MAPPINGS FOR INTEGRATING DESIGN TOOLS

Robert Amor and Dr John Hosking
Department of Computer Science
University of Auckland
Private Bag 92019
Auckland
New Zealand

ABSTRACT

The ability to integrate a range of disparate design tools has been an area of intense research throughout the world. One of the hardest tasks in this research has been to define the correspondences between the model of a building used by a design tool and the integrated data model. This paper presents a method of defining mappings between different models of a building. The implementation of a mapping management system which controls the flow of data between the varíous models and guarantees the consistency of the models is also described.

INTRODUCTION

An *integrated simulation environment* coordinates interaction between multiple users and simulation tools, permitting numerous types of simulation to be performed on a single model of a building. In most approaches to such environments a central model of a building holds all information required by the set of interacting design tools. Combined with this model is a management system permitting users to enter information on the building, run various simulations and display the results of the simulations. One of the functions of the management system is to transfer relevant building data to simulation tools and then to extract the data resulting from simulation runs for the user to analyze. An environment to accomplish this will be examined in this paper.

Transferring data back and forth between a central model and a design tool can be a very complicated procedure because of the disparate building models used by the various participants in the exchange. The central model will be very general, with data on a building spread out through many levels of abstraction and in many different classes. In contrast, the model of a building used by a particular design tool will be very specific, with most data on a particular object aggregated into one place (see Figure 1). The two models may also have quite different notions of the structure of a building depending upon what is required for the analysis. Herein lies the crux of the problem. In order for the integrated system to function properly it must be able to transform the data in one model through to the form required in another (and vice versa) without loss of information content and assuring that the models are internally consistent.

Most integrated simulation environments handle the mapping of data between a simulation tool and their central model through specialised internal programs (written in a conventional procedural language) which can extract the required information and manipulate it into the form required by the other side (Augenbroe 1992; Böhms and Storer 1993; ISO/TC184 1993). More recently there have been projects which have looked at formalising the specification of the mapping through the use of specialised mapping languages (Bailey 1994; Hardwick 1994; Clark 1992). This makes the addition of a new simulation tool a much easier task and makes explicit many of the correspondences between the models which would otherwise be lost in coded versions. However, these new languages are still basically procedural in nature, providing a very low level view of the correspondence between two schemata. Also, where the mapping is to be bi-directional, it is necessary to specify two separate mapping definitions, one for each direction.

In the remainder of this paper we describe VML, a new declarative language for the specification of mappings between various models, and an implementation of the language which is capable of mapping data between various design tool models and manages the consistency of the data in all models.

A VIEW MAPPING LANGUAGE (VML)

VML (Amor, 1994; Amor and Hosking, 1994), is a declarative high level language for describing correspondences between structures in schemata. Unlike procedural mapping languages, VML mapping specifications can be automatically applied in either direction. The language has both graphical and textual forms, and is supported by a specification environment that permits rapid construction of mapping definitions in both forms.

To specify the mapping between two schema using VML requires the definition of a collection of *inter-class* descriptions such as the one shown in Figure 2.

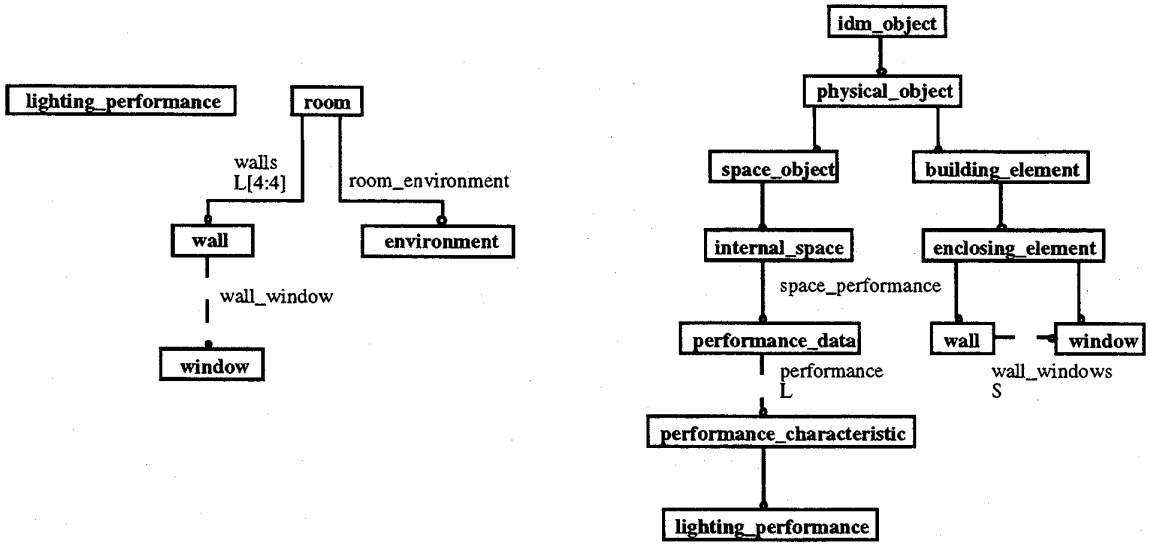

Figure 1. A specific and generalised model of a similar domain

```
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)
      ).
```

Figure 2. A textual mapping specification

Each inter-class description defines classes in each of the two schema which are involved in a particular mapping. The schema involved will have previously been defined in EXPRESS (ISO/TC184 1992) using a tool such as EPE (Amor et al 1995), but the mappings may require extension of these schema as described below.

For example, in Figure 2, the `trombe_wall` class of one schema and the `trombe_wall` and `trombe_type` classes of a second schema are involved. The mapping specifies the conditions under which that mapping may take place through an *invariant* definition, and specifies the equivalences between attributes, relationships and objects in the specified classes. Equivalences can take three different forms: a specification as a mathematical expression; a functional specification; and a procedural specification for those mappings which are not able to be defined in the previous forms. Mathematical expressions and functions can be automatically utilised when mapping in either direction, while procedural specifications must be hand-coded for application in each direction.

The specification environment will try to reconcile all attribute references from a textual mapping specification, but may need user intervention to determine the exact class and attribute being specified. For example, in Figure 2 the invariant definition is not completely explicit as the attribute reference `trombe_wall.trombe_type` is not a unique reference, as the `trombe_type` attribute occurs in both `trombe_wall` classes.

Figure 3 shows the graphical version of the inter-class definition of Figure 2. The graphical notation has icons to denote classes which display the class name and its attributes, both simple and aggregate types. The inter_class definition is also represented iconically. Direct equivalences between attributes are very easy to denote and, unlike the textual definition, the class or attribute referenced in a mapping is totally explicit, as is illustrated by the graphical representation of the invariant.

The mathematical equations supported in VML cover a large range of mathematical functions as well as specialised functions of use in schemata. Figure 2

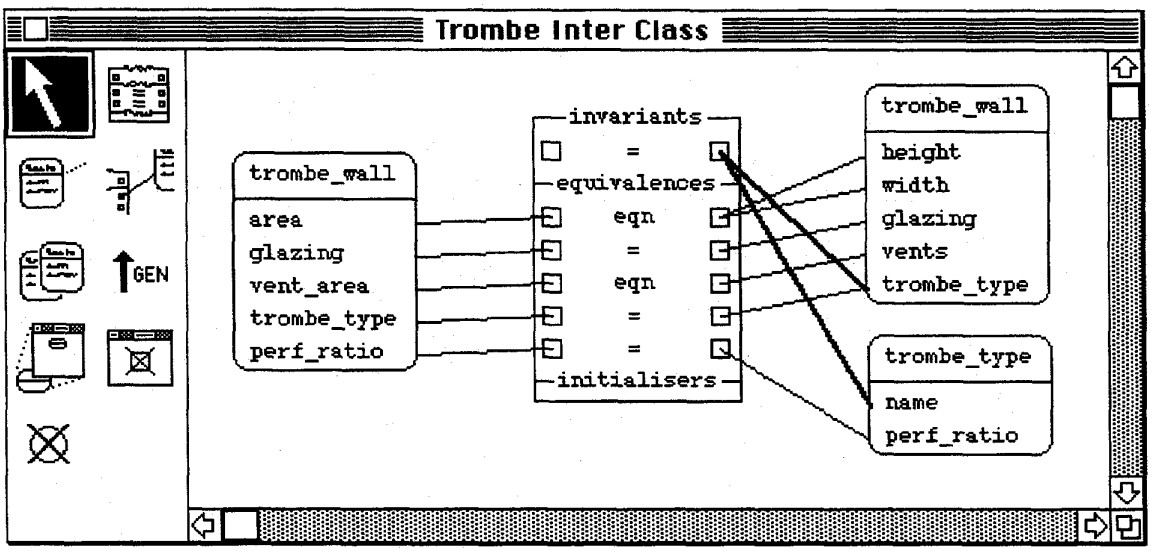

Figure 3. A graphical mapping specification

shows some of the mapping specific functions supported. In particular, chains of references can be specified with the => operator to allow the mapping to access values through relationships to other objects. There is also a full range of summary functions (ie. sum, average, minimum, maximum, count) which range over a list of values or references. In Figure 2 the specification `sum(vents=>(height * width))` sums the product of height and width for every vent object referenced in the vents attribute of *trombe_wall*. In VML the = operator is overloaded to provide a short-hand method of specifying equivalence between attributes which contain object references. For example, in Figure 2 the equivalence between *glazing* attributes refers to attributes whose type is a list of glazing element. The equivalence denotes that for each glazing element referenced in one glazing attribute, the other glazing attribute should have the object ID of the glazing element that was created in the mapping specified by an inter_class for glazing classes.

The mapping specification environment can also be used as an aid in defining and extending the central model. This is achieved through the provision of facilities for assisting with the task of schema integration. For example, during the mapping definition the user is able to specify classes and attributes of classes which do not currently exist in the schemata being manipulated. When such additions are noted the schemata may be modified to incorporate the newly referenced classes and attributes. In a similar manner the environment can also be used to describe mappings between different versions of a schema and the datasets which are associated with each version.

Taken as a whole, the collection of inter-class definitions relating two schemata define the overall relationship between the two schemata, with the inter-class invariants determining which particular mappings are relevant for a given dataset.

A MAPPING MANAGEMENT ENVIRONMENT

A *mapping manager* provides an execution time interpretation of VML mappings. This manager (developed in a Prolog enhanced with object-oriented extensions, Hosking et al 1994) takes the VML descriptions of the correspondences between two schemata of a building and ensures the correct data is transferred between the two models whenever either is modified. To achieve this the mapping manager needs the ability to detect changes in either model and, using the VML descriptions, determine which data to move between the connected datasets. Rather than transferring changes incrementally between models (although this would be possible), a transaction based modification system is used. A transaction denotes a portion of work completed in one view (where a view is one user's model of a building defined using the structures specified in a schema). For example, a transaction could be the initial layout of a building, or the results of a simulation, or the results of actioning a change request.

A transaction handler helps maintain the consistency of the integrated system by managing the sets of outstanding transactions between connected views. Transactions which have not been mapped to a connected view are made available to the user to map across to the other view. Figure 4 shows the transaction handler for two views connected by a

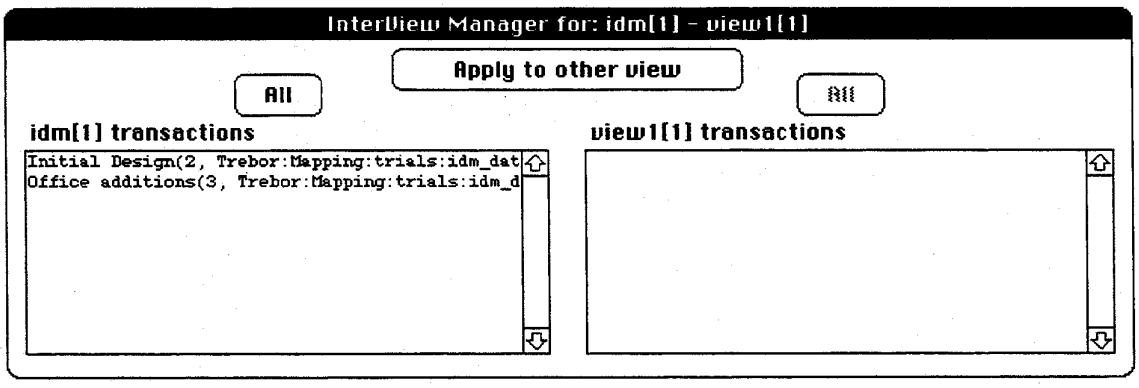

Figure 4. Transaction management

mapping where there are two transaction to be mapped to *view1*.

When linking an integrated model (which may be shared with many design tools and users) with an individual user's view, the transaction manager prevents the user applying transactions to the integrated model before all outstanding transactions on the integrated model have been applied to the user's model (though not vice-versa). This allows a user to make modifications and experiment within their view, but ensures that the user's view is consistent with the global model before the local modifications can be passed to the global model.

When the user selects a set of transactions to be applied to a view or integrated model, the set of changes made to objects in the integrated model or modified view are collated together and passed to the mapping system to apply. What the mapping system can do with each of these changes is determined by the type of change that is being made to each object. For example:

- Where the change is the creation of an object with values for attributes, the type of the new object is used to identify inter-class mappings which might be applied. The mapping system will cycle through these selected inter-class mappings and determine whether new objects should be created in the view being mapped to (by checking whether the invariants are satisfied). If an inter-class mapping can be applied, the mapping manager will use the equivalences to instantiate values for attributes of the newly created objects (in the cases where this is possible).
- Where the change is a modification to an existing object's attributes the affected data is used to determine which of the existing mappings is affected by the changed data. These affected mappings are re-examined, checking that the invariants still hold and re-evaluating the equivalences involved with the modified attributes.
- Where the change is a deletion of an object the affected inter-class mappings are examined to determine whether it is necessary to delete objects in the view being mapped to.

To ensure that the mapping system can efficiently propagate changes between views it collects information from every mapping that is applied determining all objects in both views which take part in the mapping. By collating this information the mapping system can immediately identify every inter-class mapping and equation which must be re-evaluated whenever an object is modified or deleted.

To determine the values for attributes of an object involved in an equation, the mapping system has to handle the three types of equivalence that can be specified, as well as making use of invariants which explicitly define a value for attributes. For example:

- Given a mathematical expression, the mapping system will re-arrange the equation to solve for an unknown attribute (or one which should be updated). If the equation can be re-arranged and there is only one unknown attribute then the re-arranged equation will be evaluated and the attribute set to the calculated value.
- Given a function, the mapping system will call the function with all known values and instantiate attributes with any values calculated by the function.
- Given a procedure, the mapping system will just invoke it and allow the procedure to determine all values for attributes it wants to change.

When creating new objects through an inter-class definition the mapping system can use invariants which are defined on the objects being created to define values for attributes of the new objects.

The mapping system tracks who or what asserts values for attributes to help determine what values can be calculated from an equation. Data from different sources has a worth ranking given to it. For example a user specified value has much higher precedence than a default value. This enables the equation solver to re-evaluate equations which have previously been calculated to solve for the attribute with the lowest precedence (eg. default values) and catches the overwriting of one user's data by another user. The latter is treated as a conflict in the model and notified separately to the users involved to allow them to negotiate a solution.

In some cases it is not possible to re-arrange an equation to solve for particular attributes, for example, in Figure 2 if *area* is specified in a newly created *trombe_wall* it is not possible to determine *width* and *height* for a *trombe_wall* in the other schema. In this case the value of *area* is used to constrain the values of *height* and *width* by asserting a constraint against the product of these attributes to equal the value of *area*. If at a later stage one of these attributes gets set, then the other can be calculated, or if both are specified then their product must equal the value of *area*.

As can be seen from the description above, the mapping manager is a complex system that must operate at many levels to ensure consistency between views of a building connected by a mapping specification. However, by handling this complexity in the mapping manager it is possible to considerably reduce the amount of effort required when describing the mappings in VML in comparison to that required when using procedural code.

CONCLUSIONS

We have presented a framework for modelling correspondences between views of a similar domain and have outlined a system that implements the specified correspondences. The declarative language, VML, was presented as a method of detailing the correspondences between two schemata of a building. The ability to specify correspondences between schemata is enhanced through the use of a specification environment which supports both the textual and graphical notations of the language. The actual mapping of data between views specified through a mapping definition is handled by a mapping support system. This system uses a transaction based approach to view modifications to determine which parts of the mapping specification to apply to the modified data. The mapping system can determine when to create or delete objects in the view being mapped to and which equations to re-apply when a particular attribute value is modified. The final system guarantees that data from any one view can be mapped through to all connected views to maintain the global consistency of the integrated system.

REFERENCES

Amor, R., Augenbroe, G., Hosking, J., Rombouts, W. and J. Grundy, "Directions in Modelling Environments", accepted for publication in Automation in Construction, 1995

Amor, R. "A Mapping Language for Views", Department of Computer Science, University of Auckland, Internal Report, 30p, 1994

Amor, R. and J. Hosking, "Mappings: The Glue in an Integrated System", The First European Conference on Product and Process Modelling in the Building Industry, Dresden, Germany, 5-7 October, 1994.

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, pp 731-744, 1992.

Bailey, I., "EXPRESS-M Reference Manual", Product Data Representation and Exchange, ISO TC184/SC4/WG5 N51, 66p, 1994.

Böhms, H.M. 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.

Clark, S.N., "Transformr: A Prototype STEP Exchange File Migration Tool", National PDES Testbed Report Series, NISTIR 4944, US Department of Commerce, National Institute of Standards and Technology, 13p, 1992.

Hardwick, M., "Towards Integrated Product Databases Using Views", Technical Report 94003, Rensselaer Polytechnic Institute, Troy, New York, USA, 18p, 1994.

Hosking, J.G., Mugridge, W.B., and S. Blackmore, "Objects and constraints: a constraint based approach to plan drawing", in Mingins, C. and B. Meyer, Technology of object-oriented languages and systems TOOLS 15, Prentice Hall, Sydney, pp 9-19, 1994.

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.

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, 1993.