

## **Chapter 2**

### **The Project Development Environment**

Chapter 1 provided an introduction to an area in which integrated design environments are required. In Figure 1.2 and Section 1.4, one framework for an integrated design system was discussed, with internal components and connections which are generally agreed upon. However, even with the framework in place, a large gap still exists between what must be created and how to create it. The integrated design system shown in Figure 1.2 can be implemented in a variety of system with links to relational and object-oriented databases, 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 chapter, the environment required to develop, implement, and test the sub-systems of an integrated design environment is specified. These environment requirements form the basis for the work presented in this thesis.

The development of an integrated design system involves a number of actors playing various roles. These actors may belong to different organisations. For example, the developers of DT schemas and mappings are usually from the organisation which developed the design tool. The developer(s) of the IDM may be from the organisation implementing the integrated design system, or, as in the ISO-STEP development, from many companies and research institutes throughout the world. The project coordinator is usually from the organisation utilising the integrated design system. It is clear that whatever environment is chosen for the modelling of aspects of the integrated design system, there needs to be open communication links between all actors involved in the specification.

Considering only the modelling and specification components of the integrated design system outlined Section 1.4, the models which need to be developed are: the IDM schema; the schemas

for the DTs; the mapping specification between IDM and each DT; the interface between a DT schema and its data-files; and the project model. Though there needs to be very close links between each of the environments used for the development of each of these models, each modelling aspect and its requirements can be considered independently, as detailed in the following sub-sections. The structure and requirements section below presents informal requirements for a project development environment, as do all the sections considering the various components of such an environment. This approach is taken as the proposed system is very large, and complex, which makes the use of formal approaches infeasible in their current state.

## 2.1 Structure and Requirements

The development of an integrated design system is not a linear process. There is a large amount of interaction between developers of every sub-system of the environment. Examples of the types of cycles and interactions which occur are detailed below:

- The schema for a DT is developed either by the developers of the DT or by the creators of the integrated design system, in consultation with the developers of the DT. The DT schema describes the external interface of the DT as seen by the integrated design system, along with constraints imposed by the DT which must be taken into account by the integrated design system. During this process, the modeller must detail the DT schema in collaboration with those developers responsible for the implementation of different aspects of the DT. These developers are aware of the implicit constraints imposed by their implementation methods for these aspects of the DT. During this process the schema must be reviewed by the DT developers, reasons for constraints imposed on the schema documented, and fixes to the schema actioned by the modelling coordinator.
- During the IDM development, the requirements of all DTs which are going to be used in the integrated design system must be taken into account. This requires an iterative process of development by the IDM modeller(s), and checking by all the DT schema modellers, to ensure requirements are met. IDM modification requests by the DT modellers must be actioned and coordinated with possibly conflicting requirements from other DT modellers.
- At the point where the IDM has reached a fairly stable state (few major restructurings taking place), the specification of what needs to be mapped between the IDM and a particular DT may be undertaken. This specification lays the groundwork for the implementation of the mapping between the implemented IDM repository and the DT interface. This mapping specification is likely to be developed by the DT modeller in conjunction with one of the IDM modellers. The workers from these two teams need to coordinate their work and may develop the mappings at the same time as the IDM is being developed and checked against DT models. The mapping specification needs minor updates as the IDM changes. This entails coordination and communication between IDM developers and the affected DT mapping specifiers.

- The specification of the use of an integrated design system for a particular project can be undertaken at this stage. Generally, the project manager specifies the actors who will work on a project and the design roles they will play in the project. Individual actors are likely to specify the DTs they will use to complete their various design roles, and in collaboration with the project manager develop a flow of control specification for the project. This requires negotiation and collaboration between various actors and the project coordinator to identify exact roles and responsibilities, and to ensure that the resources requested by actors are available for the design roles they need to complete.
- Though the processes described above appear to flow in a continuous manner from modelling through to project implementation, this is a major simplification of the actual process. It is likely that there will be changes at all levels described above, requiring continuous restructuring of the integrated design system as new DTs appear on the market which particular actors may wish to utilise. New DTs may force changes in the IDM as well as new mapping specifications, mapping testing and changes to the project specification. It is also likely that the needs of the project require adjustment during the lifetime of the project, adding new actors as unusual problems are encountered, removing actors who may not be performing as required or become ill, adding new flows of control to force some aspect of checking that was previously not deemed important, or adding new paths to speed completion of the project. Thus changes are likely at every point of development and use of the integrated design system. This requires a close coupling between the implementation of the integrated design system and the environment used to specify the structure and requirements of the conceptual integrated design system.

The types of interaction detailed above indicate that the development environment for an integrated design system requires a diverse mix of modelling tools and implementation tools. However, to analyse the requirements of the development environment, and in the presentation of solutions to these requirements, the problems are specified in two categories: the modelling and specification tools required; and the implementation and testing tools required (see Chapter 8). Though this is an unnatural split, given the interactions specified above, it does enable the issues for each of these sections of the environment to be considered independent of competing and distracting issues from the other category of requirement.

## **2.2 Schema Modelling and Development**

In all integrated design systems to date, the development of schemas has consumed a large proportion of the development time of the system. The amount of time expended in this phase is likely to change over the next decade as the schemas of the most commonly used design tools become available for general use. It is envisioned that these schemas will be offered, though perhaps not developed, in a common schema language (e.g., EXPRESS in the A/E/C community)

as part of the distribution of the design tool. Hopefully also in the next decade, a standard IDM will be developed for use in individual design areas (e.g., major APs from ISO 10303 for the A/E/C community). However, until this development occurs, and probably for some time after that, there will be a significant requirement for the specification of schemas.

### **2.2.1 Requirements of a schema modelling environment**

A very general and widely applicable schema development environment must satisfy many diverse requirements. Some of the most important follow:

**Modelling languages:** a wide range of languages is used by schema specifiers. For example, in the A/E/C modelling domain, data specification languages such as ER (Chen 1976), EER (Gogolla 1994), NIAM (Nijssen and Malpin 1989), IDEF1X (General Electric 1985), and EXPRESS/EXPRESS-G (ISO/TC184 1992) are commonly used. As these languages take a considerable time to learn and to become proficient with, many specifiers would resist moving to a new standardised language. These languages have various strengths and weaknesses (e.g., ER has a direct mapping to relational database specifications making ER models very useful when a relational database is required to implement the system). Hence, using one language to the exclusion of all others can limit the comprehensiveness of the developed system. Schemas are not usually initially developed as pure data models. Most are developed in association with activity and process models which detail related aspects of the domain being modelled. In some A/E/C projects, models are developed detailing aspects of the domain such as activities, data definitions, process, and ontologies using interlinked modelling methods (see Mayer 1994 for the IDEF family). For example, AP 228 (the HVAC model in STEP; ISO/TC184 1995) is being written in EXPRESS in association with an IDEF0 activity model (Mayer 1990). Therefore, some link between the data specification environment and other associated specification environments needs to be supported. The schemas currently being developed consist of hundreds of object types requiring a structured approach to their development. This often means that the modelling environment needs to offer graphical and textual notations of the languages to allow views of varying levels of complexity and generality to be specified (Meyers 1991).

**Consistency and negotiation:** the development of a large schema can involve experts from many institutions, and, especially with ISO standards, from several countries. To enable schema development with multiple developers, a strategy needs to be adopted to ensure the consistency of the final schema. This strategy must be capable of ensuring that all modellers are aware of recent changes to the schema and can be certain that they are working on the most up-to-date version. In situations where modellers are geographically dispersed, or where the modellers are not working in close collaboration, large numbers of changes to the schema may need to be notified to modellers when they come to coordinate. With several experts involved, conflicts between the modellers are certain to arise. The modelling environment needs to facilitate negotiated settlements to conflicts, and to document the settlement process.

Documentation and navigation: decisions affecting the development of a schema are made throughout its specification, by individuals, institutions, and by consensus at meetings. The reason for, and reasoning behind, these decisions is important when resolving conflicts, as well as allowing for later validation of the schema. This creates a requirement to collate documentation regarding these decisions during the schema development. Though documentation at this level can be unpopular, especially where there is very little justification for a decision (e.g., the committee wanted to go home early), its collation is vital to ensure a schema which can be further developed as well as being trusted. During the development of a large schema the specifiers will need to navigate and evaluate current versions of the schema at various levels of generality. With documentation associated with the schema there are many new ways of indexing, and therefore accessing, schema information which need to be made available through the modelling environment (e.g., viewing all modifications by a particular specifier).

The different types of schemas which need to be developed in an integrated design system are described in the sub-sections below. Chapter 3 provides a more detailed description of the requirements of a modelling environment and presents a new modelling environment (EPE) which supports many of the requirements specified both here and elaborated in Chapter 3.

### **2.2.2 IDM schema**

As described in Section 1.4 the IDM schema is the central schema through which all information is communicated. It provides structures to satisfy the requirements of all DTs and actors who will use the integrated design system. The development of the IDM is an arduous process requiring collaboration and communication between the various domain experts as well as between the integrators of the design tools. Computerised modelling systems currently in use provide little high-level modelling support to the developers of the schema. 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.

### **2.2.3 DT schemas**

As described in Section 1.4 the DT schemas reflect the information requirements of design tools, both their input and their output. The DT schemas contain many constraints upon their data structures to reflect the capabilities of the design tool which utilises the data from the schema. In most cases this equates to a very restricted subset of what can be described in the IDM. However, with all DT constraints described these schemas allow an integrated system to perform checks to ensure that valid models are passed through to the DT to manipulate.

## **2.2.4 Actor schemas**

In many integrated design systems it is necessary to model the allowable roles of the actors. This task, usually undertaken by the project manager for each new project, allows the specification of an actor's frame of reference in a project. Actor schemas are defined inside the integrated design system and are defined in terms of the IDM in use. There are two schemas associated with every actor. One schema specifies the subset of the IDM which is viewable by the actor, the other specifies the subset of the IDM that the actor has the authority to create or update. The actor schemas provide a limiting boundary for an actor. Their implementation ensures that no actor can modify a portion of a design outside their area of responsibility or expertise. Actor schemas can also be used in conjunction with DT schemas to regulate the updating of a central model with results from the DT to those that are within the scope of the actor. Actor schemas are likely to change from project to project to reflect the actors' responsibilities in each project. They are also likely to be modified within a project to cope with changed responsibilities due to changing requirements during the project.

## **2.2.5 Related schema modelling environment research**

There are a number of commercial tools used for schema development in the A/E/C world. Some of the most widely used include Design/IDEF (IDEFine 1995) and FirstSTEP (PDIT 1993). Boyle and Watson (1993) review a wider selection. There are also many tools developed at research institutes, many of which are available commercially. Some of the most common are CGE (Vogel 1991), PMShell (Luijten 1992), XP-IDEF and XP-EXPRESS-G (Poyet et al. 1990). Boyle and Watson (1993) also cover a range of the research environment tools. Almost without exception these tools provide very simple environments for defining schemas, mostly supporting only the graphical notation of a language (e.g., EXPRESS-G rather than EXPRESS). This often means only simple schemas can be developed in the tool (e.g., EXPRESS-G does not allow for constraints, functions, and procedures to be defined though they exist in EXPRESS). None of the tools allow overlapping views of a schema to be defined, and the majority only allow a single view to be used to define all attributes of an entity.

In the main, these drawbacks are due to the perceived market of the tools, which is the development of a published schema. With a paper copy of a schema, navigation amongst multiple views of a single entity is much more difficult to achieve. Though all the tools allow EXPRESS to be generated from the graphical notation (e.g., EXPRESS-G or NIAM), no connection is maintained between the graphical schema definition and the textual equivalent. This hinders a cyclic design process, as any additional specification in the textual form is lost when a new version is generated from the graphical representation. The XP-IDEF and XP-XPRESS tools from CSTB in France are the only tools in this area which maintain correspondences between textual and graphical definitions. This is achieved by generating the graphical views from the textual definition. However, this limits all graphical views to hierarchical decompositions of the textual

schema, rather than allowing predefined views of portions of the schema.

In the integrated software development research community, more sophisticated modelling environments are being developed. Integrated software development environments (ISDEs) which allow multiple overlapping graphical and textual views of a program have been available for many years (PECAN, Reiss 1985; Dora, Ratcliff et al. 1992; MViews, Grundy 1993; Meyers 1991). However, these tools are usually tied very closely to a particular programming language (e.g., Pascal for PECAN or Smart with SPE in MViews). This eases the compilation and testing requirements of such environments in comparison to abstract modelling languages (e.g., EXPRESS which has no direct mapping to any implementation language, and has constructs which can not be directly implemented in many popular languages, like C++). Recent advances in these environments (Grundy and Venable 1995) enable multiple modelling notations (e.g., EER and OOA/D) to be utilised in the same ISDE, and maintains consistency between overlapping areas modelled by the two different notations as well as between the multiple overlapping views within each notation.

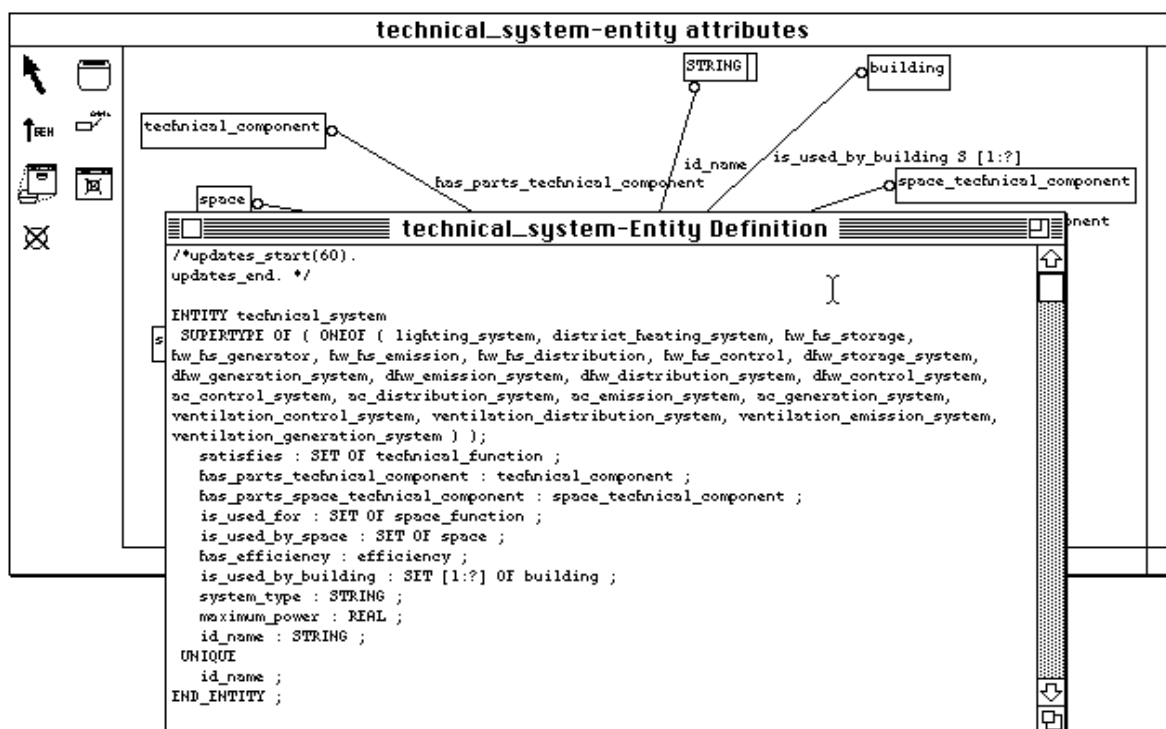


Figure 2.1 EXPRESS and EXPRESS-G views in the EPE modelling environment

## 2.2.6 Approach to a schema modelling environment

To be useable in the domain chosen for this thesis (i.e., A/E/C), the proposed schema modelling environment needs to support the EXPRESS and EXPRESS-G notations which are used in the development of the majority of schemas, but must provide much greater functionality than current commercial and research tools in the area, equalling that of ISDEs. This includes support for multiple overlapping graphical and textual views of the schema along with global consistency management, documentation procedures and powerful navigation features. The developed

environment (EPE) achieves all these goals, utilising the MViews system (Grundy 1993) and specialising it for EXPRESS and EXPRESS-G notations (see Figure 2.1).

Though only the EXPRESS and EXPRESS-G notations were implemented in EPE it is capable of extension to further modelling languages (e.g., Snart, NIAM, ER, OOA/D) due to the open architecture of the MViews system (EPE is currently capable of manipulating Snart models, but not modelling with the Snart formalisms). The full requirements and capabilities of the EPE system are discussed in Chapter 3.

## **2.3 Inter-schema Relationship Modelling**

Specifying the relationships between entities in a DT schema and the IDM is currently performed either by actors from the DT development teams who wish to see their DT used in a particular integrated design system, or by integrated design system developers who want to show the utility of their system with links to as many design tools as possible. Currently, there is no formal modelling of this stage of the integrated design system development, and mappings are hand-coded for particular implementations. There are also no tools available to support the development of mappings between schemas. This is slowly changing as the benefits of modelling this stage of the development become more obvious. There are three possible scenarios envisaged for the specification of relationships between schemas.

- During the development of the IDM, the schema modellers for the IDM work in collaboration with the schema modellers of individual DT's. In this scenario the DT modellers specify a mapping from their DT schema to the IDM to help define the structures and attributes that are required in the IDM. This also guarantees that the IDM provides for all the data needs of their particular DT.
- After the development of the initial integrated design system a new DT is added, in which case the DT schema modeller specifies a mapping between the current IDM and their DT schema. If the IDM schema is comprehensive then no changes should be necessary to the IDM. If changes are required they can be passed through to the IDM modellers to handle.
- When the integrated design system is being used for a project, a new view of the data in the IDM may be requested by an actor. In this scenario either the actor, or specialised support personnel, will define a mapping from the IDM schema to the schema of the view required by the actor. This type of mapping has no possible effect on the IDM and does not involve modellers of either the IDM or DT.

### **2.3.1 Requirements of an inter-schema relationship definition language and modelling environment**

To be useful in the development of an integrated design system, the language and modelling environment for the specification of relationships between schemas must offer the following facilities:

**Mapping specification language:** no mapping languages exist which allow a general high-level specification of correspondences between two schemas to be specified. However, in the same way that schema modelling notations are required which are independent of a final implementation, mapping modelling notations are required which are independent of their final implementation. To be easily understood and used by actors involved with IDM and DT schema development the mapping notation needs to be a high-level declarative language (i.e., leading to small definitions) rather than a low-level procedural language (i.e., leading to verbose definitions). Given the range of schema modelling languages used by actors the mapping notation should not be aimed at complementing a single schema modelling language. It should instead aim to encompass the generic facilities required in mapping in general.

**Links to schema development environments:** there is a close dependency between schemas and the definition of mappings between them. A mapping modeller needs to be informed of all changes a schema modeller makes to the schemas referenced in their mapping specification. Conversely, the schema modeller needs to be informed of all modifications needed in their schema to meet the requirements of the mapping specification from individual DT's which have information requirements not met in the current IDM structures.

**Consistency and negotiation:** though the definition of a mapping is only likely to involve one or two actors conversant with the schemas being mapped between, there is a strong requirement for them to remain consistent with each other as well as with the schemas the mappings reference. As new mapping notations are developed, it is likely that mapping portions will be specified in different languages and in views of varying levels of generality. These will need to be kept consistent with each other. For similar reasons to schema modelling (Section 2.2.1) it is imperative to ensure that all modellers are aware of recent changes to the mapping specification and that they can be certain that they are working on the most up-to-date version, and that conflict resolution strategies are provided.

**Documentation and navigation:** the mapping specification details the transformation through which data is moved between models. The reasons for using particular transformations and the perceived constraints being addressed by each mapping need to be readily accessible by those debugging or validating the mappings at a later stage. The modelling environment should help to document all reasons for the mapping being specified, as well as points where modifications are made to the schemas being mapped between. This includes tracing the modeller responsible for particular mappings, the reasons for individual or mass

changes, and offering documentation views to complement the mapping views. The modelling environment also needs to facilitate navigation through the mapping views, both to all variants which perform mappings on the same types of objects (e.g., to check for total coverage of the specified mappings) as well as to schema views for the various entities in the mapping (e.g., to check the types of attributes being mapped between).

### **2.3.2 Related inter-schema relationship modelling languages**

Until very recently no general purpose mapping notations were available. However, with the development of the ISO-STEP schemas, the need for mapping specifications has been recognised and new notations developed. Prior to this, relevant formal work has concentrated on relational database views and schema integration (which are analogous to the mapping problem). To fit with the formal grounding of relational databases, these languages have provided very restricted capabilities, but ones which guarantee various properties of the final system (e.g., that all data can be mapped in both directions under all conditions). Relational database views (Ullman 1982; van der Lans 1988) can define arbitrarily complicated mappings as long as they are unidirectional. Bidirectional views, however, can only be described with a severely restricted set of relational operators (Banchilhon and Spyrtos 1981; Dayal and Bernstein 1982; van der Lans 1988; Harrison and Dietrich 1994). Schema integration and heterogeneous database systems (Batini and Lenzerini 1984; Batini et al. 1986; Navathe et al. 1986; Motro 1987; Bright et al. 1990; Kim and Seo 1991; Qutaishat et al. 1992) implicitly allow bidirectional views between schemas or databases. However, to achieve this result, they restrict the range of operators allowed to merge the schemas. This greatly reduces the number of schemas that can be integrated, and entails that semantic mismatches in schemas have to be resolved (by modifying the original schema) before a mapping can be attempted. This is not a viable approach where existing tools with fixed schema structures exist.

Research outside of the database area comes from various sources, mapping being a general problem in all computer science domains. Garlan (1986) describes a system for integrated programming environments based on the ability to define a type conversion. Lee and Malone (1990) describe a scheme for communication among groups with different type hierarchies with specific reference to a mail/message system. These are similar to the approach taken by Eastman et al. (1995) in EDM-2, where mappings are assumed to traverse a hierarchy in a type lattice. However, many mappings can not be achieved by traversing a type lattice, yet may still be described through functional or procedural code. The following research systems allow sophisticated views to be defined in their respective domains. However, they only allow unidirectional mappings from a single specification, and provide no high-level language for the final specification of the mappings. Views for objects in OO-environments are discussed in Hailpern and Ossher (1990). The KIF (Knowledge Interchange Format) allows agents to map between different knowledge representations (Genesereth and Fikes 1992; Khedro et al. 1994). Views can also be specified in visual programming environments (Ambler and Burnett 1989, and

see MViews, Grundy and Hosking 1993b). Constraint-based systems are one of the only domains which provide a notation for bidirectional views of information with automated consistency (Bowen and Bahler 1991, 1992; Mugridge et al. 1995; Eastman et al. 1995). While these languages provide most of the structures required to map between different schemas, they were developed mindful of control structures which are not suitable for a general integrated design system. Hence they tend to be weak in some specification areas required by integrated design systems, namely the definition of conditions upon which sets of constraints can be used, and default values prior to application of constraints. Constraint systems also tend to assume all data and tools are present at all times, as well as being constructed in the same development language, in contrast to an IBDS where whole tools are connected and disconnected over time and exist and operate independently. Constraint systems therefore tend to model correspondences as though a single system is in existence.

In the A/E/C domains several integrated design systems have provided for mappings between their schemas (Gielingh 1988; Willems 1988; ESP, Clarke et al. 1989; Luiten and Tolman 1992; Wong et al. 1992; ATLAS, Greening and Edwards 1995 and ATLAS 1993; COMBINE, Augenbroe 1995a and 1995b; COMBI, Scherer 1995 and COMBI 1995; CIMsteel, Watson and Crowley 1995 and CIMsteel 1995). However, all of these systems have opted for mappings defined in their implementation languages without any formal modelling of what is required to be mapped. Only a single development in STEP has formally examined the unification of models. SUMM (Semantic Unification of Meta-Models, Fulton et al. 1992) provides a notation for describing the semantics of schemas so they can be integrated. It does not, however, lead to a methodology to map between the integrated schemas. As noted previously, the work in ISO-STEP has highlighted a requirement for mappings between schemas, and a plethora of languages have been proposed to meet this challenge (Transformr, Clark 1992; EXPRESS-M, Bailey 1994; EXPRESS-V, Hardwick et al. 1994, Hardwick 1994; EXPRESS-C, Staub et al. 1994; Operation Mapping, Bijnen 1994; XP-RULE, Zarli 1995). The ISO-STEP committees have now decided to create a standard mapping language for STEP development (EXPRESS-X, Wen et al. 1996) which pulls together the best (and worst) of the EXPRESS-V and EXPRESS-M languages. These languages range from very low-level specification (almost at the C++ level) through to partially declarative in style. They are all unidirectional in their mapping specification (though EXPRESS-V allows both mappings to be specified in the same definition), which misses the main benefit that these languages could provide in defining bidirectional mappings between overlapping schemas.

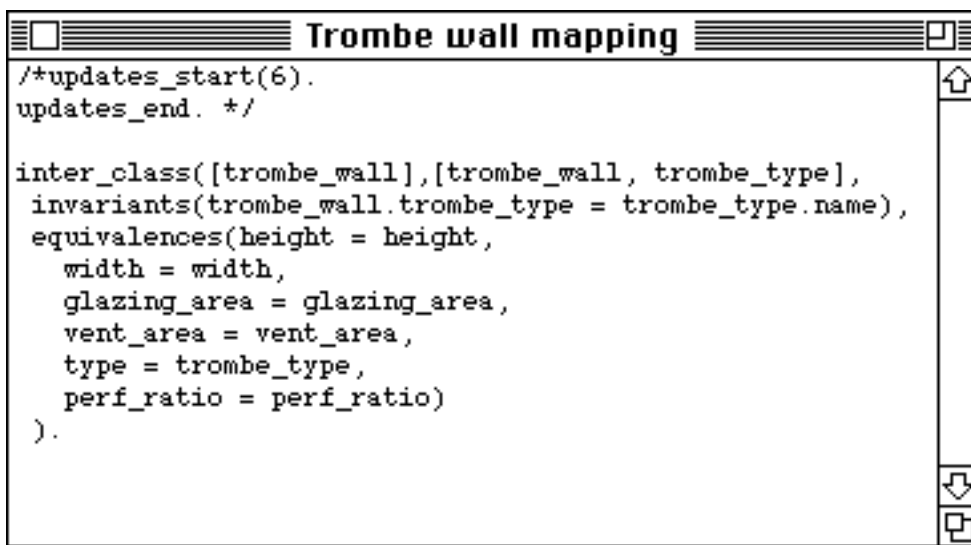
### **2.3.3 Related inter-schema relationship modelling environments**

Incredibly, given the size and complexity of mappings defined with the range of mapping languages and environments described in Sections 2.3.2 and 2.3.3, there are almost no environments available to specify mappings. It is assumed that mapping developers will utilise standard text editors to define the mapping, managing all aspects of consistency by themselves. The only tools provided are post-processors which check the mapping specification against

schema definitions when the mapping is compiled into an internal form for the mapping implementation. The only exception is Operation Mapping (Bijnen 1994) which provides a hierarchical navigation system to select classes and attributes which are to be utilised in a textually specified mapping. This is a minimalist environment providing almost no benefit to mappers except a guarantee that correct names and paths are used in the mapping specification.

### 2.3.4 Approach to an inter-schema relationship modelling language

A new high-level declarative mapping language, VML (View Mapping Language), was developed for this project and is described in Chapter 5. A sample mapping can be seen in Figure 2.2. VML has the full expressive ability found in constraint languages with equational, functional and procedural specifications of mapping between attributes. However, VML also incorporates specific notions of the conditions under which a mapping can be applied and initial values for newly created objects along with the mapping specification. VML is implicitly bidirectional with explicit definitions of the schemas being mapped between to support independent tool-based mappings. A unique feature of the language is its ability to specify method-triggered mappings. This allows mappings between object-oriented systems that can lead to a more interactive integrated design system, for example, invoking functionality in a second tool based upon a user specified action in a primary tool.

A screenshot of a graphical user interface window titled "Trombe wall mapping". The window contains a text editor with VML code. The code defines an inter-class mapping between 'trombe\_wall' and 'trombe\_type'. It includes comments for update ranges, invariants, and a list of equivalences for attributes like height, width, glazing\_area, vent\_area, type, and perf\_ratio. The window has a standard title bar with a close button on the right and a vertical scrollbar on the right side.

```
/*updates_start(6).
updates_end. */

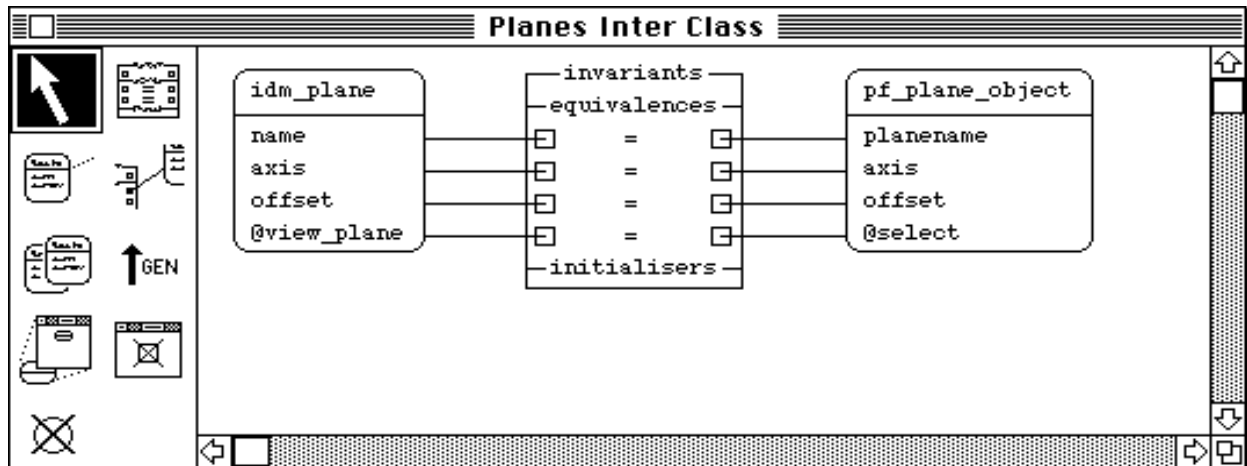
inter_class([trombe_wall],[trombe_wall, trombe_type],
invariants(trombe_wall.trombe_type = trombe_type.name),
equivalences(height = height,
width = width,
glazing_area = glazing_area,
vent_area = vent_area,
type = trombe_type,
perf_ratio = perf_ratio)
).
```

Figure 2.2 Example VML textual specification

### 2.3.5 Approach to an inter-schema relationship modelling environment

With a similar reasoning as for the modelling environment developed in Section 2.2.6 for schema models, the mapping modelling environment had to provide at least the same functionality as that found in ISDEs in computer science. It also needed to provide a tight coupling between schemas and the developing mapping. The VPE (VML Programming Environment), as described in Chapter 5, provides multiple graphical and textual views of mappings, similar to those described for EPE. VPE also manages the two schema definitions and verifies mappings being described

between the two schemas as they are developed. In a graphical view (see Figure 2.3) it provides a wiring mechanism to specify mapping relationships, whilst in the textual view (Figure 2.2) the full VML specification is available. Navigation facilities allow similar mappings and partial views to be identified. Visualisation functions allow the full class and mapping icons to be viewed, to identify features which have, or have not, been mapped. The developed VPE system utilised the MViews product (Grundy 1993) specialising it for the requirements of VML.



**Figure 2.3** Graphical mapping specification in VPE

## 2.4 Design Tool Environment Modelling

Though the schemas of Section 2.2.3 define the data structures used by particular design tools and provide a definition of the constraints on data models which can be used by the design tools, they do not specify in what form the data is used by the design tool. The inter-schema relationships of Section 2.3 describe how data from models of differing semantics can be mapped, but not the syntax utilised by the design tools. It also does not describe how to invoke the design tool with the required data, or how to recognise when the design tool has completed its tasks. All that can be assumed about the data derived from the information modelling described in the previous sections is that correct data is held for the design tool, but this data is still held in some internal format foreign to the design tool (e.g., an object-oriented database).

To make the final step from internal data models to the form required by the design tools, the design tool input and output formats must be modelled as well as its invocation requirements and methods. This task has not been tackled in this thesis. However, it is clear that an ISDE of similar capacity to those referenced in Sections 2.2 and 2.3 could be developed from a general purpose development environment (e.g., MViews).

### 2.4.1 Requirements for design tool environment modelling

To capture the information required to manage individual design tools, the design tool environment model must be capable of specifying the following aspects of the design tool:

**Input data format:** the input of design tools is structured in very specific formats. This has to be accurately recreated to invoke a design tool with selected data from an IDM. The formats will include fixed format object per line data-files (from the FORTRAN punch-card era), data-files with headers before each section of objects, and data-files which allow free format data specification. The specification of the input data format must also specify the correspondence between the design tool model, as seen in the schema development environment, and the location of data in the input data-file. For design tools which require interactive input, the format must define the format of the data to be passed through, and also define how to select the requested data from the design tool model (e.g., a query based on the question being asked by the design tool). Where the design tool input data schema can change (e.g., a new version of the design tool), changes in the input data format must be passed through to the schema in the schema development environment and reflected directly in the model seen in that tool.

**Invocation parameters:** the majority of design tools need to be invoked with parameters detailing names of files with input data and often what type of simulation to run. This invocation data may specify input data-file names, the desired level of accuracy of the final result, standard library files used when searching for material properties, etc. The source of this data must be specified, e.g., user specified before invocation of the design tool, or default values based on the type of design function the tool is being used for.

**Invocation method:** in most operating systems it is possible to schedule a design tool to start operating, though the manner of starting tools is very different in various operating systems. With this specification an integrated design system can automatically operate tools which require no operator intervention. A model of invocation specifies the environment the design tool resides in (e.g., unix, VAX, Macintosh, MS-DOS, Windows) and how the parameters and design tools must be arranged to start the design tool. It must also define the commands to start the design tool, e.g., the name of the design tool and flags used.

**Termination detection:** for an integrated design system to schedule and operate a range of design tools it must be able to determine when they are working and when they have completed their operations. A model of termination specifies how the integrated design system detects the termination of a design tool, or the completion of its design function. This specification must cover cases from those design tools which start to perform a design function and end when it is done, through to those which are continually running and whose design function is completed without exiting the design tool.

**Output data format:** to be able to retrieve results from a design tool the integrated design system must be able to process the output of design tools. The range of textual output data-files is the same as those of input data-files (I assume that all graphical outputs will have a corresponding textual output). This specification must define how the output data is tied to the data supplied as input to the design tool, and it must define the correspondences to the output design tool model, as seen in the schema development environment. For design tools which are interactive the format must identify where the resultant data resides

amongst the prompts and queries of the interaction. Where the design tool output data schema can change (e.g., a new version of the design tool), changes in the output data format must be passed through to the schema in the schema development environment and reflected directly in the model seen in that tool.

### **2.4.2 Related design tool environment modelling work**

The author has previously defined simple notations for interfacing design tools with very rigidly structured input and output data-files (Amor 1991). These methods were further augmented to provide a more flexible data format description based on DCGs (Williams 1990). In Pascoe (1994) a methodology is presented for representing GIS design tool data states including physical forms and location along with transformations to move between states. However, while this gives a method of representing the transformations that must take place to interface between various tools, it does not address representations necessary for automatic implementation of each transformation. This implementation is currently performed by hand-coded tools.

The above methods go some of the way towards what is required in the modelling of the design tool environment, but only for a limited set of design tools. Interactive design tools (e.g., knowledge-based systems) can not be handled by any of the methods described above. In Amor (1991) it was argued that defining an interface to these design tools could require as much effort as defining the design tool itself. These methods also only define the data requirements, they do not attempt to define the invocation parameters and methods required to start a design tool or to determine when the design tool terminates. Definition of these environmental parameters will be operating-system dependant and may prove to be impossible to define generically. However, the 'Tool Encapsulation Specification' project (TES 1995) does provide a standardised notation to describe tool invocation parameters and methods so that a TES system on any platform would be capable of determining how to run a particular tool. TES also provides for descriptions of methods to suspend tool execution or determine a tool's completion status.

### **2.4.3 Approach to a design tool environment modelling system**

The modelling of the design tool environment is not attempted in this thesis as there is an emerging standard (TES) which looks as though it will provide what is required. TES has currently not been developed enough to provide a modelling system to support it. However, when a design tool environment modelling system is created as part of the project development environment, it will have to provide at least the same functionality as that found in ISDEs in computer science. It will also need to provide a tight coupling between design function specification (which defines the design tool used to perform the function) in the process model and a design tool's data requirements. As with the previous modelling systems, an environment built on top of a system such as MViews (Grundy 1993) would be able to provide the functionality and integration required to bring these models into the whole project specification.

## **2.5 Project Definition**

As described in Section 1.4 the project model defines how an integrated design system will be used for a project. It provides a method to describe the users of the system, the roles they play, and the tools they will utilise to fulfil those roles. It also allows the definition of flow of control for project management purposes, allowing necessary process flows be defined, and enabling the determination of points at which concurrent design can take place. It also provides a tool to the actors to manage their design roles to ensure timely completion and hand-over of their work.

### **2.5.1 Requirements for a project definition notation and development environment**

To be of utility in the development of an integrated design system the modelling environment for the project definition must offer the following facilities:

Project definition language: project managers need to specify the designers working on a project and the activities they need to carry out in their contribution to a project. Project managers also need to keep a constant overview of the activities completed in a project and those which are next scheduled for completion. To achieve this in a computerised environment project managers will require a notation to define the actors undertaking a particular project. The notation will also need to define the roles actors play in a project and the possible, or necessary, paths between the design functions that need to be completed to fulfil the design roles.

Links to schema development environments and design tool environment: flow of control specifications are tied to particular design tools which must be controlled by the implemented integrated design system. To achieve this it must know about the input and output schemas for all design tools, and actors, to be able to tie down the responsibilities of actors in a project. To be able to control the design tools associated with particular design functions it must also be able to link with the design tool environment which details the invocation procedure and termination detection for the design tool.

Documentation and navigation: the project control system is the most visible aspect of the integrated design system. To justify flows of control in the IBDS the project definition environment should help to document all reasons for the paths between design functions being specified. This includes tracing the modeller responsible for particular paths or design function specifications. The modelling environment also needs to facilitate navigation through the project definition views, to show all places that a particular design function could be performed, and by whom, and to provide some simulation of the flows to ensure that particular configurations are useable, or possible to negotiate.

### **2.5.2 Related project definition notation work**

The majority of the project definition notations used to date have been activity modelling formalisms which are used in a specific restricted manner. For example, IDEF0 (Mayer 1990) activity models are used to model process flows by treating the relative horizontal positioning of activity icons as presupposing a temporal relationship. While this view of activity diagrams allows some forms of process flow to be defined it is very poor when attempting to handle many types of flows, e.g., recursive flows, concurrent flows, and those which are conditional on previous flow states. This is similar to the modelling capability of data flow, state diagrams, and Gantt charts which are generally linear in nature. Pert charts provide the majority of modelling behaviour required except that recursive flows are difficult to model and flows are very deterministic, which is fine for highly structured projects, but not all projects fall into this category. Many notations have been derived from the Petri Net formalism (Petri 1976, Jensen 1990). These notations range from IDEF3 (Mayer et al. 1992), which allows process flows to be described with AND and OR connections between states, through to VPL (Swenson 1993) which additionally allows specified conditions or constraints to help define paths in the work-flow. IDEF3 allows looping and recursive behaviour, but the semantics of looping with AND and OR conditions is not specified. The formalism is also developed independently of the actors involved in the flows which removes the possibility of defining the links to organisational requirements. In contrast, VPL incorporates notions of actors associated with process flows. A previous review of process modelling notations (Curtis et al. 1992) determined that a complete notation needs to specify functional, behavioral, organizational and informational requirements. This full range of requirements had not, at that stage, been incorporated into a single notation. However, during the COMBINE2 project (Augenbroe 1995a and 1995b) a two part modelling notation was developed (CombiNets, TU Delft and Amor 1993), based around Petri Nets, which models the four requirements of Curtis et al. (1992) to some extent.

### **2.5.3 Related project definition environment work**

The specification of processes in a project has been part of the project management aspect of projects well before computers appeared on the scene. Following the paper-based process management many commercial computer tools have been developed to support the specification of flows, and in some of these tools to help analyse the practicality of the prescribed flows (PowerProject, ASTA 1996; Process Charter, Scitor 1995). These tools offer a very simple interface to define and refine what are basically linear process flows. Associated with the more sophisticated process modelling formalisms are tools of correspondingly greater sophistication. In COMBINE a CombiNet tool is offered through the CGE environment (Configurable Graphical Editor, Vogel 1991), this provides a simple hierarchical view specification of process flows and utilising design functions and actors. IDEF3 tools (ProSim, KBSI 1995; System Architect, Popkin 1996) provide simple single view specifications of process flows, but with links to IDEF0 activity and IDEF1X data models. VPL tools (Swenson 1993; Serendipity, Grundy 1996) provide

more sophisticated multiple view specification environments, but without the links to associated models.

### 2.5.4 Approach to a project definition notation

An extension of the CombiNet formalism is used to define process flows in this thesis (CombiNet was developed by the author whilst a guest at TU Delft for six months). This formalism takes two parts. The first part allows the specification of actors, their design roles in a project, and the design functions necessary to complete the various design roles. The second part is based around the Petri Net formalism (though the semantics are markedly different) and defines the possible flows between the design functions as specified in the first part of the formalism. The formalism incorporates aggregate process icons to represent complete sub-flows and incorporates actor overlays to allow different actors to be responsible for the same design functions in different parts of the process (see Figure 2.4 for an example of a portion of a process flow specification).

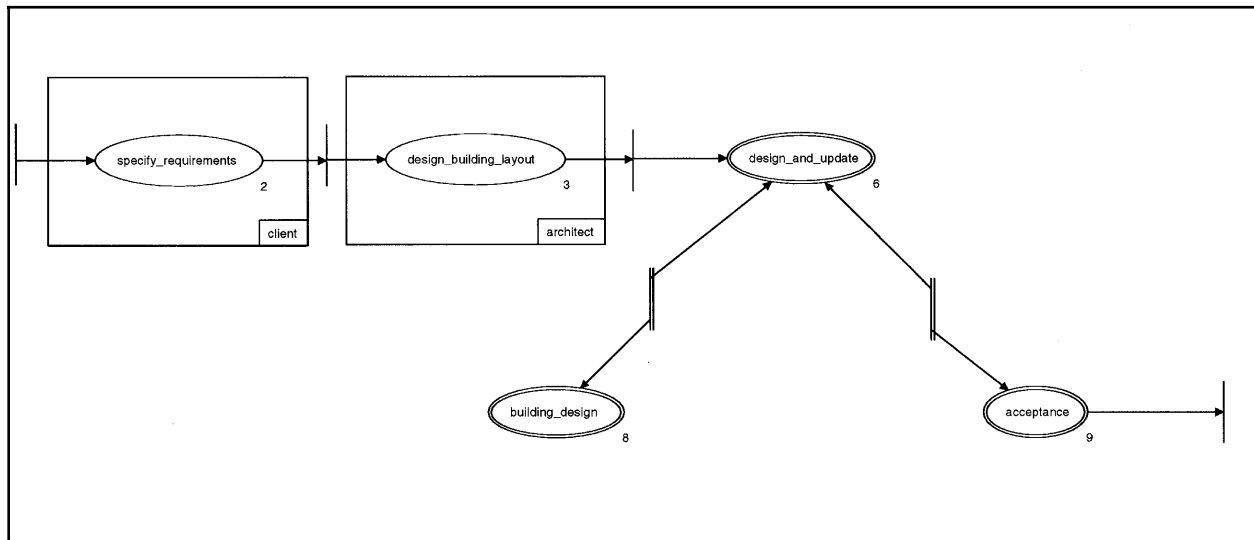


Figure 2.4 Project flow of control definition

### 2.5.5 Approach to a project definition environment

The CGE tool (Vogel 1991) was utilised in this thesis to implement the environment for defining process flows in a project. The choice of this tool was initially determined by the requirement in the COMBINE project for all modelling tools to be developed in CGE, which was available to all partners. Given the author's effort in defining the initial formalism in this tool, it was not felt necessary to re-implement the extended formalism in another environment. The CGE implementation of the formalism provides links between the actor and role specification, and the process flow specification. Though only single views of a process flow are supported during the specification, the tool provides hyper-linking between aggregate flows and their complete definition, allowing easy navigation around large process flow specifications.

## 2.6 Project Development Environment Summary

This chapter has described a range of individual modelling environments and modelling paradigms required to implement a project development environment. The individual environments have to model schemas, mappings between schemas, design tool parameters and project definitions. It is shown that these different models are all inter-related, so the total project development environment must provide for relevant communication between the different modelling environments. It is also shown that the individual modelling environments have many requirements in common. These include the ability to provide multiple views of information both textually and graphically and being able to maintain the consistency of the global model under changes in any view. The MViews development environment (Grundy 1993) is introduced and, through environment implementations, shown to provide the features required to implement the individual modelling environments as well as providing some of the interaction required between environments.

Two modelling environments, developed in this thesis for schemas and mappings between schemas, are described in detail in Chapters 3 and 6. The requirements for a schema mapping language are developed in Chapter 4, followed by the view mapping language definition in Chapter 5. The requirements for a project specification notation and an actual notation are described in Chapter 7. Taken as a whole these environments and modelling paradigms allow a full project development environment to be constructed.