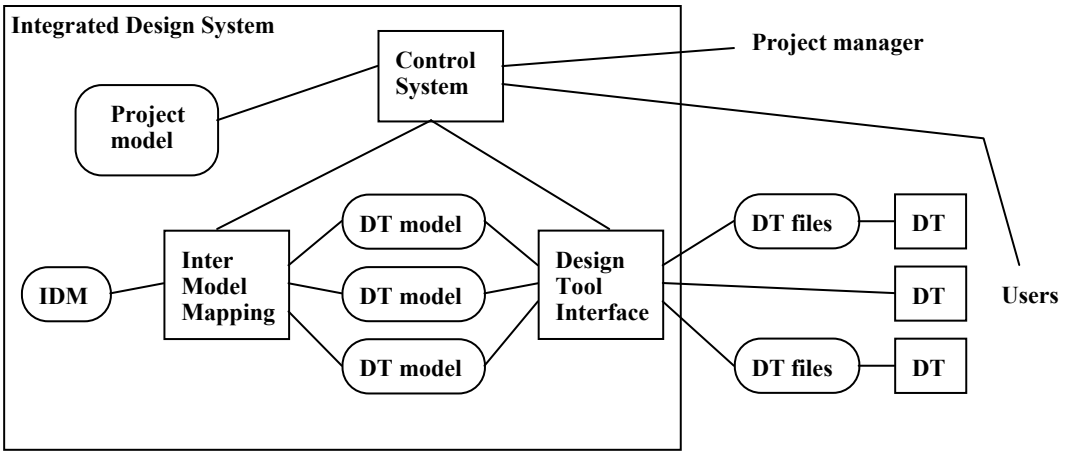

Figure 2 Refined integration model

To support this IDM-based approach, a number of tools and components are needed. These include:

- A notation for defining information models (usually called a schema definition language) for the IDM and design tools.
- Techniques for defining information models (schema) for the data needs of each of the design tools and users of the system.
- Techniques for synthesising the IDM schema from the various user and tool schemas.
- A notation for defining the translations (inter-model mappings) between the IDM schema and the user or design tool schemas.
- A means of specifying mappings from the conceptual design tool models to the physical data requirements of the design tools (the design tool interface)
- Tools to create instances of the IDM and DT models (ie actual building designs).
- Tools to move data between the IDM instances and the design tools based on the inter-model mapping and design tool interface specifications.
- A notation and tool to specify project models.
- A control system to co-ordinate user activity and tool invocation based on a project model.

ICAtect-II comprises a suite of tools covering these requirements. Figure 3 outlines the tool suite and their inter-relationship. The following sections describe the tools and their use in detail.

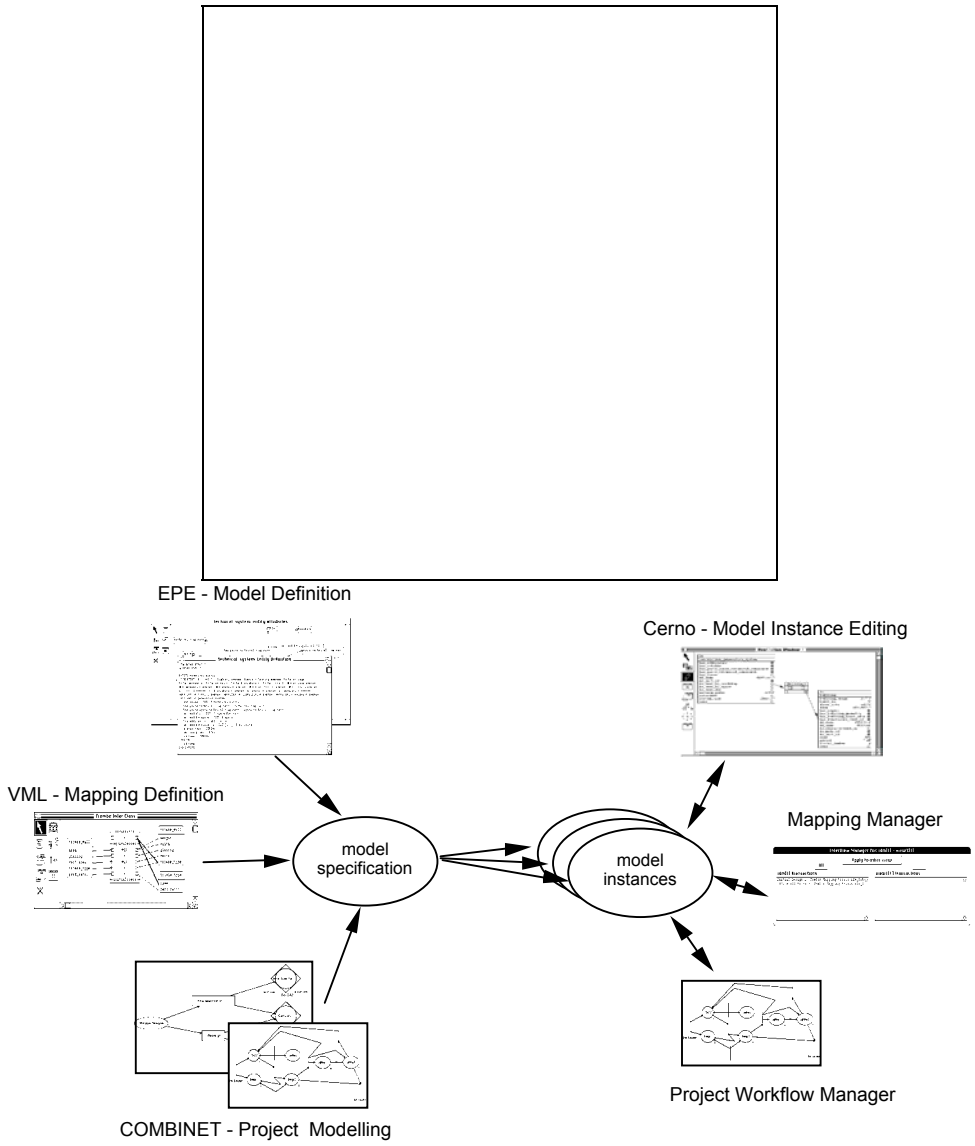

Figure 3. Tools used in constructing and manipulating integrated building models in ICAtect-II

3. Specification of Schemas

The conceptual model required for completely specifying building design information is probably the largest and most complex information model under active development in the world to date. A primary focus of this work has been the development of the ISO STEP standard [5]. This has been a massive collaborative effort, which has had an impact on, or has incorporated results from, all of the major integration projects. In particular, as part of the STEP work the EXPRESS object-oriented specification language has been developed and adopted for use as a schema specification notation [6]. For compatibility purposes, we have used EXPRESS to specify user**,** DT and IDM schemas in our work.

The creation of design tool data models is a task that must be undertaken by modellers who have great familiarity with the particular design tool being modelled. This is because the model must capture: the data structures explicitly defined by the tool; the data constraints explicitly specified by the design tool (eg. ranges on attribute values,

cardinality constraints on lists, etc.); and also 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 (or other type of interface) for the DT to execute and to determine whether a semantically correct description of the building can be created from the data available.

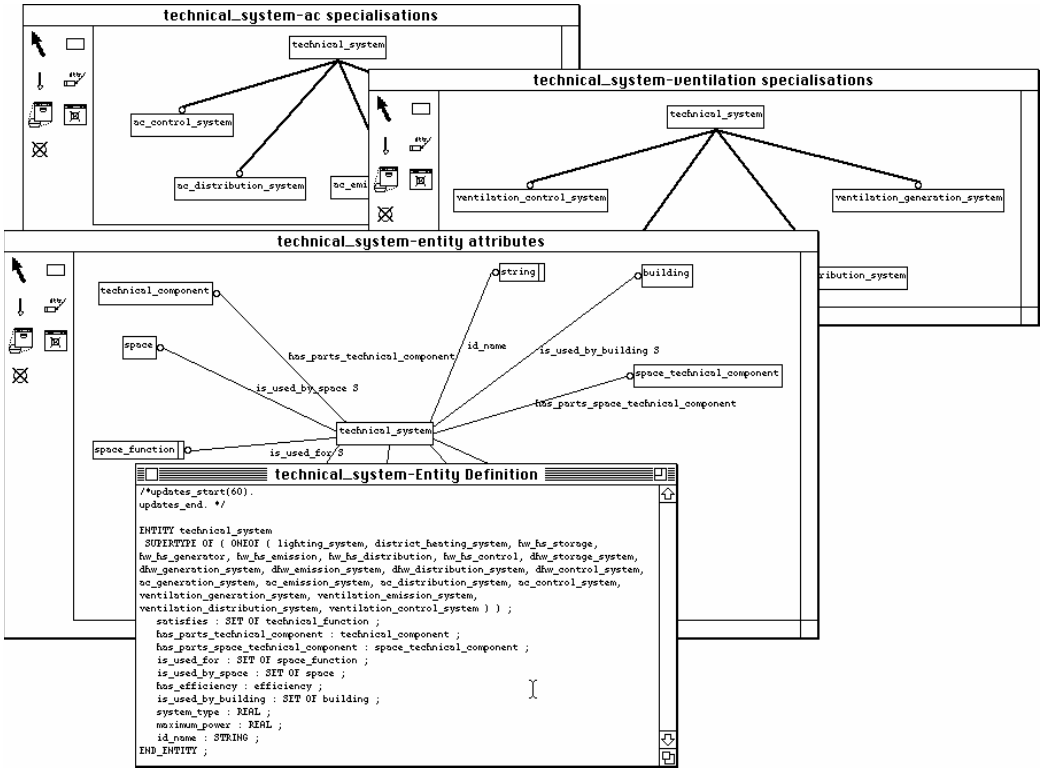

Fig. 4. Modelling the IDM in a multi-view environment

The DT model is developed as an EXPRESS schema for the tool. Concurrently, a simple translator is defined which maps from the EXPRESS schema to the actual format of data that the tool requires, such as a flat file (the Design Tool Interface of Fig. 2). Despite the widespread use of EXPRESS in integrated projects, there has been little development of tools that fully support EXPRESS modelling and programming. To remedy this, we have developed the Express Programming Environment (EPE), for the EXPRESS lexical and EXPRESS-G graphical modelling languages [6]. EPE permits multiple textual and graphical views of EXPRESS entities to be specified [7]. 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 an 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 to maintain view consistency. 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. Figure 4 shows several graphical and textual views of a *technical_system* entity modelled with EPE.

EPE supports the translation of the EXPRESS schema to and from Snart, an object-oriented implementation language [7, 8]. EPE is, in fact, a specialised implementation

of the SPE modelling environment, which provides visual and textual programming support for Snart [9][1]. The Snart translation facilities allow realisation of the EXPRESS schema in an executable form. The realised schema may also be extended with additional Snart code (method implementations, additional attributes, etc). This provides a means of defining the simple design tool interface, or, alternatively, EPE can be used as a complete development environment for designing and implementing the design tools themselves, using Snart as the implementation language.

EPE is also used to specify the IDM schema. The development of this schema 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, although the introduction of new design tools, construction techniques and building constructs force modifications to the model. Several collaborative projects have worked on general models of buildings, based on the ISO STEP fundamentals for restricted aspects of buildings [6 1, 2], which could be used directly in our work. These models are, however, preliminary in nature. To demonstrate the viability of our techniques, we have instead developed a simple IDM incorporating the requirements of a small number of design tools. This has been developed in an evolutionary manner, with the model being incrementally extended as each new tool is integrated into the evolving system [11].

4. Specification of Mappings Between Schemas

The specification of inter-model mappings is an area which has, until quite recently, been neglected by most researchers. In the majority of international integration projects, translations have been done using conventional procedural programming. This can lead to obscure and difficult to maintain code, as well as presenting problems of maintainability due to schema evolution. One of the important contributions of our work has been the definition of a high level declarative language, VML, for describing these translations and which avoids many of the practical problems encountered using conventional programming techniques [12, 13].

VML has both a lexical and a graphical notation for specifying a mapping. Fig. 5 shows a simple example mapping between classes, with Fig. 6 showing the equivalent graphical form. Mappings are specified through a set of *inter_class* definitions (as in Fig. 5) which specify: the classes involved with the mapping; the conditions under which the mapping may take place (invariants); and the attribute and procedure mappings that can be applied (equivalences).

For example, in Figure 5, the *trombe_wall* class of one schema and the *trombe_wall* and *trombe_type* classes of a second schema are involved. The invariant specifies that this mapping is only applicable when the *trombe_type* attribute of a *trombe_wall* object (second schema) has the same vaue as the *name* attribute of a *trombe_type* object (second schema**).** Equivalences are specified through the use of equations, which can be used in both directions, invertible functions and procedural code. Where procedural code is used two mappings must be specified to be able to move data between both views. Procedure calls (methods) may also be mapped by specifying the procedure

[1]SPE has also been extended to support NIAM modelling with translation to Snart [10]. NIAM is also used extensively in STEP-related modelling work.

name and any parameter values required as part of the equivalence. Chains of references can be specified with the => operator allowing the mapping to access values from other referenced objects. Thus, the *sum(vents=>(height * width))* expression in Fig. 5 sums the product of *height* and *width* for every vent object referenced in the *vents* (list) attribute of *trombe_wall*. The graphical notation is not as expressive as the lexical notation, and is analagous to the EXPRESS-G graphical variant of EXPRESS.

```
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. 5. Mapping specification in VML

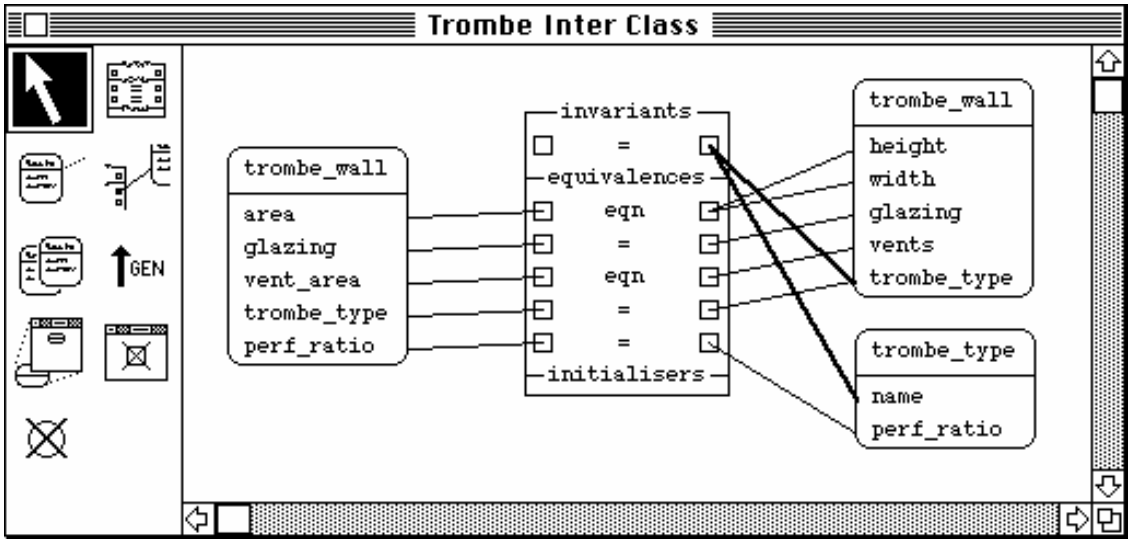

Fig. 6. Graphical specification of a VML mapping using VPE

Taken as a whole, the collection of inter-class definitions relating two schemas defines their overall relationship. The inter-class invariants determine which particular mappings are relevant for a given dataset.

The VML Programming Environment (VPE) supports specification of VML mappings using schemas developed by EPE or imported from external sources. Mappings may be developed using either the lexical or graphical forms and consistency is maintained between the two forms. Figure 6 is a screen dump of the VPE graphical specification browser/editor.

VPE can also be used as an aid in defining and extending the schemas involved in the mappings, thus assisting with the task of schema integration. For example, during the mapping definition users can specify classes, and attributes of classes, not currently existing in the schemas being manipulated. The schemas are then modified accordingly.

In addition to its use in specifying DT-IDM mappings, VML can also be used to describe mappings between different versions of a schema (DT or IDM). The resulting mappings can then be used to migrate datasets between versions.

5. Model Realisation and Instance Management

Having developed the design tools, schemas, mappings, and design tool interfaces, tools are required to both create model instances and to move data between them accordinging to the mappings. To support these activities, ICAtect-II includes the following tools:

• The Snart translation facilities of EPE are used to realise schemas as executable object-oriented programs.

• Model instances are constructed either directly by design tool invocations or by using the Cerno-II visualisation tool, originally developed as a companion for SPE [9, 14]. Models may also be imported to and exported from ICAtect systems in the form of STEP data transfer format files. Internally, Snart supports object persistence and thus provides object-oriented database facilities for storing model data.

• Model instances may be visualised using Cerno-II, which permits multiple views of a model instance to be displayed and edited, provides navigation support to follow object references, and allows custom visualisations to be developed using a user-editable display specification language. A screen dump of Cerno visualising a small portion of an object model is shown in Fig. 7. Alternatively, instances may be browsed using the InSTEP building model visualisation tool developed as part of the COMBINE project [7].

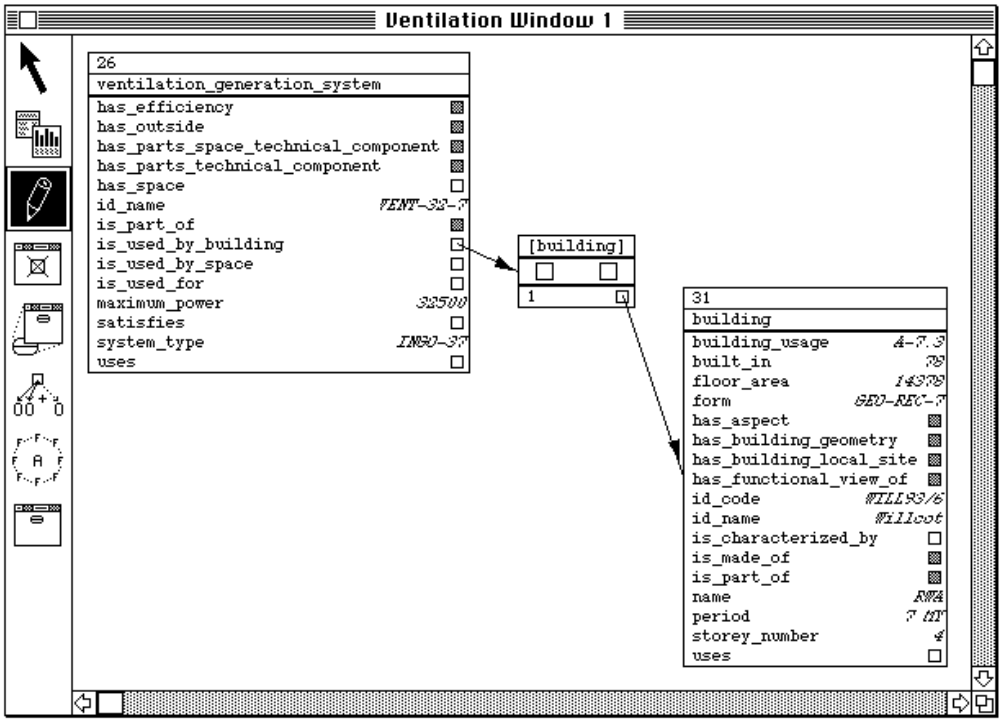

Fig. 7. Cerno-II visualising a small portion of a building instance.

• The underlying Snart system also provides a query language which allows a data model to be searched for objects matching a general query term, together with an object viewer, which can be used to provide a lower level data visualisation facility than that provided by Cerno-II or InSTEP.

• A mapping manager supports incremental or batch transfer of data between instances of the IDM and instances of tool models (controlled by the VML mapping specifications), ensuring consistency of the IDM at all times [12, 13]. The manager 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 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. The size of the transaction depends on the mode of operation of the tool. In coarse mode, transactions are large, and hence similar to a batch mode of operation. In fine mode, tranactions are small, corresponding to individual tool manipulations (eg adding a window in a plan package), providing a more incremental approach to consistency. The mapping manager also detects conflicts between various sets of data, invoking processes to negotiate over conflicting data from different design tools and users. A further important function of the manager is to maintain tracability, so that the originators or modifiers of data can be determined at any time. This can be used in the conflict arbitration process.

Combined, these facilities provide the low level "glue" for integrating the various design tools at execution time.

6. Project Management

The instance management facilities described in the previous section provide low level co-ordination between tool invocations and building designers. However, there is also a need for higher level project management tools to assist in the specification of design team responsibilities and to manage the flow of work between the designers and design tools. To provide these capabilities, we have developed, in collaboration with the COMBINE project [1], the COMBINET project modelling system. This comprises a specification tool, together with an execution time control system. The specification tool provides two interlinked modelling formalisms**.** One of the formalisms allows users to specify flows of control in a project by means of ordering constraints on design tasks. This is used in conjunction with the other formalism which models the designers and the roles they take in performing design tasks. As the design process is often not fully understood, it cannot 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 is able to be modified during the project to be able to take into account changes in personnel and new flows of control

A *project role model* defines the designers who will be working on a project, their roles in the project and the design functions required to fulfil those roles. Fig. 8 shows a small project role model, illustrating the graphical notation developed. 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. Each design function is connected to a particular design tool and the schema representing the input and output to the design tool are specified.

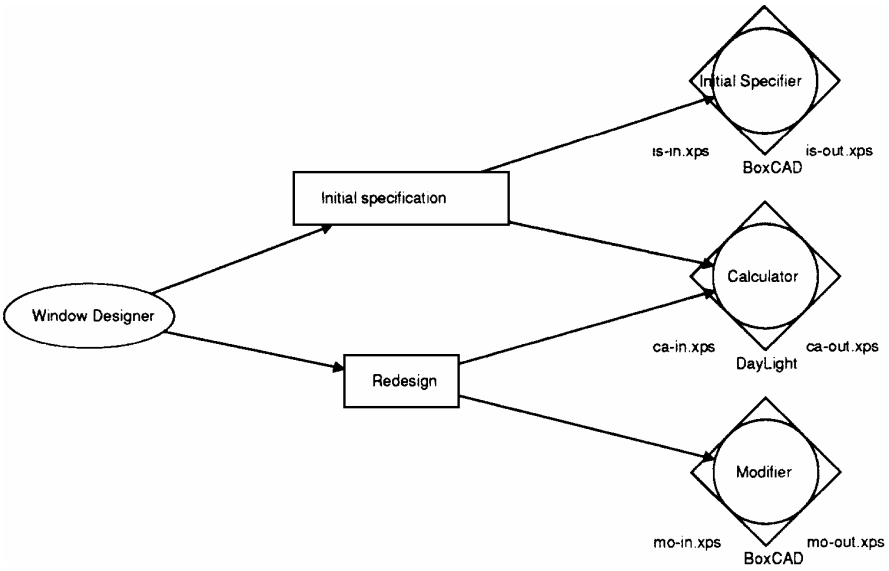

Fig. 8. Specifying actors, design roles, and design functions in a project

A *workflow model* is 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. This uses another graphical notation drawn from ideas in Petri-Nets [15]. 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 (Fig 9. shows 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). Workflow models may be hierarchically organised in *project windows*, to permit modularisation of the model.

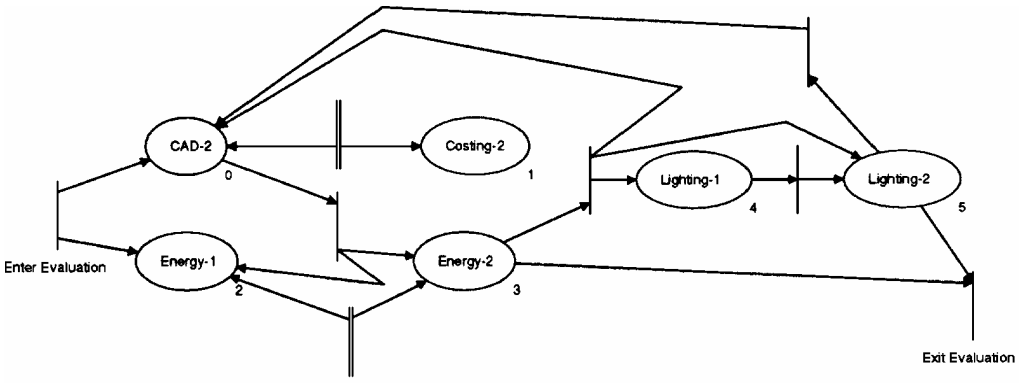

Fig. 9. Specifying flow of control

The *Exchange Executive* is the runtime control system that 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 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.

7. A Demonstration Integrated System

In order to test the value of this system, and to extend our work, we have implemented a demonstration integrated system using several tools that we have developed or had available to us. These tools are:

- ThermalDesigner, a code conformance tool [16, 17]. This calculates the thermal resistance of the elements of a building, heat losses, and heat gains, in order to check conformance of a building against a thermal insulation standard for residential buildings. ThermalDesigner was originally developed as a Kea application [16], but was recently reimplemented in Snart [17].

- PlanEntry, a novel constraint-based plan drawing system [18, 19], also implemented in Snart. This provide multiple orthogonal views of a building. A change in any view is reflected in the other views. PlanEntry permits definition of the building envelope, internal walls and openings.

- MaterialsEditor, another Snart application, for specifying materials and bracing information [20] for wall planes. Walls and other faces on a plane are provided to this tool, and the user may overlay wall, roof or floor material information on that plane.

- VISION3D, a package for rendering and manipulating 3-D models [21]. This has been used purely as a visualisation tool in this work, permitting 3-D visualisations of the building models developed by the other design tools.

The integrated system will later include WallBrace [22], another code conformance tool (for checking wall bracing requirements of a loadings code) that we have previously developed. The overall system design is shown in Fig. 10.

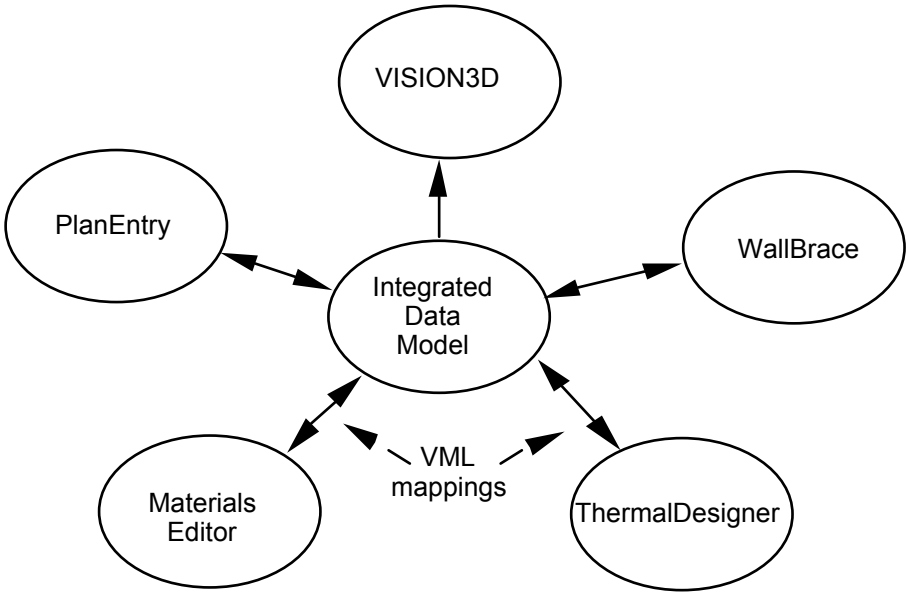

Figure 10. Demonstration System Design

Schema for the three Snart-based tools were developed by directly translating the Snart object definitions for their building models into EXPRESS using EPE's translation

facilities. The Snart implementation of these tools also obviated the need for a design tool interface to be separately developed for them. Rather, the VML mappings defined for translating the IDM to the design tool models represented all of the information necessary to directly create or otherwise manipulate Snart objects belonging to the design tools. VISION-3D is capable of a variety of data entry formats. For this application we chose a flat file description. An EXPRESS schema conceptually representing the fairly simple file format was developed, together with a Snart translator to translate from the realised schema objects into the concrete file format.

The IDM for this demonstration system was developed in an evolutionary manner, being extended as each tool was integrated [11]. Schema realisation resulted in a Snart object model for the IDM. Concurrently, VML code for the DT-IDM inter-model mappings was developed. This exposed a number of weaknesses in the DT and IDM schema, resulting in incremental improvement of both.

A very simple project model for the system was also developed, with two designers in the project role model (one responsible for the PlanEntry and MaterialsEditor, and the other responsible for ThermalDesigner) and a simple workflow model.

A user's view of the integrated system is shown in Fig. 11. This is a composite view showing all of the facilities in use specifying and visualising charactersitics of a small residential building. Changes in any component are propagated through the IDM to other, affected components. For example, the changes resulting from resizing a space in the PlanEntry model (label 1 in the Figure) are propagated to the IDM, and then to the MaterialsEditor (label 2), VISION3D (label 3), and, eventually, to ThermalDesigner (label 4). Such changes can be propagated incrementally, ie as each change is made, or can be batched together to reflect a complete design modification where a number of changes are propagated at once. Change propagation is handled using the transaction model described earlier and a user specific management tool (label 5).

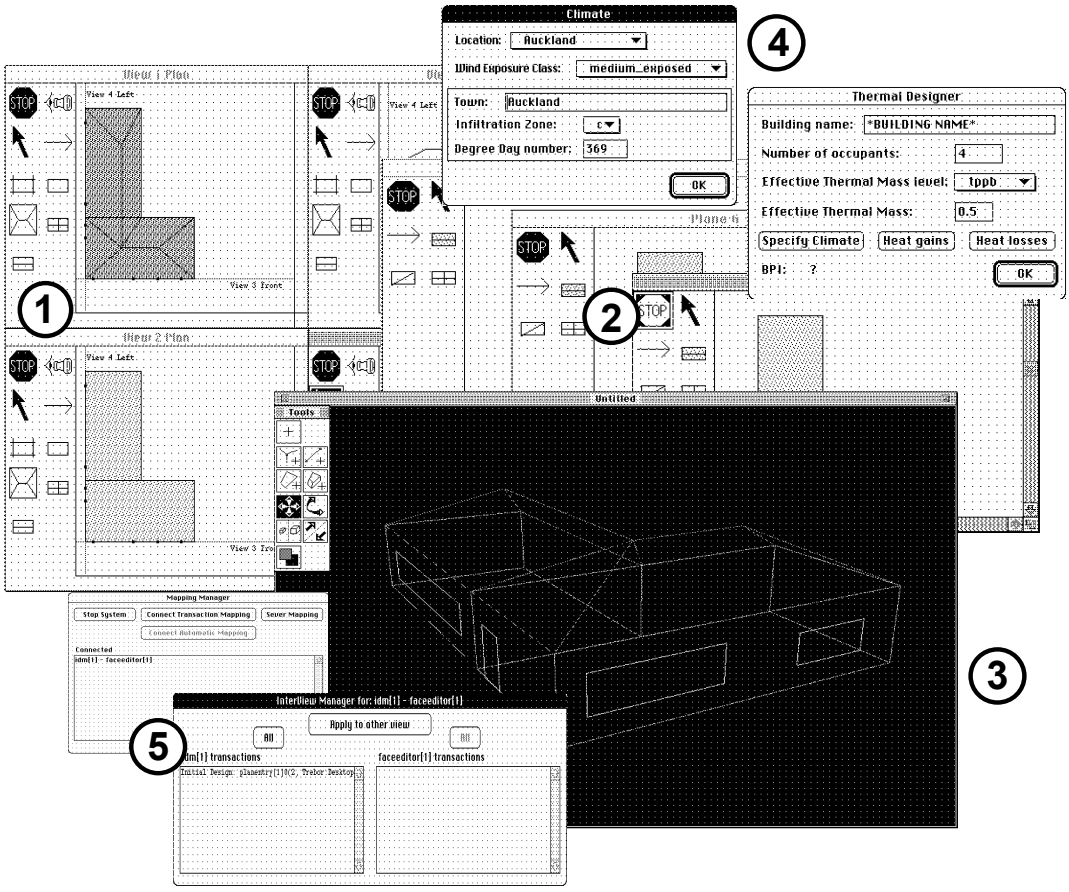

Figure 11. Example of the demonstration system in use

8. Related Work

Much research has been carried out on the schemas for integrated design models [1, 2, 5, 23, 24]. However**,** little attention has been given to the design and construction of software architectures that are suitable for managing an IDM; for example, [1] discusses several tools to help manage this process at its completion. At the design level, our work is novel in providing an integrated toolset covering the complete design and implementation of an integrated system.

At the implementation level, our work is also unusual in that we have started with the assumption that data mappings between tools will be most useful when they are dynamic, rather than necessarily assuming a batch mode of operation. We have also made extensive use of a constraint-based language (Snart) for the development of design tools that are integrated through an IDM, simplifying the task of building new tools that are to be integrated. While our IDM architecture supports such a dynamic model, it also allows for traditional, procedure-based tools, which must explicitly manage changes to data values, and may be required to act in a batch-like manner.

The work that is closest at the implementation level to the research reported here is that of Eastman [25, 26] in developing the EDM-2 system. EDM-2 is organised around a database model. It also makes use of mappings and constraints, but there are several major differences from our work:

• Constraints are only used in the declaration of the IDM. They are defined for two purposes: as database-style constraints to determine whether a changed data model is consistent with respect to the constraints (to ensure across-tool consistency) and as functions which compute new values when data values change. There are two forms of consistency constraints: variants need not be satisfied (at the users discretion) while invariant constraints must be satisfied, otherwise a commit is aborted.

• EDM-2 constraints and mappings are written in C and thus do not enjoy the declarative benefits of Snart constraints and VML's equivalences.

• EDM-2 includes "accumulations" to allow for external code to compute values. When any of the preconditions of the external code are changed, the results (post-conditions) of the external code in the IDM are marked as inconsistent. Hence other applications know that data is invalid at the moment and any affected external code can be automatically scheduled for execution (assuming that only one tool is available for computing the given result). This approach is functional, as compared to the more general mappings provided by VML.

9. Conclusions and Future Work

We have described a framework which enables the integration of a range of disparate design tools from a single domain. It incorporates various modules that are necessary to implement tasks such as flow of control and mapping of data between design tools and the central model. Where modelling tasks were found to be necessary in the specification of the integrated system, but where existing formalisms were unsuitable, we created high level modelling tools and modelling formalisms for such tasks.

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 extensive 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.

The small illustration system presented demonstrates the viability of our approach. Current work aims to extend the number of design tools in this system, to demonstrate scaling of the architecture.

The process of constructing the demonstration system led to two observations. First, the provision of an integrated data model, plus a simple method of defining and implementing mappings, encourages the development of small, specialised design tools, such as the MaterialsEditor. These interact directly with the central model, and, by change propagation, the other design tools. This approach is in contrast to existing practice, which sees individual design tools being extended to encompass additional functionality, in the process becoming difficult to maintain.

The second observation regards the implementation of individual design tools when assuming an IDM-based architecture. As remarked earlier, the ThermalDesigner application was rewritten in Snart prior to integration. The main purpose of this was to see the effect having an integrated data model available made to the development of an individual design tool. In this case, it meant almost complete elimination of ThermalDesigner's procedural code which, in the original implementation was primarily

used to drive the data gathering in an appropriate sequence. The user interface code (forms definitions etc) was also almost entirely eliminated as the data sought was often better sourced from other design tools, such as PlanEntry, via the IDM. The net result is that the new design tool is much simpler in structure, and hence easier to maintain. Also the declarative nature of the code conformance "rules" is much more obvious than in the original application.

One drawback of the current ICAtect implementation is its reliance on Snart. While Snart has proved to be a very useful vehicle for proof of concept implementation, it's reliance on MacProlog for its own implementation has two drawbacks. First, the accessibility of our work to others is diminished as a result of using this platform; portability is a problem. Second, the double interpretation involved results in a perfomance penalty that will limit the ultimate scalability of the existing implementation. For these reasons we are re-implementing the major runtime components of our system in Java and plan to port the design components at a later stage.

Acknowledgments

The authors acknowledge the financial support of the Building Research Association of New Zealand, the New Zealand Foundation for Research Science and Technology, the University of Auckland Research Committee, and the TU Delft COMBINE Group in performing the research described in this article. The first author acknowledges the support of a University of Auckland PhD Scholarship.

References

[1] G. Augenbroe, 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] H.M. Böhms and G. Storer, 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] R. Amor, ICAtect: Integrating Design Tools for Preliminary Architectural Design, MSc thesis, Department of Computer Science, Victoria University of Wellington, Wellington, New Zealand (1990).

[4] R.W. Amor, L.J. Groves and M.R. Donn, Integrating Design Tools for Building Design, ASHRAE Symposium on Artificial Intelligence in Building Design: Progress and Promise, *ASHRAE Transactions*, Vol. 96, Part 2 (1990) 501-507.

[5] A. Boyle, STEP: An introduction, Technical Paper CIPM/LU/TP/1, Computer-Aided Engineering Group, Department of Civil Engineering, University of Leeds (1992).

[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] R. Amor, G. Augenbroe, J. Hosking, W. Rombouts and J. Grundy, Directions in Modelling Environments, *Automation in Construction*, 4 (1995) 173-187.

[8] W.B. Mugridge, J. Grundy, R. Amor and J. Hosking, Snart94 User and reference manual, Report, Department of Computer Science, University of Auckland (1995).

[9] J.C. Grundy, J.G. Hosking, S. Fenwick and W.B. Mugridge, Chapter 11, *Visual Object-Oriented Programming*, M. Burnett, A. Goldberg and T. Lewis (Eds), Manning/Prentice-Hall (1994).

[10] J.R. Venable and J.C. Grundy, Integrating and Supporting Entity Relationship and Object Role Models. *Proceedings of the 14th Object-Oriented and Entity Relationship Modelling Conferece*, LNCS 1021, Springer-Verlag, Gold Coast, Australia (December 1995).

[11] W.B. Mugridge and J.G. Hosking, Towards a lazy evolutionary common building model, *Building and Environment,* 30(1) (1995) 99-114.

[12] R.W. Amor and J.G. Hosking, Mappings: the glue in an integrated system, In R.J. Scherer (Ed), *Product and process modelling in the building industry,* Rotterdam, The Netherlands, A.A. Balkema Publishers (1995) 117-123.

[13] R.W. Amor, J.G. Hosking and W.B. Mugridge, A declarative approach to inter-schema mappings, Modelling of Buildings Through Their Life-Cycle, Proc. CIB W78/TG10 Conference, *CIB Proceedings Publication 180,* Stanford (August 1995) 223-232.

[14] S. Fenwick, J. Hosking and W. Mugridge, A visualization system for object-oriented programs, in C. Mingins and B. Meyer (Eds), *Technology of object-oriented languages and systems TOOLS 15*, Prentice Hall, Sydney (1994) 93-103.

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

[16] R.W. Amor, J.G. Hosking, W.B. Mugridge, J. Hamer and M. Williams, ThermalDesigner: an application of an object-oriented code conformance architecture, in D. Vanier and R. Thomas (Eds), Joint CIB Workshops on Computers and Information in Construction, International Council for Building Research Studies and Documentation, *CIB Proceedings Publication 165,* Montreal, Canada (1992) 1-11.

[17] W.B. Mugridge, J.G. Hosking and R.W. Amor, Adding a code conformance tool to an integrated building design environment, Auckland Uniservices Ltd (May 1996).

[18] J.G. Hosking, W.B. Mugridge and S. Blackmore, Objects and constraints: a constraint based approach to plan drawing, in C. Mingins and B. Meyer (Eds),

Technology of object-oriented languages and systems TOOLS 15, Prentice Hall, Sydney (1994) 9-19.

[19] J.G. Hosking and W.B. Mugridge, A constraint based building model editor, Auckland Uniservices Ltd (June 1994) 28pp.

[20] J.G. Hosking, W.B. Mugridge and R.W. Amor, An integrated building design environment, Auckland Uniservices Ltd (June 1995).

[21] P.D. Bourke,VISION-3D User Manual, School of Architecture, University of Auckland, Auckland, New Zealand (1989).

[22] W.B. Mugridge and J.G. Hosking, The development of an expert system for wall bracing design. *Proc. NZES'88 The third New Zealand Expert Systems Conference*, Wellington (1988) 10-27.

[23] B-C. Björk and H. Pentilla, A scenario for the development and implementation of a building product model standard, Current Research and Development in Integrated Design Construction and Facility Management, CIFE, Stanford CA, 28-29 March (1989) 18pp.

[24] W.L. Carroll, S.E. Selkowitz and F.C. Winklemann, The energy design advisor: a new desktop design tool based on DOE-2, a white paper for discussion and comment, Lawrence Berkeley Laboratory, USA (1991).

[25] C.M. Eastman, M.S. Cho, T.S. Jeng and H.H. Assall, A data model and database supporting integrity management, in *International Computing Congress in Civil Engineering*, ACSE, Atlanta, Georgia (June 1995).

[26] C.M. Eastman, Managing integrity in design information flows, to appear in *Computer Aided Design*, IPC Press (1995).