Project Control in Integrated Building Design Systems

Godfried Augenbroe* and Robert Amor**

*Georgia Institute of Technology, College of Architecture
Atlanta, GA 30332-0155, USA
fried.augenbroe@arch.gatech.edu

**Building Research Establishment Limited, Garston, UK
trebor@bre.co.uk

Abstract

With the fruition of research into integrated design systems that allow communication between multiple actors and design tools, there is a growing need for control over the flow of execution of project tasks performed with the help of these systems. This calls for the specification of configurable control systems that are able to 'design' the organization and management of actual projects.

The paper argues that an IT driven approach, leading to narrow software engineering solutions, is inadequate to solve the new type of project management problems that are inherent in the use of collaborative systems. It is argued that integrated systems will only find acceptance if, apart from providing 'generic integration facilities', adequate tools are provided to design, configure and execute the management control on a case by case basis.

A three-tiered control approach is discussed and a prototype tool to aid the project manager to execute control is introduced.

Introduction

Integrated design systems are becoming more common place in many fields and especially in the engineering and building domain, where there is often a greater number of participants in a design project than in many other fields. R&D efforts dealing with integrated design and engineering systems recognize the need for the support of interoperability on different levels of granularity. Examples of such efforts are reported in Amor et al. (1997), Augenbroe (1995a, 1995b), Böhms and Storer (1993), CIMsteel (1995), and COMBI (1995).

A layering taxonomy introduced in (Augenbroe, 1995b) has identified three distinct levels where support of interoperability poses challenges, each of a different nature and each requiring a different paradigm to deal with them (Figure 1.).
The figure shows five layers that you are likely to find in any type of multi-actor integrated system. Different functionality is provided by each layer and each layer builds on the ones below it.

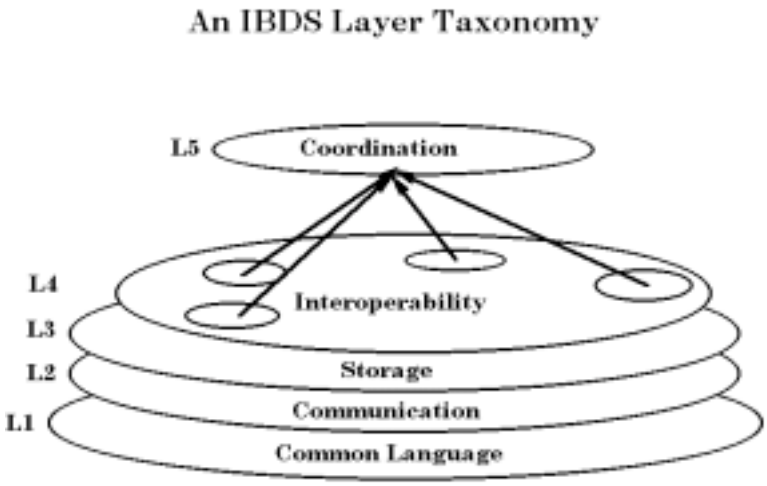

Figure 1: Layers in integrated design systems.

Functions in one layer are enabled by the functions offered in the layers beneath it. It is helpful to group the three bottom layers into one ‘Data Sharing’ layer (L1+L2+L3).

A closer look reveals which key functions should be supplied by each layer.

First Layer (L1/2/3): Data Sharing

This bottom layer is essential to any type of meaningful exchange. It represents the knowledge in the universe of discourse in a formal manner; it comprises a superset of the semantic content of the information that actors want to exchange with the system and hence with each other.

An aspect that has to be faced is the level of explicit actor view support, e.g. explicit representations of each view, keeping track of all relevant (i.e., within the scope definition) interrelations between views. Or one can choose not to support explicit representation of views. In the latter case, all actor views contribute to the definition of a common building model. The resulting common "model" is supposed to be complete in the sense that each particular view can be mapped to/from it.

Communication takes place through a variety of physical connectivities (network, diskette, etc.), communication protocols (on/off-line, OLE, CORBA, native, etc.) and messaging paradigms (exchange of states, exchange of operation, atomic messages, etc.).

Explicit and persistent storage of the state(s) of design is an essential requirement with rich features such as: versioning, audit trail, public/private versions, actor definable locking, explicit view storage.

Second layer: Interoperability

One is quick to realise that data exchange facilities for collaborative design become rather meaningless if there is no support for evolutionary design scenarios and incremental design refinements. Without any form of goal driven control the exchange events, e.g. related to design intent or predefined procedures, communication facilities become rather pointless.

The interoperability layer must provide the basic functions for executing the actual information exchange and managing the storage and evolution of design states. It is in this layer that the "team performance" of the system is determined. It is good to note that the functions performed at this layer are constrained by the support functions offered by layer 1. For instance, in the absence of explicit knowledge about inter-actor activities in a given design task (L1), lack of on-line or interruptible communication (L2), or lack of flexible locking of private data in the data store (L3), severe restrictions will apply to the level of interoperability that can be offered in this layer.

Third layer: Co-ordination

Data exchange without a controlled purpose and interaction scenario will lead to chaos. One may be able to "understand" and "interpret" all messages but one has no control over "who should be doing what next and for what purpose".

The co-ordination layer provides the means to "drive" the system towards defined design-objectives. It contains the ingredients to supervise actor activities and schedule events, and manage the work flow. It can employ different project management paradigms (reactive, monitoring, pro-active, autonomous, etc.). The co-ordination layer should add the project specific and design office specific context to the generic layers beneath it.

The Challenges

On the ‘data sharing’ level the emphasis is on a common understanding of the domain, to be formally expressed in an ontology (Gruber 1993). Judging from the ongoing debate, e.g. in the

ISO-STEP community (ISFAA electronic conference 1997), the AEC industry is still far removed from an accepted common ontology and it is becoming increasingly clear that relevant practical solutions cannot wait and should not have to wait for such a ‘grand scheme’ solution.

The second layer deals with process interoperability, i.e., the way in which actors collaboratively add and change data in a shared project information space. Much recent focus has gone towards IT solutions to provide support for this shared and simultaneous access to the building data. Much of that work concentrates on database support functions for concurrent transactions, ultimately with the aim to support design change updates made by different actors from different views (Amor et al. 1995), (Eastman et al. 1995). Merging and reconciling conflicting design versions is another, even more challenging option that future systems may offer.

These efforts will provide essential solutions for the middle layer the integrated systems. It is important to realise that. As such they will result in part solutions, as humans, engaged in a collaborative design process have much more needs than being able to exchange data through a system that caters to their particular views and domain applications. Rather, they are actors in a group decision process where the computer-based environment is instrumental as information provider and flawless transfer agent of data, but not the solution by itself. Effective control should therefore be executed on all three levels introduced above. They represent different 'layers of authority' and each layer should show the right mix of responsibility delegation to both human decision taking and computer (based) control.

It is also good to realize that each layer is in fact a hierarchical composition of many sub layers. E.g., in the common ontology layer, structuring applies to specialization and decomposition hierarchies; atomic transaction processes in the second layer can be grouped into complex processes, sessions and ultimately whole design tasks. In the project management layer, a hierarchical structure will usually reflect the different stages of decision taking and authority delegation. Detailed decisions will occur at short time intervals, involving only the directly involved team members, whereas major design changes occur less frequently and will require a top level decision, with the supervisors of different smaller design teams being the active partners.

The integration challenge for the next decade lies clearly in the development of systems that show full integration over the three layers. This implies not only a process model that is tied in with the data items in the building model ‘below’ it, but also tied in with a decision taking and task allocation and role delegation model of the design team and the ‘virtual’ organization they form part of.

It is important to recognize that a project starts with the design of the management structure on the top layer. This 'management layer' defines the actors, their roles and authorities in the design process at large. Then, the actual 'project layer' is 'designed' entailing the breakdown of tasks, their scheduling and inter-dependencies, as well as a control structure stating pre and post conditions related to their execution. The third layer is the operational layer of the collaborative system. It provides the generic functionality for exchange of data among the design actors, while they are performing their assigned roles in the project.

Other industrial sectors, faced with the same needs, have come up with part a structured approach to the above. Most notably the electronic design community has come up with initiatives such as the CAD Framework Initiative (CFI 1995), and relevant applications, such as NELSIS (van Leuken 1995) and SIFRAME.

Acceptable solutions for real life problems in building design do however not yet exist, as many of the complex relationships between items on different layers have not yet been explored in these systems. One of the projects that has tried to integrate the three layers is the EU funded COMBINE project (Augenbroe 1995a). The COMBINE project is seen as leading the way towards the practical development of IIBDS's (Intelligent Integrated Building Design Systems) through which

energy, services, functional and other performance characteristics in planned buildings can be modeled and integrated

This project explored prototypes of the next generation of integrated building design systems with a configurable project control layer. It introduced a modeling technique, that allows design and configuration of the project control structure and provides a flexible way to tie the control in with the actual data being exchanged, at different level of detail (‘deep’ and ‘shallow’).

‘Project windows’ are defined to encompass the flow of control for a small part of the design process; a formalism for describing project windows was also developed in the COMBINE project.

The COMBINE approach to the layered architecture

Layer 1: Shared OO database with STEP interfaces

Based on 12 actor views in the field of architectural design with energy and HVAC as the primary scope, an integration exercise led to a common building model. EXPRESS was used as knowledge representation language.

The scope of the model caters for the tools that make up a small, but for practice relevant, part of a building design process (encompassed by a so-called "Project-Window"), dealing with thermal/energy/comfort simulation, HVAC design with HVAC CAD support, lighting design, shape design and space layout with architectural CAD support, cost estimation, building regulation checking, component database access for HVAC and fabric components, and on-line document browsing.

Actual data is transferred as STEP data files. They are parsed and mapped to local entities on input at the start or a design tool session and the reverse takes place when the session ends. These tools are called the off-line tools; they exchange static product descriptions with a central repository.

The CAD systems are deployed as on-line tools; they interact with the central building model through a native API; they can both exchange STEP entity instances with the central repository and they may activate operations on the model in the repository.

The central Data Exchange System keeps a coherent representation of the designed building in a central persistent storage location. This component acts as a central repository, keeping track of versions and is able to update a design according to the new information supplied by one of the actors.

Second layer: Interoperability through (specialized) subschemas

COMBINE has managed to develop a basic, but crude, interoperability support mechanism based on subschemas for design tools. The granularity of interoperability is chosen to be at design tool function (DTF-level). Each design tool (both on and off-line) has a range of specific DTF's that are pre-defined and used as the basic granular activities in a design scenario. Each DTF is associated with a subschema of the complete building model.

A subschema may contain any (valid) subset of entities and subset of relationships of the complete building model. The validity of a subschema is ensured by sticking to certain rules when stripping the building model to subschema proportions.

A subschema can be further "specialised" by adding tighter cardinality constraints, add or remove optionally constraints and add clauses which act as a query and thus reduce the set of instances that fit the subschema. The latter serves to limit the building instances to those that make sense to a particular DTF.

Third Layer: Project Supervision Module

COMBINE has addressed the issue of project supervision and developed a so-called "Exchange Executive" module. It can be fed with sequences of actions (scenarios) that a certain project demands and along the way it is able to detect what results are affected by design modifications so that it requests the redoing of certain evaluations. The granularity of a scenario is again at the DTF level.

All actors, DTF's, and constraints with respect to transitions between DTF's are formally modelled using an adapted form of Petri-Nets. Each DTF in the model has its input and output subschemas (L4) associated with it. This serves to detect overlaps between DTF's (disallowing concurrent use) and also enables us to maintain a list of DTF's that have to be reinvoked because a downstream DTF has changed its input. The following section will explain this in detail.

A model of design task control

This section explains the approach to a task scheduling model, i.e., a formal, computer interpretable representation of the tasks, the actors that perform them and the constraints that control their execution and sequencing. It is based on the metaphor of a Project Window, within which a Petri-net modeling technique serves to derive the formal Project Window reference model.

Project Window

The Project Window (PW) metaphor is introduced as a ‘window’ on any suitable, limited portion of the overall design process. Such a portion is usually limited to a small time period of the project (typically a project phase with a well defined start and end state) and only a limited numbers of actors (typically limited to a specific sub-system in the building or dedicated to the exchange that takes place among actors in a particular setting, i.e., to reach a predefined decision point). Each actor in a project is assumed to play some number of design roles in the project, and these design roles are comprised of some set of design functions.

The PW approach has been proven to be an adequate instrument to implement integrated building design system prototypes directed at a particular industrial need. COMBINE has generated a number of PW models with input from industry, and IBDS prototypes were developed for them.

A PW model describes actors, their roles and tasks, the input and output data for each task and pre and post conditions that regulate the execution of the tasks. It should be noted that a PW model is no attempt at a generic process description. Rather, it is the result of a work floor approach to the project at hand. PW modeling tools should allow easy configuration at the start of the project and ‘on the fly’ reconfiguration. The execution of the PW model is done in interaction with the Project Manager, who is always in control over what task to execute next.

The Combi-Net technique

To model an individual PW a formalism is required which allows the specification of all the information described above. The formalism that has been defined has two distinct parts. One part is for the definition of the actors, the design roles they play and design functions associated with each role. The other part is for the specification of flow of control utilising the design functions defined in the dual part of the formalism. The control flow formalism is based loosely around a Petri-net definition of flow of control (Jensen 1990), but as it is quite different in its mode of operation it has been renamed a CombiNet.

PW models are expressed using a flavored Petri-net technique. Petri-nets are well suited to model control over the sequence of tasks. Several types of controls apply:

- conditions determining when a specific actor can perform a particular operation. (temporal logic control).

- conditions to ensure that the model remains in a consistent state while the design progresses (data integrity control).

A PW model consists of two diagrams The first one captures all the “Actors”, their “Design_Roles”, and the “Design_Tools” they can execute. The diagram describes all the communicating entities in the system and their inputs and outputs.

Specification of actors, design roles and design functions

Being able to specify the actors, their design roles and design functions requires only a very simple formalism. The relationship between these items is strictly hierarchical. An actor performs several design roles in a project, though two actors may be involved in the same design role. Each design role is realized by a set of design functions, though several design roles may utilise the same design function. The graphical formalism that was designed for this specification has three types of icons in it. One to denote an actor, another to denote a design role and a third to denote a design function. These icons are connected by lines to denote the association of design roles with an actor and design functions with a design role.

Figure 2 shows a small example of this formalism in the CGE modelling environment. There are three types of objects in this view. The ellipse on the left of the diagram represents an actor. The rectangles in the center of the diagram represent design roles, and as such are referenced by actor objects. The right hand objects represent design functions and are referenced by design roles. A design function has some associated information, in the center of the circle is the name of the design function, at the point of the diamond is the name of the design tool which will perform this design function, and to the left and right of the design tool name are the names of the files which contain the IDM subset models describing the input and output requirements of the design function. The specification of input and output models for a design tool allows for the possibility that a design tool can perform several design functions, sometimes through the use of different data models.

Figure 2 shows a example for one *actor* (the structural consultant), having *design_roles* to perform the structural system design and to ensure structural integrity, both of which utilise *Design Tool Functions*, associated with expert performance evaluation tools. The diagram also shows the explicit reference of each function to a predefined subset of the building model.

The second diagram type, known as the "Combine_Petri" diagram, describes the order in which these DTF's are allowed to execute in the system. These diagrams use the basic structures of

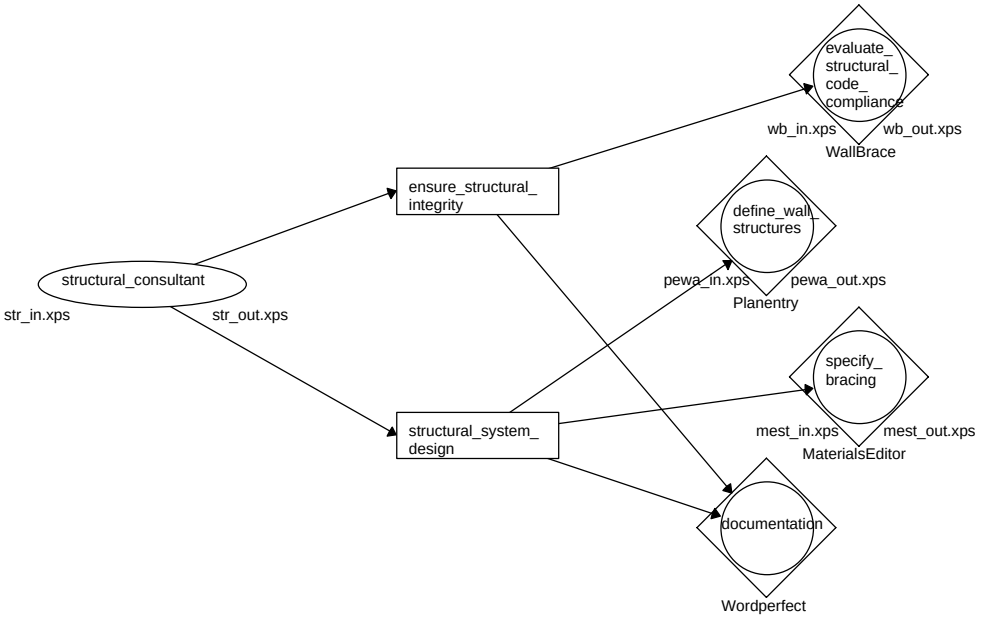

Figure 2. Combine PW Diagram

Petri_net diagrams, i.e., *Places* and *Transitions*, where *Places* represent (on the lowest granularity) design tool functions and *Transitions* represent all potentially allowed sequences from one Place to the other, subject to constraints.

Specification of flow of control

The flow of control between the various design functions associated with a design role and actor can be described through a permutation of the Petri-net formalism. By equating a design function with a place a transition with a choice point and a token with an actor it is possible to connect the various design functions in a manner which can mimic the flows that will occur in a project. In the graphical formalism that was developed for COMBINE the basic Petri-net formalism has been extended to contain aggregation and short-cut descriptive facilities, all of which are shown in Figures 3 and 4. The views presented in these figures represent a distinct portion of the flow of control as evidenced by start and end transitions, the role of these portions of the flow of control is detailed later in this section.

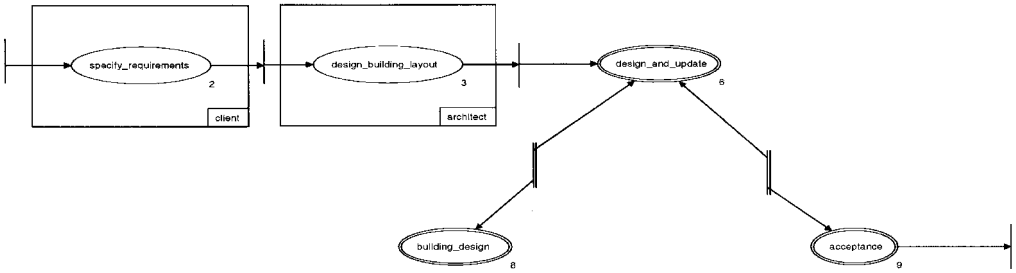

Figure 3: Combine_Petri Diagram

These diagrams are hierarchical in nature. "Aggregate_places", such as the three on the right-hand side Figure 3 refer to lower level "Combine_Petri" diagrams, such as the one in Figure 4 until the level of plain single Petri_Net places is reached. There is no theoretical limit to the level of diagram nesting which may be accomplished using "aggregate_places".

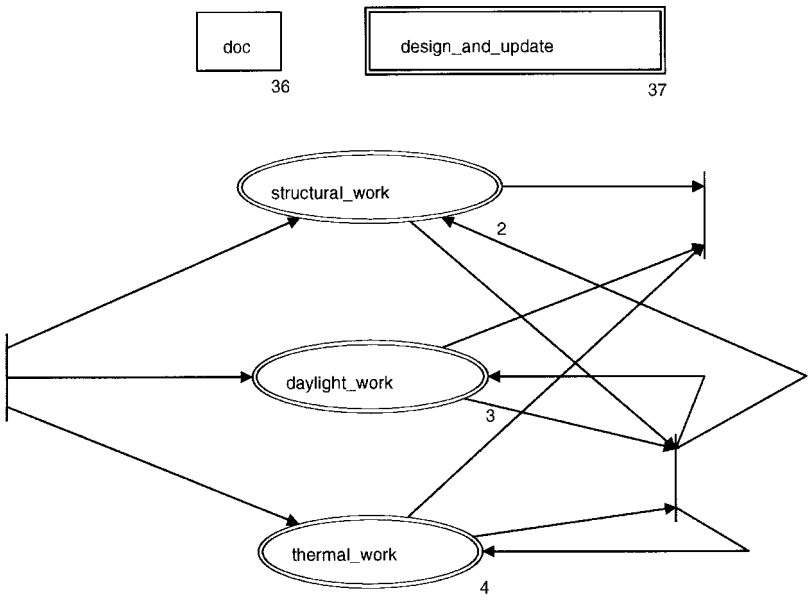

Figure 4: Lower Level COMBINE_Petri diagram

Place

A place in the CombiNet view, denoted by an oval, represents a single design function. A place can be reached from some set of transitions, and will exit to some other set of transitions. The movement of a token in and out of a place represents three distinct actions. The movement of a token into a place denotes the invocation of the design tool associated with the design function along with the input data required for the design tool as modelled with the input DTF model. The existence of a token in the place denotes the execution of the design function by the attached actor. The movement of a token from a place through an exit transition denotes the termination of the design function and the passing of resultant data back to the IDM. Any of these actions may fail, either the correct input data can not be formulated, the actor may not succeed in the design function, or the output data may not match the specifications set in the output DTF model. In the case of a failure the token representing the actor is passed back to the transitions of the place it last exited to perform some other (or perhaps even the same) design function.

Transition

A transition in the CombiNet view, denoted by a single bar, works as the choice-point between one design function and the next. A place can exit to a number of transitions and a transition can branch out to any number of places. In this formalism there are two special kinds of transition that can be created. A transition which is not exited to by any place is called a start transition and denotes the entry point into a CombiNet view. A transition which exits to no places is called an end transition and denotes the exit from a CombiNet view.

Aggregate place

An aggregate place, denoted by an oval drawn inside another oval, stands in for a whole CombiNet view, allowing the modeller to group complex interactions into logical sub nets and reference them through a higher level mechanism. All transitions which invoke an aggregate place are connected to the start transition(s) of the CombiNet view represented by the aggregate place. In a similar manner, the exit transition(s) of an aggregate place are all tied to the set of end transitions in the CombiNet view represented by the aggregate place.

Global place

The global place, denoted by a rectangle, represents a single design function that can always be activated when in the current CombiNet view. The global place is also available to all places found in all aggregate places referenced from the current view. The global place is a short-hand notation to cut down on the transitions required to model a place that is always accessible.

Global net

The global net, denoted by a double-lined rectangle, stands in for a whole CombiNet view, similar to an aggregate place. However, this global place can always be activated when in the current CombiNet view, similar to a global place.

Double transition

The double transition, denoted by two parallel bars, is a short hand notation replacing the use of two transitions which provide a loop between two places.

Token

Tokens are not shown in the flow of control model, but are used when simulating the flow of control represented in such a model. A token represents a single actor performing a particular design function. Unlike Petri-nets a token passing through a transition in the CombiNet only propagates a token to one of the reachable places. This equates to the actor starting a new design function. Where several design functions may operate concurrently the project manager must create new tokens to represent each of the actors and start them in the appropriate part of the flow of control simulation.

Figure 7 shows a snapshot of a part of a Petri_net, in the runtime support tool that we will briefly introduce below. The next section explains the inner workings of the control execution.

Execution of control

Underlying the specification of flow of control in the COMBINE system is a very simple observation on the connection between DTFs and the IDM. This is, that knowing the subset of the IDM represented by a particular DT (for both input and output), and also the subset of IDM instance objects used by a running DT, it is possible to determine the affect of a particular DTF's invocation upon all other DTFs in the system.

If the schema information about a DT is modelled, in terms of both the input to the DT and the output of the DT, then it is possible to statically determine whether it is possible for two DTFs to interfere with the running of each other. We can see that if the input schema of a DT intersects with the output schema of any running DT then it may not be possible to invoke the DT, depending upon the actual objects used in the invocation. Also, if the output schema of a DT which has completed has an intersection with the input schema of a DT which previously completed, then it may be necessary to re-invoke the previously run DT, again dependent upon the actual objects involved.

As stated above, the schemas for the DTFs can only provide a static pre-computed indication of whether there are constraints on the running of a particular DT or not. If the pre-computed check indicates that there may be constraints then the actual objects which are used in the running of a particular DT must also be checked to provide the final confirmation, or otherwise, of whether two DTFs interfere with each other.

The working of the comparison between DTs is illustrated in the figures below. These figures represent both the schemas of the DTs and the objects used as input and produced as output from the DTs.

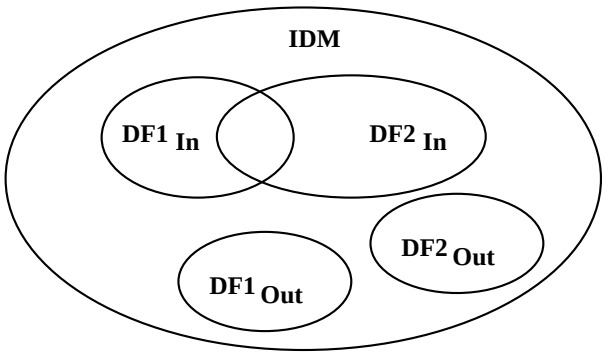

Figure 5 Runable DTFs

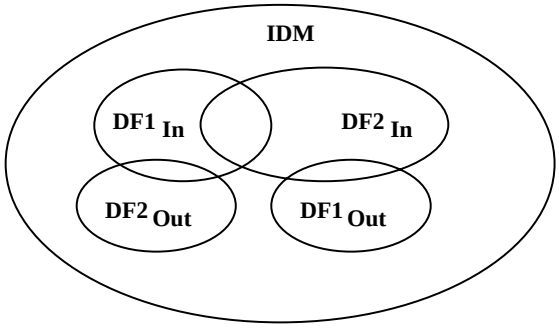

Figure 6 Constrained DTF1 and DTF2

In Figure 5 we see the situation where even though the two DTs share data in their input (DF1 $_{In}$

and DF2 $_{In}$), neither of the DT inputs have an intersection with a DT output (DF1 $_{Out}$ and DF2 $_{Out}$). In this case both DTs may run concurrently, and the result of either tool will not cause the re-invocation of the other DT.

In Figure 6 we see a case where the two DTs may not run concurrently, as the input of each DT (DF1 $_{In}$ and DF2 $_{In}$) intersects with the output of the opposite DT output (DF1 $_{Out}$ and DF2 $_{Out}$). This figure also illustrates a situation where the invocation of either of these DTs can cause the other DT to be a candidate for a rerun. This may also be seen as a cycle, as running one DT causes the other to be a candidate for a rerun, which in turn causes the initial DT to be a candidate for a rerun, etc. This is not considered a problem because each of these DTs are just candidates to be run, whether they actually do get re-run is based on the flow of control specification and the project manager for the PW.

Run-time Control

In COMBINE the exchange executive (ExEx) is the system which controls the flow of control defined in a project window.

Supervision and Exchange Executive

In order to translate the Combi-Net control model into an agent actively controlling the building design system, an application called the Exchange Executive (ExEx) was developed within the COMBINE project (Augenbroe 1995a) The project control based on the PW model was essentially developed to control scenarios in which DTFs were allowed to access the central database. The ExEx is capable of reading in the PW model, creating a binary image of the model in RAM and executing it. The ExEx tool determines data dependencies between all individual DTF's and evaluates the state of the system based on the evaluation of the pre and post conditions. After the execution of a DTF the ExEx will show a list of all possible next DTF's that the project manager can choose from. It will also show the list of already executed DTFs which might have to be re-executed because part of their input data has changed.

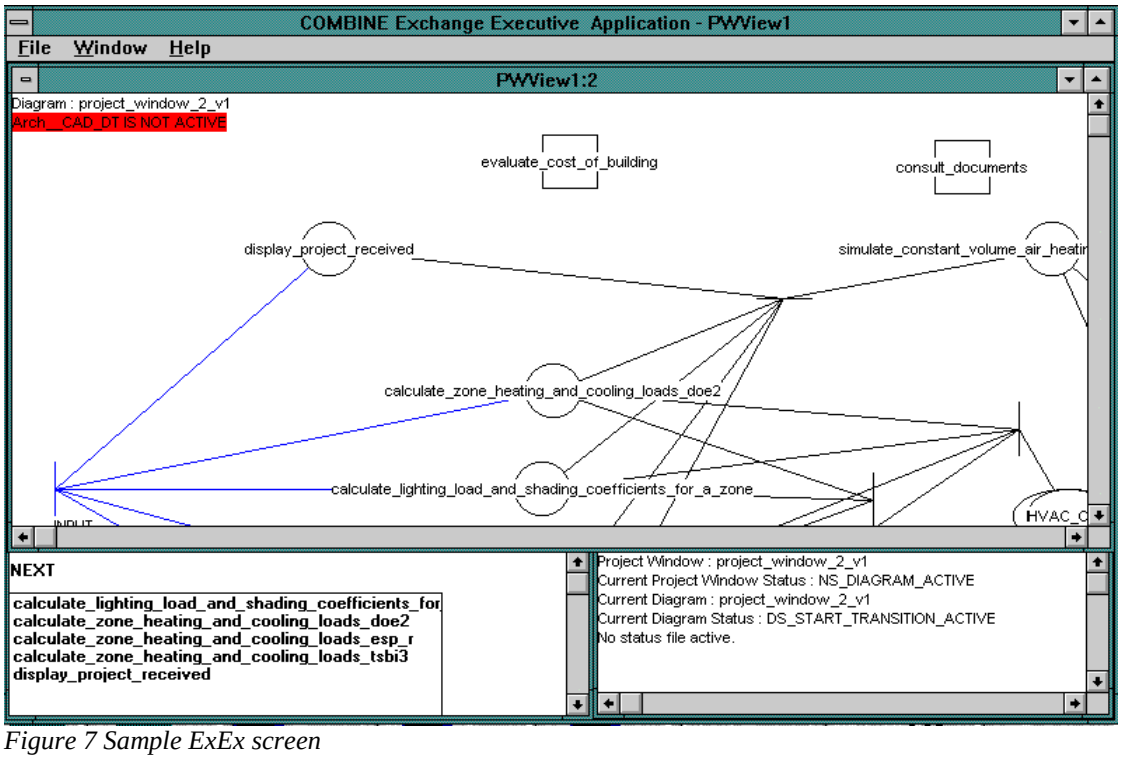

Figure 7 Sample ExEx screen

The ExEx is initially configured for a particular PW definition, but to perform the analysis described above it must receive status messages from the various design tool interfaces stating whether a particular DTF was invoked and is running, or whether it failed to get started due to data dependency requirements not being met (this is assumed to be checked at the design tool interface). The same sort of information is required from the design tool interface regarding the termination of a DTF, the ExEx needs to know if the DTF terminated correctly, whether it failed to complete its function, or whether the data it tried to pass through to the IDM model was rejected. From the DES the ExEx needs to know what IDM objects are requested by a particular DTF, and what IDM objects were modified by the running of the DTF. With this knowledge about the operation of the IBDS the ExEx can calculate the set of DTFs which may be invoked at any time, and passes that information through to the project manager, who decides which DTF(s) to invoke next.

To calculate the CombiNet's flow of control in the ExEx we use tokens as defined in the Petri-net formalism to denote our place in the CombiNet. A single token represents the actions of one actor performing their design roles with various DTFs. A Token in a place denotes the running of a particular DTF, when the DTF has terminated the transitions are used to determine the candidate DTFs from the current place. When the next DTF is decided upon the token crosses the choice-point created by the transitions and invokes that DTF. If a DTF is unable to start, or terminates unexpectedly, then the token is sent back to the place it previously came from and must re-evaluate the transitions available from that place. Since a token represents a process, the propagation of a token through a transition is handled in a different manner to that normally used in Petri-Nets. When a token reaches a transition in the Petri-Net we are at a choice point, at which time the project manager is choosing which DTF should run next. When the project manager chooses a DTF the token is moved to the place which represents that DTF, hence only one token is ever sent out of the transition.

It should be noted that there is an ordering in which the information describing the state of the PW is used. When the ExEx calculates the state of the PW it will work at two levels, these being:

1) Determining whether any of the DTFs are candidates to run based on the two set constraints defined earlier (i.e., input doesn't intersect the output of a running DTF, or the DTF is a candidate to be rerun due to the input data it used to perform calculations being modified by another DTF).
2) Determining which of the candidate DTFs returned from the analysis in 1) can be invoked when compared with the state of the PW and the Petri-Net specification of control of flow of the DTFs.

This calculation will lead to the identification of a set of DTFs which may be run at the current time. In practice this set of DTFs will be calculated every time a DTF is invoked by the system (giving the project manager a set of DTFs which may run concurrently with a running DTF), and every time a DTF terminates (giving the project manager a set of DTFs which follow from the DTF which terminated). In the best case where all DTFs run and terminate correctly this set of DTFs presented to the project manager will reflect a path through the Petri-Net following the arrows from places to transitions. However, when a DTF fails to start up, or terminates with an error condition, the Petri-Net definition of the flow of control will be used in reverse to determine the transition prior to the DTF which had problems, and to recalculate the set of DTFs which may run from that transition. It is also conceivable that after a problem with a DTF the ExEx could find itself in a position of not being able to find any DTFs capable of running, it is at this time that the project manager will be able to use discretionary powers to force the start of a DTF in the system, redirecting the flow of control of the PW.

Conclusions and further work

One could roughly define the following levels of control:
No control: one is merely able to 'ship 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,

one needs to establish work procedures, task scheduling, etc. among the team of users. The system is a "dumb" data router which provides no guidance on what needs to be, or should be, done next.
Shallow control: a limited level of control, that we informally denote as shallow. It was one of the challenges of COMBINE to define a meaningful yet limited level of control integration. It is assumed that a workable approach to shallow control is a "scheduler" where one deals only with the control over "who's next" and where one monitors the states resulting from exchange events. A project manager can thus enforce a lot of rigor in the use of the system. Although this might suit PWs for parametric (routine) design, it is to be expected that most PWs will require a more flexible approach, where the human users may interact with the control layer as well as with each other. In principle the control layer will give advice, but any user can always supersede the suggested action.
Deep control: an extended level of control. This type of control deals with much more than scheduling. It deals with the pre and post conditions themselves related to exchange-events. It can look "inside" exchange events and DT interfaces and propose remedial actions, and propose and control how the system can recuperate from deadlocks. It furthermore can deal with declarative knowledge on meaning and purpose of activities and thus can support goal-driven strategies.

Although the mainstream developments in COMBINE were on the shallow control level, some experiments with deep control were done in a small Blackboard based prototype. Future work will address deep control issues by expanding the current approach to handle entity instances rather than just entity classes.

It would also seem fruitful to extend the ExEx from the shallow level of control to use the schema definitions and constraints that we have defined in the reverse direction to that specified above. In this case one may state that they wish to see a particular result. Then given the current state of the project (i.e., the set of DTFs that are available to run) and the schemas of the DTFs it should be possible to decide which DTFs can give the required information. If the runable DTFs can not calculate the information from the current data then you apply this decision making process recursively until you get what is basically a flow of control definition of which DTFs must be invoked in which order to reach the desired result.

Other future work will address extensions to the PW reference model to allow multi structured management levels and links to a decisional model, organizational issues like time allocation of the individual team members, and incorporation of Internet based work flow model (WFM) approaches.

References

Amor, R., Mugridge, W. and Hosking, J. (1995) A declarative approach to inter-schema mappings, CIB W78 - TG10 Workshop on Modeling of Buildings through their Life-cycle, Stanford University, California, USA, 21-23 August, pp. 223-232.

Amor, R.W., Clift, M., Scherer, R., Katranuschkov, P., Turk, Z. and Hannus, M. (1997) A Framework for Concurrent Engineering - ToCEE, European Conference on Product Data Technology, PDT Days 1997, CICA, Sophia Antipolis, France, 15-16 April, pp. 15-22.

Augenbroe, G. (1995a) COMBINE 2, Final Report, CEC-JOULE program, Brussels, http://dutcu15.tudelft.nl/~combine/

Augenbroe, Godfried (1995b) The COMBINE project: A Global Assessment, CIB W78 - TG10 Workshop on Modeling of Buildings through their Life-cycle, Stanford University, California, USA, 21-23 August, pp. 163-171.

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

CFI (1995) The CAD Framework Initiative, 4030 W. Braker Lane, Suite 550, Austin, Texas 78759, USA, cfi@cfi.org.

CIMsteel (1995) Computer Integrated Manufacturing for constructional steelwork, Eureka project 130, Brussels.

COMBI (1995) Computer-integrated Object-oriented product Model for the Building Industry, ESPRIT CIME, Brussels.

Eastman, C., Jeng, T-S., Assal, H., Cho, M. and Chase, S. (1995) EDM-2 Reference Manual, Center for Design and Computation, UCLA, Los Angeles, USA, 50pp.

Gruber, T.R., Tenenbaum, J.M. and Weber, J.C. (1992) Toward a knowledge medium for collaborative product development, Proceedings of the Second International Conference on Artificial Intelligence in Design, Pittsburgh, pp. 413-432, Kluwer Academic.

ISO/TC184 (1993) Part 1: Overview and fundamental principles in Industrial automation systems and integration - Product data representation and exchange, Draft International Standard, ISO DIS 10303-1, ISO-IEC, Geneva.

Jensen, K. (1990) Coloured Petri Nets: A High Level Language for System Design and Analysis, Advances in Petri Nets 1990, Rozenberg, G. (ed), Lecture Notes in Computer Science 483.

van Leuken, R. and van der Hoeven, A. (1996) Framework Services: Design Data and Design Flow Management for Engineering Environments, The Second World Conference on integrated design and process technology, Austin, Texas, 1-4 December, (to be published).