

Chapter 8

The Project Testing and Implementation Environment

Although the development of an integrated design system requires the development of schemas for the IDM, design tools, and actors, as well as mappings, design tool environment models, and project flow definitions, they can not be developed in isolation. After defining the various schemas, the developer(s) must be able to test what is being created to verify that it fulfils its required purpose. To achieve this, the schemas must be instantiated with test data from the domain they will be used in. Mappings between schemas must be tested to ensure that the result of mapping from a model is equivalent to the original. Design tool environment models must be tested to ensure that the design tools start and finish as expected and that they find no problem with the data presented to them.

To make this possible requires testing environments that allow models to be quickly instantiated and validated to ensure that relationships and dependencies are as expected. To test mappings requires a system which simulates the functionality of a full mapping system, coupled with model visualisers to enable the correctness of the mapping to be ascertained. Similar testing is required for project models and design tool environment models. Though the environment developed for this testing could be the final integrated design system there are many reasons why it should not be. The major reason is that integrated design systems can take many forms and assuming that several are to be developed it is likely that each would vary in implementation to some extent. For example, the model implementation may be very different in different systems, from a relational database through to an object-oriented or frame-based system. To test models in each of these variations would require very different testing environments. However, if a single model visualisation tool was created for testing all integrated design systems during development, the final schema could be specified in any paradigm with the developer being sure of the correctness of the specification. Having a single suite of testing tools also allows closer integration between the testing tools, with flow-through benefits from the different stages of testing. For example, the

model visualisation tool can be used to inspect a derived model after performing a mapping between two models.

In this section, the various testing environments required to test the range of models detailed in Chapter 2 are described. A testing environment is described for each class of model described in Chapter 2 and where they can benefit from close links between each other and the modelling environments this is specified. Some of the environments are major pieces of work completed as part of this thesis, and described in more detail in subsequent chapters. Others are tools developed in other projects or are yet to be implemented, and are only described below to provide an idea of the functionality they can provide in this integrated framework.

8.1 Structure and Requirements

Complementing the non-linear modelling process described in Section 2.1, there is a similar non-linear implementation and testing process. Examples of the types of cycles and interactions which occur at this stage are detailed below:

- During any schema development stage a test suite needs to be set up to validate the schema development. This test suite is likely to be developed incrementally as the schema progresses, with tests representing normal, as well as unusual, design situations which the schema should be capable of representing. This evolving suite of tests must be able to be run against each new schema version to ensure that restructurings have not reduced the schema's expressive power. Later in the project whole building models must be able to be loaded in, not just as a test of the whole schema, but to enable testing of mappings to and from other design tools.
- During the IDM development, and particularly towards the final stages of its development, the IDM needs to be tested with an instantiated model. This is a paradoxical problem, as the IDM is developed to integrate DTs which can create an instance model, but the DTs can not be integrated until the IDM is developed. Bootstrap tools are thus required to create, visualise, and manipulate instances of the schema. These tools are used by the IDM developers to test their model, by the DT modellers to ensure the actual data in the IDM matches their conceptions of what should be there from the IDM development process, and by the DT interface developers to test the interfaces as they are developed. The IDM is still likely to change at this stage, so the instance model needs to be constantly restructured to take into account all modifications to the IDM. Changes in the instance model must then be passed through to all DT developers who rely on the instance model for their development.
- When the mapping specifications are being developed they need to be tested with the instance models of the IDM, to ensure that the mapping is correctly specified (the 'artwork' on page *v* shows the 'small' problems that are easily picked up from this type of testing, this problem originated from an incorrect specification of axes in a building model

mapping). These tests are run by the mapping developers. Where problems are detected they require assistance from the schema modellers of the IDM and the DT. Problems in test mappings may require changes to the mapping specification, or to the schemas of the IDM and DT, where the problem is one of missing information or constraints in the schemas. All of these changes have flow-on effects to anyone using the IDM as specified above.

- The specification of a project's utilisation of an integrated design system is a negotiated process and, especially in large projects, may need to be simulated to ensure that all required design roles can be completed in the available time-frame. Results from simulations may highlight areas of concern in the specified project flow, and lead to further modifications and re-negotiations between actors in the project.

Modelling tool and integrated environment requirements have already been described in Chapter 2. Similar requirements exist for the implementation tools and these are described throughout the rest of this chapter. Again, this chapter looks at implementation environments as if they are independent from the modelling stage. This is not the case, but this unnatural split does enable the issues for each of these sections of the implementation to be considered independent of competing and distracting issues from the other category of requirement.

8.2 Schema instance development

To help reduce the schema development time mentioned in Section 2.2, it is necessary to provide methods to instantiate sample buildings for the evolving schema and to navigate the resultant structures to ascertain that they meet their required purpose. There are many tools which can provide this level of utility. Four which have been used throughout this project are described in Section 9.2, however, requirements for this type of tool are detailed below.

8.2.1 Requirements of a schema instance maintenance system

To provide maximum utility to designers these environments must satisfy the following diverse requirements:

Instance manipulation: developers instantiating or browsing a model need to inspect tailored views of an instance, to specify values for attributes, and to specify references to other objects. The views of the data should be able to be presented in forms which are of a similar semantic level to that in which the developer envisages it (e.g., a graphical view of an object's geometry). Attribute values should also be able to be specified at a similar semantic level (e.g., clicking on an iconic representation of objects to collate identifiers to be referenced, rather than typing in all the object identifiers).

Correspondence with an evolving schema: as schemas evolve the previous model instances need to be brought forward into the new representation. While a mapping specification and mapping system can be used to achieve this syntactic transformation of the data, the model

maintenance system must be made aware of new schema versions. The system can then ensure that new objects are created with the correct parameters and attributes.

Multiple views: for model developers, multiple graphical and textual views are useful beyond the schema development stage. Similar methods for navigation, perusal and checking of the instances in a model are essential when dealing with a model as complicated as a building. Thus, the ability to view graphically the physical components of a building and to navigate through the building structure at either a textual or a graphical level and with the ability to switch between different representations of a selected instance, is particularly useful. Tools offering support for viewing and manipulating instances of the model are used by both the IDM development team and the aspect model teams to check instances of schemas, determining whether the structure is as they imagined when detailing their schemas and whether it accurately represents the information requirements of their DT.

Consistency: the development of an instance model can involve many of the schema developers, especially where a schema covers many domains in a field (e.g., steel structures, concrete structures, HVAC systems). Where several developers view instance model portions in varying levels of detail and representation, the various views need to be kept consistent with each other. This ensures that all modellers are aware of recent changes to the instance model and can be certain that they are working on the most up to date version.

Navigation: even a relatively simple building model (e.g., a domestic dwelling) from a schema with a hundred or more classes can contain thousands of objects. Navigating such a model can be a major task in itself. Methods of overcoming this problem are to provide high-level navigation mechanisms (e.g., walking down *part-of* structures utilising a GUI) and the incorporation of query languages which enable the identification of elements of interest. For elements which have a graphical representation, graphical navigation methods should also be provided.

8.2.2 Related schema instance maintenance systems

The majority of the schema modelling tools surveyed in Section 2.2.5 have no associated instance creation and perusal system. Though this is mainly due to the vendors' perception of market requirements, many of the modelling languages lack an ability to translate models into implementation languages (e.g., multiple conditional inheritance in EXPRESS has no direct translation into languages such as C++). The majority of development systems assume that the developed schema can be correctly specified without testing against actual data and can be transferred to the final implementation of the developed system which will contain the methods for creation of instances of the model. One modelling system (for software development) which allows instances to be created against the developing schema is SPE (Grundy and Hosking 1993a) and the associated development of CernoII (Fenwick et al. 1994) provides for many of the requirements described above.

The lack of associated instance manipulation environments in large schema development projects has proven to be a problem. In COMBINE, COMBI, ATLAS, etc. example buildings were created for final proof-of-concept demonstrations. These model instances were invariably developed late in the project and, with no complete tool to create the model, relied upon hand instantiation and laborious checking to ensure validity. This meant that the models developed were small (10-20 spaces and not always utilising all classes defined in the schema) and required hand-patches as schemas continued to evolve at the late stages of the project or as missing information was discovered (Augenbroe 1994b).

8.2.3 Approach to a schema instance maintenance system

To work with the modelling environments described in Section 2.2 the schema instance maintenance system has to comprehend EXPRESS schema definitions and data models sent in the associated STEP Part 21 data-file format (ISO/TC184 1994). The tools built for the maintenance system all operate in the same environment, which provides parsers for these formats and allows persistent databases to be created and manipulated according to the resultant specifications and data. The environment in which the maintenance system is built is the same as used to build the modelling environment, which eases the burden of providing connections between the two systems. The maintenance system tools provide the majority of requirements in Section 8.2.1, including instance manipulation and modification in a persistent store, viewing and navigation in both textual and graphical formats, and guaranteed consistency of the persistent store for data being manipulated in multiple views. The tools created and used in this project for instance creation and navigation are described in Chapter 9.

8.3 Mapping handler and controller

This module performs the translation of data between the IDM and the DT models. It must determine whether it is possible to create a consistent model from the IDM for any one of the DT models based upon the constraints in that DT model, and it must ensure that the IDM remains consistent upon update from a design tool's output. It must detect conflicts between various sets of data and must be able to invoke processes to enable negotiation over conflicting data from different design tools and users. An implementation of a mapping controller meeting these requirements is described in Chapter 10.

8.3.1 Requirements of a mapping handler and controller

A generalised mapping controller must offer the following facilities:

Consistency of data models: as developers or actors complete design functions with a particular DT they must be able to move relevant data through to the IDM whilst maintaining global consistency of the developing model. They must also be able to extract the current IDM

model for use in a particular design function. Through all of this the data in various data models must be kept consistent with each other when mapped through different models. In this way every user of the integrated system knows that they are utilising the same data as the other users of the system, or if they are out of date they will know of any possible problems and be provided with methods to become consistent. If consistency of the data models can not be maintained automatically then the integrated system must provide negotiation methods to allow the possible conflicts to be resolved.

Traceability of modifications: in any developed model a number of developers and actors will have been involved in the specification of various parts of the model. Modifications made to a particular model must be able to be traced to a particular mapping and ultimately to the user or DT which asserted the values. Tracking the modifications allows complete mappings to be applied and undone if necessary. It can also provide leverage in determining what mappings can be applied dependent upon the source of the data, or it allows a negotiation to be initiated with the specifier of the data to resolve conflicts.

Utilisation with different DT regimes: different developers and actors require different types of DTs to perform their design functions in a project. The mapping controller must be able to handle the requirements of various types of DTs. Some of these require all their data at the start of a simulation or design task, and others have interactive requirements for data based upon previous data (e.g., knowledge-based systems).

Actor and project manager links: the mapping of data between tools is a small part of the overall development of a coordinated design in a single project. To allow coordinated and managed design tasks amongst all participants in a project the mapping controller must be tied closely to the project management system. With close links to a flow of control system mappings can be activated to initiate a new design task, or used at the termination of a design task to capture the output of the task. To this extent the mapping component of most integrated design systems should be almost invisible to its users, providing many services in the system but unseen. The only point where it need be visible is to notify its user of model conflicts and allow a negotiated settlement.

8.3.2 Related mapping handler and controller work

The range of mapping handlers and controllers is as wide as the range of mapping notations described in Section 2.3.2. However, as none of the notations satisfy the requirements for a mapping language in that section, their environments can not provide for the requirements in this section. RDBMS provide the most sophisticated environments of any of the previously referenced notations, allowing tight management of modifications and their flow through to affected users. Though they can not perform the types of mappings required in this problem domain many of their facilities for management of transactions (including roll-back and roll-forward), view updates, traceability, and view interfaces are utilised in this thesis.

All of the mapping languages appearing in this domain (e.g., EXPRESS-M, EXPRESS-V, EXPRESS-X, Transformr, View mapping, etc) are associated with implementations. These implementations are very limited in their scope as they implement unidirectional mappings from one data-file of a model to the transformed data-file. Therefore they provide no management of global consistency of data in a project. Though all mappings in the COMBINE project were hand-coded their integrated building design system at least managed the mapping of data between tools (in a very conservative manner) to ensure that the IDM model was globally consistent.

8.3.3 Approach to a mapping handler and controller

The user interface of the mapping controller developed for this thesis is shown in Figure 8.1. This system implements mappings specified with the VML notation, utilising either a transaction-based, or individual, update mechanism to move data between related models whilst maintaining the consistency of the models. The mapping handler allows a VML mapping specification to be used in either direction to map data bidirectionally between models. Dependent upon the type of mapping specified, it will ensure that only coordinated updates are performed on an IDM (i.e., ensuring consistency of the IDM). It also automatically manages the notification of changes to all actors in a project who are affected by a particular modification to the IDM. The system allows both on-line and off-line DTs to be utilised with a mapping and incorporates full tracking of modifications, down to changes to individual attributes and permissions to change values at this level. Chapter 10 provides a full description of the implemented mapping handler for VML.

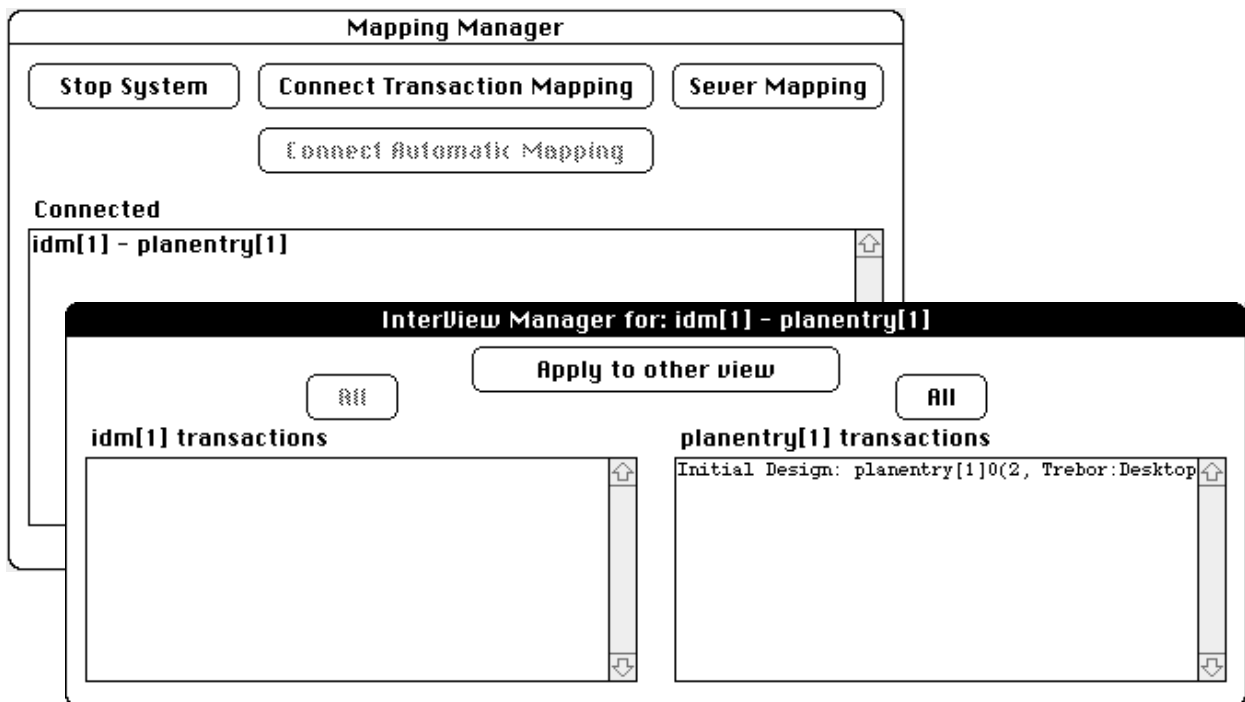


Figure 8.1 Sample mapping controllers

8.4 Design tool connection

This module provides the connection between design tools and the integrated design system. It should also be capable of invoking design tools when required, and determining when they have terminated. In a generic integrated design system it should be capable of performing these tasks over a range of hardware and software platforms, providing the interface to the integrated design system for all actors.

8.4.1 Requirements for a design tool connection system

A design tool connection system must offer the following facility:

Interface with IBDS: to lessen the workload on the actors all of the modelled requirements in Section 8.3.3.1 must be automated. For an off-line DT this module must be able to create the input files required for the DT (from the DT model in the system). It must then be able to invoke the DT to perform its work and finally retrieve the output from the DT into the DT model in the system. For interactive DTs it must perform the same task but driven by the demands of the design tool.

8.4.2 Related design tool connection systems

Traditionally this module is implemented by hand-coding the parsing and pretty-printing for each new design tool, and leaving it to the actors to run their particular design tools with the generated input. This is the approach seen in all of the EU funded projects (COMBINE, ATLAS, COMBI, CIMsteel, etc.). The systems of Amor (1991) and Pascoe (1994) provide the parsing and pretty-printing of data-files without the need for hand-coded tools (i.e., created based on the design tool data-file definitions). However, the design tool invocation still needs to be performed by the actor. No known system can detect design tool invocation and termination in a general environment (though TES environments (TES 1995) should, when implemented, provide this ability). However, it is possible for an IBDS to provide user activated procedures which allow it to be notified of a design tool's invocation and termination. Such a system is used in the implemented project flow of control module described in Chapter 11.

8.5 Flow handling

This module directs both the inter-model mapping and design tool interface modules to perform their tasks as required by the designers, or as directed by the project manager. This module simulates the flow of control defined in the project model, determining what design functions are able to be performed at any particular time. It also interfaces with designers to let them specify the tasks they wish to perform next, and interfaces with the project manager for decisions about which designers should be involved in new workflows in the project. The system directs the inter-model

mapping module to map data between different models, or to ascertain whether it is possible to perform the mapping. It also directs the design tool interface to invoke a design tool with appropriate data and to collect results upon termination.

8.5.1 Requirements of a flow of control manager

A flow handling system must offer the following facilities:

Calculation of possible design functions at any time: in a large project, with many design functions and where actors are working concurrently, it becomes very difficult to determine which design functions can operate without affecting current work. The flow handler must be able to simulate the flow of control specified with the tools described in Section 2.5. This initial simulation indicates design functions which are candidates to be performed at any instant in time. In a concurrent design environment many of these candidate functions will not be available due to possible conflicts with the data used by design functions already underway. Problems in the design are also registered by the flow handler which must backtrack through the specified flow paths to determine previous design states that can be worked from.

Visualisation of project state: the project manager must always be able to determine the current state of the project, and actors need to see what tasks they must still complete. Through coordinating a project the flow handler holds much information about the state of the project, both current and past. This information can be utilised to compare against expected or required deadlines as well as to estimate completion time for the project. Time-lines of expected versus actual time to completion of design functions can be derived from information held by the flow handler.

Control interface for actors and the project manager: all users of an IBDS need an interface with the system. The flow handler is likely to be the only interface the user has with the integrated design system. It will provide the actor with an interface for initiating design functions and for signifying the termination of design functions assigned to them. It will also allow the project manager an overview of the state of the project and the design functions being undertaken by different actors. This interface will allow the project manager to move and manage resources as required in the project to ensure its timely completion.

8.5.2 Related flow of control manager work

While Sections 2.5.2 and 2.5.3 describe several notations for describing flow of control in a project, as well as environments to define project flows in, only one of these systems follows through to a flow of control manager (TU Delft and Amor 1993). The majority of tools associated with the standard flow definition notations (IDEF0, IDEF3, Pert charts, state diagrams, etc.) either provide simulation capabilities or can output the model in a form understandable by stand-alone simulation tools. These simulations provide as output a likely schedule for a particular project, often presented in the form of a Gantt chart. Such charts are used by project managers to direct and

manage design functions on a project. As a paper system, however, they are not useful in managing design tools and the flow of information to and from them. They also become difficult to use when a very large number of design functions are specified on the chart(s), as the project manager needs to be mindful of all running and upcoming tasks and their relationships at the same time.

The CombiNet (TU Delft and Amor 1993) does have an associated management tool. This tool can load in a CombiNet definition and uses it to inform actors of their upcoming design functions, as well as to determine allowable design functions at particular times in the project. An initial prototype of the management tool was developed by the author for the COMBINE project. This prototype was later used as the specification for a more robust tool written in 'C' which was demonstrated at the end of the COMBINE project as the top level controller of their whole integrated building design system.

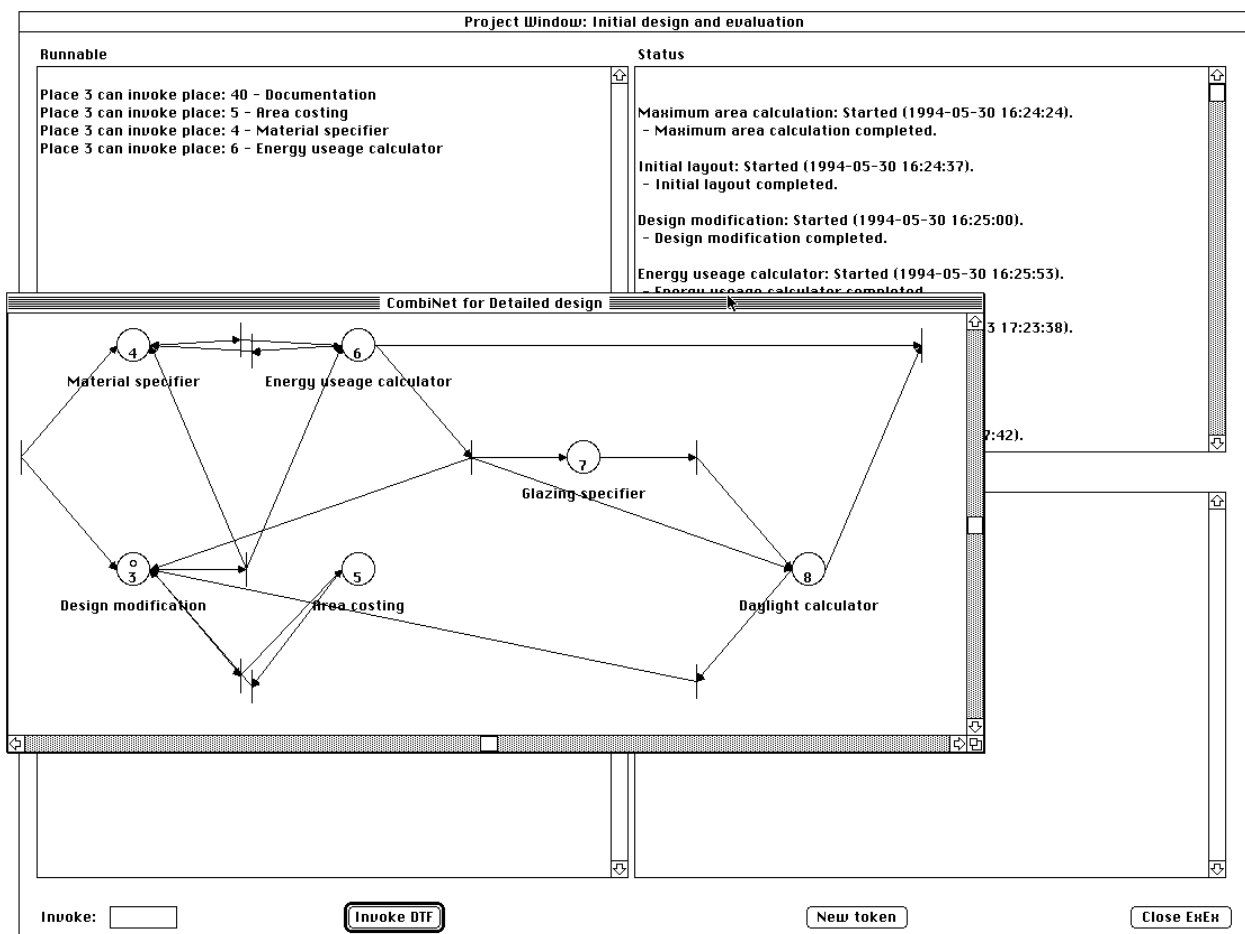


Figure 8.2 An operating flow of control manager

8.5.3 Approach to a flow of control manager

The flow of control manager, described in Chapter 11, is an extension of the system developed by the author during his work on the COMBINE project and provides for the majority of requirements as specified above. As seen in Figure 8.2, the interface provides a visual reference to an actor's location in a project flow, as well as the calculation of invocable design functions, those which are

stalled, and those which may be required to be reinvoked. Though the current project state is always ascertainable there is no global project state assessment or simulation. There is, however, enough information held in the flow of control manager to make it possible to output data which could be used with standard scheduling simulation software to calculate estimated time to completion. Actors and the project manager are provided with control interfaces of slightly different functionality. Project managers may investigate and control work over the whole project flow, whilst actors are restricted to the flows related to their design roles. As design tool invocation and termination can not be automatically determined by this flow of control manager, each actor is provided with an interface to specify when a particular design tool is working and when it has terminated.

8.6 Project Testing and Implementation Environment Summary

This chapter described the tools required to provide for the testing and implementation of the models developed in the types of environments specified in Chapter 2. The provision of tools and an environment as described in this chapter compliments the development of models in their modelling environment by allowing testing and development to be totally inter-connected. The tools described in this chapter taken as a whole provide an environment which has the majority of the functionality of an actual implementation of an integrated design system. Thus the validity of such an environment can be tested with all design tools and users of a final system prior to a commitment to a large implementation process. The major tools required for this testing and implementation environment are further described in the rest of this thesis. Chapter 9 describes schema instance maintenance in the Snart environment along with connections to EXPRESS and STEP Part 21 data-files, which have instance maintenance systems as well. Chapter 10 describes an implemented mapping system which can maintain data correspondences between many participating users, design tools and integrated data models. Chapter 11 describes a project management tool which controls flows of control between the various users and a project manager, helping to ensuring that the final designed artifact has been created with the input and attention desired by the client and project manager.