

Chapter 7

Project Modelling

To utilise the types of integrated design system that can be described with the modelling and mapping specifications described in Chapters 3, 5 and 6, it is also necessary to manage the tasks and people involved in the projects in which the integrated system is used. This level of modelling enables an integrated design system to be customised for use in a specific project. Project modelling is a task that must be undertaken at the start of every project as participants and their tasks will vary for almost every project. This task is usually assigned to a project manager, though project models must be checked and accepted by all participants.

The work on project modelling presented here includes and extends work undertaken by the author during a six-month visit to the COMBINE group at TU Delft in the Netherlands, the results of which are reported in TU Delft and Amor 1993, and Amor and TU Delft 1993.

7.1 Introduction

A project encompasses all stages of work from inception through to demolition and possible reuse of a particular artifact. To manage and understand what happens in a project, a model is required of the various actors involved and the responsibilities and tasks they play in the project. The development of a project model also provides an understanding of which data models are required in a project and what data transfer is required between these models. This influences the work done on schema development and on mappings between schemas. A project model provides a formal description of the various users who will be involved in the project, and formalises the roles they will be fulfilling in the design task. The various design roles can be further refined to individual design functions which can, in turn, be associated with the design tools available to

perform them. The set of design functions must be able to be scheduled, to allow a project manager to ensure that design policies are followed, quality assurance conventions are maintained, handover points between contractors are formalised, and design progress can be monitored. Without the ability to model and manage these aspects, an integrated design system offers a meaningless data transfer mechanism unrelated to a real-world project.

A large design and construction project involves many development phases, some of which may be independent of each other, and, in many cases, need not be specified (and may not even be known) at the start of the project. To understand and manage the large scale of projects, and their initial indeterminacy, they are often conceptualised as happening in stages, though these stages often overlap. Common stages in a project may include inception, management, design, construction, maintenance and demolition. Stages may be very broad, or quite specific; for example, the structural design stage in a particular design office. In COMBINE, the term ‘project window’ was introduced to describe these coherent portions of a project (see Figure 7.1). This term is used throughout this thesis. The project windows represent a given time-slice of the project, or some sub-process. Therefore, a project is considered as being divided into multiple, conceivably overlapping, project windows which are specified prior to the execution of a new phase of a project.

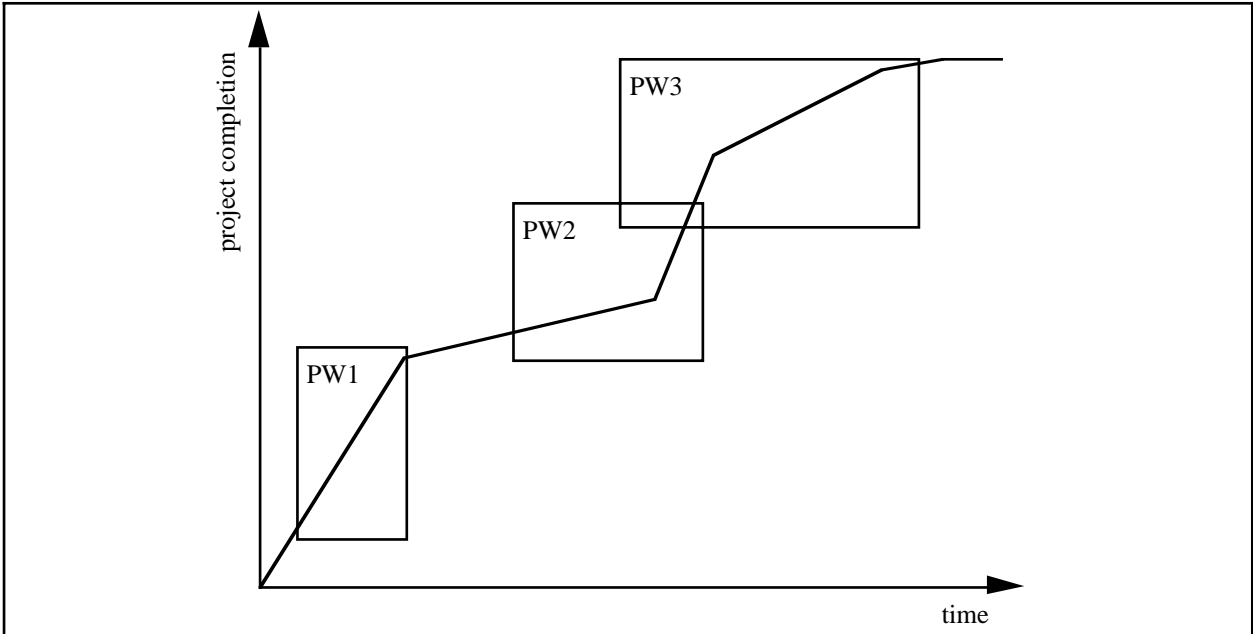


Figure 7.1 Multiple project windows in a project

Even splitting a project into multiple project windows is unlikely to provide enough flexibility for real projects. A single project window may model weeks of work, over which period the participants in the design team could change (e.g., bringing in an expert to help with unforeseen problems) and various flows between design functions may need to be modified (e.g., to ensure new aspects of the design are checked). To cope with this variability, a previously specified

project window model must be able to be modified and updated as the design progresses, with immediate flow-on effects to the running control system.

The modelling of projects and project windows requires many aspects of a project to be captured. Other research in this area has been examined to provide an understanding of the requirements for project modelling. The notion of project models outlined above has great similarity to the concepts of project and process in software engineering. Curtis et al. (1992) provide a review of process modelling which categorises requirements and various approaches to process modelling. Their four most commonly utilised perspectives in process representation (functional, behavioral, organisational and informational) are used to rate previous process models, and are also used to evaluate the CombiNet model developed here. The meaning of these four perspectives (from Curtis et al. 1992, page 77) are:

Functional: represents what process elements are being performed, and what flows of informational entities (e.g., data, artifacts, products) are relevant to these process elements.

Behavioral: represents when process elements are performed (e.g., sequencing), as well as aspects of how they are performed through feedback loops, iteration, complex decision-making conditions, entry and exit criteria, and so forth.

Organisational: represents where and by whom (which agents) in the organisation process elements are performed, the physical communication mechanisms used for transfer of entities, and the physical media and locations used for stored entities.

Informational: represents the informational entities produced or manipulated by a process; these entities include data, artifacts, products (intermediate and end), and objects; this perspective includes both the structure of informational entities and the relationships between them.

	Functional	Behavioral	Organisational	Informational
Procedural programming languages	*	*		*
Systems analysis and design	*		*	*
AI languages and approaches	*	*		
Events and triggers		*		
State transition and petri-nets	*	*	*	
Control flow		*		
Functional languages	*			
Formal languages	*			
Data modeling				*
Object modeling			*	*
Precedence networks		*		
CombiNet	*	*	*	*

Table 7.1 Capabilities of project specification languages (after Curtis et al. 1992)

Curtis et al. (1992) evaluate different styles of project specification against these perspectives, summarising their results in tabular form. Table 7.1 replicates this summary (where asterisks represent the ability of a language to support a perspective) and extends it to include the work presented here (CombiNet entry). The styles of project specification presented below cover all the different process notations and tools presented in Sections 2.5.2 and 2.5.3.

7.1.1 Requirements

Considering the four most common perspectives offered by Curtis et al. (1992) led to the definition of a set of views that must be able to be supported. These views include users, tasks, data and workflow. The requirements of these views are as follows:

User View: all the actors involved in a project window and the tasks they must undertake to ensure completion of the project window role. This view must capture the responsibilities of the various actors and their rights in terms of viewing and modifying information in a project.

Task View: all design functions that can be performed by the design tools within the project window. The granularity of a design function can vary from atomic changes to a design, through to a complete design. The level of granularity is determined by the requirements of individual projects. The only fixed requirement is that each design function must happen between a starting and ending exchange event of information with respect to the integrated design system. Everything that goes on between these exchange events is invisible in the project window model, being the domain of the design function. Thus, design functions can be both on-line atomic CAD design tool operations and off-line batch mode design tool runs.

Data View: all schemas concerned with design functions and users. In the project windows described here this includes:

- The complete IDM schema.
- The IDM subschemas corresponding to the input of design functions.
- The IDM subschemas corresponding to the output of design functions.
- The IDM subschemas corresponding to the input view of actors.
- The IDM subschemas corresponding to the change access of actors.

This offers a purely static view of the project window definition, not addressing aspects of control over instances (unless it can be modelled in the schema). The subschemas do, however, define the state that the project must reach before a particular design function may be utilised, or before an actor can be called into the project.

Workflow View: defines all possible flows from a particular point in a project. It ties the design functions together and provides the way of describing the handover between various actors in a project. The workflow defines how tightly managed and controlled a project is going to be, from highly specified to very open. To enable the workflow view to be used, a management system will have to be able to determine all states of a design function, the state of the integrated design system, and control over exchange events according to control flow constraints. This will include the following aspects:

- whether a design function is being performed.
- whether a design function is a candidate to be invoked (in view of design functions that it depends on).
- whether a design function is a candidate to be re-run because the running of another design function changed its original input.
- the design function state and integrated design system state resulting from previous exchange events.

The views specified above describe aspects of a project which must be modelled to ensure that required project perspectives can be supported. However, they do not provide a measure of the level of control that will be exerted on a project through the views. The level of control could vary widely, from very rigid and autocratic management through to very free and autonomous management. Current perspectives on this include:

- No control as typified by current integrated design systems. These systems are merely able to transfer data around without any in-built process context and purpose of data exchange events. For such a system to be operational in a particular design project, a project manager needs to establish work procedures, task scheduling, etc. among the team of users. The system is an insensible data router which provides no guidance on what should be done next.
- Shallow control is seen in several of the more recent integrated design system efforts, e.g., COMBINE, and ToCEE. This approach uses a scheduler which deals only with the control over who, or what task, is next, and which monitors the states resulting from exchange events. At this level of control, a project manager can enforce much rigour in the use of the system. Although this might suit project windows for parametric (routine) design, it is to be expected that most project windows will require a more flexible approach, where the human users may interact with the control layer as well as with each other (outside the system).
- Deep control is the level that integrated design system developers aspire to and AI researchers still ruminate over. This type of control deals with much greater complexity than just scheduling. Deep control must deal with the pre- and post-conditions which are related to exchange events. It should have the ability to look inside exchange events and DT interfaces, and propose remedial actions for any failure of a DT invocation or analysis, or propose and control how the system can recuperate from deadlocks in design scheduling. Deep control should also deal with declarative knowledge on the meaning and purpose of design functions and thus support goal-driven design strategies.

A deep control system, while perhaps being the type of control system that should be aimed for in the future, is out of the scope of this thesis. Deep control, as defined above, requires the solution of many hard research problems in the artificial intelligence field. As a first step towards introducing process control into integrated design systems, shallow control has been selected for

use in all the large EU funded projects and this is the approach implemented by the author and shown in this chapter.

7.1.2 Structure

To cover the four most commonly utilised perspectives specified in Section 7.1, and meet the requirements of Section 7.1.1, the CombiNet formalism developed here defines three main types of information to be modelled for a project window: project window requirements, defining part of the informational perspective; user requirements, defining most of the functional, organisational and informational perspectives; and flow of control requirements, defining the behavioral perspective. Project window requirements allow the specification of starting conditions for entry into a project window, and exit conditions (e.g., data that is required by the end of the project window). User requirements allow the specification of the participants and their design functions in a particular project window. These participants (actors) perform certain design roles in a project and these design roles can be completed through the application of various design functions. A design function can be represented as a particular design tool used in a certain manner, e.g., to perform one type of analysis from the range a design tool offers. Flow of control requirements allow the specification of the paths that may be followed between various design functions to complete the design phase encapsulated by the project window.

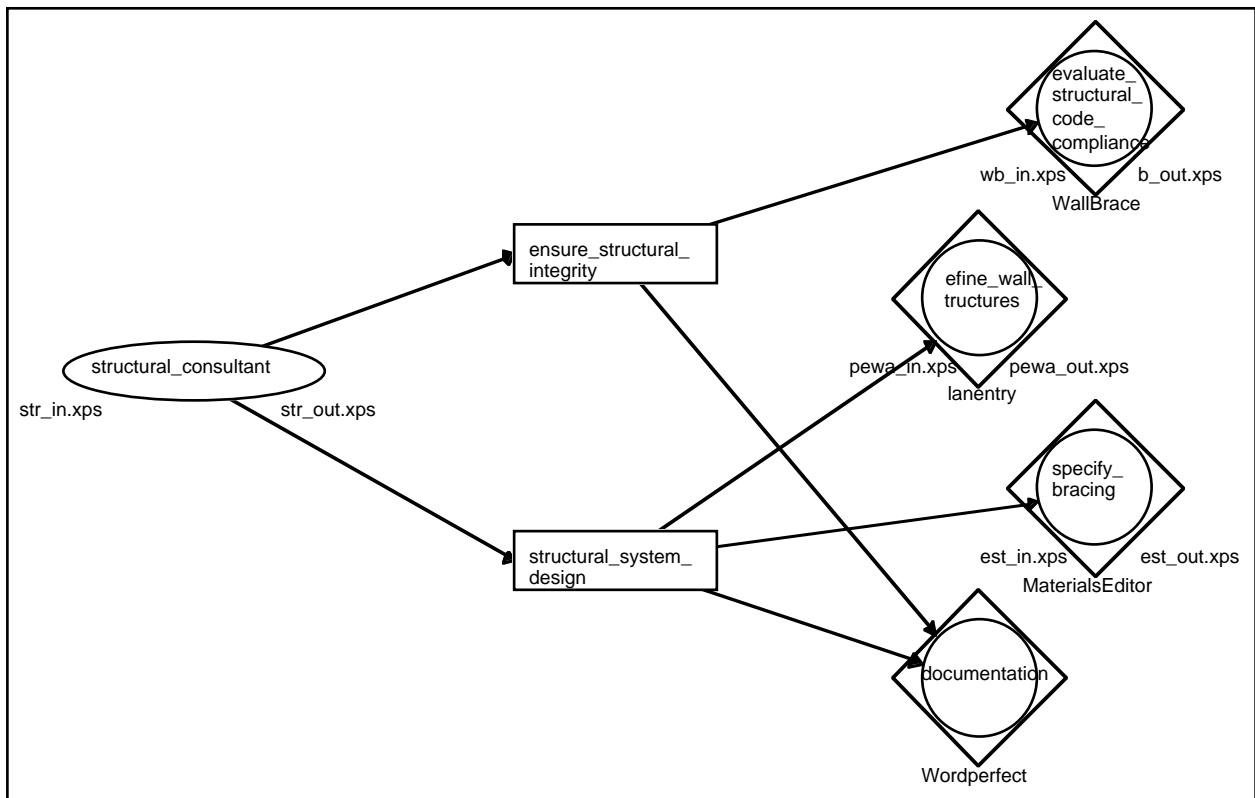


Figure 7.2 Example of user and function specification

The CombiNet formalism utilises two specification notations to model user requirements and flow of control requirements. Though two notations are described, they are used in an integrated manner in the specification environment, with explicit links between diagrams in both notations.

7.2 User and Function Modelling

A majority of the functional, organisational and informational perspectives, as introduced in Section 7.1, can be grouped together and defined in a single formalism. Functional and informational perspectives can be defined for design functions, actors and the project window through the definition of input and output models. These schemas specify structures and constraints of the models used by, and produced through, the use of a particular design function, actor or project window. Organisational perspectives can be partially defined in the same model through connections between actors, their design roles, and the design functions required to complete the design roles.

In Figure 7.2, a graphical specification of user and function, detailed in a CGE (Configurable Graphical Editor, Vogel 1991) project window formalism (developed by the author), is presented. There are three main types of icon in this diagram, connected by arrows representing *performs* and *requires* relationships for actors and design roles respectively.

Actor: the oval icons in the left column of the diagram represent actors (users) participating in the project window being specified. An actor has a name defining either the actor or, as in Figure 7.2, the type of actor which will be performing particular roles in the project window. Each actor is associated with a pair of schema defining both the subset of the IDM that they are able to view, and the subset of the IDM that they are allowed to modify. These two schemas define the area of responsibility of a particular actor, and are used to ensure the actor acts within a specified domain. These schemas can also be used to check that the roles an actor plays are not outside the actor's area of responsibility. Actor responsibility can be checked against design role responsibility, as the area of responsibility of a design role can be determined by the union of the schemas associated with each design function it utilises. An actor is associated with one or more design roles. In Figure 7.2 each actor has unique design roles, but this is not mandatory, multiple actors are allowed to perform the same design role.

Design Role: the rectangular icons in the centre column of the diagram represent the various design roles which are performed by actors in this project window. Each named design role can be fulfilled through the application of various design functions (the scheduling of which is specified at a later stage). Several design roles can utilise the same design function in the completion of their role. For example, the documentation design function is used by most of the design roles.

Design Function: the icons in the right column of the diagram represent the various atomic design functions which can be carried out in the completion of design roles in the project window. A design function has a name defining the type of function it performs and directly under the icon the name of the design tool which will be used to perform the named function. Each design function is associated with two schemas defining both the subset of the IDM which will be the input to the design tool, and the portion of the IDM which can be updated at the completion of the design function. These two schemas define the responsibilities of the design function. Though not shown in Figure 7.2, several design functions can be accomplished with the same design tool, but often with different input and output schemas defining responsibilities for the tool. The definition of the different functions performed by the same design tool is indicated by the two schemas specified along with the design function.

In the CGE environment the modeller can navigate from this user and function view to the top level flow of control view through a menu item in the environment. The CGE environment is not as sophisticated as the MViews environment utilised in Chapters 3 and 6, limiting the type of environment that can be offered to the modeller. The main restrictions are that it can only provide a single view of a model (i.e., one user and function view) and sophisticated navigation facilities are not available (e.g., it is not possible to navigate from a design function to all flow of control views which reference that design function).

7.3 Flow of Control Modelling

A Petri net formalism provides much of the behavioral perspective required for project specification, and, along with the DT schemas for actors and design functions defined in the user specification detailed in Section 7.2, many properties of a project's state can be calculated. In this section the calculable properties of a flow of control specification are defined, along with a formalism for describing them. This formalism is based around the design functions defined in the user specification. However, it also overlays actor's areas of responsibility to provide the link to organisational perspectives for the project specification.

7.3.1 Set theoretic background for flow of control

The control view states defined in Section 7.1.1 are calculable from analysis of data flows in the integrated design system based on the schemas associated with each design function (DF). The basis of the analysis is to assume that the input and output schemas for a design function together describe a subset of the IDM schema. This is not strictly true, as the schema for a design function is likely to have cardinality constraints, keys and value constraints which differ from the IDM. However, these added constraints can be ignored when checking subset relationships and when

performing intersections of various schemas. What will be used for the set operations will be the definition of entity names and their inherited entities, and attribute names and types which appear in the design function schemas and the IDM. In the following conditions and constraints i_{In} is used to denote the input schema to a design function or actor, and i_{Out} to denote the output schema of a design function or actor.

There are two conditions which must hold on the design function and actor schema definitions:

Condition 1: $\forall i \in (DF \cup Actor) (IDM \supseteq i_{In}, IDM \supseteq i_{Out})$

Condition 2: $\forall i \in Actor, \forall j \in DF \text{ of Actor } i (i_{In} \supseteq j_{In}, i_{Out} \supseteq j_{Out})$

This just states formally that the input and output schemas of all design functions and actors must be a subset of the IDM, and that the input and output of any design function used by an actor is a subset of that actor's schemas. In practice, this means that the input and output schemas must be defined in terms of the IDM (i.e., using the same entity, relationship, method and attribute names), as well as being defined by the model structure used in the actual design tool or actor view. Using these definitions enables a static check of all schemas in the integrated system. The schemas can be checked against the IDM to determine whether they are valid subsets of the IDM (i.e., whether the schema has been defined properly. This will mainly pick up typing errors). The allowable differences in schemas of a design function or actor over that of the IDM are that the former may define: different uniqueness constraints (keys); different constraint clauses or ones which are additional to those defined in the IDM; different cardinalities on attributes; and different optional specifications for attributes.

Given a system which contains the IDM schema and the input and output schemas of the various design functions used in a particular project window, and the definition of a flow of control for a given project window, there are various properties which can be calculated. To derive these properties from the design function schemas two constraints are defined:

Constraint 1: $\forall i \in \text{running DF}, \exists c \in DF (i_{Out} \cap c_{In} = \emptyset)$

This constraint is a concurrency check, or an invocation check, stating that a design function (c) is a candidate to be invoked if its input schema has no intersection with the output schema of any of the running design functions (written as running DF in the constraint).

Constraint 2: $j = \text{completed DF}, \exists i \in \text{previously run DF} (i_{In} \cap j_{Out} \neq \emptyset)$

This constraint is a re-invocation check, stating that if the output schema of a design function which has just terminated ($DF_{j_{Out}}$) intersects with the input schema of any other design function which was previously run ($DF_{i_{In}}$), then the previously invoked design function (DF_i) is a candidate to be rerun.

It must be noted at this point, that intersections between design functions are described only at the schema level (i.e., static determination). Where the design functions require a model of a full building this will be sufficient to determine the properties detailed above. However, if a design function models only a small portion of a building (e.g., calculates properties for a single space) then the properties calculated above could present a design function as a candidate to be rerun in more cases than necessary. In the implementation of the flow of control system, this is handled by also tracking the objects which are used by each design function. The intersections between design function schemas gives a static determination of whether a design function needs to be further examined at run time to determine the properties defined above.

The working of the two constraints described above is illustrated in the figures below:

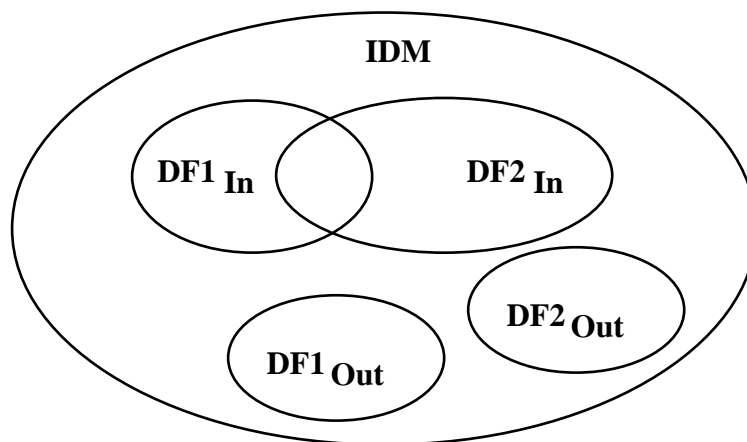


Figure 7.3 Invocable design functions

Figure 7.3 presents the situation where, even though the two design functions share data in their input schemas, neither of the design function input schemas have an intersection with a design function output schema. In this case, both design functions may run concurrently, and the result of either tool will not cause the re-invocation of the other design function.

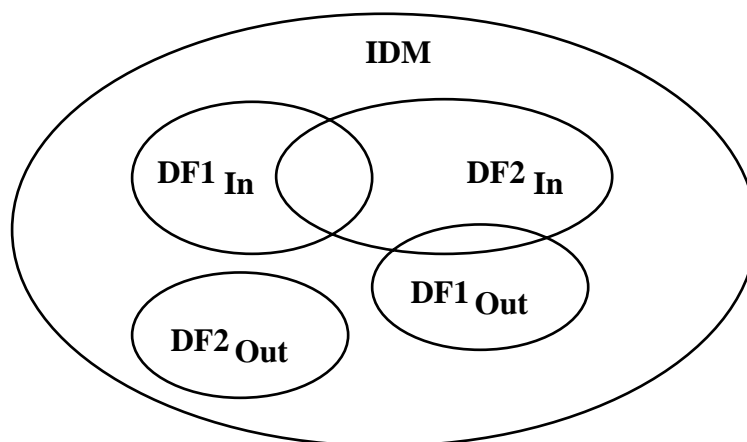


Figure 7.4 Constrained design function

Figure 7.4 presents an example where there is an intersection between an input and output schema of two design functions. In this case, if DF1 is running then DF2 may not start. This figure also illustrates what may happen with Constraint 2. If DF2 was run at some time in the past, and then DF1 is run, then the output of DF1 may overwrite what was previously supplied to DF2, so DF2 becomes a candidate to be rerun.

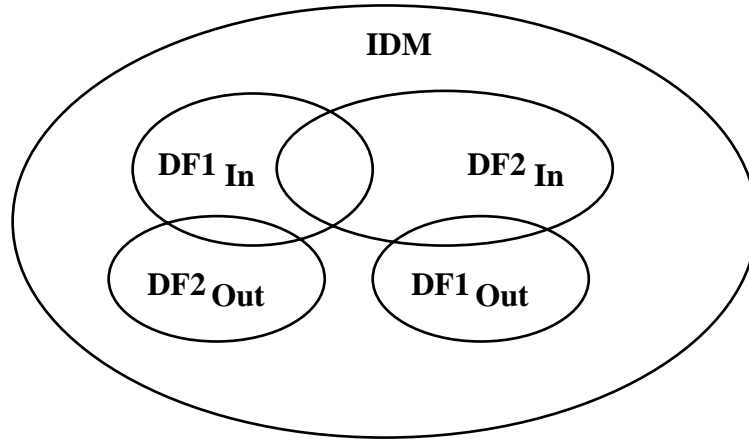


Figure 7.5 Constraints between two design functions

Figure 7.5 presents a case where the two design functions may not run concurrently, as the input schema of each design function intersects with the output schema of the opposite design function. This figure illustrates a situation where the invocation of either of these design functions can cause the other design function to be a candidate for a rerun, apparently causing a cycle. However, this is not problematical as the design functions are only candidates to be run. The final decision of what can be run at any particular time is made via the project window control flow information.

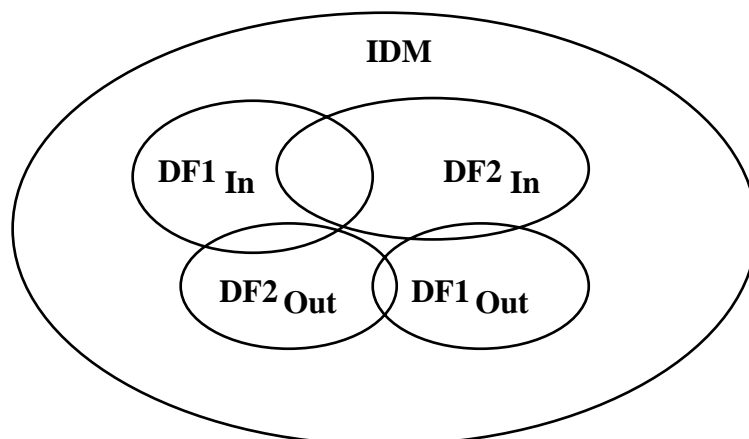


Figure 7.6 Design function constraints leading to apparent inconsistencies

Figure 7.6 presents a definition which could apparently lead to inconsistencies. The two design functions have the same properties as those in Figure 7.5, but with the additional complication that they overwrite portions of each others output. Again, this is acceptable, as the running of each of these design functions is determined by the project window definition which by its own definition gives rights to a design function to place its output in a portion of the resultant data store. The

termination of either design function will mark the opposing design function as a candidate to be rerun, making explicit the fact that the output of the design function has been overwritten.

While these examples demonstrate the interaction between just two design functions, the constraints described earlier are general and the results can be applied over any number of design functions.

7.3.2 Flow definition

The graphical formalism developed to specify flow of control in a project window is based very loosely on the Petri net formalism (Jensen 1990). Icons representing places, transitions and tokens are used in the modified definition, along with new icons to allow further functionality to be described. Although the new formalism (called a CombiNet) looks very similar to a Petri net, the semantics are very different. The icons in this formalism and their functionality are described below, with reference to Figures 7.7 and 7.8.

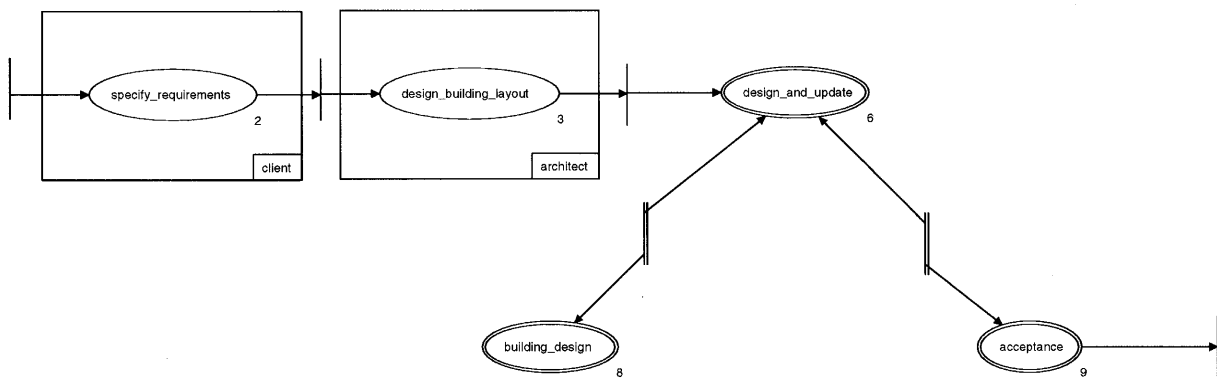


Figure 7.7 Top level flow of control specification

Place: A place in the CombiNet (the single lined oval icon in Figure 7.7) represents a single design function. It can be reached from some set of transitions and it will exit to some set of transitions. Each place represents exactly one design function, but the same design function can appear many times in a CombiNet.

Aggregate Place: An aggregate place (the double lined oval icon in Figure 7.7), as its name suggests, represents a set of icons defining an identifiable subset of the process being modelled, defined here to be all icons drawn in a window. Thus, allowing the modeller to group complex interactions into logical subnets and reference them through a higher level mechanism. Aggregate places representing the same CombiNet may be referenced from any number of CombiNets or many times from a single CombiNet. This allows a common sequence of events to be represented by a single icon in a CombiNet. It is also used to distinguish the flow of control of different actors, or different design roles, into separate definitions which can be linked together appropriately at a higher level CombiNet.

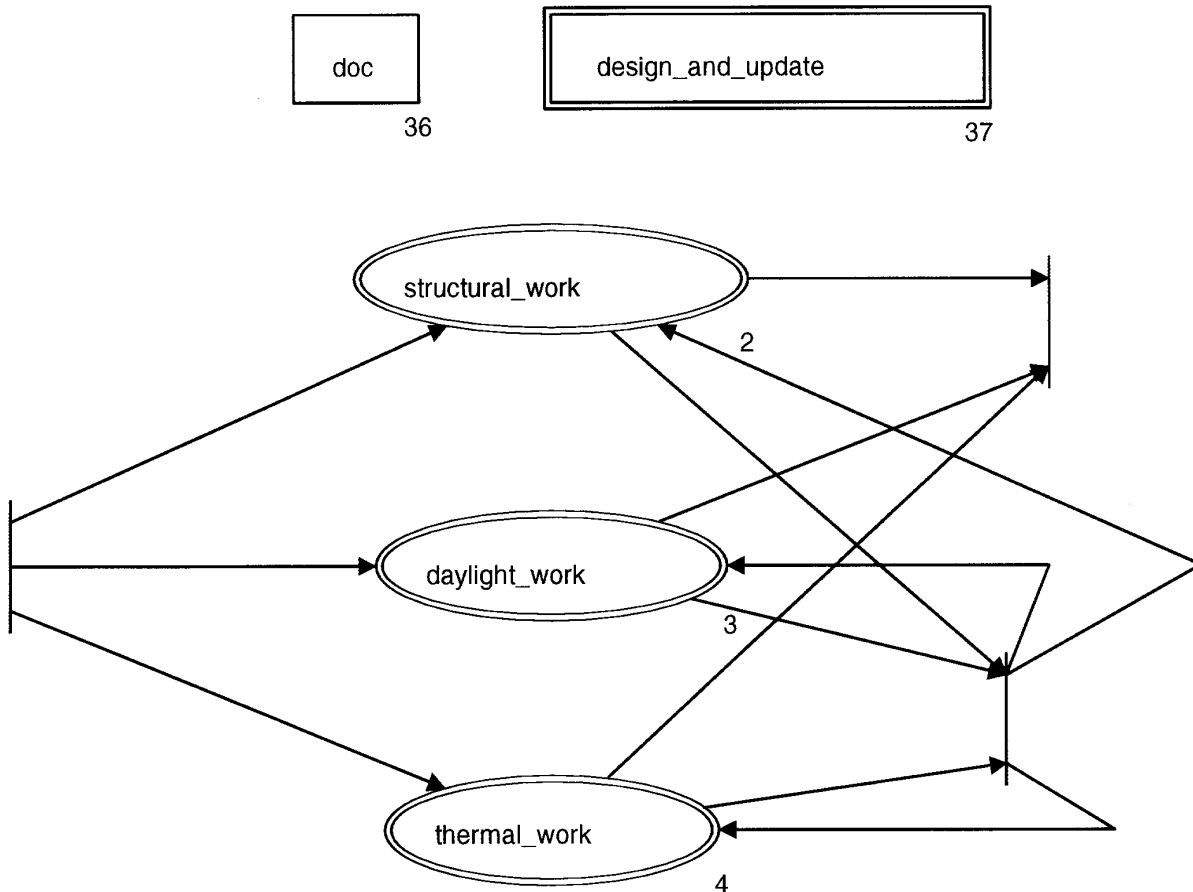


Figure 7.8 Flow of control with global elements

Global Place: The global place (the single lined rectangular icon in Figure 7.8) represents a single design function exactly like a place, but it has no connection to the transitions. The global place denotes a design function that can always be activated when in the current CombiNet (or in any descendant CombiNets, i.e., those represented by an aggregate place or global net). Global places allow design functions which can always be invoked (e.g., documentation functions) to be specified in the CombiNet without having to draw links from every single place and aggregate place in the diagram. It also allows a specific flow semantics to be associated with the global place. This semantics is that after invoking the design function in a global place the flow of control passes back to the place or aggregate place which invoked the global place (rather than any place or global place in the diagram, as would occur if everything was wired back and forth to a single place to achieve this effect).

Global Net: This is a combination of an aggregate place and a global place. The global net icon (the double lined rectangular icon in Figure 7.8) represents a complete CombiNet and is invocable from any place in the current net, and all descendants of the current net. When inside the global net all other global elements, other than those that are already entered (i.e., a global net can not call itself recursively), can be seen. The set of visible global nets reduces as each one is stepped into, and grows as each one is completed. The CombiNet representation of a global net is the same as a normal net, so it can have its own global

places and global nets as well as normal place-transition flows and aggregate places. A global net is used for many of the same reasons as a global place (mainly for functions which can occur at any time). Its main benefit is that a whole flow of control can be defined in the global net rather than a single design function. For example, after completing some set of design functions in a global net it would be possible to force the flow of control through an evaluation function before returning control back to the original CombiNet. The semantics of entering and exiting a global net are the same as for a global place.

Transition: A transition in the CombiNet (the single vertical line icon in Figure 7.7) works as the choice point between one design function invocation and the next. It can be used to work out the set of candidate design functions that can be invoked after a running design function has terminated. There are two special kinds of transition that can be created. A transition which has no entering arrows (i.e., is not exited to by any place or aggregate place) is called a start transition and denotes an entry point into a CombiNet. A transition which has no arrows exiting to other places is called an end transition, and denotes an exit from a CombiNet. In the current version of the project window definition, a CombiNet is allowed to have only one start transition, though it may have any number of end transitions. Having a single start transition in a CombiNet is purely a reading convenience for a user, as this guarantees that having found one start transition the reader has found the only entry into the CombiNet (a single CombiNet can be very large and complicated and it could be easy for the modeller to overlook some starting transitions if more than one was allowed). A single start transition also eases the computation required when attaching to aggregate nets. All transitions which invoke an aggregate place are connected to the start transition of the CombiNet represented by the aggregate place. In a similar manner, the exits from an aggregate place to a transition are all tied to the set of end transitions in the CombiNet represented by the aggregate place.

Double Transition: The double transition (the double vertical line icon in Figure 7.7) is a shorthand notation for a loop between two places, which is represented by two normal transitions, one in each direction, in a running system. In Figure 7.7 the two double transitions provide two design function choices when exiting the *design_and_update* aggregate net, either *building_design* or *acceptance*. If the *building_design* aggregate place is entered, there is only one exit, back to *design_and_update*. If *acceptance* is entered then there are two exits, either back to *design_and_update* or to terminate the CombiNet (and in this case the whole project window).

Actor: The actor overlay (the large box icon with the actor name in the lower right corner, see Figure 7.7) defines the actor responsible for a set of places, aggregate places, global places and global nets in a CombiNet. This allows a clear distinction to be made between areas of responsibility of various actors, and indicates the point of transfer of control from one actor to the next. The actor overlay is needed in cases where multiple actors and design roles utilise the same design function to perform their tasks. In this case, without the

overlay, there is no way of determining the actor responsible for a design function in the CombiNet. When implementing the flow of control, moving from the design function of one actor to that of another is treated as indicating an acceptance and sign off of all responsibilities for the initial actor. This includes accepting and terminating all outstanding re-run requests for the actor handing-over control. However, the new actor will see re-run requests generated by modifications that affected run states of their previous design function work (if any) since the last time they were involved in the project. New re-run requests will be added for the new actor as they work through their design functions, as in the previous definition. Again, at the hand-over to another actor all current re-run requests for the completing actor are removed. This helps control the proliferation of re-run requests caused by changes which may impact on every design function which has previously run. This management of re-run requests is local to a single project window, and will not affect the same actor's work in another project window of the same project.

Although tokens are never shown in the CombiNet diagrams, they are used in the implementation to illustrate the flow of control in the CombiNet. A token represents the actions of one actor performing their design roles with various design functions. A token in a place represents the running of a particular design function. When the design function has terminated, the transitions are used to determine the candidate design functions from the current place. When the next design function is decided upon, the token crosses the choice point created by the transitions and invokes that design function. If a design function is unable to start, or terminates unexpectedly, the token is sent back to the place it previously came from, and transitions available from that place must be re-evaluated. If a token moves to a global place or global net then after completing that function it will return to the place from which the global function was invoked. A token enters a CombiNet from a start transition and leaves on an exit transition. In every flow of control specification there is a top level CombiNet from which a single token is started. When a token moves across a boundary between areas of responsibility of different actors then the user the token represents changes, and this represents a formal hand-over between the two actors.

The differences between the working of a Petri net and a CombiNet are seen most clearly in the flow of tokens. In a Petri net there must be a token in each place entering a transition before a token can cross the transition. When a token crosses a transition it places a token in every place pointed to by the transition. In the CombiNet it is completely different; a token, which represents an actor, is only constrained from traversing transitions by the state of concurrency constraints as defined in Section 7.3.1. Transitions provide a choice point between functions in the design. Also, as a token represents the workflow of an actor, the same number of tokens are passed over a transition as approach it. Therefore the number of tokens (workflows of actors) in the project window remains static in the flow of control system unless a new workflow (which would require a new token) is activated by the project manager (e.g., concurrent design in the project window). As the project manager is shown the analysed state of a running project window, it always clear

when multiple workflows can be scheduled concurrently. This is shown by the set of design functions calculated able to be run at the same time as currently running design functions.

7.4 Appraisal of Project Specification

The two part project specification formalism described above has been used (see Augenbroe 1995a and the example in Appendix E) to describe, and then implement, a shallow level of control in integrated design systems. The formalism, although simple, allows the definition of complicated project models and actor interactions. In relation to the four perspectives of project representation it meets them in the following manner:

Functional: this perspective is addressed in both notations. The definition of what process elements are performed can be ascertained from the set of design functions specified in the flow of control diagrams. The informational entities relevant to each process element are specified through the input aspect model of each design function in the user and function diagram.

Behavioral: this perspective is addressed in the flow of control diagrams. The CombiNet allows specification of all sequencing requirements of design functions as related to the design roles of individual actors in a project.

Organisational: this perspective is partially addressed in the user and function diagram. This hierarchical diagram associates actors with specific design roles, and through those design roles to individual design functions. CombiNet's actor overlays are used to distinguish actor responsibilities for particular design functions where multiple actors need to utilise the same design function. Actor responsibilities are rigidly defined through the use of aspect models defining an actor's view of a schema and the schema components they are allowed to modify. Physical communication mechanisms and physical media are not addressed in this formalism as this is currently managed independently in integrated design systems.

Informational: this perspective is addressed in the user and function diagram. The definition of an output aspect model specifies what is produced by a particular design function, while the input aspect model defines what is manipulated by a particular design function.

The CombiNet formalism for flow of control specification provides a simple yet powerful notation to define control flow in a project. The main benefits of the formalism are:

Explicit flow of control: The use of places and transitions allows for the definition of explicit paths through the design process. There can be iterations in the design process and choice points in the direction that the design progresses. The flow of control can be tightly constrained to follow set paths and invoke certain functions in a specified sequence.

Aggregation: The introduction of aggregate places reduces the complexity of individual nets by allowing subnets to be identified and packaged as individual components. The aggregate

places allow common tasks or processes to be described once and referenced many times throughout the design process. Aggregation also allows processes pertaining to individual actors to be specified separately from all other actors, providing a very visible separation between actor tasks.

Global icons: These icons provide a simple notation to encode flows of control which can occur at any time in the design process. Multiple global nets in a CombiNet represent a wiring structure which would be very intricate and cumbersome to define with other process models.

Wide range of control: The combination of global components and connected components provides for a wide range of control strategies to be implemented. These range from totally unconstrained, through to totally constrained. The various states are characterised by:

Totally unconstrained: all states are defined by global places, so that anything can start at any time. Flow between various actors in the system is still monitored, but any actor can do anything at any time.

Partially constrained: some states are defined by global nets, therefore invoking certain states requires passage through other states in a controlled manner.

Partially unconstrained: most states are described through the usual place-transition flows, with some global places or global nets to represent states that can occur at any time.

Totally constrained: there are no global places or global nets defined, so all states must be invoked by following the flows defined in the place-transition flows.

Well defined hand-over points: the hand-over of control between different actors in a project window is explicitly modelled in the flow of control specification. This provides well defined and easily distinguishable areas of responsibility in a design project.

There are also certain weaknesses to the formalism as developed, the most important being a poor handling of complicated boolean conditions between design functions in a CombiNet. For example, it may be required to inhibit the execution of a particular design function until a set of other design functions have executed, without consideration for the order in which this other set of functions is executed. This type of specification will prove hard to code in this formalism due to the ability to have loops in the flow of control. For example, does satisfying all the conditions for the first time make these conditions satisfied every other time they are encountered? Although these types of conditions can not be made explicit in the formalism, a very weak form of these conditions can be implicitly defined in the input schema definition of a design function. An input schema can specify a requirement for input which could only be met through the execution of a given set of design functions (again only useful for the first iteration in the flow of control). An approach to alleviating this problem would be a formalism for explicitly specifying the set of conditions that must hold before a design function is invoked. This would allow for both the semantics of repeat invocation to be specified and for full system-state, and data-state, pre-conditions to be defined.

The project specification provides a way of describing shallow control in a project with a degree of

flexibility sufficient for small through to large projects. However, there are aspects of the specification which could be extended:

Deep control: the shallow control offered in the current project specification should be extended towards the requirements of a deep control system. Initially this would cover greater flexibility for specifying constraints on the project model. These could be constraints to be satisfied before the invocation of particular design functions or constraints which guide the flow of control based on the results of a design function. A large amount of research would be required to try to implement a more comprehensive control system based on design intent and it would have to be carefully crafted so as not to intimidate the users of such a system.

Greater specification of environment in model: an assumption in the project specification was that physical communication mechanisms and physical media need not be addressed in the formalism, as it is currently managed independently in the implemented integrated design systems. This assumption could be lifted to provide a more general project specification which can model design tool parameters. For example, the environment in which they operate, their invocation parameters, the format of input and output data-files (if they use data-files), etc. Previous research has considered this problem and the papers on TES (1995), and to some extent by Pascoe (1994), describe notations which allow the majority of these parameters to be defined.

Flexible specification environment: the CGE environment was used by the author to implement this formalism as it was the modelling tool used in the COMBINE project. Re-implementation in an MViews-like environment would provide a more sophisticated modelling environment, allowing multiple overlapping views of the flow of control specifications, as well as all the coordination and documentation benefits offered in an MViews environment. An MViews environment would also offer the possibility of tying the project specification through to the schema and mapping specification environments described in Chapters 3 and 6. This type of specification environment has subsequently been demonstrated in the Serendipity system (Grundy 1996). Serendipity allows the specification and coordination of process models alongside software development for multiple developers, with consistency management and developer conflict resolution and management.

In summary, this chapter presents the requirements for project specification in an integrated design system as well as a formalism to allow its specification. This formalism allows a shallow level of control to be specified between actors in a project, the design roles they play, and the flow of control between design functions used to satisfy particular design roles. The project specification, along with schemas (from Chapter 3) and mappings (from Chapter 5 and 6), is sufficient to implement an integrated design system which can be used in a specific project. Chapter 11 describes such a control system and gives examples of its use with the project definition used as

examples in this chapter, and fully specified in Appendix E.