A Set Theoretic Exchange Executive

COMB2-93-34 (DRAFT)

Robert Amor
TU Delft COMBINE Team
November 21, 2003

Abstract
This draft document describes the thinking behind the definition of an Exchange Executive (EE) based on simple set theory. We describe how if a particular Design Tool Function (DTF) is described in two sections, one being the input schema and the other the output schema, then many properties of a working EE can be calculated automatically. We will show that it is possible to determine whether a DTF is able to be invoked at a particular time, whether it needs to be re-invoked after a change to the base model, and whether a set of DTFs can be invoked concurrently. Based on this analysis the implementation details and working of a demonstration exchange executive are discussed. In the last section we propose further extensions to this analysis which would give the EE the ability to handle multiple Project Window (PW) definitions, and in the future determine the possible DTFs which would have to be invoked, and in what order, to calculate a particular result.

Introduction
The basis of this analysis is the observation that the definition of input and output schemas for a DTF describes a portion of a global set which is the schema of the IDM+ (the Integrated Data Model of COMBINE2). This is not strictly true as the schema for a DTF is likely to have cardinality constraints, keys and value constraints which will be different 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 DTF schemas and the IDM+.

For this system to work we need only define two conditions which should hold on the DTF schema definitions.
Def. 1. IDM+ □ DTF In
Def. 2. IDM+ □ DTF Out

This just denotes formally that the input and output schemas of any DTF must be a subset of the IDM+. In practice this means that the input and output schemas must be defined in terms of the IDM+ (i.e. using the same entity and attribute names). Using these two definitions gives us the first result from defining the DTFs in terms of the IDM+. The DTF 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).

In the rest of this paper when we talk about a schema, whether IDM+ or DTF we will be envisaging a schema defined in the EXPRESS modelling language. The differences we will allow in schemas of a DTF over that of the IDM+ are that they may define: different UNIQUE clauses; WHERE clauses which are additional to those defined in the IDM+; different cardinalities on attributes; different optional specifications for attributes.

The only other information required from the PW definition is the diagram for DTF management. This diagram is based on a Petri-Net definition of flow of control (Jensen 1990). We will define a *place* in the Petri-Net as being equivalent to a DTF schema. Then the PW definition can denote the running of a DTF, the order in which specific DTFs can be run, and what DTFs must have run before a specific DTF can be invoked (using the transition mechanism). The Petri-Net formalism is modelled in the EXPRESS modelling language with the following entities and attributes:

```
ENTITY petri_net ;
 places : SET [1:?] OF place ;
 transitions : SET [2:?] OF transition ;
END_ENTITY ;

ENTITY place ;
 represents : design_function ;
 exits_to : SET [1:?] OF transition ;
END_ENTITY ;

ENTITY transition ;
 invokes : SET OF place ;
END_ENTITY ;
```

Figure 1. EXPRESS model of the Petri-Net formalism

Calculable Properties from DTF schemas

Given a system which contains the IDM+ schema and the input and output schemas of the various DTFs used in a particular PW, and the definition of a flow of control for a given PW, there are various properties which we can calculate. To derive these properties from the DTF schemas we need to define two constraints:

Constraint. 1. $\forall_{i \in \text{running DTFs}} (DTF_i \text{ Out} \leftrightarrow \text{DTF In} = \emptyset)$

This constraint can be thought of as a concurrency check, or an invocation check. What is being stated is that a DTF is a candidate to be invoked if its input schema has no intersection with the output schema of any of the running DTFs.

Constraint. 2. $\exists_{i,j} (DTF_i \text{ In}^{(T1)} \leftrightarrow DTF_j \text{ Out}^{(T2)} \neq \emptyset, T2 > T1)$

This constraint can be thought of as a re-invocation check. What we are saying is that if the output schema of a DTF which has just terminated intersects with the input schema of any other DTF which was previously run then that DTF is a candidate to be rerun.

It must be noted at this point that we are considering intersections between DTFs only at the schema level. Where the DTFs require a model of a full building this will be sufficient to determine the properties detailed above. However, if a DTF models only a small portion of a building (e.g. calculates properties for a single space) then the properties calculated above could present a DTF as a candidate to be rerun in more cases than necessary. Where efficiency is not a problem, this can be ignored as we are always guaranteeing that we will come to a consistent state. This problem will need to be tackled at some stage, and is allowed for in the current EE through a connection to the DES along which the EE can collect information on the objects which are used and modified by a particular DTF. This list of object IDs can be used to supplement the properties calculated above, but will not replace the above calculations. This is due to the fact that we must look for intersections at attribute level, so even finding an intersction of objects is not at a low enough level of granularity. Using the DTF schema definitions (which specify the attributes that are required for each entity) and the objects used by a DTF, it will be possible to determine if a DTF is a rerun candidate only in the case where its input data actually has been modified.

The working of these two constraints can be illustrated in the figures below:

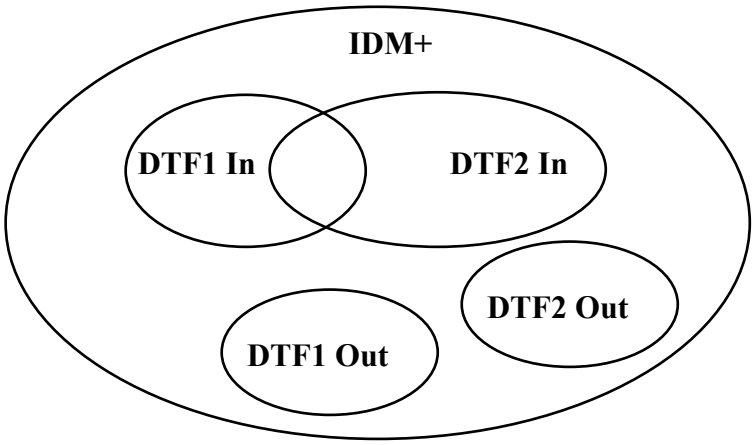

Figure 2. Runable DTFs

In Figure 2 we see the situation where even though the two DTFs share data in their input schemas, neither of the DTF input schemas have an intersection with a DTF output schema. In this case both DTFs may run concurrently, and the result of either tool will not cause the re-invocation of the other DTF.

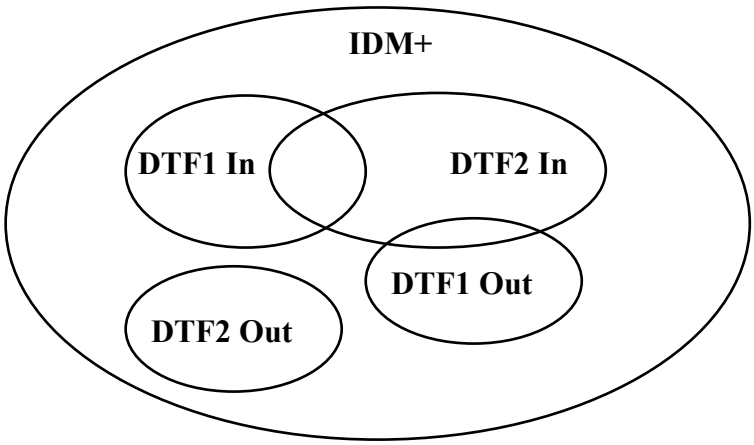

Figure 3. Constrained DTF2

In Figure 3 we have an example where the intersection between an input and output schema is non nil. In this case if DTF1 is running then DTF2 may not start. This figure also illustrates what may happen with constraint 2. If DTF2 was run at some time in the past, and then DTF1 is run, then the output of DTF1 may overwrite what was previously supplied to DTF2, so DTF2 becomes a candidate to be rerun.

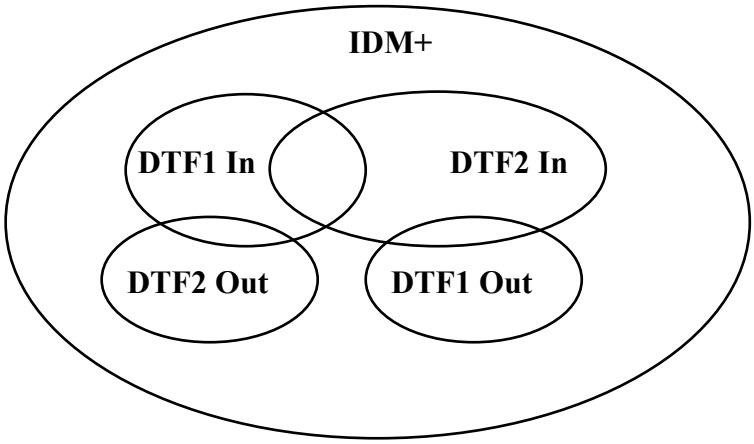

Figure 4. Constrained DTF1 and DTF2

In Figure 4 we see a case where the two DTFs may not run concurrently, as the input schema of each DTF intersects with the output schema of the opposite DTF. This figure also illustrates a situation where the invocation of either of these DTFs can cause the other DTF to be a candidate for a rerun. This may also be seen as a cycle, as running one DTF causes the other to be a candidate for a rerun, which in turn causes the initial DTF to be a candidate for a rerun, etc. The reason why this is not considered a problem is because they are indeed just candidates to be run. The final decision of what can be run at any particular time is made through the PW control flow information.

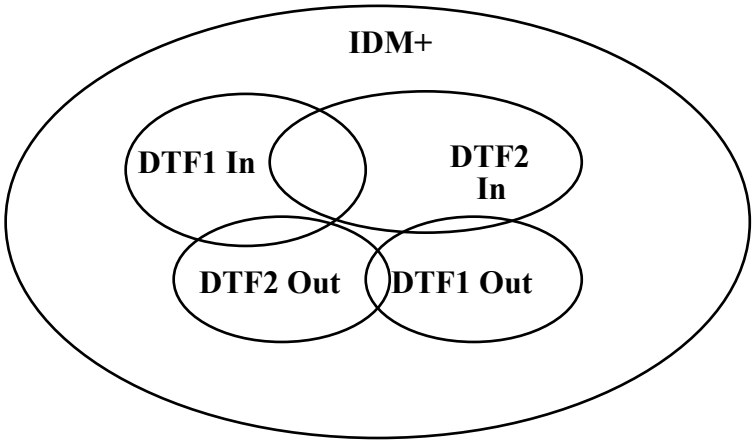

Figure 5. A definition leading to inconsistencies?

In Figure 5 we see what appears to be a definition which could lead to inconsistencies. The two DTFs have the same properties as those in Figure 4, but with the additional complication that they overwrite portions of each others output. Again this is acceptable as the running of each of these DTFs is determined by the PW definition which by its own definition gives rights to a DTF to place its output in a portion of the resultant data store.

While all these examples demonstrate the interaction between just 2 DTFs, the constraints described earlier are of course general and the results can be applied over any number of DTFs.

Working Method for an Exchange Executive

The set intersection work described above is of course just a small portion of what an EE must be able to perform. To accomplish its tasks there must be several events in the IBDS (Integrated Building Design System) system that the EE is notified about. The information that the EE needs to know about is shown graphically in Figure 6.

Figure 6. Place of the EE in the IBDS

The EE 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 EE 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 EE 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 EE 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.

Figure 7. Internal state and functionality of the EE

Figure 7 details the internal state and functions of the EE. The most important information kept by the EE can be seen at the top-right of the figure where we see the project window state entity which keeps track of the candidate DTFs to be invoked at a particular time, those DTFs which are candidates to be rerun, and using these sets of candidates, the Petri-Net definition and the current PW status (as represented by the tokens in the Petri-Net) it can calculate the set of DTFs which are able to be run at a particular time. This is the message which is passed through to the Project Manager. The other important information detailed at the top of this diagram is the invocation history entity which is created for each DTF that is invoked and is used to record the status information which is passed through by the design tool interface and the DES.

It should be noted that there is an ordering in which the information describing the state of the PW is used. When the EE 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 schema doesn't intersect the output schema 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 EE 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.

At this point it is useful to explain how the state of the PW is managed in the EE. To record the state of the PW, tokens are placed in the Petri-Net and propagated from place to place through the transitions in the Petri-Net. The meaning of a token in the Petri-Net is that a token is a single process in the system, so two tokens in the Petri-Net will allow the running of two DTFs concurrently (subject to PW and set constraints). 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.

Modelling of the PW definition

To enable the EE to be configured for a particular PW it is necessary to load in a definition of the PW. To enable this we have defined a schema for PW definitions (see Appendix A) which describes all the information required for the shallow control of an IBDS. A STEP data file of an instantiated model of this schema may be loaded into the EE to configure it for that particular PW definition. However, the creation of this instantiated model can cause some problems. Figure 8 shows the STEP file for a very simple PW definition with a Petri-Net flow of control as detailed in Figure 9. As will be noted by the reader it is difficult to see what exactly is described in the STEP file, so it is obvious that we would not wish to create the STEP file by hand. Therefore, we must look at the tools available to instantiate a model of an EXPRESS schema for such a file.

```
   ISO-10303-21;
   HEADER;
   FILE_DESCRIPTION(('Project Window model for COMBINE2'),'PETRI_NET');
   FILE_NAME($,'6th Dec. 93 4.29pm',('RWA'),('TU
Delft'),'Petri_Net','PETRI_NET',$);
   FILE_SCHEMA(('petri_net'));
   ENDSEC;
   DATA;
   #1 = PLACE(#10, (#6) );
   #2 = PLACE(#11, (#8) );
   #3 = PLACE(#12, (#5, #7) );
   #4 = TRANSITION((#1) );
   #5 = TRANSITION(() );
   #6 = TRANSITION((#3) );
   #7 = TRANSITION((#2) );
   #8 = TRANSITION((#3) );
   #9 = PETRI_NET((#1, #2, #3), (#4, #5, #6, #7, #8) );
   #10 = DESIGN_FUNCTION('Initial spec', 'Box-CAD', 'dt1a-in.xps', 'dt1a-
out.xps' );
   #11 = DESIGN_FUNCTION('Modifier', 'Box-CAD', 'dt1b-in.xps', 'dt1b-out.xps'
);
   #12 = DESIGN_FUNCTION('Calculator', 'DayLight', 'dt2-in.xps', 'dt2-out.xps'
);
   #13 = DESIGN_ROLE('Initial specification', (#10, #12) );
   #14 = DESIGN_ROLE('Redesign', (#11, #12) );
   #15 = ACTOR('Window designer', (#13, #14) );
   #16 = PROJECT_WINDOW('Window design', (#15), #9 );
   ENDSEC;
   END-ISO-10303-21;
```

Figure 8. STEP datafile of PW definition

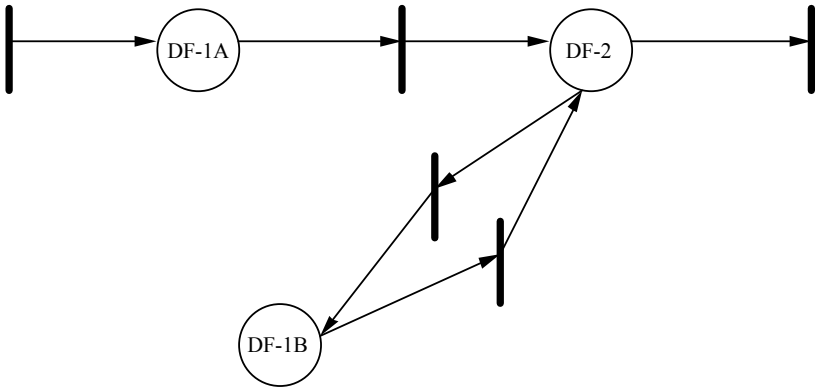

Figure 9. Petri Net definition for a simple project window

The tools readily available for instantiating an EXPRESS schema consist of: the InSTEP tool, which gives a forms based interface to entities defined in a schema and performs the requisite type checking of data as it is entered; and a modified version of the Cerno system (Fenwick 1993, Grundy and Hosking 1993) which gives a graphical formalism for instantiating a schema with type checking as in InSTEP. The Cerno diagram of Figure 10 gives an example of the definition of the top level project window information of Figure 8. The advantage of this method over InSTEP is that a view allows several objects to be shown at any one time and the relationships between objects are displayed by the arrows between objects in the view.

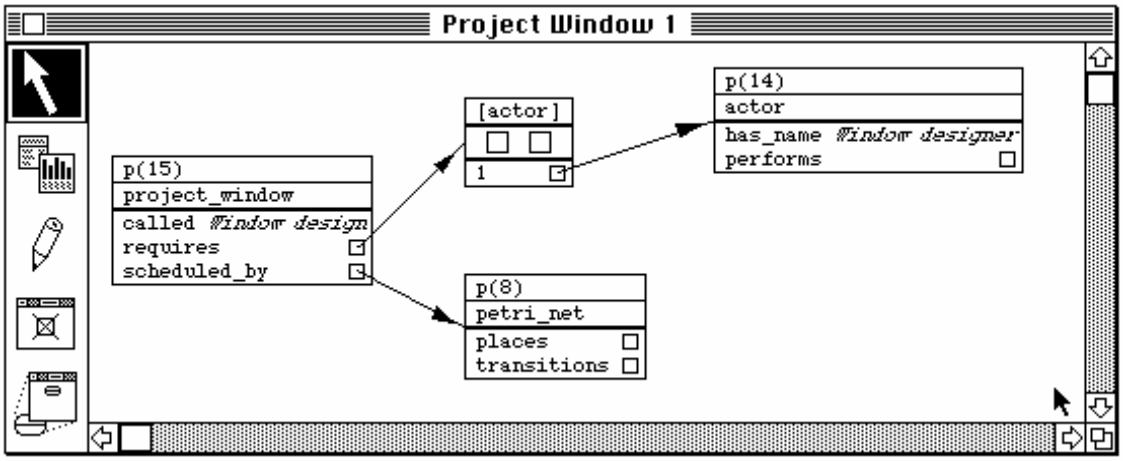

Figure 10. Creating a project window definition in Cerno

However, even this notation can become awkward when trying to view large amounts of the PW definition at one time. Cerno does allow multiple views of objects in the system, so it is possible to create new views, such as in Figure 11 where we detail the relationships specified from a particular actor in the PW definition. But Figure 11 is already becoming cramped, and is not displaying a large amount of the data in the PW definition.

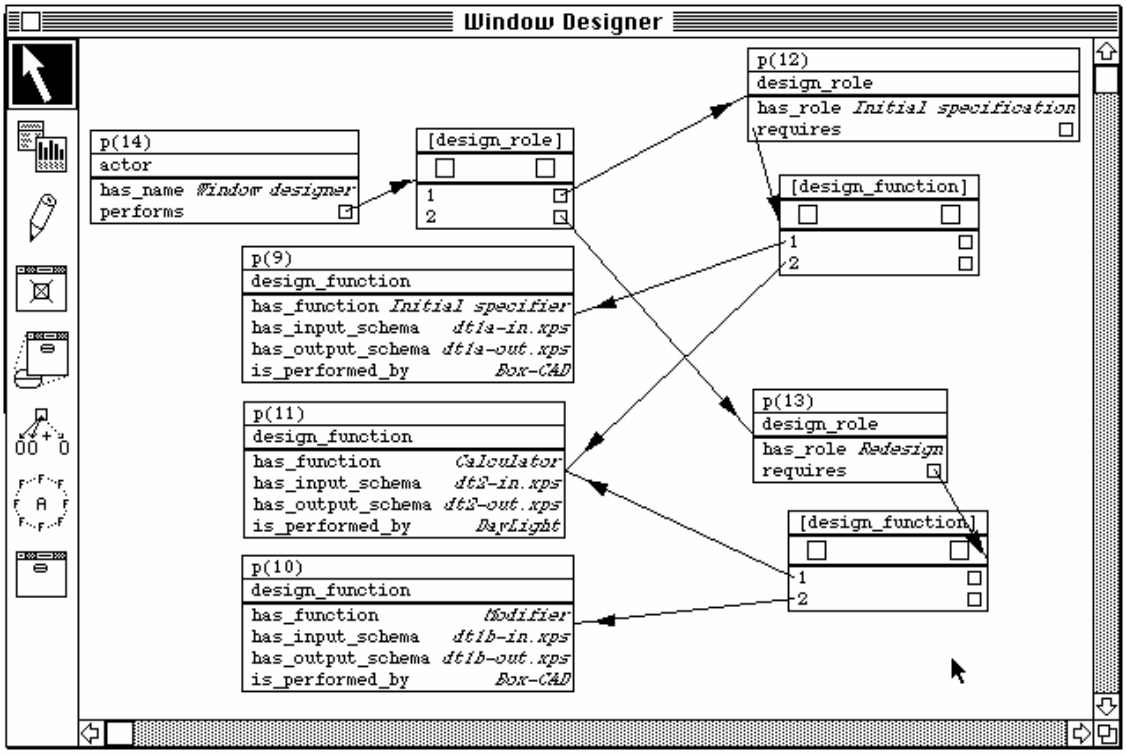

Figure 11. Creating an actor definition in Cerno

The problem gets much worse when we try to model the Petri-Net definition in this manner (see Figure 12 and compare with the definition in Figure 9). It is obvious that we would wish to specify the Petri-Net in a method as close to that of Figure 9. Finding a tool to do this is not easy as the Petri-Net definition references other portions of the PW model, and hence whatever method is used to model the Petri-Net for the PW must be capable of modelling the rest of the PW definition.

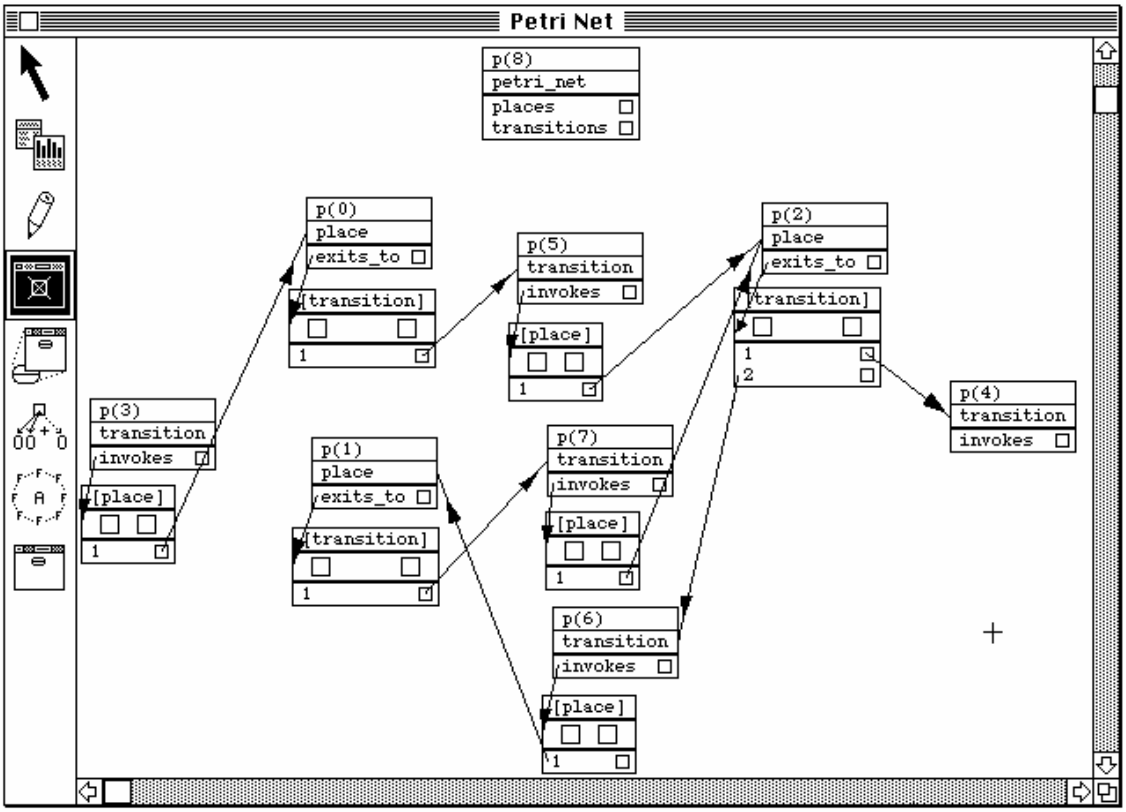

Figure 12. Petri Net as viewed from Cerno

It is clear from out efforts to instantiate PW models that we need to further investigate methods for detailing a model. The chosen methd must allow the PW model to be defined with a combination of forms based, graphical and Petri-Net type input. This problem will be looked at from the Cerno system (which will allow different graphical representations for different types of entities) and from the graphics interface to the Prolog system that the demonstration EE has been created in.

Future Work

It would appear possible to extend the definition of the EE described in this report in two major areas.

The first area the EE could be extended is to incorporate multiple PW definitions. The checking of DTF availability to run a particular time is PW independent and based purely on the information held in the IDM+ model, and the set of DTFs which are running at the current time. This analysis just provides a set of available DTFs to each PW definition, and the PW manager makes the decision as to which DTF should be invoked at the next phase of the project. This being so the EE could well manage several PWs running concurrently by storing all PW definitions, and only answering questions about the runable DTFs in the context of a specific project window.

The second area the EE could be extended is 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.

References

Fenwick, S. (1993) Cerno-II, Masters thesis, Department of Computer Science, University of Auckland, Auckland, New Zealand.

Grundy, J.C. and Hosking, J.G. (1993) Constructing multi-view editing environments using MViews, Proceedings of 1993 IEEE Symposium on Visual Languages, Bergen, Norway, IEEE Press, August 1993.

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

Appendix A. EXPRESS schema for the PW definition

```
SCHEMA pw_reference_model;


END_SCHEMA;
```