Modelling the Work Flow of a Nuclear Waste Management Program

Kjeld H. Mortensen
Computer Science Department
Aarhus University
Ny Munkegade, Bldg. 540
DK-8000 Aarhus C, Denmark
Phone: +45 8942 3188
Fax: +45 8942 3255
E-mail: khm@daimi.aau.dk

Valerio Pinci
Meta Software Corporation
125 Cambridge Park Drive
Cambridge, MA 02140, U.S.A.
Phone: +1 617 576 6920
Fax: +1 617 661 2008
E-mail: pinci@metasoft.com

Abstract. In this paper we describe a modelling project to improve a nuclear waste management program in charge of the creation of a new system for the permanent disposal of nuclear waste.
SADT (Structured Analysis and Design Technique) is used in order to provide a work-flow description of the functions to be performed by the waste management program. This description is then translated into a number of Coloured Petri Nets (CPN or CP-nets) corresponding to different program functions where additional behavioural inscriptions provide basis for simulation. Each of these CP-nets is simulated to produce timed event charts that are useful for understanding the behaviour of the program functions under different scenarios. Then all the CPN models are linked together to form a single stand-alone application that is useful for validating the interaction and cooperation between the different program functions.
A technique for linking executable CPN models is developed for supporting large modelling projects and parallel development of independent CPN models.

1 Introduction

A large nuclear waste program[1] in U.S.A. is responsible for permanently disposing used nuclear fuel and similar high-level nuclear waste. The objective of the program is to establish a capability to accept, transport and store nuclear waste by 1998, and to start the storage of nuclear waste in a geological repository by 2010. The program has quite unique characteristics; provide safe nuclear waste isolation for 10,000 years with unprecedented oversight and control by different affected and interested groups. Additionally the program must take into

[1] By program we mean an organised set of activities directed toward a common purpose. A program is typically made up of, e.g., technology based activities and projects.

account changing conditions in its environment as, e.g., changes in the current legislation. The program director therefore decided to develop a new approach to improve the current nuclear waste systems management strategy and to carefully design the program, much like physical systems are designed. A general design methodology is used to capture the functionality of the system[2]. The methodology includes use of Structured Analysis and Design Technique (SADT) [4] and Coloured Petri nets (CP-nets or CPN) [1].

The physical waste management system is composed of groups of people, documents, and equipment. They need to cooperate and interact with each other. A group of people in charge of a specific domain needs to exchange many kinds of information between other groups often with very different domains. In order to ensure an efficient and consistent cooperation and interaction between groups, the Nuclear Waste Management System (NWMS) is modelled (with SADT) and the resulting model is analysed by translating it into CP-nets which subsequently are simulated. Major components of the system are identified which are called the programmatic functions. The process of modelling and analysing will be referred to as the programmatic functional analysis (PFA). The modellers take the perspective of functional behaviour on the nuclear waste management system. E.g., one of the programmatic functions characterises sites for storage of nuclear waste, and another major function performs systems engineering.

The PFA of the nuclear waste management program is the effort of identifying all activities which must be performed in order to clarify interdependencies between the activities and bring the physical system into being (thus meeting the program objective). Models will provide a means for people in the program to understand their position in the overall program, thus becoming able to identify their functions, e.g., how cooperation and interaction should take place with other people in the system.

The result of the analysis is to be used for developing policies and guidelines that determine how to actually implement the system itself. The programmatic functions are the functions that bring the physical nuclear waste management system into being. The main stages of the physical system functions are simply stated: acceptance, transportation, storage, and disposal of nuclear waste. These operations comprise the stages in the nuclear waste management program.

We will look at some of the programmatic functions in this paper, namely the functions that are prepared for simulation in a CP-net design and simulation tool.

1.1 CP-nets, SADT, and Similarities

CP-nets are recognised as a useful modelling language for validation and simulation of complex concurrent systems (see, e.g., the book [2]). CP-nets have a hierarchy concept, which is very similar to the activity hierarchy resulting from functional decomposition in SADT. However, SADT is a top-down method while modelling with CP-nets do not need to be. See [4] for an in-depth description of

[2] By system we here refer to a complex organisation.

SADT. There is also a direct correspondence between non-decomposed functions in SADT models and transitions in CP-nets. These similarities make it possible to do a semi-automatic translation from SADT diagrams into CP-nets, where the hierarchy and the connections between activities of the SADT diagram are preserved. Places are automatically created, but net inscriptions with behavioural information have to be added manually. Adding inscriptions to the model provides a basis for simulation, and thus also a means for validating the original intention of the SADT model. See [11] and [9] for a more detailed description of the process of going from an SADT model to a CP-net.

Above, we have described that SADT and CP-nets have a number of important similarities. But the two languages also complement each other in several interesting ways. SADT does not have a formal framework while CP-nets have. SADT cannot be simulated where CP-nets can. Furthermore, CP-nets do not have explicit guidelines for a structured and systematic approach to the model creation process — but SADT does. Both methods build on principles and concepts that are easy to learn and understand. The use of CP-nets in conjunction with SADT implies that the dynamics of an SADT model easily can be examined. Hence, we obtain a better understanding of behavioural properties.

SADT in conjunction with CP-nets is the chosen method in this project for doing the PFA. We use two tools in the project. One is for editing SADT diagrams (Design/IDEF [6]), the other for editing and simulating CP-nets (Design/CPN [5]). Design/CPN is able to load SADT diagrams and translate them into CP-nets.

1.2 The Top-level SADT Function

With the purpose of further introduction to SADT and the waste management system, let us take a look at the top-level SADT function from the modelled system (see figure 1).

An SADT diagram consists of three basic elements: boxes, arrows, and labels.

1. The boxes denote functions or activities. A box will always contain a short text describing what the activity stands for. A box can be decomposed, giving a more detailed description of the function by means of another SADT diagram. Each box also has a number indicating its sequence number on the page and the contents of its decomposition. E.g., "A0" means the top-level function, "A3" is the third function on the first level of decomposition, and "A31" is the first function on the second level of decomposition, of the function "A3". The notation "3.1" is a variant of "A31".
2. The arrows denote incoming and outgoing flows of functions. Incoming flows can be of three different types; controls, inputs, or mechanisms. The type of an arrow is determined by the side of the box it enters on or exits from. A function maps input (arrows coming in on the left of the box) to output (going out of the right), possible under some constraints (control arrows entering from top), and by means of mechanisms (arrows entering from below). Mechanisms are often used for modelling resources. Several arrows can join

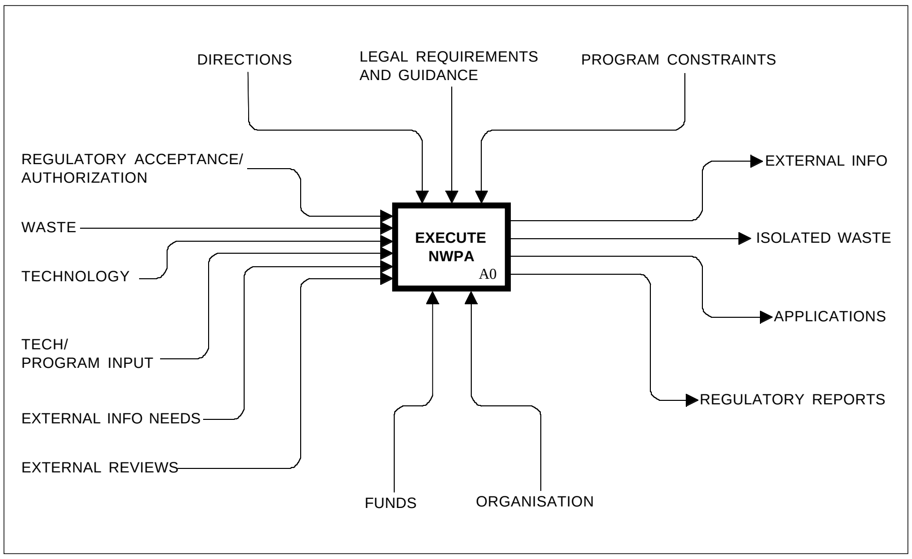


Fig. 1. The topmost SADT function ("Execute Nuclear Waste Policy Act").

or one arrow can branch into several. (If not explicitly stated, input, control, and mechanisms will go under the common term "input".)

3. Labels are associated with arrows. A label gives the name of the kind of information flowing on the arrow.

As the function above is the topmost SADT function, it provides an overview of what kind of interaction the waste management system has with the external environment. The function takes, e.g., waste and technology as input. When the function is performing, it isolates nuclear waste in sites which are allocated by functions found on lower levels of decomposition. The waste management program, represented by the above function, operates under a number of constraints imposed by its external environment. The program gets directions (a control arrow) from a department in the U.S. government. It also operates under requirements and guidance from the legislative environment, e.g., the Congress. The resources or mechanisms necessary for the nuclear waste management system to operate are an organisation (people and machinery) and funds. On the lower level of decomposition, the resource arrows (mechanisms) are omitted which means that resources are not taken into account in the rest of the model.

The rest of the paper is organised as follows: In section 2 we provide the description of the project and look at some of the SADT diagrams from the model. Section 3 describes the work involved as the SADT diagrams are translated into a CPN model. Examples will be given. We proceed with simulation of the individual submodels in section 4, and in section 5 we explain how the submodels are merged together, resulting in one model which is then simulated. Finally, we have the conclusion where we also discuss work in progress and suggest further work — all in section 6.

2 Project Description

In this section we elaborate on our participation in the project. The improvement of the nuclear waste management program is still an ongoing activity, but the work described in this paper was confined to six months (in 1991).

2.1 Configuration Overview

Two groups of people with very different backgrounds and qualifications participated in the project. The main group performing programmatic functional analysis (which we will refer to as the PFA team) was responsible for designing the systems management strategy. The PFA team produced, among other things, functional descriptions and SADT diagrams. The size of the team varied between 15 and 25 people.

The waste management program interacts with and is restricted by a variety of groups in the world outside. The program has to work within laws (which may change) of the government, and the program has to handle and interpret many sorts of data coming from, e.g., geologists that analyse potential storage sites. Therefore, many people with knowledge and experience in various domains like geologists and lawyers are present in the team. These people do not have any SADT or CPN modelling background.

The CPN team consisted of the authors of this paper having several years of experience with SADT and CPN. Instead of training the PFA team in using CPN, we, with the CPN modelling expertise, were hired by the program management. The CPN team took the produced SADT diagrams, and based on some additional written descriptions of the intended functional behaviour, the CPN team created a corresponding CPN model which simulates some aspects of the dynamic behaviour of the system. Simulation and timed event charts gave the necessary means for validating properties and answering questions about the behaviour of the model. One interesting question is if there are any unintended blocking of the information flow in the system. If the flow is blocked and other activities are dependent on blocked information, it can under some circumstances lead to a deadlock in the model. This does not mean that the system in the real world also will deadlock. But it means that the set of activities in the model that are going to be implemented in the real world will operate less optimal, because the system will discover delays. These flaws then need to be fixed (if possible) which is more expensive than fixing the model in the first place.

The purpose of our involvement was to provide an executable model of the PFA team's SADT model, in order to give a basis for validation of the accuracy and completeness of the programmatic functional analysis they performed.

From the PFA team's point of view, the interaction and cooperation between the various submodels can be compared with a protocol. As the SADT submodels are made independently by different people it is not guaranteed that the submodels will fit together when the model is viewed as a whole. The SADT modellers made the choice of CP-nets, because CP-nets could help them to validate, by means of the simulation tool, that the models really would be able to interact and cooperate correctly. (Formal analysis of CP-nets was also of interest but never used because of lack of tool support at that time.) The modellers also wanted to be able to investigate the dynamics of the models in more detail and obtain a concrete understanding, since this cannot be accomplished by just looking at SADT diagrams.

2.2 The Major Components of the Nuclear Waste Program

In this project we focus on seven major programmatic system functions. Many other functions are also in the SADT model, but they do not have nearly as high priority as the seven major functions we describe in this paper and are thus not considered. Furthermore, it would not be possible to complete the CPN modelling efforts with the available resources if we did not limit our scope. The following is a short description, which explains the main purpose of the considered major functions:

- Provide Program Control (PPC): This function controls and provides overall management direction for the NWMS program.
- Ensure Regulatory Compliance (ERC): Identifies regulations which applies to the program and the physical system, and ensures they comply with these regulations.
- Perform Systems Engineering (PSE): The function translates the NWMS program mission requirements into a set of functions, requirements, and interfaces for the physical system.
- Design Engineered System (DES): The function is divided into the four phases; conceptual, preliminary, final, and as build design.
- Identify and Characterise Sites (ICS): Provides site information for consideration in system evaluations, and also identifies and screens potentially acceptable sites.
- Evaluate Integrated System (EIS): The purpose of this function is to reduce program technical performance risks.
- Perform Confirmation/Construction/Operational Testing (PCOT): This function plans, conducts, and documents tests to verify that the NWMS physical system conforms to, e.g., technical requirements.

The latter five functions (PSE, DES, ICS, EIS, and PCOT) are the basis for the decomposition of a function called "Configure System", shown in figure 2. PPC and ERC are located outside the Configure System function and the relationship with PSE, DES, ICS, EIS, and PCOT is not easily visualised by a simple figure.

The five functions in figure 2 interact throughout the NWMS program. Typically PSE provides input information to one of DES, ICS, EIS, or PCOT. They in turn provide a result which is either a success (a final result) or a request for more information or additional action. The result is processed by PSE, sometimes in cooperation with PPC. And so it goes on through all stages of the NWMS program.

2.3 Understanding the SADT Model

Throughout the project the work procedure of the CPN team was as follows. Typically the PFA team finished a first version of one of the submodels. Our task was then to translate the SADT diagram into a CPN diagram and finish the model by adding behavioural inscriptions so that a simulation could take place.

As an example of a diagram page from the SADT model, the decomposition of the function "Analyse Performance Variances" is shown in figure 3, which is a function within PPC. (We will henceforth base our examples on this function.) The purpose of the function "Analyse Performance Variances" is to identify performance variances (e.g., delays) in the waste management program. It bases its analysis on, e.g., information about the physical system provided from PSE and general information about the status of the program assembled from many origins. If there are variances, the function is also responsible for identifying the cause and issue alternative corrective actions (with associated risks) in order to reduce the variance. It also determines where the action should be taken and develop recovery plans for future actions. The analysis results from the function in the figure are used to make change requests of which approved changes are sent to many functions in ERC as a part of general program information. ERC is then responsible for realising the requests. Thus, the scope of the function really covers most of the waste management program.

To help the CPN team to understand the intended behaviour of the SADT models, these came together with clarifying function descriptions. Initially the form was only unstructured textual descriptions. The textual descriptions contained information about the relationship between input and output when a function is activated. Figure 4 shows an example of a textual description for the function Identify Corrective Actions" from figure 3.

The textual description is divided into the mission and the scope of the function. The mission just explains briefly what the function itself is supposed to do in terms of input and output, and the scope explains to what extent the rest of the waste management program is involved in the actions of the function.

In the beginning of the project, this kind of textual description was the only available explanation of the model. Communicating the intended functional behaviour in the above form showed quickly to be insufficient and ambiguous, therefore error prone and a waste of time for us. To support the textual descriptions, a more formal description language for functional behaviour was introduced. The exchanged information became decision tables and descriptions which both speeded up the process of understanding the behaviour and reduced the number of misunderstandings dramatically in comprehending the intended functionality. Decision tables were useful for describing exactly when output is

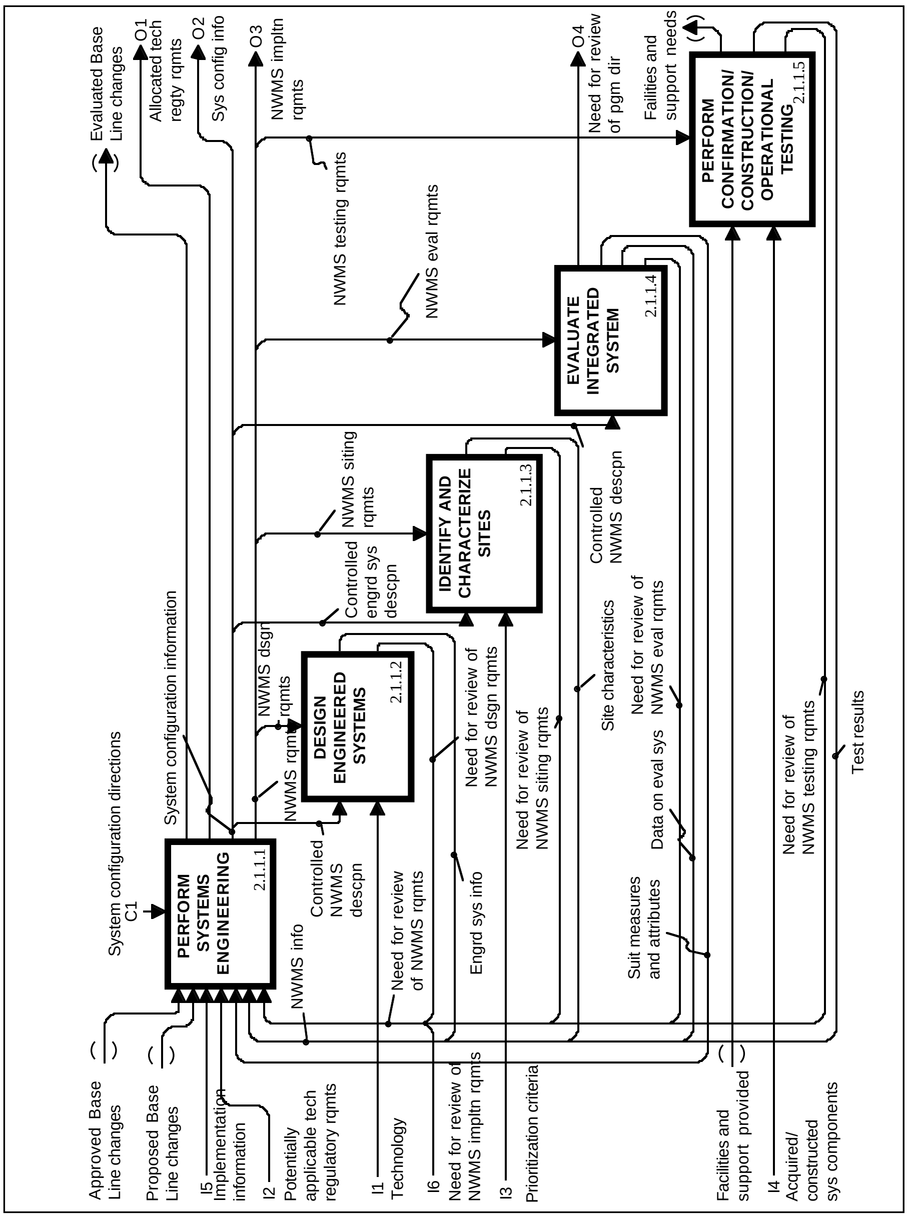

PERFORM SYSTEMS ENGINEERING 2.1.1.1
DESIGN ENGINEERED SYSTEMS 2.1.1.2
IDENTIFY AND CHARACTERIZE SITES 2.1.1.3
EVALUATE INTEGRATED SYSTEM 2.1.1.4
PERFORM CONFIRMATION/ CONSTRUCTION/ OPERATIONAL TESTING 2.1.1.5
Approved Base Line changes
Proposed Base Line changes
I5 Implementation information
I2 Potentially applicable tech regulatory rqmts
I1 Technology
I6 Need for review of NWMS impltn rqmts
I3 Prioritization criteria
Facilities and support provided
I4 Acquired/ constructed sys components
System configuration directions C1
System configuration information
Evaluated Base Line changes
O1 Allocated tech regty rqmts
O2 Sys config info
O3 NWMS impltn rqmts
O4 Need for review of pgm dir
Failities and support needs
NWMS testing rqmts
NWMS eval rqmts
NWMS siting rqmts
Controlled engrd sys descpn
NWMS dsgn rqmts
NWMS rqmts
Controlled NWMS descpn
NWMS info
Need for review of NWMS rqmts
Engrd sys info
Need for review of NWMS dsgn rqmts
Need for review of NWMS siting rqmts
Controlled NWMS descpn
Site characteristics
Need for review of NWMS eval rqmts
Data on eval sys
Suit measures and attributes
Need for review of NWMS testing rqmts
Test results

Fig. 2. Decomposition of the function “Configure System”, which shows the relation between five of major programmatic functions; PSE, DES, ICS, EIS, and PCOT.

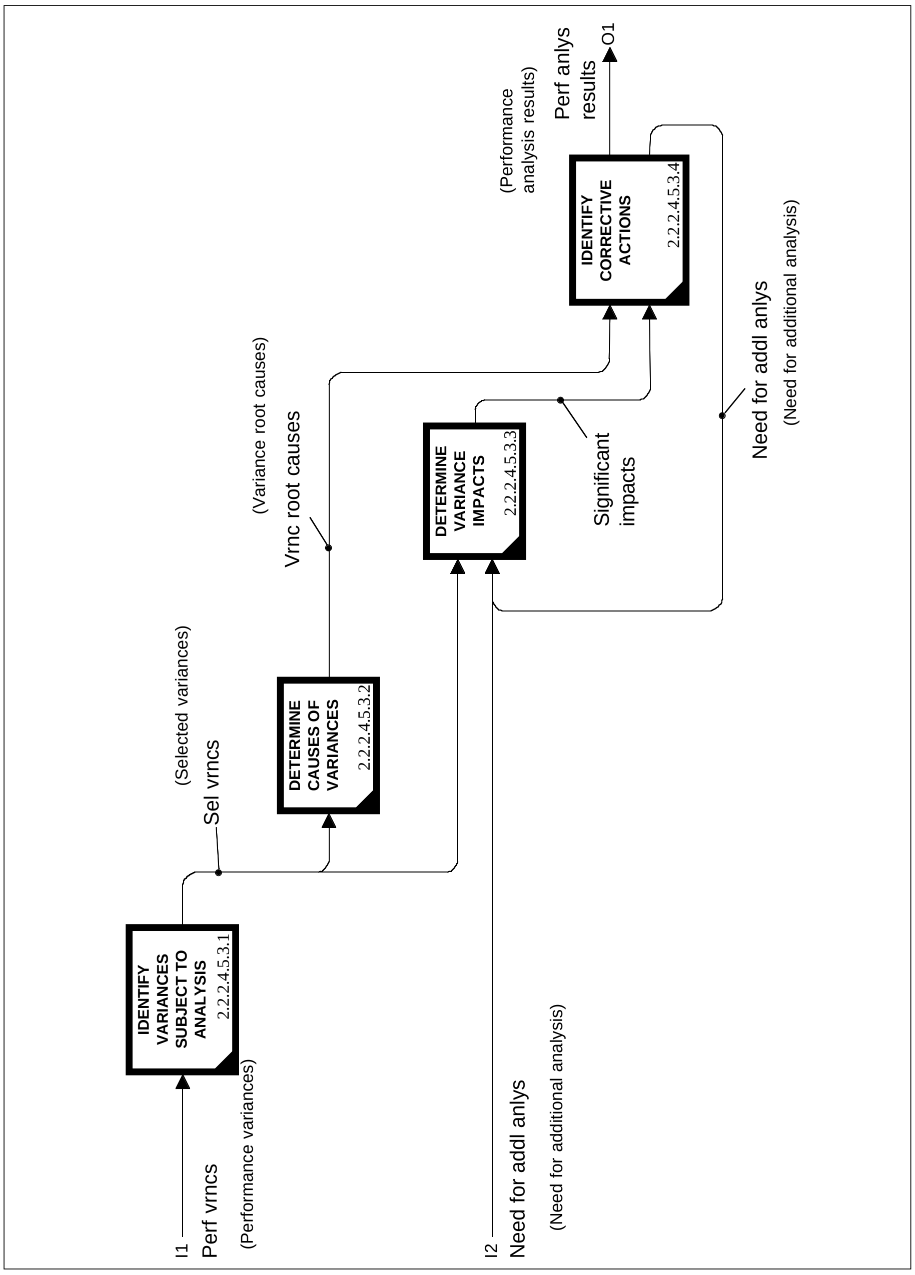


Fig. 3. Decomposition of the function “Analyse Performance Variances”.

```
2.2.2.4.5.3.4  IDENTIFY CORRECTIVE ACTIONS
MISSION: Develop alternative actions to correct root causes and
mitigate impacts of the variances affecting the program.
SCOPE: Analyse the variance root causes and impacts and develop
alternative corrective actions to correct the cause or mitigate
the impact. Analysis includes identifying and evaluating the
risks associated with each corrective action.
```

Fig. 4. Description provided with the function "Identify Corrective Actions" (figure 3).

produced, given that some combination of inputs has changed. An example of this format can be seen in figure 5.

Arrow labels give the name of the arrow from the SADT diagram. Types of the arrow can be "in", "control", or "out", as provided by the SADT terminology. The operative cycles should be read as follows. Conditions ("x") for the first execution of the function is specified in columns labelled "1". Subsequent execution is specified by columns labelled "S". Each column in the table describes one alternative. Thus, the function above will initially produce either "Performance analysis results" or "Need for additional analysis" when both inputs are available (have changed). Similar for subsequent execution.

Arrow label	Operative Cycles					
	Type	1	1	S	S	Notes
Significant impacts	In	x	x	x	x	1,2
Variance root causes	In	x	x	x	x	1,2
Performance analysis results	Out	x		x		
Need for additional analysis	Out		x		x	

1) Initial execution requires both the Significant Impacts and Variance Root Causes and will produce either the Performance Analysis Results (which includes alternative corrective actions is applicable) or the Need for Additional Analysis.
2) Subsequent executions of the function behave in the same manner as the initial execution.

Fig. 5. Decision table for the function "Identify Corrective Actions" from figure 3.

As mentioned earlier, the decision tables provided a significant speedup in the process of understanding the SADT model. We were also, at an early stage, able to predict malfunctioning behaviour in the model. E.g., with the above decision table we can identify a blocking of the information flow on the page in figure 3. From the table one can read that both inputs are always required for the function to execute. In the case that "Identify Corrective Actions" issue "Need for additional analysis", subsequent execution of "Determine Variance Impacts" results in a change in only one of the inputs of "Identify Corrective Actions". But this function requires both inputs to change in order to execute and therefore erroneously waits instead of processing the local feedback immediately. This was reported to the PFA team and they modified the decision table so that subsequent execution of “Identify Corrective Actions” only required a change in one of its inputs. These flaws and similar errors were reported to the PFA team on a very early stage, and thus saved time later when they were not so easy to fix.

3 Entering the Environment of CPN

This section is about the CPN model that was built. We do not show the total model, but only a typical example to give a feeling for the general intentions.

The final SADT model has seven major submodels. A total of 116 SADT diagram pages were drawn summing up to more than 300 non-decomposed functions, which in the corresponding CPN model are transitions. Each diagram page was automatically converted into CPN and manually extended with behavioural inscriptions.

3.1 The CPN Submodels

In the following we describe the translation process. We take each SADT submodel and translate it automatically into a CP-net. In the translation process, places are added in the net where necessary to get a syntactically correct CP-net, but the structure of the original SADT diagram is preserved. Now, based on the textual descriptions and decision tables, we add inscriptions to the CP-net so that we end up with a complete model ready for simulation. As an example we have taken the SADT diagram from figure 3 and completed the corresponding CP-net with inscriptions. The result can be seen in figure 6. Grey places and arcs are extensions to the original SADT diagram. Inscriptions are also added manually, except from colour sets.

There are a lot of details in the figure, but we will only explain the most important ones. The basic colour set used for most places in the model is a specific record called `InfoObject`. All other colour sets are structurally equal to `InfoObject`:

```
color InfoObject = record Ver:int * Info:string * Av:bool;
color Perf_vrncs = InfoObject;
color Sel_vrncs  = InfoObject;
...
```

The `Ver` field contains a version number. It is used to identify changes when new tokens are issued. `Info` is the information field. The `Info` field of a token from an output arc in the CP-net will have the name of the corresponding arrow in the SADT diagram. `Av` denotes the availability of the token. This field is only used rarely in the model, thus not considered futher in this paper. As most places use the colour set `InfoObject` we could have chosen just to use this colour set name on these places instead of using many different names. But this would violate one of the principles of the SADT method, viz. that the type (or kind) of information flowing on arrows is associated with the arrow name, not with the implementation of the type. As it is the intention that the PFA team, who are used to read SADT diagrams, should be able to read the CPN model, it is preferable to use the arrow names from the SADT model as colour set names in the CPN model.

The grey places are modelling local information about the current version number of the tokens on the input places, except the place called "st" which has information about whether it is the first time the transition occurs or not. Initially all places contains one token with the record value `{Ver=0, Info="", Av=true}` or first on "st" places. With this at hand we can easily implement the decision tables by writing a guard that take advantage of the local information.

The guards in this figure determine when a new version of a token has arrived on the input places. We use the convention that a guard is placed above its transition and the guard-expression is enclosed in "[]". Notice that the third guard ("Determine Variance Impacts") has logic for the case when it occurs the first time and subsequently. Recall the decision table in figure 5.

Then there are the code segments, which are actions called as a side effect when the transition occurs. The "C" in the lower left corner of a transition indicates that the transition has a code segment. The contents of the code segments are not visible in the figure. The code segments are responsible for creating, among other things, values used for generating the output tokens, determined by a set of configurable reference variables. These are: the maximum number of times a transition will fail (or produce abnormal output), the probability of failure, and the amount to decrement the probability of failure each time it fails. In the next paragraphs we will discuss these variables in detail. The code is also responsible for updating the graphics in the time event charts, which shows the relation between time and the occurrence of transitions. We will not show the contents of the code segments because it is just a lot of trivial code for manipulating reference values and the graphical output.

Finally we associate with each transition a delay value modelling that the corresponding activity in the SADT diagram takes time to complete. The value is a global reference to an integer typically of the form:

```
val tA2224533 = ref 1;
```

here meaning that the activity "Determine Variance Impacts" (labelled A2224-533) takes one time unit to process its inputs in order to make a complete output. We have chosen to represent the global reference values like this, because it is then convenient (for the user) to configure the model in the beginning of the simulation (or at any other time) by just reading external files with a set of values and then assigning them directly to the references. The contents of the reference value is accessed by means of the function called par. In general, models with global references are not safe, since side effects might influence the enabling calculations as the global scope of references violate the locality principle. But we

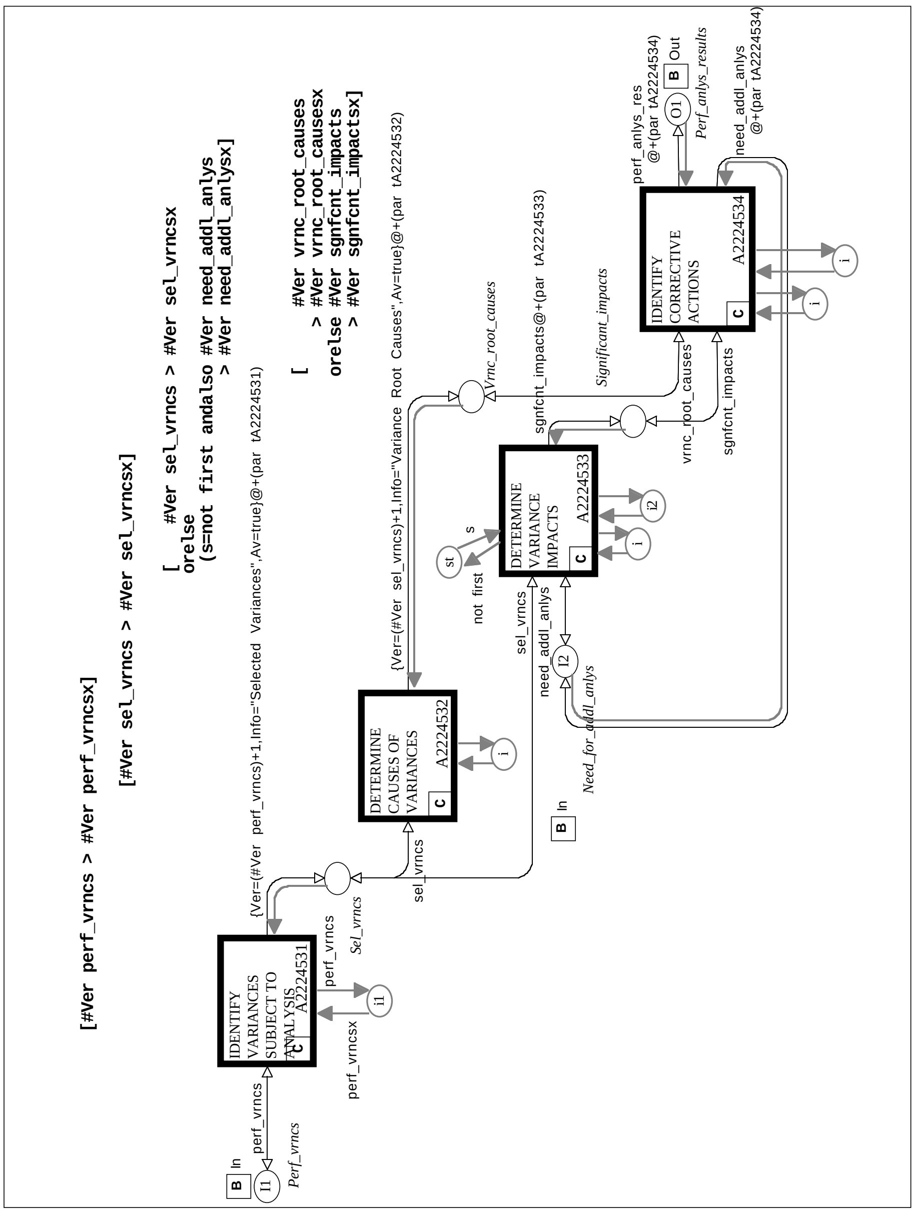

[#Ver perf_vrncs > #Ver perf_vrncsx]
[#Ver sel_vrncs > #Ver sel_vrncsx]
[#Ver sel_vrncs > #Ver sel_vrncsx
orelse
(s=not first andalso #Ver need_addl_anlys
> #Ver need_addl_anlysx]
[#Ver vrnc_root_causes
> #Ver vrnc_root_causesx
orelse #Ver sgnfcnt_impacts
> #Ver sgnfcnt_impactsx]
IDENTIFY VARIANCES SUBJECT TO ANALYSIS
A2224531
DETERMINE CAUSES OF VARIANCES
A2224532
DETERMINE VARIANCE IMPACTS
A2224533
IDENTIFY CORRECTIVE ACTIONS
A2224534
C
B In
I1
Perf_vrncs
perf_vrncs
perf_vrncsx
i1
Sel_vrncs
sel_vrncs
{Ver=(#Ver perf_vrncs)+1,Info="Selected Variances",Av=true}@+(par tA2224531)
{Ver=(#Ver sel_vrncs)+1,Info="Variance Root Causes",Av=true}@+(par tA2224532)
Vrnc_root_causes
st
s
not first
need_addl_anlys
I2
Need_for_addl_anlys
i2
i
sgnfcnt_impacts@+(par tA2224533)
Significant_impacts
vrnc_root_causes
sgnfcnt_impacts
perf_anlys_res
@+(par tA2224534)
O1
B Out
Perf_anlys_results
need_addl_anlys
@+(par tA2224534)

Fig. 6. The CP-net as a result of translating the SADT page from figure 3. Grey places and arcs are extensions to the original SADT diagram.

only use reference values on output arcs, which are independent of the enabling rule in CP-nets.

The four lines related to the CP-net in figure 6 from a typical configuration file have the following format:

```
1     (* A2224531 IDENTIFY VARIANCES SUBJECT TO ANALYSIS *)
1     (* A2224532 DETERMINE CAUSES OF VARIANCES          *)
1     (* A2224533 DETERMINE VARIANCE IMPACTS             *)
1 2 100 20   (* A2224534 IDENTIFY CORRECTIVE ACTIONS     *)
```

The first number is always the duration information. If there are additional numbers on the same line, in groups of three numbers like the last line above, then they have the following meaning: the first determines the maximum number of times the function will fail or generate a request for something, i.e., something unusual. E.g., in this case the abnormal or unusual output is "Need for additional analysis". The second parameter determines the initial probability of failure, and the third and last parameter determines how much the probability of failure is decremented each time the transition produces abnormal output. In the above output file we have that the transition will fail at most two times. The first time, it will fail with a 100% probability and the second time it will fail with a 80% probability. Subsequently the transition will never fail. This simple scheme of handling non-determinism in the model was satisfactory from the point of view of the PFA team.

3.2 Adding Inscriptions, an Example

To give a better idea of the process of adding inscriptions to the CP-net that comes from the automatic translation of SADT diagrams, we will complete the function "Identify Corrective Actions". There are many approaches on adding inscriptions to the CPN model. We chose to keep the amount of structural changes in the model to a minumum, so that it resembled the SADT model as much as possible.

Most of the efforts of adding inscriptions will appear to be rather trivial. Work is in progress to automate this process [10]. Let us first take a look at the corrected table from figure 5 (recall that it had an error that could lead to a blocking of the information flow). The corrected table is located in figure 7. If we cut out the example function "Identify Corrective Actions" from figure 6 with surrounding arcs and places we get the result as in figure 8 (with added inscriptions).

In the process of adding inscriptions we first add extra places (labelled "i"); one for each arc coming in from the left of the transition (the SADT input arrows). This is in order to be able to detect that input to the function has been updated, reflected by an increment in the version number (the `#Ver` field of the token). Therefore we store the latest information on "local" places.

Next, we add initial markings, which is the multi-set `` 1`{Ver=0, Info="", Av=true} ``. On these places there will always be just one token. Thus in order

Arrow label	Operative Cycles							
	Type	1	1	S	S	S	S	Notes
Significant impacts	In	x	x	x		x		
Variance root causes	In	x	x		x		x	
Performance analysis results	Out	x		x	x			
Need for additional analysis	Out		x			x	x	

Fig. 7. The corrected decision table of “Identify Corrective Actions” from figure 5.

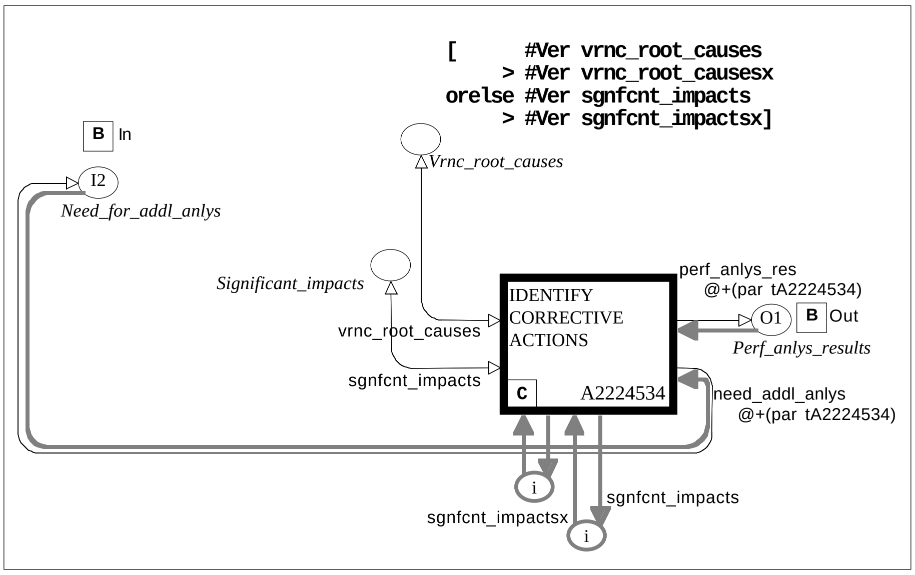


Fig. 8. The transition “Identify Corrective Actions” cut out of figure 6.

not to consume the token upon occurrence of the transition, we add extra arcs (indicated in the figure with grey) in the other direction to put back the same token (input) or an updated token (output and local copies of input). Arc inscriptions are then added so we can refer to tokens from the surrounding places (on output we also add time delay expressions “`@+delay`”). Now we are ready to determine when the transition can occur by creating the guard. We do this by first inspecting the decision table from figure 7. From the table we get that the first time the transition occurs, we need new versions on both inputs. Subsequently the transition can occur if just one of the inputs changes. Thus, a guard that has not been optimised, can look as follows:

```
[ (<first time>     andalso <Variance Root Causes changed>
                    andalso <Significant Impacts changed>)
orelse
(not <first time> andalso (<Variance Root Causes changed>
                   orelse <Significant Impacts changed>)) ]
```

Based on the knowledge that the two precedent transitions initially change the two inputs to this example transition at the same point in time, and that the Boolean expression "(X and Y) or (X and Y)" is equivalent to "Y" we get the optimised guard as seen in figure 8:

```
[        <Variance Root Causes changed>
 orelse <Significant Impacts changed> ]
```

Finally, we get to the code segment. Apart from updating the graphics in the timed event charts, it also produces values for variables used on the output arcs. This transition can issue "Need for additional analysis" to which we will associate a probability. We say that the transition (or the function) fails if it issues "Need for additional analysis" otherwise it will behave normally if it issues "Performance analysis results". The probability of failure will decrease on every failure with some user specified amount (which crudely models that activities learn from their failures). Following is a very rough sketch of the code segment:

```
input <empty>
output <perf_anlys_res, need_addl_anlys>
action
  if <graphics output is on> then
    <update timed event chart>
  else
    <do not update chart>

  if <failed less than the max # of times> then
    if <we should fail this time> then
      <decrement probability of failure>
      <increment number of times failed so far>
      <output> need_addl_anlys
    else
      <output> perf_anlys_res
  else
    <output> perf_anlys_res
```

3.3 Work Schedule

We quickly got experience in predicting how long it would take to complete a CP-net with inscriptions and test it. This knowledge is very valuable when making work schedules that are realistic and thus possible to keep up with.

One part of the work is to add inscriptions to the CP-net that is generated from the SADT diagrams. A minor part of the inscription work is to understand the textual descriptions and the decision tables as provided by the PFA team. The major part is to add the actual inscriptions. The amount of inscription work is somewhat proportional to the number of non-decomposed transitions. These transitions require most of the work, while adding arc inscriptions and extra places are less laborious. The time it took to test a major submodel tended mostly to depend on the complexity of the inscriptions.

Just to give an idea of the amount of work it takes to complete a CP-net, we give a few examples of how long it took to complete three of the major submodels. The numbers in the following table are based on the work of one person.

Model	Number of Pages	Non-decomposed transitions	Time to do inscription work	Time to complete testing
PPC	12	34	6 days	5 days
ERC	17	45	7 days	6 days
PCOT	11	18	3 days	3 days

From the table it can be seen that it takes almost as long time to test and fix a model as it takes to build it. Work is in progress with the intention to reduce the time spent doing the inscription work by automating the process. As mentioned in the previous subsection, adding the inscriptions manually tend to be more or less trivial. Most likely, adding many of the inscriptions automatically to the CPN model can significantly reduce the time spent on testing, because syntactic and some semantic errors are not introduced in the process.

3.4 Pushing the Technology

Because of the size of the total SADT model, it was evident from the beginning of the project that we were forced to break it down into smaller and more manageable submodels. It was fairly easy to do so, because of the inherent modularity of the SADT model. Another argument for breaking down the full model was that it would enable more people to be able to work on the model at the same time, creating a flexible work environment. However, as the technology has advanced significantly since then, it should today be possible to have the full model on one machine and even to have more than one person working on the same document at the same time. Alas, this was not possible during the period of our modelling project.

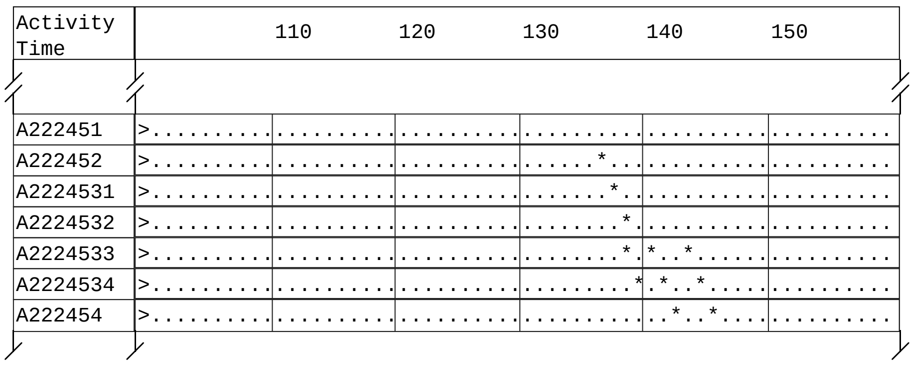
As more and more complex inscriptions were added to the CPN models, we soon realised, that they would not fit into the memory of the machines. This was simply because the machines were running Mac OS version 6.0.7, which only supports 8Mb addressable memory. We wanted to be able to simulate each of the major CPN-submodels individually, with a minimum of extra efforts in setting up an initialisation environment for each submodel. The initialisation uses the knowledge based on assumptions about how the major submodels interact with each other. The interaction is not as well defined on a lower level of decomposition. In fact, that is what we would like to get more information about from the simulation of each major submodel. Therefore it was unfavourable to break up the submodels even further, because it would not make the test of each major submodel very convincing.

Two solutions were found to the memory problem. We applied both in order to be able to proceed our modelling efforts. The first solution was to upgrade our Macintosh machines to Apple's new OS System 7, allowing us to add more physical memory. The second solution was to improve the simulation code generated for the executable models. The effort of investigating different approaches resulted in less code generation and also improved code. This reduced the size of the generated simulation code by an average factor of 40%. As a side effect, the execution time of the simulation models was also reduced significantly.

4 Simulating Individual CPN Submodels

After all the inscription work on the CPN model of the waste management system (as described in the previous section), we next generate simulation code from the CPN model and enter simulation mode. In this section we will look at the output from a typical simulation run.

The CPN model takes input in form of an input file, specified by the user, with parameters to initialise the model. The output of the model is a set of graphical reports showing when in time and how often a transition occurs. We will refer to these reports as the timed event charts. Whenever one of the transitions occurs, it updates the graphics in the timed event charts. It is these reports which were sent back to the PFA team for review. The intention was to give them a better and more detailed understanding of the dynamics behind the SADT models they created.



```
Activity |       110       120       130       140       150
Time     |
---------+----------------------------------------------------------
A222451  |>.........|..........|..........|..........|..........|..........
A222452  |>.........|..........|..........|......*...|..........|..........
A2224531 |>.........|..........|..........|.......*..|..........|..........
A2224532 |>.........|..........|..........|........*.|..........|..........
A2224533 |>.........|..........|..........|........*.|*..*......|..........
A2224534 |>.........|..........|..........|.........*|.*..*.....|..........
A222454  |>.........|..........|..........|..........|..*..*....|..........
```

Fig. 9. Extract of a timed event chart illustrating the relation between time and occurring transitions. See also figure 3 with the corresponding SADT diagram page.

Each CPN submodel produces a page with a graphical report. Figure 9 shows an output from the submodel PPC (Provide Program Control). The discrete time is displayed from left to right and transitions are shown from top to bottom. A "." in the body of the report, means that the transition (or function) did not occur at that time, while an "*" means that the transition occurred once at a particular time. If the transition occurs more than once at the same point in time, a number (instead of an "*") will indicate how many times it occurred. (A "?" will show up if the transition occurs more than nine times, without the time advancing.) The timed event charts are updated during simulation and more pages with charts are created as needed.

4.1 Informal Validation of the SADT Models

The main purpose of our involvement is to validate the accuracy and the completeness of the programmatic functional analysis. More specifically: do the models reflect the intended behaviour and how well do each submodel interact and cooperate with other submodels?

To answer such questions, simulation is used extensively. As submodels are completed, each of them is simulated individually. In order to be able to do that, a primitive initial state is generated based on assumptions on how the environment of the model would behave. This kind of simulation is not only able to identify the worst misunderstandings and bugs, but also to give important and valuable feedback to the PFA team so that they can modify, if necessary, their SADT models accordingly.

Especially a lot of blockings in the information flow (like the one mentioned in section 2.3) have been discovered during the construction and simulation of the individual submodels. Blocking of the flow caused by the interaction of a submodel with the other models, are not investigated or discovered until the models are merged together.

The timed event charts were inspected by the team responsible for the SADT models in order to determine whether the functions, in the waste management program, will be performed satisfactory or not. Often the inspections resulted in requests for changes in the CPN model or redesign of pages in the SADT model. The reports provide important knowledge about how the waste management system can be improved.

5 Merge of the CPN Submodels

We have built a set of submodels that can be simulated individually. In order to validate that they actually work together it is necessary to find a way of merging them together into one large model. Once we have one CPN model containing all the submodels which can be simulated, it is possible to look at the interaction between the major functions.

The effort of having translated the SADT models into CPN, has produced seven CPN submodels each having 7–20 pages, 150–350 places, and 20–50 non-decomposed transitions. Because of the size and complexity of each submodel, the attempt of generating one single graphical CPN model was quickly abandoned[3]. Instead we developed a technique for linking the binary formats of the simulation code from the CPN models.

5.1 Using Standard ML

We use the functional programming language Standard ML (SML) for the purpose of generating code for the CPN simulator. In this project we use an ML compiler, that is developed at the University of Edinburgh, as the simulation engine, and for generating stand-alone executables.

ML was originally designed with the purpose to be used as a meta language for theorem proving. Robin Milner is one of the persons behind the design of Standard ML and David MacQueen introduced the module concept [7,8].

Standard ML is a strongly typed functional programming language supporting, among other things, parametric modules (functors) and type inference. Most ML-compilers have an interactive environment, where it is possible to save compiled declarations to files (e.g., functors), which then later can be loaded into a new environment. These features will show to be useful for the purpose of merging models.

5.2 The Interface between the Major Functions

One SADT diagram page (see figure 2) gives the necessary information about the interface between the functions PSE, DES, ICS, EIS, and PCOT. The remaining two (PPC and ERC) are on a different level of decomposition, and thus not shown here. An arrow between two major subfunctions shows that there is an interface relationship, and thus the location where the subfunctions interact and need to cooperate.

5.3 Generating the Stand-alone Executable

Viewed from outside, only the interface of a CPN submodel is interesting. The interface consists of a set of places from which tokens can be added or removed. The colour set (type) of these places should be specified. As a submodel has to be executed we also need ML-functions in the interface for calculating the enabling and execution of transitions.

The SML compiler (Edinburgh version) has a feature for saving individual parametric modules (called functors) as binary data in a file for later retrieval. The interface specification will therefore be captured in a functor and saved, resulting in a binary file with all the information about the submodel but only the possibility for manipulating markings and execution of transitions through the interface.

For all the CPN submodels we have one common environment with all the colour set declarations among other things. This common environment is a model

[3] However, the technology today will not inhibit one from handling models of this size.

in itself. When we need to insert each submodel into the common environment, SML provides a safe way of doing this. SML is a strongly typed programming language, and thus ensures that the types in the interface of a submodel match with any other submodel it is connected to.

5.4 The Principle of Model Merge

Merging models with our developed technique is similar to modular linking features used in software development environments. The principle of model merge is simple: given a set of models with a well defined interface, create a common environment for controlling the simulation of each submodel (see figure 10).

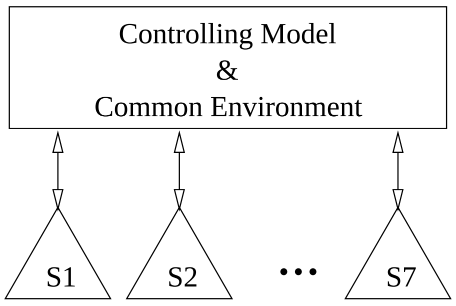


Fig. 10. The principle of merging submodels into a common environment.

Each submodel has an interface which is a set of places. The interface is determined by how the submodels are connected with arcs on the higher level. E.g., from figure 2 we can determine the common places (i.e., the interface) between PSE, DES, ICS, EIS, and PCOT. What do we need in order to control a submodel for the purpose of simulation? We need to be able to read and write markings in the interface places, and we need to have a facility for calculating the enabling and executing a step. The signature (type specification) of the functor in which the submodel is encapsulated and later accessed, typically has the following appearance:

```
functor SubM () :
  sig
    val SubM'calc_enab : unit -> TIME
    val SubM'exec      : unit -> unit

    val SubM'assign :
      (((P1 tms) ref) * ((P2 tms) ref) *
       ((P3 tms) ref) * ((P4 tms) ref) *
       ((P5 tms) ref) * ((P6 tms) ref)) -> bool
      val SubM'get : unit ->
        (((P1 tms) ref) * ((P2 tms) ref) *
         ((P3 tms) ref) * ((P4 tms) ref) *
         ((P5 tms) ref) * ((P6 tms) ref))
    end
  = struct ... end;
```

Here we have a submodel with six places (P1, P2, ..., P6) in the interface. The `tms` denotes the type timed multi-set. Furthermore, there are four ML-functions for accessing and manipulating the places in the interface and for executing the submodel. The first `SubM'calc_enab` calculates the enabling and returns the minimal time at which something is enabled. The function `SubM'exec` executes one step in the submodel. The two last functions `SubM'assign` and `SubM'get`, respectively assigns and reads markings to and from the places in the interface. The assign function returns true if the marking assigned is different from the previous marking in the interface otherwise false. We use this functionality to determine if a recalculation of the enabling is needed (an expensive operation), i.e., when something changed in the interface marking. The four ML-functions above will be the only way of accessing the submodel, once saved to a file.

We can then save each submodel, encapsulated in the functors, into separate files. How does the controlling model look like? Submodels are passive components, so we need to implement a simple simulation engine in the controlling model. The engine should do the following during simulation:

1. Read minimal enabling time from the individual submodels. This step is necessary in a timed simulation, and it identifies what the minimal time is, when one or more transitions are enabled. If the time found is less than the current time (the global model time), we choose the current time instead. The global time is managed by the controlling model, while each submodel has its own local time. The global time is derived from the local time of the submodels; thus the global time is the model time.
2. Select an appropriate candidate model for execution. Candidates found in this step will have the smallest enabling time. We only execute models with the smallest enabling time less than or equal to the global model time (if any exists).
3. Execute one step in the selected model.
4. Transfer markings between places in the interfaces by means of the assign and get functions.
5. Recalculate the enabling and minimal time for involved models, if interface marking has changed. In this step, only submodels with changed markings on interface places, need recalculation of enabling. This step can also advance the current model time if it is needed for anything to be time-enabled.
6. If anything is enabled, restart at 1, else terminate the simulation.

The above simple algorithm is implemented as part of the controlling model.

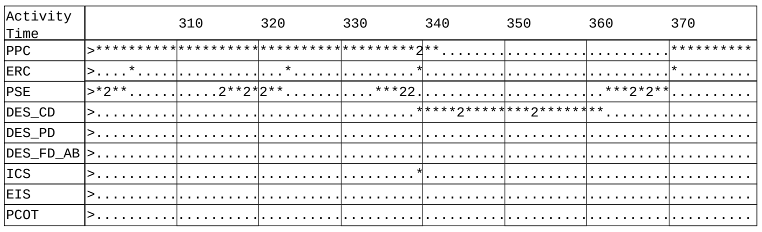
Each submodel is then loaded into the controlling model while the SML type system ensures that there are no type conflict. The SML type system will capture human errors, typically when trying to glue places together between interfaces where the places have differing types.

We now have one single model consisting of all the submodels and we can save this model as a stand-alone executable, ready for simulation.

5.5 Simulating the Merged CPN Model

We are now able to simulate the whole model. The size of the model is large, as a result from merging the major submodels: there are more than 110 pages, 2000 places, and 300 transitions (not counting substitution transitions). The file size of the binary simulation image is in the order of 5MB.

As the simulation proceeds, we end up with report pages from the submodels. Typically a simulation session will produce in the order of 60 windows/pages of timed event charts, which makes it rather difficult to find and view interesting pages. To make it easier to navigate through all this information we introduce the concept of hierarchical reports. It is simply a page where you can get a high level view of what is happening in each of the submodels. An example of a hierarchical report can be seen in figure 11.



Activity Time		310	320	330	340	350	360	370
PPC	>**********	**********	**********	*********2	**........	....	....	**********
ERC	>....*.....	....	...*......	.........*	....	....	....	*.........
PSE	>*2**......	.....2**2*	2**.......	....***22.	....	....	..***2*2**	....
DES_CD	>..........	....	....	.........*	****2*****	***2******	**........	....
DES_PD	>..........	....	....	....	....	....	....	....
DES_FD_AB	>..........	....	....	....	....	....	....	....
ICS	>..........	....	....	.........*	....	....	....	....
EIS	>..........	....	....	....	....	....	....	....
PCOT	>..........	....	....	....	....	....	....	....

Fig. 11. A hierarchical report, giving a high level view of the activity in the model.

A row has a link to the page where the timed event chart from the submodel is updated. To follow a link, the user just double clicks on a row and then the corresponding timed event chart of the submodel will show up on the screen. It is also possible to follow the link backwards, in order to return quickly to the hierarchical report.

In the above figure one can see that PPC first is active. Later the conceptual design phase (DES CD) takes over, then shortly PSE before PPC regains control. Therefore, we also find the hierarchical report useful for giving a quick overview of submodels in where transitions occur. It quickly shows if anything is wrong, e.g., if a submodel unintentionally starts at a wrong time. In the above figure one can see that ICS does something at time 339. This is not intended and needs to be investigated further.

6 Conclusion and Future Developments

Both the CPN team and the PFA team have learned useful lessons and gained valuable experience in this project. The PFA team obtained an improved SADT model. They were "forced" to provide a more formal description of the behaviour of the activities in the SADT model. To accomplish this level of description the PFA team needed to peruse each activity in the model more carefully. As a result they discovered many ambiguities and inconsistencies — the SADT model was essentially incomplete. Consequently, more concise SADT diagrams were created and more importantly many errors were found at a very early stage. Additionally, the ability to simulate the models gave the PFA team new insight into behaviour. Studying the SADT model in itself does not necessarily provide unequivocal information. At the stage of simulation, more errors were discovered and improvements to the model carried out.

The CPN team identified and implemented improvements in the modelling and simulation tool as a consequence of practical work with a large and non-trivial model. Hardware limits forced us to reduce the memory requirements. By rewriting the code used for simulation we improved performance. We developed a technique for merging CPN models and we also created support for parallel working on models. Both techniques performed well in practice.

When we deal with work flow models, a significant part of the translation from SADT to CPN can be automated. In this way the CP-net methodology becomes available for non-CPN experts. More importantly; the project described in this paper contributes to the opening of a path in this area of automatic translation, reducing the turn-around time. Work is in progress to create support for a more automated translation from SADT to CPN [10]. A translation tool is under development, which is specific for work flow models. The tool supports automatic generation of a CPN model, based on the SADT diagrams with additional behavioural information. Furthermore, the tool will also add simulation code for various kinds of analysis of the work flow (e.g., bottle neck analysis), and simulation code for generating textual and graphical reports. The textual output will be in a format such that advanced external analysis tools can be included as needed.

The purpose of the PFA team's work is to describe the work flow of the nuclear waste management program and ensure that their model is consistent and complete. An obvious extension to the model is to take into account mechanisms, i.e., resources like funds and people. This gives the possibility to create more sophisticated and detailed scenarios by means of simulation. An extended analysis of the resulting information can then be performed to determine more precisely the configuration of the nuclear waste management system. Another useful aspect to have in models is time. Time delays are already included in this model, but they were never exploited fully. For most simulations all time delays were set to 1. This was sufficient for validating consistency in the flow of the model. But more advanced results can be obtained by giving individual functions non-trivial delays. The model can then be used to do completion time analysis of functions with realistic simulation scenarios.

Acknowledgements. We would like to thank Kurt Jensen, Jens Bæk Jørgensen, Charles Lakos, and Søren Christensen for patiently reviewing draft versions of this paper, providing constructive critique and food for discussion.

References

1. K. Jensen. Coloured Petri Nets: A High-level Language for System Design and Analysis. In G. Rozenberg, editor, *Advances in Petri Nets 1990*, number 483 in Lecture Notes in Computer Science, pages 342–416. Springer-Verlag, 1991. Also in [3], pages 44–122.
2. K. Jensen. *Coloured Petri Nets — Basic Concepts, Analysis Methods and Practical Use. Volume 1, Basic Concepts.* Monographs in Theoretical Computer Science. An EATCS Series. Springer-Verlag, 1992.
3. K. Jensen and G. Rozenberg, editors. *High Level Petri Nets.* Springer-Verlag, 1991.
4. D. A. Marca and C. L. McGowan. *SADT.* McGraw-Hill, New York, 1988.
5. Meta Software Corporation, Cambridge, Mass. *Design/CPN User's Manual*, 1992.
6. Meta Software Corporation, Cambridge, Mass. *Design/IDEF User's Manual*, 1992.
7. R. Milner, M. Tofte, and R. Harper. *The Definition of Standard ML.* MIT Press, 1990.
8. L.C. Paulson. *ML for the Working Programmer.* Cambridge University Press, 1991.
9. V. O. Pinci and R. M. Shapiro. An Integrated Software Development Methodology Based on Hierarchical Coloured Petri Nets. In G. Rozenberg, editor, *Advances in Petri Nets 1991*, volume 524 of *Lecture Notes in Computer Science*, pages 227–252. Springer-Verlag, 1991. Also in [3] pages 649–666.
10. V.O. Pinci and R.M. Shapiro. Work Flow Analysis. Meta Software Corporation. Cambridge, Massachusetts, 1993.
11. R. M. Shapiro, V. O. Pinci, and R. Mameli. Modelling a NORAD Command Post Using SADT and Coloured Petri Nets. In P. E. Lauer, editor, *Functional Programming, Concurrency, Simulation and Automated Reasoning*, volume 693 of *Lecture Notes in Computer Science*, pages 84–107. Springer-Verlag, 1993.