Product and Process Modelling in the Building Industry, Scherer (ed.) © 1995 Balkema, Rotterdam. ISBN 90 5410 584 8

Mappings: The glue in an integrated system

Robert W. Amor & John Hosking
Department of Computer Science, University of Auckland, New Zealand

ABSTRACT: We describe a new high level mapping language which assists in solving the problems of schema evolution, schema integration, multiple perspectives of models, and frameworks for integrated systems. In contrast to both existing methods and the procedural mapping languages currently being developed, this language uses a declarative style with graphical and textual notations to allow users to specify more easily the equivalences between entities in various schemata. The mapping language implementation is capable of propagating modifications from one view to all dependent views, ensuring consistency of information at all times across all views.

1 INTRODUCTION

During the last five to ten years considerable research effort has been directed to the development of integrated environments for architectural and engineering applications. In that time much emphasis has been placed on the development of a representation of buildings and the implementation environment for such a representation to act as the repository for all information in the integrated system. This work has resulted in significant contributions to the fields of schema modelling languages (ISO 1991b) and collaborative development environments (Boyle & Watson 1993). However, other components of the integrated frameworks have received less research effort.

The area that our work is concerned with is the mapping of information back and forth between the central store of building information and the stores for the various design tools. This may appear to be a straight-forward problem, as in many cases the central schema has been partially derived from, or at least checked for coverage by comparison with, a range of existing design tools. However, even when design tool schemata have been used to help derive the central model, the correspondences between data in the design tool model and the central schema are often not retained or formally modelled during the integration process. Also, when using a set of design tools to check the coverage of a central model it is possible to ascertain that the required information is modelled without specifying all of the correspondences between the design tool and the central model.

Thus developers find there is a gap between the central model and the design tool models which must be bridged in some manner. In working integrated systems this is achieved by hand coding translators to transfer information from one model to another, building in all the assumptions and constraints which are implicit in the various models (Augenbroe 1992; Böhms & Storer 1993; ISO 1991a). These translators work well provided the semantics of the two models are well understood by the developer of the translator. Problems arise when new tools are to be integrated into the system. The addition of a new design tool may force the extension or restructuring of the central model to accommodate the information and structures modelled by the new design tool, possibly violating assumptions used in constructing existing translators, requiring their revision. Integrating a new design tool which can modify its input data, such as an interactive design tool, a design tool which allows parametric simulations, or a knowledge-based system, requires writing mappings which map the same set of data in both directions, duplicating effort and allowing greater opportunity for inadvertent programming errors.

For these reasons we believe it is imperative to model the correspondences, constraints and implicit semantics of various models using a high level mapping language. Use of such a language allows the integrated system to cope better with the addition of new design tools as well as modifications to the existing models and their mapping definitions. In the remainder of this paper we examine the place of a mapping system in an integrated environment and its use both to integrate design tools and to support system and schema evolution.

2 STRUCTURE OF AN INTEGRATED SYSTEM

ICAtect (Amor et al. 1993) provides a framework for design tool integration. Its approach is very similar to that of the majority of integration projects being

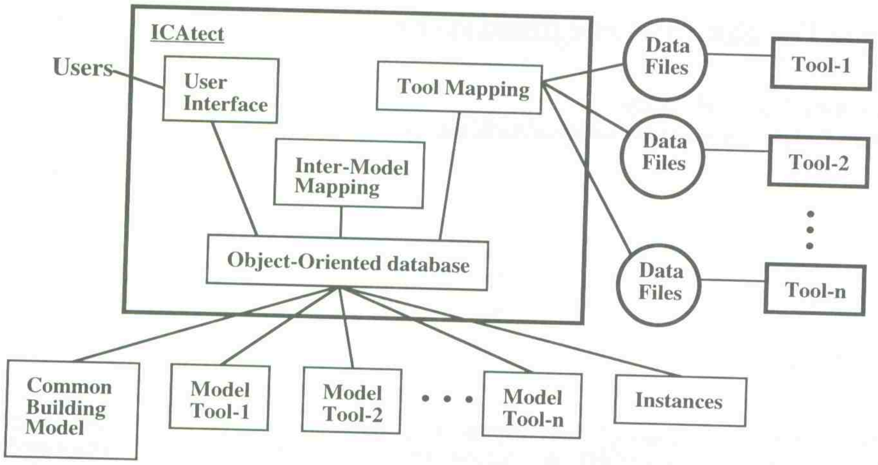

Figure 1. The structure of ICAtect.

undertaken; its broad structure is shown in Figure 1. In this framework, the common building model (CBM) is the central schema, representing canonically the union of the information requirements of each of the attached design tools. Schemata representing information needs of each individual tool are represented in the same object-oriented language (Grundy 1993) as the common building model and allow the capture and evaluation of the structure and constraints imposed by the various design tools inside the integrated environment. The user interface permits the user to both control the invocation of various tools and to peruse or modify the data in the system.

In ICAtect the mapping process is broken into two stages. One is concerned with mapping data between the different models in the system, the other with mapping from the internal model of a tool's data to the actual form used by the tool. The second case is a purely one-to-one mapping and is concerned mainly with the straightforward tasks of parsing and "unparsing". The inter-model mapping system handles a very difficult task and is the one we concentrate on in this paper.

Major tasks that the inter-model mapping system must be capable of performing include:

1. Building a new model from an existing store of data. To achieve this the system must apply all the mappings between the schema of the existing data and the new schema to determine the objects and data required in the second model.

2. Capturing any updates to a model and propagating to all dependent models. Any change to the data in a model, or any invocation of a method is captured. Appropriate modifications to dependent models can then be made by checking against the mappings applicable to the model in which the change was captured.

3. Ensuring that propagated data still maintains the consistency of the model to which it is propagated. If there is a contradiction between the data being propagated and the constraints in the model to which it is being propagated, then it must be identified and handled in an appropriate manner.

To support these requirements, we have developed a view mapping language (VML, see Amor 1994) and support environment which allows for the specification of correspondences between schemata. These specifications can then be used by the integrated system to provide multiple views of the CBM.

3 INTEGRATING DESIGN TOOLS WITH VML

Integration of a new design tool to an existing system commences with the definition of the design tool's schema. Then follows the definition of a mapping specification between the design tool schema and that of the central model.

The major task at this point is to provide support to the developer in describing correspondences between the design tool's schema and the CBM. This has an impact in two areas of the system development. First, the language for describing mappings needs to be well suited to describing correspondences between structures, attributes and relationships between classes in two different schemata. Second, to manage the complexity of large schemata and large mapping definitions it is necessary to have a language support environment which manages and facilitates the definition of mappings. These two points are further discussed below.

To focus the discussion, we consider the integration of a simple design tool, the EXPRESS-G schema of which is shown in Figure 2, with a demonstration CBM derived from the COMBINE-1 project (Augenbroe 1992), of which a portion of the schema is shown in Figure 3.

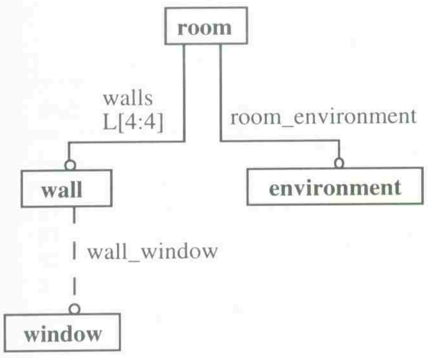

Figure 2. Design tool schema.

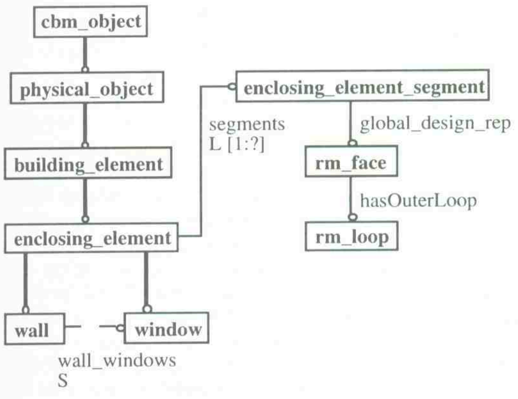

Figure 3. Partial CBM schema.

3.1 *VML: a mapping language*

The development of a new language is generally a task to be avoided (given the broad range of existing languages), unless there is a very good reason for the development time required. In this case we based our decision to create a new language on the lack of languages well suited to the problem domain: the description of correspondences between schemata. Existing approaches (as in the use of C, C++, EXPRESS-M (Bailey 1994), Transformr (Clark 1992), etc.) use a procedural approach which means that programmers must concentrate on the low level mechanics of *how* the conversion is to be done, rather than describing *what* the equivalences are. Also, a common requirement is the ability to map the same set of data in both directions. To be able to achieve this in a procedural language requires two mapping specifications. A more declarative approach solves both of these problems.

For these reasons we have developed VML, a declarative language for specifying mappings. The language has one main construct, the *inter_class* definition, used to describe correspondences between classes in two schemata. This description consists of three parts, as illustrated in Figure 4. First is the definition of the class (or classes) from each schema which is to take part in the mapping. In the example we show a mapping between the wall classes of the two schemata. Second is the definition of *invariants* which limit when this particular mapping may be used. In the example we see that there are limits on the cardinality of windows and segments in a wall. Third is the definition of *equivalences* between attributes and relationships in the various classes. In the example we see simple equivalences between attributes and relationships from the two classes. Each of these parts is described in more detail below.

3.1.1 *Classes in a mapping*

The class specification section of the *inter_class* definition describes which entities in a data store will take part in the described mapping. The example in Figure 4 depicts that there is a mapping between all wall objects in the first schema and all wall objects in the second schema which match the specified invariants, and the equivalences describe how to move the data between them. The class specification can name several classes for each schema. When more than one class is defined for a schema then all occurrences of combinations of the entities are collected together (normally in association with an invariants definition).

In some cases, it is necessary or convenient, to describe a mapping to temporary objects, e.g. where two mappings can reuse a partial mapping in their transformations. To be able to distinguish mappings to temporary entities from those which create real objects in a particular view a special notation is used for temporary entities.

3.1.2 *Invariants in a mapping*

Invariants are an optional part of an *inter_class* definition and describe the conditions under which it is possible to use a particular *inter_class* definition. The example in Figure 4 specifies that it is only possible to use this *inter_class* definition for walls in the CBM which have one segment and at most one window. Presumably there would be other *inter_class* definitions specifying how to map other types of wall. The invariants thus provide criteria to decide which *inter_class* definition to apply to any given object. The invariants also create constraints on the objects involved in the mapping which, in some cases, can be used to fill in values in an object, or create constraints on an object.

Invariants can be thought of as boolean expressions which must be true to allow the mapping to proceed. Invariants can be composed of functions or programs which succeed or fail, as well as expressions containing any of the standard relational operators (e.g. =, >=, <, etc.), and boolean operators.

```
inter_class([wall], [wall],
 invariants(count(segments) = 1,
         count(wall_windows) <= 1),
 equivalences(name = id,
         wall_window = wall_windows[1],
         x1 = segments[1]=>global_design_rep=>hasOuterLoop=>X1,
         y1 = segments[1]=>global_design_rep=>hasOuterLoop=>Y1,
         x2 = segments[1]=>global_design_rep=>hasOuterLoop=>X2,
         y2 = segments[1]=>global_design_rep=>hasOuterLoop=>Y2
         )
 ).
```

Figure 4. A mapping specification.

3.1.3 *Equivalences in a mapping*

The equivalence descriptions specify how a particular piece of data in a class of one schema is related to a piece of data in a class of another schema. These equivalence descriptions can describe one-to-one mappings, mappings which involve the arithmetic manipulation of values, and mappings which may involve many attributes in an entity or even whole entities. The equivalences shown in Figure 4 illustrate a few of the different forms of allowable equivalences. The first equivalence describes a direct equality between the name and id attributes of the two wall classes. The second equivalence specifies a relationship between a wall_window reference in the tool model and an element of the wall_windows set of the CBM. The remaining equivalences show relationships between location attributes in the tool model and corresponding values in the CBM obtained via a chain of references. The chain of references accesses derived location attributes of an object of class rm_loop, found by following the segments[1], global_design_rep and hasOuterLoop references is shown in Figure 3.

There are three basic forms that equivalence descriptions can take:

1. Expression_1 = Expression_2

This covers one to one mappings between attributes (eg girth = circumference) and many to many mappings between attributes (eg area = height * width, or sqrt(x * x + y * y) = length). All normal arithmetic expressions can be used in this form of specification. Attributes described on the left hand side of an equivalence statement are assumed to belong to a class defined in the first schema and those on the right hand side to the second schema, though this is checked and the user may have to further qualify attribute definitions to avoid ambiguity

2. Function(Attributes)

This allows the description of functions mapping attributes between entities which can not be described simply with arithmetic expressions. In the ICAtect system the function is implemented as a Prolog predicate. These predicates are expected to work bi-directionally.

3. map_to_from(Function_1, Function_2)

In some cases it is not possible to describe the mapping between attributes in a simple invertible fashion. To cope with this it is possible to define mappings using a piece of code for each mapping direction. The appropriate piece of code is called whenever a mapping is required in a particular direction.

3.2 *A VML support environment*

To facilitate specification of mappings we are developing a modelling environment, VPE (VML programming environment), which allows both textual and graphical descriptions of a mapping to be developed in the same context as the schema definitions. This environment is based upon the MViews programming environment framework (Grundy & Hosking 1993). This framework eases the task of developing software environments which provide multiple overlapping graphical and textual views with consistency maintained between the views. MViews has been used to develop EPE, an environment for EXPRESS and EXPRESS-G modelling (Amor et al. 1994), which is used for developing schema definitions in ICAtect. VPE acts as a complementary tool to EPE to support mapping specifications.

Using VPE the user is able to navigate through the EPE schema views and select various classes to participate in a single mapping definition. The mapping definition can then be developed textually, as in the example of Figure 4, or graphically, or through a combination of both methods. In the graphical approach, icons representing each of the selected classes are placed into a new view window with a mapping descriptor icon between them, as shown in Figure 5. The user then specifies invariants and equivalences which hold between the classes by “wiring” between the class attributes and the mapping icon. In some cases, e.g. the location attribute mappings, supplementary textual descriptions must be used. This is due to the fact that the graphical notation provides a subset of what may be expressed in the VML language. Users may shift between the textual and graphical representations and changes to one will be propagated to all others to maintain consistency.

Graphical views of a mapping can also be used to display a simple subset of the total mapping. This is especially useful where class definitions are large and

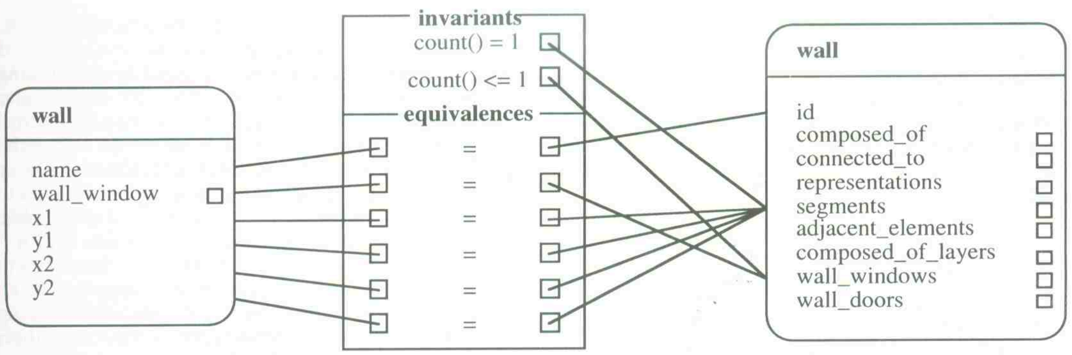

Figure 5. Graphical mapping specification.

the user wishes to concentrate on a smaller portion of the mapping problem in a single view. VPE maintains the canonical definition of a mapping and propagates all additions and modifications to any view of a mapping to all other views which incorporate that mapping. This ensures consistency between all views of the mapping definition and provides instant feedback to the user on changes which affect other mappings in the system.

4 SCHEMA MANAGEMENT WITH VML

VML and VPE can, in addition to allowing the specification of mappings, be used to manipulate the schemata being mapped between. This can be useful in three distinct cases as discussed below.

4.1 *Schema updates with VML*

Defining the mapping between a schema for a new design tool and an existing CBM can, in some cases, require modification of the CBM. This may be because new attributes need to be stored in a class, new relationships need to be modelled between classes, or entire classes need to be created due to a new area being handled by the design tool being attached.

Rather than having to modify the relevant schema before constructing the mapping, VPE provides support for doing both simultaneously. When constructing a schema mapping using VPE, it is possible to refer to attributes and relationships which do not currently exist in one or other of the schemata. VPE is capable of detecting that modifications to the corresponding schema are required and automatically making those changes. The changes will then be reflected in the canonical form of the schema, and all views of that schema constructed using EPE. This form of automatic update can also be used to specify totally new classes for a particular schema. An audit trail of modifications to the schema is also maintained, permitting developers to keep track of which previously integrated schema caused the addition or restructuring of particular components of the current schema.

4.2 *Schema integration with VML*

In a similar fashion to the schema update approach, it is possible to perform schema integration using VPE. If the user has several design tool schemata, but no CBM, then a CBM can be constructed through the definition of mappings from the design tool schemata to an initially empty schema.The additions to the CBM resulting from each mapping will be amalgamated into the canonical form for the CBM. Where the addition of a new design tool mapping modifies portions of the CBM which are seen by other design tool mappings, these modifications are propagated to the affected mappings resulting in changes to their definitions.

4.3 *Version management with VML*

Another variation on schema updates is the ability to define new versions of a schema using VML. To perform this function the user can select a class (or classes) and ask for a one-to-one mapping to the new version's schema. This mapping can then be altered to produce the modifications required for the new version. For example in Figure 6 we see three variations of an initial CBM with a VML definition to specify the difference between each version. These VML mappings can then be used to move data stores from older versions of a schema through to newer versions and vice-versa if desired.

If new versions of the CBM are created, it is often necessary to migrate all mappings between the old CBM's schema and integrated design tools to the new schema definition. With VML mappings between versions of the CBM this can be handled without the need for the user to re-specify mappings for the new schema version. The necessary modifications to the existing mappings for design tools can be determined by examination of the mappings between the versions of the CBM. This method will also work with new versions of a design tool schema mapping to an existing CBM schema.

Using this suite of updating mechanisms it is possible to manage modifications to an integrated system and existing data stores with little or no need for user modifications to the previously specified mappings.

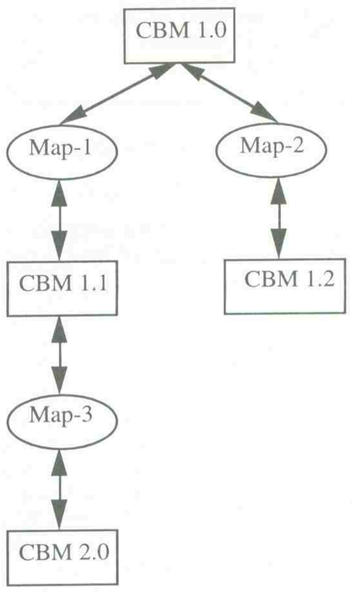

Figure 6. A version tree.

5 IMPLEMENTATION OF VML

The implementation of the inter-model mapping system has four major components, one of which is only used when building up new views from an existing store, while the other three manage data modifications and the VML equivalences. While the complete details of these components is beyond the scope of this paper, we provide here a broad outline of the tasks performed by each component.

The most active component of the system is the *monitor*. This component tracks all modifications made to data stores belonging to any design tool or the CBM. It also traps all method calls made by objects in an object store (if the data store is an object-oriented system). Whenever a change is made, or a method invoked, the monitor determines what currently defined mappings are affected by the modification and passes through commands to force the mappings to be re-evaluated. The monitor also determines if the modification affects attributes referenced in the invariant section of a mapping. If this is the case, it forces the invariant to be re-evaluated, which potentially could require the application of a different mapping definition (if there is another mapping the invariant of which is satisfied by the new value).

An *equation solver* handles the evaluation of equivalences. This component is capable of re-writing simple equivalences to solve them for a particular attribute, or of invoking a function with the correct parameters to produce the desired result. When the value for an attribute has been calculated from an equivalence specification, it is propagated through to its object. In some cases it is not possible to calculate a value from an equivalence (e.g. area = width * height, if area is changed there is insufficient information to determine appropriate changes to width and height). In these cases a constraint is set up between values in both data stores and their consistency is managed by the system.

When a modification is propagated to a new data store through a mapping, its modification may trigger other existing mappings to be re-evaluated. To manage the possible flood of changes a *change propagator* ensures that deadlock situations are not created by the changes being requested.

The final component of the inter-model mapping system is a *query system* for use when creating a new view from an existing data store. When a new view is established, the complete mapping definition for the view is applied against the existing data store. This requires many queries for objects which match the invariants of particular mappings. Every set of objects which match such a query can be passed through to the equation solver to be mapped to the new data store, using the equivalences of the mapping definition.

6 CONCLUSIONS

In this paper we have presented a new language and associated support environment for the specification of mappings between two schemata. This declarative language was designed to match closely the requirements of mapping specifications and, in contrast to other mapping languages, the definitions can be used bi-directionally. We have detailed a support environment for the mapping language which allows the user to easily specify and manage the associations between classes in schemata and then further specify the equivalences between individual class attributes. This environment is capable of supporting schema updates, schema integration and version management. Finally, the implementation of VML in the ICAtect framework creates an integrated system which is capable of managing multiple views of a CBM. This system is also able to guarantee the integrity of data in any of the views against changes made to the CBM or any other design tool view.

ACKNOWLEDGEMENTS

The authors acknowledge the support of the University of Auckland Research Committee in pursuing this research.

REFERENCES

Amor, R., J. Hosking & M. Donn 1993. Integrating Design Tools for Total Building Evaluation, *Building and Environment*, 28(4):475-482.

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

Amor, R., G. Augenbroe, J. Hosking, W. Rombouts & J. Grundy 1994. Directions in Modelling Environments, submitted to *Automation in Construction*.

Augenbroe, G. 1992. 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:731-744.

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

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

Boyle, A. & A. Watson 1993. *STEP Tools Review: Phase 2*, Computer-Aided Engineering Group, Department of Civil Engineering, University of Leeds, England, Technical report CIPM/LU/TP/7:26.

Clark, S.N. 1992. *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:13.

Grundy, J. 1993. *Multiple Textual and Graphical Views for Interactive Software Development Environments*, PhD thesis, Department of Computer Science, University of Auckland, New Zealand.

Grundy, J.C. & J.G. Hosking 1993. Constructing multi-view editing environments using MViews, *Proceedings of the 1993 IEEE Symposium on Visual Languages*, IEEE Computer Society Press, Los Alamitos, CA:220-224.

ISO 1991a. *Introduction to ISO 10303 (STEP)*, ISO TC184/SC4/WG5, Technical report.

ISO 1991b. *EXPRESS Language Reference Manual*, ISO TC184/SC4/WG5 Document No. 14, April.

Motro, A. 1987. Superviews: Virtual Integration of Multiple Databases, *IEEE Transactions on Software Engineering*, 13(7):785-798.

Ullman, J.D. 1982. *Principles of Database Systems*, second edition, Computer Science Press.