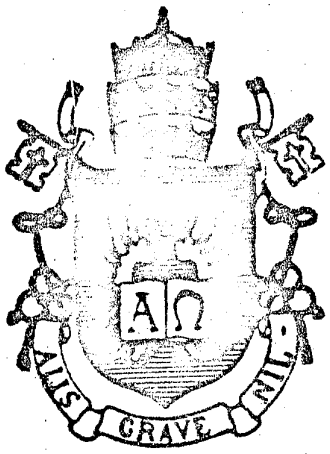


PUC

Series: Monografias em Ciência da Computação,
No. 16/91

TOWARDS A RIGOROUS INTERPRETATION OF ESML - EXTENDED SYSTEMS MODELING LANGUAGE

Gernot Richter
Bruno Maffeo

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 - CEP-22453
RIO DE JANEIRO - BRASIL

PUC RIO - DEPARTAMENTO DE INFORMÁTICA

Series: Monografias em Ciência da Computação, No. 16/91
Editor: Carlos J. P. Lucena July, 1991

TOWARDS A RIGOROUS INTERPRETATION OF ESML - EXTENDED SYSTEMS MODELING LANGUAGE *

Gernot Richter
Bruno Maffeo

* Part of this work was performed while Prof. Richter was visiting the Departamento de Informática of PUC Rio, partially supported by the Convênio Especial GMD/CNPq sobre Cooperação Científico-Tecnológica. Prof. Richter present address: Institut fur Methodische Grundlagen der Gesellschaft fur Mathematik und Datenverarbeitung mbH, Schloss Birlinghoven, W-5205 Sankt Augustin 1.
Part of this work was performed while Prof. Maffeo was visiting the Departmnet for Foundations of Information Technology of GMD, partally supported by the Convênio Especial GMD/CNPq sobre Cooperação Científico-Tecnológica. The main support for this author's research activities is given by the Secretaria da Presidência da República Federativa do Brasil.

This work has also been published by the GMD,as Arbeitspapiere der GMD no. 537.

In charge of publications:

Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC Rio - Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22453 - Rio de Janeiro, RJ
Brasil

Tel.:(021)529-9386 Telex:31078 Fax:(021)511-5645
E-mail:rosane@inf.puc-rio.br

Contents

Part of this work was performed while the author was visiting the Departamento de Informática of PUC-RJ, partially supported by the Convênio Especial GMD/CNPq sobre Cooperação Científico-Tecnológica.

Part of this work was performed while the author was visiting the Department for Foundations of Information Technology of GMD, partially supported by the Convênio Especial GMD/CNPq sobre Cooperação Científico-Tecnológica. The main support for the author's research activities is given by the Secretaria de Ciência e Tecnologia da Presidência da República do Brasil.

Abstract

A graphics-based language known as extended systems modeling language (ESML), which is an extension of the data flow diagram notation for representing control logic in models of real-time systems, is analysed and summarized aiming at a rigorous interpretation of ESML symbols and their combinations. Based on elementary and compact or "high-level" Petri nets (PN), for which a succint introduction is given, formal foundations for ESML, in particular for its transformation schema (TS) notation, are proposed. Translation principles as well as examples of usual transformation and flow patterns are presented in TS notation and PN notation. The obtained PN models can be executed due to their formally defined "token game" which includes concurrency of events and values as tokens. This permits a rigorous analysis of the dynamics of real-time systems with signals, prompts and data flows of various kinds included in a single formalism. The conclusions list general features of the approach relevant to its application and indicate further research tasks.

keywords

concurrent systems, data flow diagram, Petri net, real-time systems, state/transition diagram, systems design, systems modeling, transformation schema

1 Introduction

Several efforts have been undertaken in recent years to equip Data Flow Diagrams (DFD) with extensions for modeling real-time systems [12, 9, 3, 11, 1]. In general, such a system is a subsystem of a comprising system including an external environment with complex dynamics which is monitored and controled through reactions of the real-time system.

To the best of our knowledge, [1] is the latest proposal in this direction. It presents an Extended Systems Modeling Language (ESML) combining features from several independent attempts for representing control logic. Networks of active and passive elements, known as Transformation Schemas (TS), are used as integrated system models in the sense of modeling both data flow and control flow structure. A TS is thus, at the same time, a DFD and a CFD or "Control Flow Diagram". The TS notation is intended to be a system modeling language for (real-time) systems. It includes graphic notation, formation rules and a way of determining the behaviour of the modeled system through "a formalism for token-based execution" which is based loosely on the execution of Petri nets [9].

Approaches like ESML address system modeling on a level considered "natural" to the problem at hand or "abstract" when seen from an implementation point of view. Starting from such a level is certainly more promising than to begin system construction with a

programming language which entails the well-known results like lack of understandable documentation, poor reliability of the system, costly maintenance and virtually no possibility to modify the system. Graphics-based languages support the development of systems in particular at the initial stages of investigation, analysis and design by facilitating comprehension of the problem to be solved and communication with people from the application area.

In a paper on graphics-based languages for modeling software systems [10], the "lack of rigorous, enforceable definitions of the meanings of symbols and symbol combinations in the graphics-based languages" has been identified as one of the major problems encountered by system developers. One way to incorporate formal methods into the process of system development without giving up the benefits gained from informal and semi-formal languages like ESML is to establish principles for the transition from a more intuitive model in terms of these languages to a rigorous model with a formal semantics. This requires two things: First, a mathematically founded, fairly general model for systems and processes which allows to represent all dynamic properties of a system and to explore them by execution or formal analysis, second, mapping rules which define an at least intuitively evident correspondence between an informal model and its formal counterpart. A recent proposal for such an approach is presented in [4]. Preferably, such a technique of formalization should support visualisation by conspicuous symbols with a rigorous interpretation and should also allow to consider parts of the system as autonomous modules. Moreover, a well-defined connection with other system modeling languages (e.g. ERD – Entity/Relationship Diagrams, PDL – Program Design Language) should be possible.

The present article attempts to show that Petri nets (PN) can be useful for defining a rigorous interpretation of graphic symbols and their combinations and demonstrates how the mapping can be done in the case of ESML as the "source language". Independently of these authors, a partial approach to specifying control transformations through Petri nets is pursued in [5]. If such an effort were done for several graphics-based system modeling languages, comparison of their expressive power would become easier and cross-fertilization among these languages as advocated in [10] would be encouraged.

For the reader less familiar with ESML, concepts and terminology of Transformation Schemas are outlined in Section 2. A succint introduction into elementary and compact or "high-level" Petri nets is given in Section 3, however confining itself to the features needed in this paper. Section 4 gives a first account of translation principles from ESML models to PN models, which are then applied in the subsequent Sections 5 to 7. These sections illustrate by means of examples drawn from other sources, how frequently occurring ESML constructs (Transformation Schemata and State/Transition Diagrams) can be "converted" into formally defined executable PN diagrams. Section 8 presents some conclusions from this exercise and suggests further research tasks.

2 Outline of the Transformation Schema Approach

2.1 Elements of a Transformation Schema

The basic elements of a general transformation schema (TS) are "transformations", "flows" and "stores". For reasons explained in Section 4, Principles of Translation, we shall use the terms *operational activity* and *control activity* instead of "flow transformation" and "control transformation", *flow connection* instead of "flow" and *store connection* instead of "store".

With these elements a transformation schema can be constructed being a network where the nodes are activities and the arcs are connections. The present section outlines the main properties of TS elements, the principal TS formation rules and the dynamic properties expressed with transformation schemas.

2.1.1 Activities

Activities are the active elements of a TS. An activity models processing actions upon the content of surrounding passive elements which connect it with other activities. Examples of content are flow values, store values and control information. They are used or consumed as input to the activity or produced as output from the activity.

There are two categories of activities according to their *purpose*:

- A *control activity* models a portion of controlling work inside the system and determines when, and for how long, other activities will execute their actions.
- An *operational activity* models a portion of operating work inside the system, which consists in processing information, material or energy. This includes actions such as accepting, manipulating, producing, storing, transporting and retrieving flowing or stored values within the system and monitoring or monitoring-controling values of the external environment, either in an uncontrolled way or under the control of a control activity.

Activities may also be classified according to their level of *aggregation*:

- A *primitive activity* is considered as a functional unit which is not further detailed in another TS.
- A *non-primitive activity* represents the aggregation of several activities and has a lower-level detailing TS.

Applied recursively, this aggregation process, which obeys an "equivalence principle" (i.e., the low-level TS is a model which may replace the non-primitive activity), aims at reducing the complexity of schemas containing an excessive number (more than 10) of activities.

As a rule, a non-primitive *operational* activity may aggregate operational and control activities whereas a non-primitive *control* activity may only aggregate control activities.

2.1.2 Connections

A TS should be thought of as a network of inscribed nodes denoting activities, interconnected by two categories of also inscribed directed arcs denoting connections.

Flow connections

Flow connections establish causal relationships between the connected activities, expressing that the production of an output by the origin activity, which defines the content of the flow connection, causes the destination activity to immediately respond when this content reaches it.

A flow connection or *flow*, for short, may be further classified according to its kind of *content*:

- A *non-value-bearing flow* (also called control information flow) has no value content. That is, each instance of the flow conveys the same control information, designated by the unique flow inscription.
- A *value-bearing flow* (also called data information, material or energy flow) carries a value content. That is, each instance of the flow conveys a possibly different value for data information, material or energy.

A non-value-bearing flow is either a signal or a prompt. A *signal* serves to indicate an occurrence of an event which is important for the control behaviour of the TS. A *prompt*, which comes with a standard interpretation, models control imposed by one activity on another activity. An input prompt is one of the following:

- trigger prompt: triggers an activity, which cannot be reset during execution;
- enable prompt: permits the execution of an activity;
- disable prompt: aborts the execution of an activity;
- activate prompt: combines the functionality of an enable and a disable prompt;
- suspend prompt: interrupts the execution of an activity;
- resume prompt: continues the execution of an interrupted activity;
- pause prompt: combines the functionality of a resume prompt and a suspend prompt.

A value-bearing flow, associated with data information, material or energy, conveys, respectively, data information, material or energy values which are made available to or are provided by activities. Value-bearing flows are further classified according to the *structure of their content*:

- A *non-structured flow* carries atomic values only.
- A *structured flow* carries structured values of arbitrary complexity.

Another classification of flow connections is by their *dynamic properties*:

- A *continuous flow* carries content (data information, material or energy) made available permanently in a given time interval.
- An *intermittent flow* carries content (control or data information, material or energy) made available occasionally at discrete points in time.

Finally, flow connections are classified according to their level of *aggregation*:

- A *primitive flow* is not further decomposed inside the TS model.
- A *non-primitive flow* is decomposed in a lower-level TS.

A non-primitive *continuous* flow may represent an aggregation of a set comprising continuous and intermittent flows whereas a non-primitive *intermittent* flow represents an aggregation of a set of intermittent flows only.

Store connections

Store connections establish non-causal relationships between the connected activities, expressing that the destination activity has an independent access to the content made available by the origin activity.

So a store connection or *store*, for short, is a repository which may have an associated capacity and whose content possibly has an internal structure. It may be further classified according to *persistence of its content*:

- The content of a *non-depletable store* (data information) is only used, not consumed, i.e., it persists after any input access (which may only occur at discrete points in time).
- The content a *depletable store* is consumed when accessed for input. More precisely, content of a depletable store is organized as FIFO (queue) or LIFO (stack) whose items — associated to signal, data information, material or energy content — are consumed when an input access is made. In the case of material or energy content, this may occur continuously in time.

As with flow connections, also store connections are classified according to the *structure of their content*:

- A *non-structured store* holds atomic values only.
- A *structured store* holds structured values of arbitrary complexity.

In order to avoid unnecessary terminological variety in the definition of the representation language, we are not considering *terminators* as basic elements (as original ESML does). Terminators are used to represent active and passive entities situated in the immediate conceptual neighbourhood of a system modeled in terms of a TS. They specify the interacting environment external to the modeled system.

As the concept of system is recursive, we may consider the modeled system and the external (interacting) environment as components of a comprising system which may, in turn, be modeled in terms of a larger TS. In such a case, the active terminators will be represented by activities and the passive ones will be represented by flow and store connections; no other concepts will thus be needed.

When ESML is applied in systems modeling, the graphical symbol used to represent active terminators (usually called "external entities") is different from the one used to represent an activity. But this should be understood as an aid to make a conspicuous representation of the system border. Passive terminators (stores shared by the system and some external entity) use the same graphical notation as the ones used inside the TS to represent store connections. These terminators and the flows input to and output from the system characterize the interface between the modeled system and the external environment.

2.2 Principal Formation Rules for a Transformation Schema

Viewing a TS as a network of activities interconnected by flow and store connections, a syntactically valid TS must obey formation rules which are summarized in the sequel.

A notation for TS elements (slightly different from that used in ESML) is presented in Figure 2.1. The term *descriptor* stands for a legend which concisely describes the functionality of the activity or the content of the flow/store connection. The term *id* is intended for a unique identification of the activity or store (in the case of flows the descriptor acts also as an identifier). The lowest field of an activity symbol is reserved for a possibly necessary comment, trace identifier, etc.. Only primitive control activities are considered in this paper since there is no functional distinction from non-primitive ones.

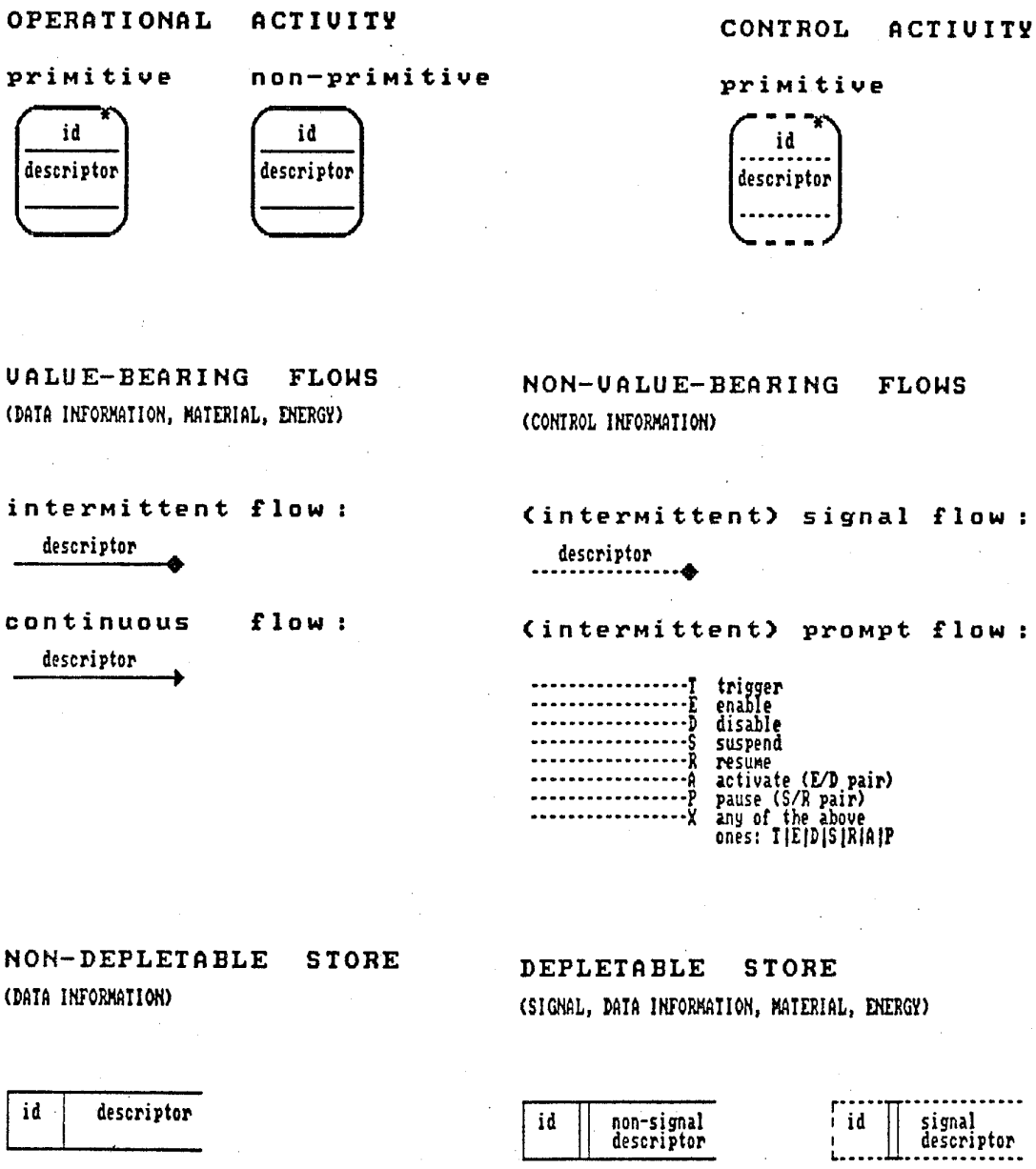


Figure 2.1: notation for the basic TS elements

Figures 2.2, 2.3 and 2.4 are intended to represent graphical "blueprints" for syntactically correct TS constructs. In each of these figures, any primitive activity can be substituted by a non-primitive one, but not vice-versa. Also, not all shown connections need to exist.

Interconnection of activities by flows is shown in Figure 2.2. Firstly, we consider operational activities only. In this case, if a non-primitive activity receives a signal or produces a prompt flow this means that the activity comprises at least one control activity. Secondly, we consider control activities only. If in this case the intention is to model hierarchical control, a prompted control activity cannot send prompts back to the sender of the prompt, neither directly nor indirectly. The last group of flow constructs combines, in each of them, operational and control activities.

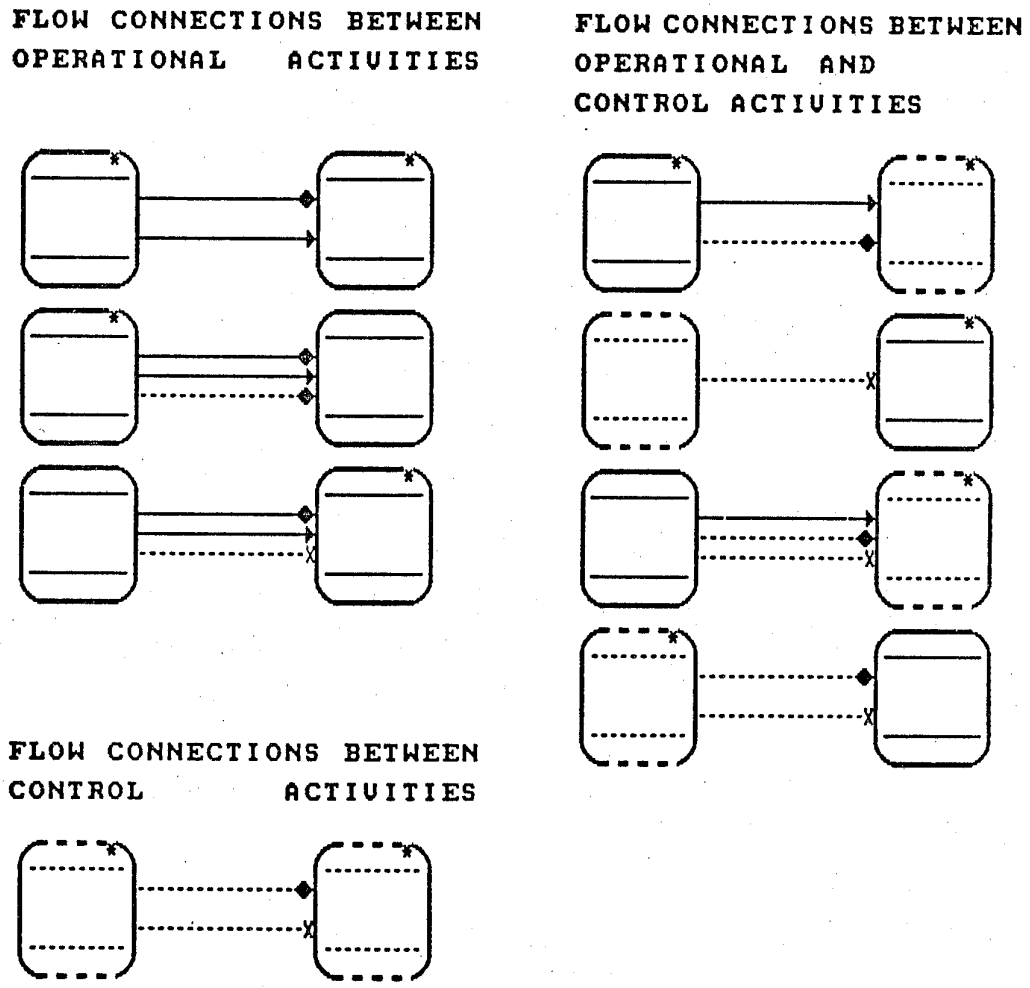


Figure 2.2: activities interconnected by flows

Interconnection of activities by non-depletable stores is shown in Figure 2.3. When only operational activities are involved, an input from the store models a non-destructive use of the store content (reading of stored value(s)) and an output to the store models a change of the store content (insertion, deletion or modification of stored value(s)). When only control activities are involved, an input from the store models a non-destructive use of the store content (reading of stored value(s)), while an output to the store models a modification of stored value(s). Note that the store may only contain data information.

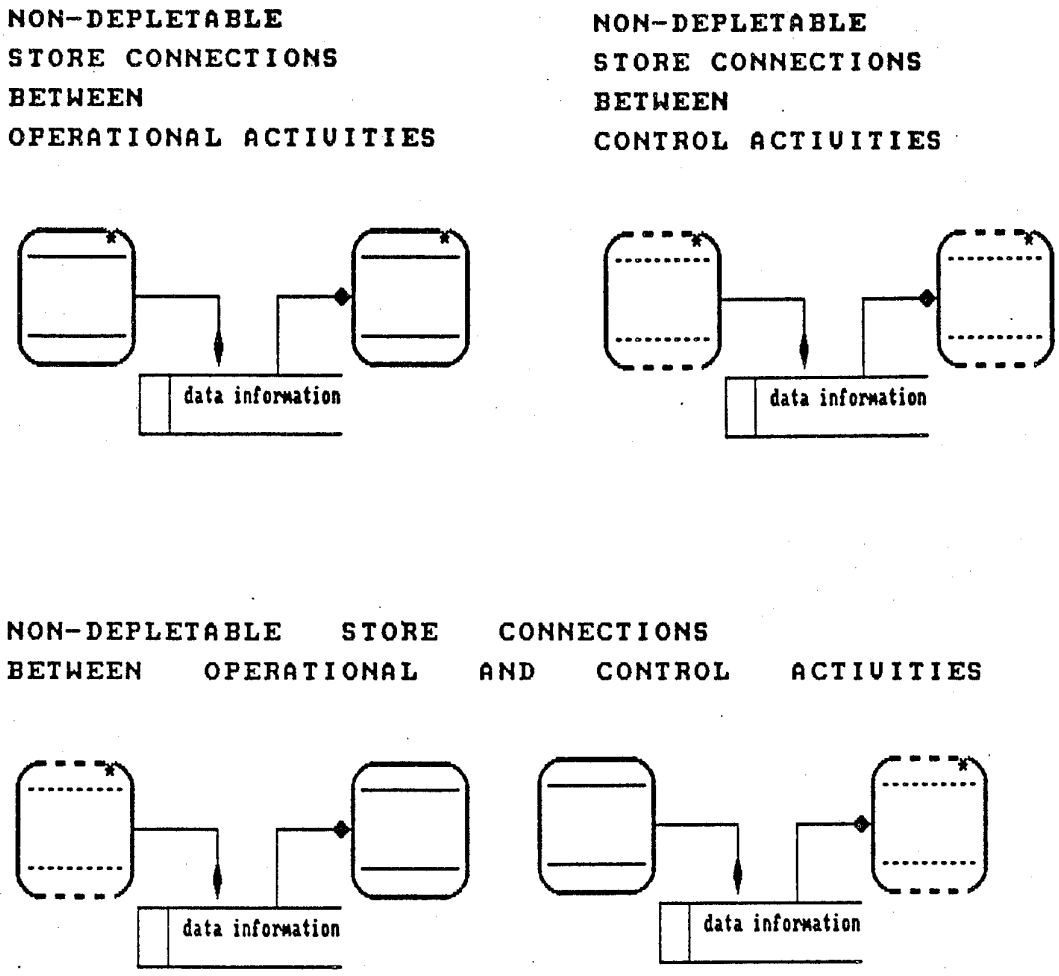
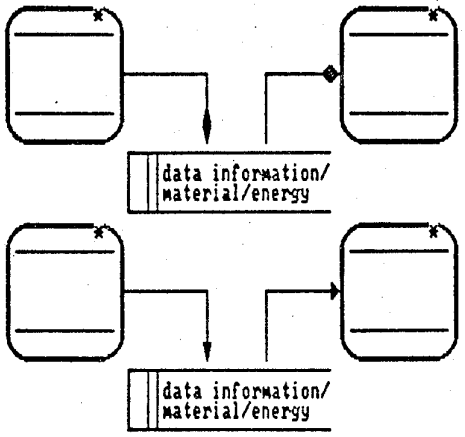


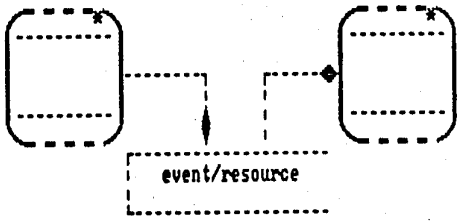
Figure 2.3: activities interconnected by non-depletable stores

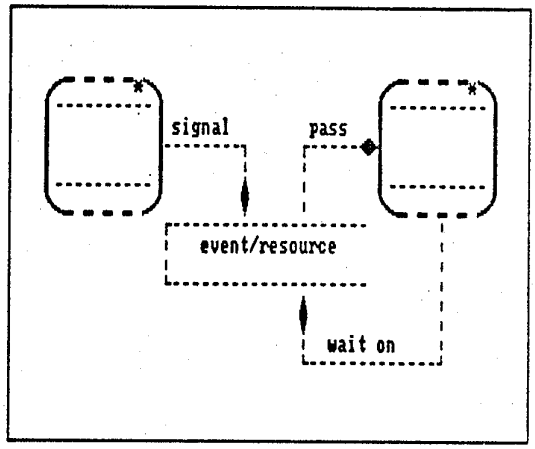
Interconnection of activities by depletable stores is shown in Figure 2.4. When only operational activities are involved, the content of the store may be data information, material or energy. In the case of time-discrete access, input from the store models consumption of an integer number of units of store content, output to the store models production and storage of an integer number of units of store content. In the case of time-continuous access, input from the store models continuous consumption of store content, output to the store models continuous production and storage of store content. When only control activities or both operational and control activities are involved, the content of the store is a signal which indicates the occurrence of some event or the availability of some resource (i.e., a particular case of an event). In the latter two cases, the four ESML constructs include an abstraction of a kind of semaphore control mechanism which is modeled in the framed diagram of Figure 2.4.

DEPLETABLE STORE CONNECTIONS BETWEEN OPERATIONAL ACTIVITIES

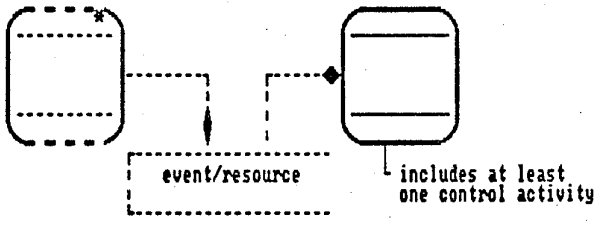


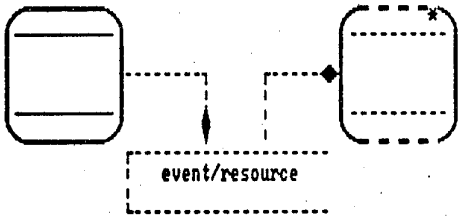
DEPLETABLE STORE CONNECTIONS BETWEEN CONTROL ACTIVITIES





DEPLETABLE STORE CONNECTIONS BETWEEN OPERATIONAL AND CONTROL ACTIVITIES





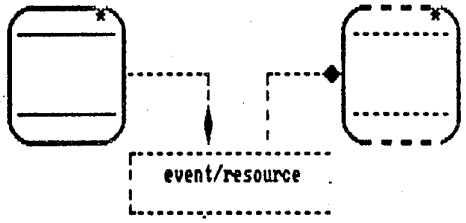


Figure 2.4: activities interconnected by depletable stores
`event` = occurrence of some event, `resource` = availability of some resource

2.3 Dynamic properties

Taking into account the possible interactions between activities, further features of activities are distinguished, which concern their dynamic properties.

The *control mode* of an activity is one of the following six combinations:

- *standing* (with or without a *pause prompt*): The activity can permanently perform actions, it can neither be triggered nor be enabled/disabled;
- *triggered* (with or without a *pause prompt*): The activity starts to perform actions immediately after having received a trigger prompt. The actions of a triggered activity terminate "naturally" (i.e., without intervention of a control activity).
- *enabled/disabled* (with or without a *pause prompt*): The activity can only perform actions after having received an enable prompt and before having received a disable prompt. A disabled activity "forgets" any intermediate results and starts anew when next enabled.

These features are depicted in the following two figures. Figure 2.5 represents the possible changes of state owing to input prompts by means of a transition table, which is an enhanced version of the table found in [1]. Figure 2.6 displays the same changes of state through Transition/State Diagrams.

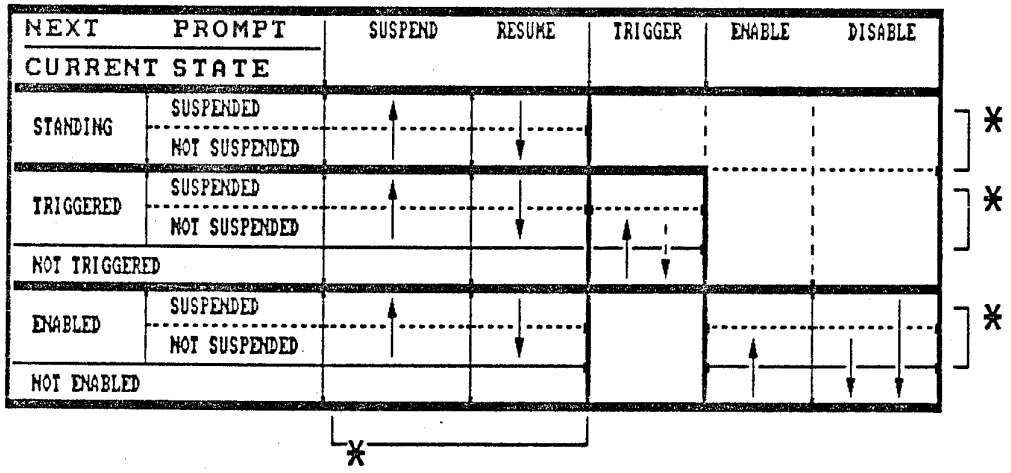


Figure 2.5: changes of state owing to input prompts: Transition Table

An activity with a pause prompt may be suspended and resumed by a corresponding prompt, once its execution was initiated, i.e., once it can perform actions. In all the above three cases, a suspended activity "remembers" its intermediate results and the system context and picks up where it left off when it receives a "resume prompt".

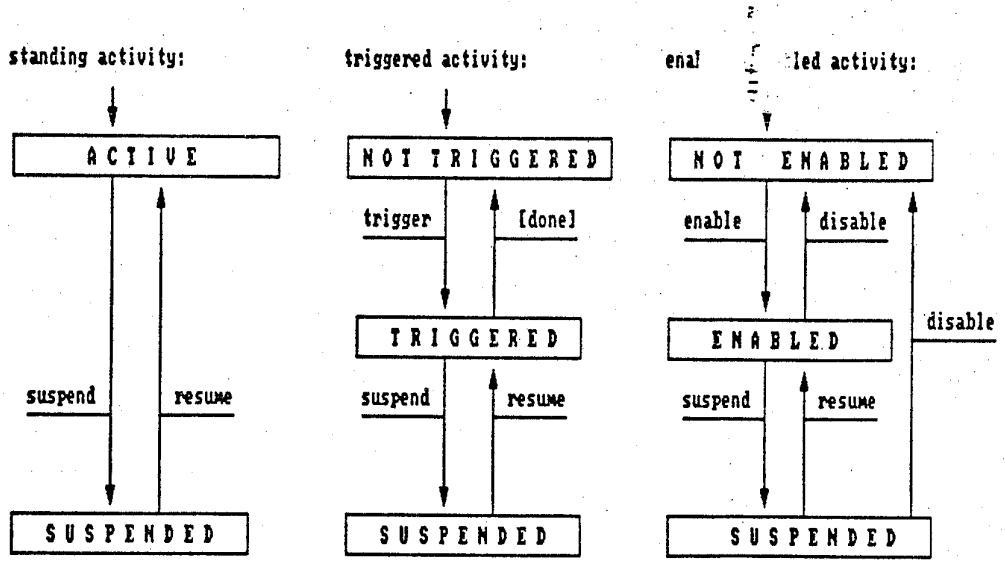


Figure 2.6: changes of state owing to input prompts: State/Transition Diagrams

The *stimulation mode* of an operational activity indicates the way it is caused to perform actions. An operational activity is

- *trigger stimulated*, when, not having intermittent flow(s) as input, its actions are initiated by an instance of a trigger prompt.
- *enable stimulated*, when, having continuous flow(s) as input, its actions are initiated by an instance of an enable prompt.
- *value stimulated*, when:
  - not having continuous flow(s) as input, its actions are initiated by an instance of an intermittent (value-bearing) flow;
  - having a continuous (value-bearing) flow as input, its actions are initiated by the (re-)start of the output production of an activity which is at the origin of that flow connection whose destination is the value stimulated activity under consideration;
- *time stimulated*, when, having neither continuous nor intermittent flow(s) as input, its actions are initiated by the arrival of "point(s) in time" specified as a *built-in* value stimulation which is a special type of value stimulation.

Figure 2.7 illustrates these features by showing, through the use of State/Transition Diagrams, the various ways operational activities may be activated and deactivated.

The details of a time-discrete interaction depend on whether or not a non-zero output delay has been associated with the activity. An *output delay* is interpreted as the amount of time between the establishment of the conditions for producing output and the actual production.

The salient element of a TS, upon which the more complex aspects of the modeled dynamics rely, is the control activity. In ESML, there are two manners of specifying the behaviour of a control activity, both corresponding to a finite automaton with output.

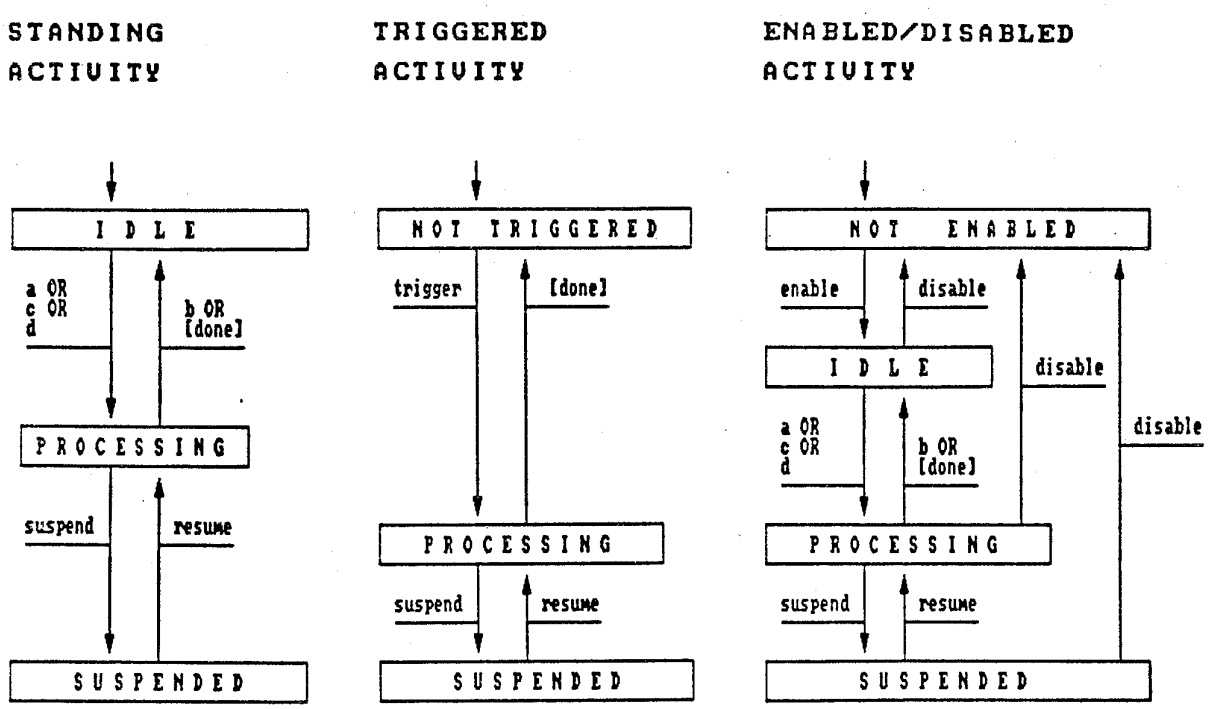


Figure 2.7: activation and deactivation of operational activities (a: continuous flow production (re-)start, b: continuous flow production (re-)end, c: intermittent value-bearing flow arrival, d: point in time)

The first one, called a Moore-type automaton and represented in tabular form (known as Activation Table), is characterized by the absence of input signals and prompts. Rather, each state of the control activity, which normally reflects a situation that prevails in the external environment, is specified by a combination of input values only (of continuous flows or stored elements). They remain constant during a certain period of time (this is the time duration of the state associated to this value combination). It is this combination of input values that determines the output of the control activity. Each change of the input values provokes actions of the control activity which may consist in sending signals and prompts and modifying values of stored elements. It is also admitted to only change the internal state without producing output.

The second way, called a Mealy-type automaton and represented by a State/Transition Diagram, differs from the previous one in that the changes of state of the control activity are determined by the arrival of signals and possibly by prompts and input values, which are values of continuous flows or stored elements accesssed by the control activity. These modeling elements characterize conditions for a change of state. A state is specified by a *descriptor* (a piece of text that indicates the modeled external environment situation). This automaton can produce the same kind of output as those of a Moore-type.

3 Elementary and Compact Petri Nets

Data and Control Flow Diagrams as well as State/Transition Diagrams are the principal modeling tools of the ESML approach. Their usefulness could considerably be increased if one could base their interpretation on formally defined semantics which reflects the dynamic properties — the behaviour — of real-time systems in a precise and unique way without showing unnecessary details.

Dynamic systems modeling on any level of abstraction is the most common application area of Petri Nets (PN). Their structure as well as their dynamics are formally defined. Hence, if we manage to express ESML constructs in PN terms, we have a formal semantics for ESML models of real-time systems.

We assume that the reader is familiar with elementary (one token per place) nets called C/E-Systems in [6]. Formally, a net is a triple $(S, T; F)$ where S and T are two non-empty disjoint sets and $F \subseteq (S \times T) \cup (T \times S)$ is a set of ordered pairs representing adjacency of S- and T-elements. An S-element, also called a *place* (represented by a circle), models a local state or *condition*, a T-element (represented by a box) in the setting of its adjacent places (a subset of F) models a local transition or *event*. In a PN diagram, adjacency of two net elements is represented by an arrow from the first to the second component of the pair. Figure 3.1 shows a PN diagram (PND) modeling two conditions and two events.

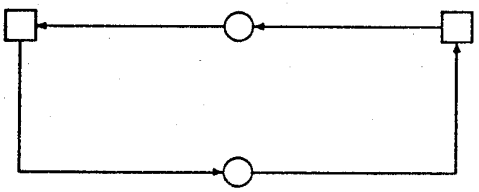


Figure 3.1: PN model of two conditions and two events

Whether a condition holds or not is indicated by a *token* (a black dot in the circle): presence of the token at a place models that the condition is holding, absence the contrary. A distribution of present and absent tokens over the whole net is called a *marking* of the net.

For system specification use we prefer to see a T-element and its setting as a whole and call it a *link*, since the places corresponding to the conditions involved in an event are linked to model the event. In a PN diagram, a box with its ingoing and outgoing arrows or *arcs* and surrounding circles represents a link, while the arcs represent the *branches* of the link. The PN of Figure 3.1 has two links with two branches each.

The dynamic aspects captured by a PN model are expressed by its links. A link defines a particular *change* of the marking of its places. In a (usual) PN with two kinds of arrows, an *occurrence* of the change turns present tokens absent and absent tokens present. A change can only occur if it is enabled. The change defined by a link is enabled if and only if at all entry places of the link the token is present and at all exit places the token is absent. This is the familiar transition rule for Petri Nets with single-token places.

Ease of modeling can considerably be enhanced by introducing two further kinds of branches (represented by special arrows) which are obtained from a particular folding of ordinary branches, i.e. without modifying or extending basic semantics of elementary nets [8]. With respect to a given place, one of these arcs expresses that the condition modeled

by the place ceases to hold and, coincidently, starts to hold — the presence of the token is "restored". Of course, these are two different holdings of the same condition. Such a construct is also referred to as a "side-condition". The other arc expresses the contrary: absence of the token is "tested" (see Figures 3.3 and 3.4 below). It resembles what some authors call an "inhibitor arc".

With these new arrows we distinguish the following four kinds of branches: *altering entry* branches, *altering exit* branches, *restoring entry* branches and *restoring exit* branches. Now that we have more than two kinds of branches, the F-relation can no longer be formalized as $F \subseteq (S \times T) \cup (T \times S)$. Rather we use *branch labels* $\beta = \{\texttt{a-entry}, \texttt{a-exit}, \texttt{r-entry}, \texttt{r-exit}\}$ to label branches according to their kind. A branch label indicates the way in which an S-element is connected with a T-element or, in other words, the way in which a place participates in a link.

Figure 3.2 summarizes the drawing conventions for branches we use in net diagrams.

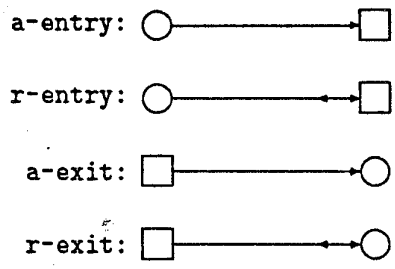


Figure 3.2: representation of labeled branches

The meaning of the four branch labels should be evident from the example given in Figures 3.3 and 3.4. The change defined by the link with three entry places and two exit places is the transition from the marking shown in Figure 3.3 to the marking shown in Figure 3.4. Note that the places with restoring branches do not change their markings.

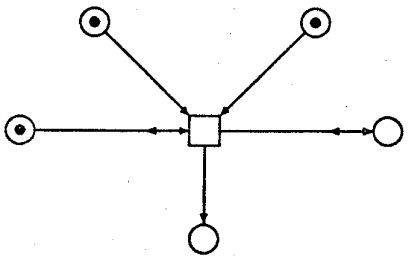


Figure 3.3: marking before change

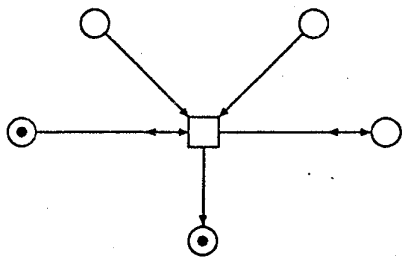


Figure 3.4: marking after change

Non-value-bearing flows shall be modeled with *elementary* nets. For the notions of concurrency, sequentiality, conflict, process etc. the reader is referred to standard PN literature (e.g. [6]). For nets with restoring branches as *basic* elements, some of the standard net-theoretical concepts have to be adapted in a straightforward way.

Value-bearing flows, in order to support visualization and at the same time keep their models in a manageable size, require a compact form of Petri Nets which is achieved by attaching formal annotations to place, link and branch symbols [2].

This allows to collect the possible values of a considered flow into a set which is assigned to a place of a *compact* net. The set of possible values is called the *domain* of the (compact)

place. Each element of the domain represents a place of an *underlying* elementary net. For example, the compact place p shown in Figure 3.5 represents the four elementary places depicted in Figure 3.6.

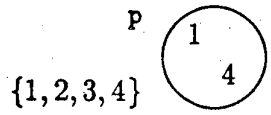


Figure 3.5: compact place

Figure 3.6: underlying elementary places

The domain of a place is defined by its annotation (attached to the circle, e.g., place p in Figure 3.5 is annotated with the term $\{1,2,3,4\}$). The example shows also that presence of the token in the elementary place corresponding to a particular value (e.g. place (p,4) in Figure 3.6) is represented by writing that value in the compact place symbol.

Compact places call for compact links. In the same way as a compact place comprises a set of elementary places, a compact link comprises a set of elementary links and thus defines a set of changes. How are they defined? Annotations of the individual branches (attached to the arrows) and of the link as a whole (attached to the box) specify the set of *underlying* elementary links as will be explained in the sequel.

For the purpose of this paper it is sufficient to consider terms as branch annotations which denote *exactly one element* of the domain of the branch's place (in general, subsets of the domain are considered). A link annotation is a formula in terms of the variables used in the branch annotations. The underlying links defined by the compact link result from the evaluation of the formula. Each assignment of values — drawn from the respective domains — for which the formula yields *True* represents an underlying elementary link whose places are those elementary places which correspond to the assigned values (remember Figure 3.5 and its expansion Figure 3.6).

Given the compact net of Figure 3.7 we get the elementary net of Figure 3.8, since $y = 3$, $x = 5$ and $y = 4$, $x = 7$ are the only combinations which meet the above requirements.

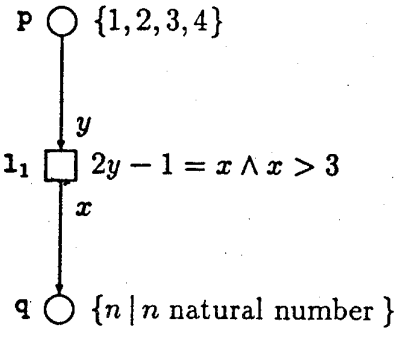


Figure 3.7: compact net

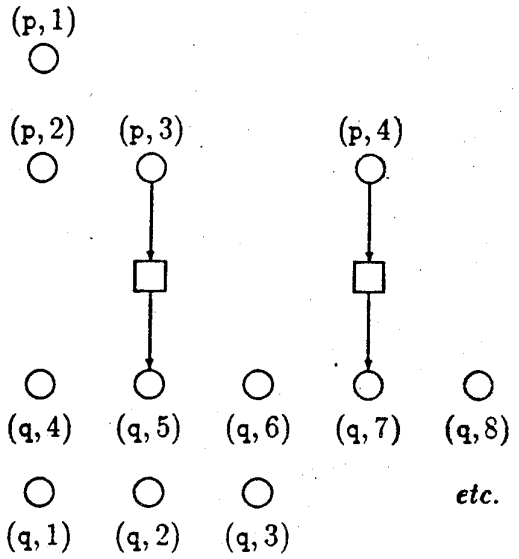


Figure 3.8: elementary net

The "token game" of a compact net is thus defined in terms of its underlying elementary

So, without necessarily suggesting this terminology for use with ESML, we shall use the terms

- *operational activity* instead of "flow transformation" or "data transformation",
- *control activity* instead of "control transformation",
- *flow connection* instead of "flow",
- *store connection* instead of "store".

Roughly speaking, branches with a common place will represent a flow connection or a store connection, a link will represent a control activity or an operational activity. Similarly, given a State/Transition Diagram (STD), a place will model a state, a link will model a transition. This seems to be a quite obvious set of translation principles.

In the world of DFDs, activities with input *flow* connections are sometimes assumed or required to react "immediately" as soon as an input flow has been produced. Apparently, this rule is intended to allow for "time-free" models in the sense of disregarding actual time and duration of flows and activities. However, we do not think that time-free modeling implies or requires "zero-time" interpretation of flow connections. Rather, the idea is to hide actual time relationships between activities without ignoring that any implementation of a flow connection is a technical construct which causes the receiving activity to react "without delay", i.e. without a delay *additional to* the one given by the physical implementation. In other words, arrival or change of an input flow causes the destination activity to start (or continue) within a non-zero time span which depends on the physical properties of the implementation. Although a functional model will not reflect non-essential time aspects of the object system, one should keep in mind that in any implementation the effects of activities can only propagate with finite velocity. Consequently, we shall base our translation from ESML to PN on the principle that an "essential" model should disregard actual time values as far as they are not relevant for the considered functionality without assuming them to be zero.

As opposed to input flow connections, activities with input *store* connections are not assumed or required to react immediately as soon as an input content is available. So, the production of an output content by the emitting activity does not cause the destination activity to react immediately or, in terms of [12], "has no immediate effect on" the destination activity. From an implementation point of view the interpretion is obvious: Arrival of an input content establishes only one of usually several conditions which enable an activity, but it doesn't cause it to operate. The store content remains available for any period of time. Of course, it can't be accessed earlier than permitted by the physical properties of an implementation of the store connection. In other words, an additional delay is technically possible, though not required.

Our approach to translating ESML connections into PN diagrams is to omit any timing on the level of functional modeling, i.e. not to represent time related properties of flow and store connections. This does not preclude other dynamic properties from being captured by PN constructs, e.g. loss of information if the destination activity is not ready or production of some error signal when a flow arrives before a particular store content became available. Where time *is* considered relevant for a model one can model the course of time by means of PN constructs with a formal condition/event semantics (see e.g. [7]). Where it is important to model that an activity should not be considered as an instantaneous transition from one state to another, a "first event" and a "last event" could be introduced in order to capture the notion of duration.

net. However, it is only in special cases that reference to the underlying net becomes necessary. In most cases, a diagram of a compact net allows to directly visualize the dynamics of a system.

For the purpose of this paper we can adopt a further restriction: *no more than one value* can be present at any place (in general, several values are allowed). In the model of a value-bearing flow, presence of a value means that this is the current value or content of the modeled flow.

Occurrence of a change means that the *values* (like "high-level tokens") at each altering entry branch disappear, at each restoring entry branch remain present, at each altering exit branch appear, and at each restoring exit branch remain absent (same principle as with Figures 3.3 and 3.4).

As an example, consider the preceding compact net (Figure 3.7) where link l_1 defines two changes (see Figure 3.8). Let 3 be the current value in place p (place (p,3) with token present). If value 5 is absent from place q (place (q,5) with token absent), then that one of the two changes is enabled which turns value 3 absent in p and value 5 present in q. When this change occurs, 3 disappears from p and 5 appears in q — or, in terms of the underlying net (Figure 3.8): the token becomes absent in (p,3) and present in (q,5).

Due to their "elementary net semantics", compact nets can easily (and formally) be combined with elementary nets. It seems to us that it is this feature which provides a powerful tool for real-time systems modeling, since it allows to combine control flow and data flow in a natural manner. We shall try to substantiate this claim in the following sections.

4 Principles of translation

In the sequel an attempt is made to underlay ESML diagrams with a formal semantics based on Petri Net Theory. It is not our intention to propose any extension or modification with respect to ESML, but to show common dynamic features of ESML constructs by using a uniform and mathematically founded notation for systems with concurrent processes, i.e. Petri Net diagrams (PND).

As the reader might have observed in section 2, we use a slightly modified terminology because, to our mind, the terms "flow" and "control" in connection with "transformation" do not adequately reflect the interpretation intended in ESML. They are even prone to misunderstanding, for *both* kinds of transformation represent activities of flow processing. Thus, for the purpose of this presentation, we suggest a terminology which stresses what is specific by referring to the organizational functions those activities have in a system. While "control" indicates influence over some other activities which are then controlled, the latter are activities in the sense of units of work in which data, material or energy is processed under the control of control activities. We propose to contrast them with control activities by calling them "operational" activities.

Moreover, we are not comfortable with the use of "flow" in the sense of a system component. Strictly speaking, a flow is an act of flowing or an uninterrupted movement or continuous transfer of something or the like. But it is not a medium through which a flow is achieved. It seems therefore more appropriate to allude to the purpose of a "flow" which is to connect several activities in the sense of a generalized channel. The term *connection* has been chosen to reflect this meaning.

A refinement of an ESML control activity like the STD of Figure 4.1 is represented by a PN diagram with a basic structure as outlined in Figure 4.2. It goes without saying that the concrete PN diagram will depend on the details of a given STD.

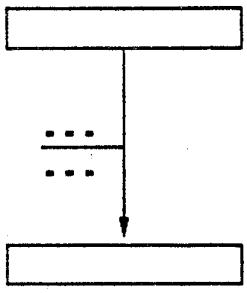


Figure 4.1: State/Transition Diagram schema

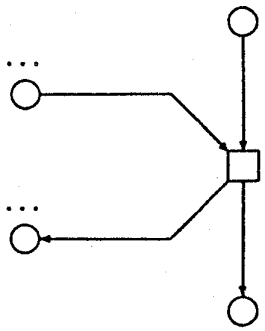


Figure 4.2: PN diagram schema

The variety of features dealt with in ESML is clearly not captured by these schematic examples. They rather should outline the general idea of our approach to a formal ESML semantics. We shall use combinations of elementary and (simple) compact nets in order to reflect the dynamics of ESML constructs, i.e. their execution rules. Sections 5 to 7 elaborate on the functional characteristics of various kinds of flow connections, store connections, control activities and operational activities.

5 Connections between activities

5.1 Flow connections

The following examples are to illustrate the proposed translation of ESML connections into PN constructs. In particular, one should take into account that the connected activities are not shown in a complete functional context where usually several flows enter into or originate from a considered activity. In Figure 5.2, for instance, link u alone would not suffice if the task were to model an activity which produces a signal whenever the temperature exceeds a given value. Not only would it be necessary to add a further branch from u to an elementary place (as e.g. in Figure 5.7, left arrow) and attach an annotation, say, $t > 30$ to the box, but also had one to provide further PN constructs to ensure that only after the temperature has dropped below the limit a situation of exceeding will be signalled anew.

Value-bearing flows:

A value-bearing flow connection ("data flow" in [12]) is modeled by at least two links with a common *compact* place. The domain of the place is the set of possible flow values. Remember that, in the context of this paper, we impose a restriction with respect to the presence of values in a place: No more than one value can be present at any time.

For a flow connection with *continuously available* values the correspondence is shown in Figures 5.1 and 5.2.

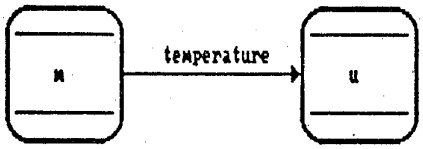


Figure 5.1: TS of a value-bearing continuous flow connection

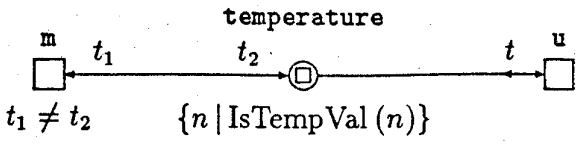


Figure 5.2: PND of a value-bearing continuous flow connection

The small box within the symbol for the place `temperature` stands for a particular value drawn from the domain of that place which is defined by the place annotation $\{n \mid \text{IsTempVal}(n)\}$. Place `temperature` pertains to two branches of link `m` whose symbols are drawn one over the other according to the drawing convention depicted in Figure 5.3, which is also used for branches without annotations.

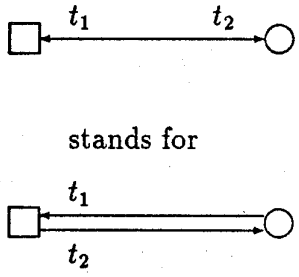


Figure 5.3: double arrows

Overlaid symbols for altering branches should, however, not be confused with symbols for restoring branches.

Link `m` models an activity which *modifies* the current value of temperature, i.e. which causes the old value to disappear and the new value (different from the old one) to appear in place `temperature`. Link `u` models an activity which *uses* the current value of temperature, i.e. which reads the current value without changing or consuming it. The initial value of temperature is either given by an initial marking of the net or produced by another activity not modeled in the example. Remember that a missing link annotation (as with link `u`) is equivalent to the formula *True*. This is to say, that the set of events defined by the link is not further restricted.

For a flow connection with *intermittently available* values the correspondence is shown in Figures 5.4 and 5.5.

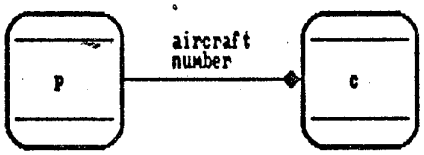


Figure 5.4: TS of a value-bearing intermittent flow connection

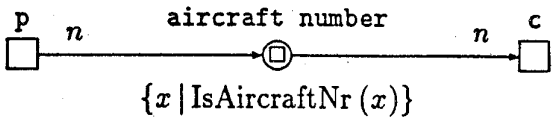


Figure 5.5: PND of a value-bearing intermittent flow connection

Link `p` models an activity which *produces* an aircraft number, i.e. which causes a value drawn from the domain of place `aircraft number` to appear in that place. Link `c` models an activity which *consumes* the aircraft number, i.e. which reads it and causes it to disappear.

Non-value-bearing flows:

A non-value-bearing flow connection ("event flow" in [12]) is modeled by at least two links with a common *elementary* place. The distinction between "signals" and "prompts" is expressed by the way the places are linked.

For a *signal* the correspondence is shown in Figures 5.6 and 5.7.

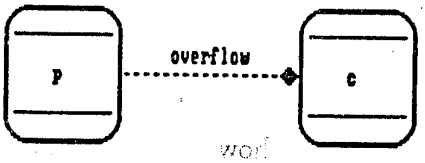


Figure 5.6: TS of a non-value-bearing (signal) flow connection

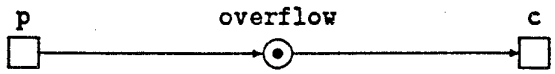


Figure 5.7: PND of a non-value-bearing (signal) flow connection

Link `p` models an activity which *produces* a signal indicating the occurrence of an overflow. In other words, the token in place `overflow` turns present which means that condition `overflow` holds. Link `c` models an activity which *consumes* the signal, i.e. which "reads" it and causes it to disappear. In other words, the token in place `overflow` turns absent which means that condition `overflow` no longer holds. Here we have an example of a potentially misleading place designation. It is not the condition "overflow exists" which is modeled by `overflow` but, strictly speaking, the condition "overflow has been indicated, no activity has responded so far".

For a *prompt* the correspondence is shown in Figures 5.8 and 5.9.

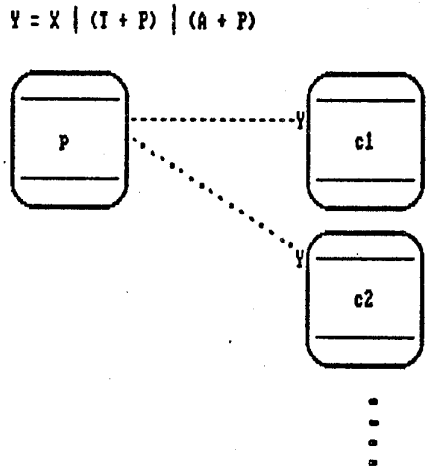


Figure 5.8: TS of a non-value-bearing (prompt) flow connection

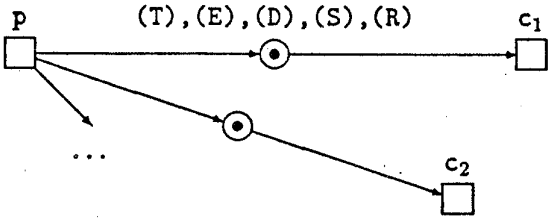


Figure 5.9: PND of a non-value-bearing (prompt) flow connection

Link `p` models an activity which *produces* one of the five possible prompts (of course, only one of the five place designations `(T)` ... `(R)` is attached to a place modeling a prompt). Presence of the token in a place `(T)`, for example, means that a trigger prompt has arrived or, more precisely, that the condition "trigger has arrived, but was not yet responded" holds. In Section 6.2 we shall see that activities modeled by links c_i are more complex than suggested by Figure 5.9.

The purpose of a non-value-bearing flow connection is to control the destination activity. As soon as the latter actually reacted, the condition which ceased to hold can start anew

to hold, since the reaction on signals and prompts is always "destructive": presence of the token is consumed because the condition does no longer hold for the intended purpose once it has "done its job".

5.2 Store connections

Non-depletable stores:

A non-depletable store is a repository for a value-bearing flow content. It is modeled by a compact place. Its "value" (content) is usually a complex information object (not shown in this presentation). The initial content of a store is either given by an initial marking of the net or produced by another activity not modeled in the example.

For a non-depletable store connection the correspondence is shown in Figures 5.10 and 5.11.

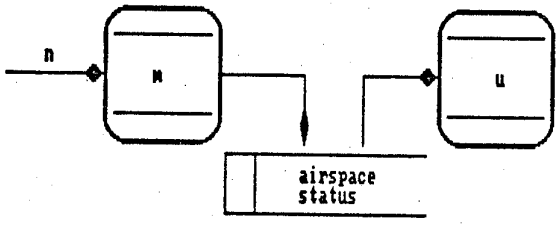


Figure 5.10: TS of a non-depletable store connection

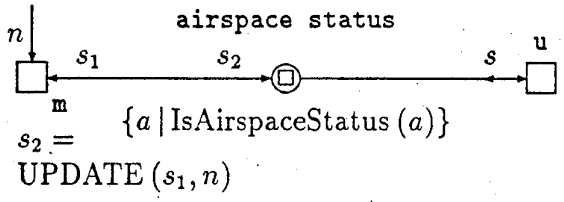


Figure 5.11: PND of a non-depletable store connection

Unlike the model of a flow connection with continuously available values this model specifies modification as being a function of the current value. In the example, a new item n is obtained from somewhere, and link m models an activity which *modifies* the current store content (see annotation beneath box m): the new status s_2 which is to appear in place airspace status is defined to be the result of applying an operator UPDATE on the the old status s_1 and the new item n. Link u models an activity which *uses* the current airspace status, i.e. which reads the current content without changing or consuming it. Of course, u is assumed to be activated by some further condition not shown in the example.

Depletable stores:

A depletable store ("buffer") is a repository for a value-bearing flow content or for several copies of a non-value-bearing flow content. The latter case will not be elaborated on in this article. A depletable store connection for value-bearing flows is modeled by a compact place whose content is accessed according to an internal organization (FIFO or LIFO).

For a depletable store connection the correspondence is shown in Figures 5.12 and 5.13.

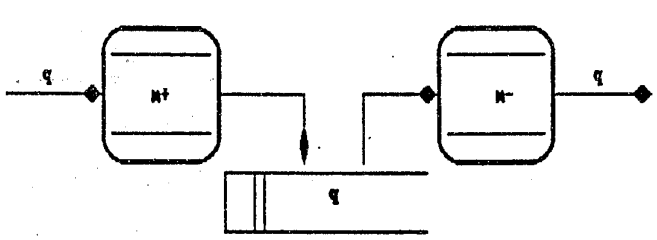


Figure 5.12: TS of a depletable store connection

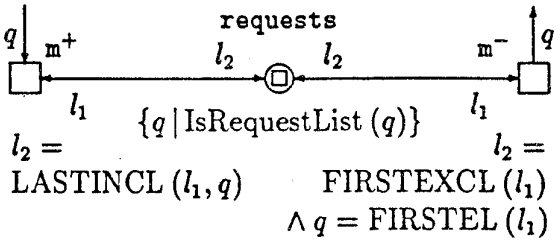


Figure 5.13: PND of a depletable store connection

Link $\mathtt{m}^+$ models an activity which *modifies* the current store content by *adding* some item to it. In the annotation of link $\mathtt{m}^+$, we assume an operator LASTINCL which includes a request q (obtained from somewhere) as the last element in a list of requests. The list l_2 which is to appear in place `requests` as its new marking is defined to be the result of LASTINCL. Link $\mathtt{m}^-$ models an activity which *modifies* the current store content by *removing* some item from it. In its annotation, we assume an operator FIRSTEL which yields the first element of the list of requests l_1 and another operator FIRSTEXCL which excludes that element from the list. The list l_2 which is to appear in place `requests` as its new marking is defined to be the result of FIRSTEXCL, while request q excluded from the old list appears in some place not shown in the example. (Remember that the scope of a variable is a link rather than the whole diagram.) The underlying organization of the store connection is assumed to be of type FIFO. There are, of course, many ways of modeling a FIFO organization, in particular when concurrent access to both ends of a queue is desired or required.

6 Control activities

ESML distinguishes two modeling levels for control activities:

- the level of "transformation schemata",
- the level of "transformation specifications".

While the transformation schema of an ESML model shows the causal relationships between control and operating activities, the specification of a control activity gives a detailed description of its logic. We'll restrict ourselves to the use of State/Transition Diagrams (STD) for specifying control activities, although this is not the only representation allowed in ESML.

6.1 Transitions between states of control activities

Figure 6.1 shows a TS for a control activity CF1 with value-bearing (a and b) and non-value-bearing (signal $\mathtt{s}_1$) input and with value-bearing (c) and non-value-bearing (signal $\mathtt{s}_2$, prompt `A`) output.

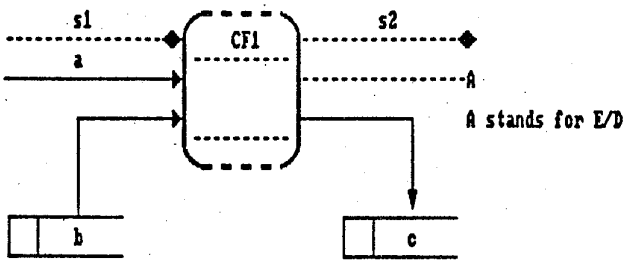


Figure 6.1: TS of control activity CF1 with flows and non-depletable stores

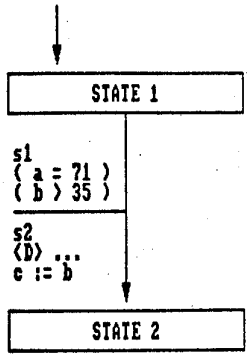


Figure 6.2: STD specifying the behaviour of control activity CF1

A specification of the behaviour of CF1 is given by the STD shown in Figure 6.2. The transition will occur only if the three input conditions hold together, i.e., the signal s_1 is present and $a = 71$ and $b > 35$ are both *True*, and if the activity is in STATE 1. The signal will be lost, if any of the three other conditions does not hold when it arrives.

The PN depicted in Figure 6.3 provides a formal definition of the dynamics expressed by the previous STD.

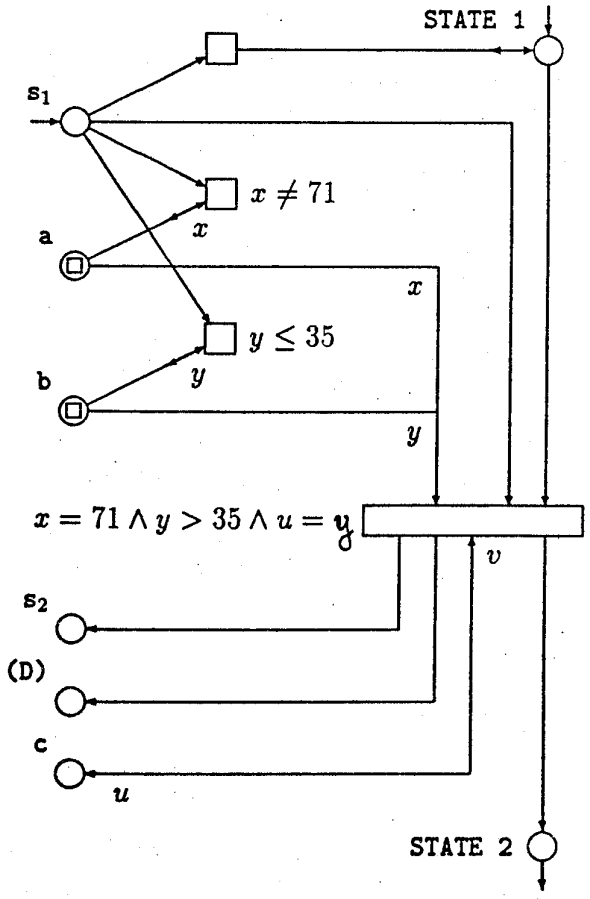


Figure 6.3: PND for Figures 6.1 and 6.2

Figure 6.4 shows a TS for a control activity CF2 without value-bearing input, but with two independent non-value-bearing inputs (signals s_1 and s_2).

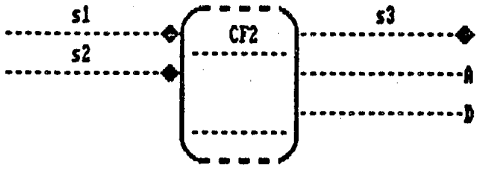


Figure 6.4: TS of control activity CF2 with non-value-bearing flows

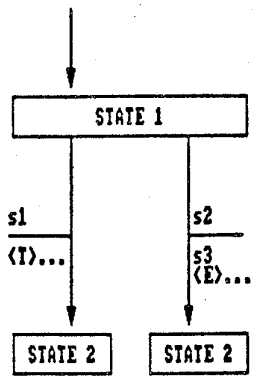


Figure 6.5: STD specifying the behaviour of control activity CF2

A specification of the behaviour of CF2 is given by the STD shown in Figure 6.5. There are two mutually excluded state transitions. If both signals are present and the activity is in `STATE 1`, a conflict exists between the two possible transitions. The STD does not specify, which one will occur. A signal will be lost, if the activity is not in `STATE 1` when it arrives.

The PN depicted in Figure 6.6 provides a formal definition of the dynamics expressed by the previous STD.

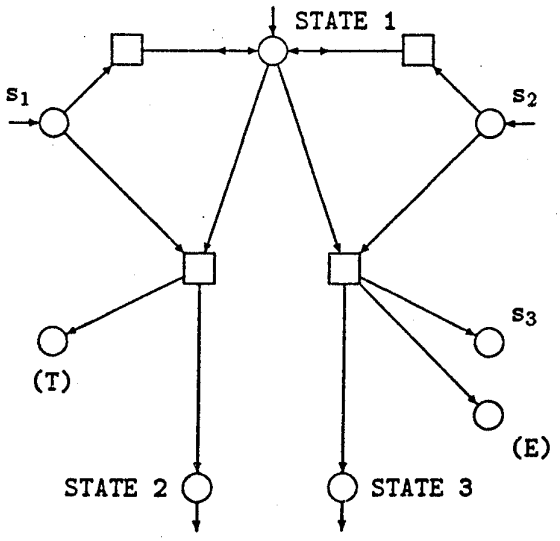


Figure 6.6: PND for Figures 6.4 and 6.5

Figure 6.7 shows a TS for a control activity CF3 without value-bearing input, but with two independent non-value-bearing inputs (signal s_1 and stored control information s_2). Control information s_2 is supposed to be obtained from a depletable signal store.

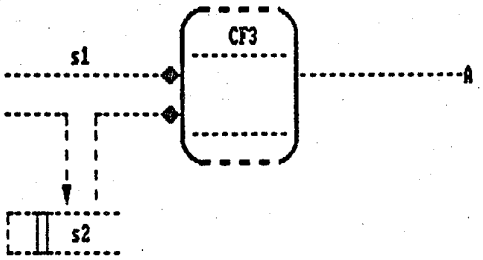


Figure 6.7: TS of control activity CF3 with non-value-bearing flows and depletable store

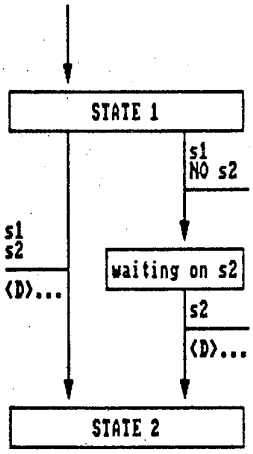


Figure 6.8: STD specifying the behaviour of CF3

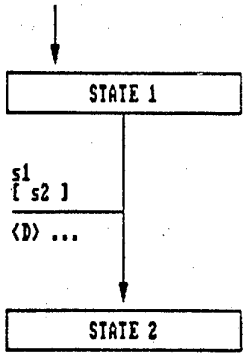


Figure 6.9: abbreviation for the STD of Figure 6.8

A specification of the behaviour of CF3 is given by the STD shown in Figure 6.8. In this case, signal s_1 will not be lost if there is no item in the control store when it arrives. Rather, CF3 will enter in an intermediate waiting state `WAITING ON` s_2 until s_2 has arrived. If upon arrival of s_1 signal s_2 is already present, the transition from `STATE 1` to `STATE 2` occurs in one step. However, the signal s_1 will be lost, if the activity is not in `STATE 1` when it arrives.

As this is a rather common structure in control activities, we propose in Figure 6.9 an abbreviated representation, using `[...]` to indicate optional presence of s_2 upon arrival of s_1.

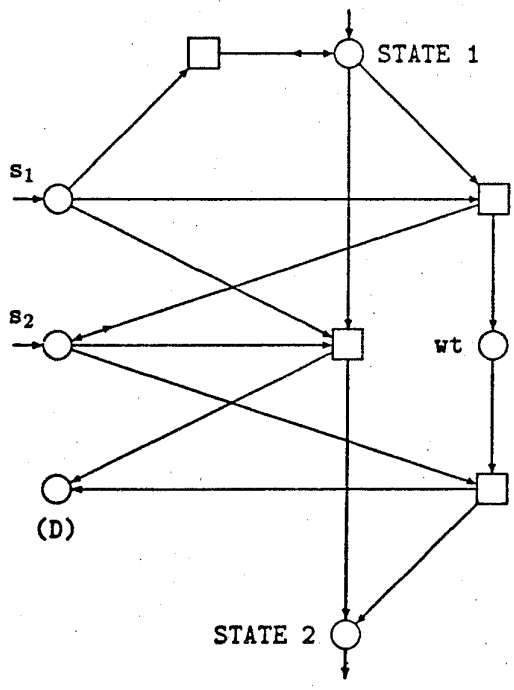


Figure 6.10: PND for Figures 6.7 to 6.9

The PN depicted in Figure 6.10 provides a formal definition of the dynamics expressed by the previous STD (the place `wt` models the state `WAITING ON` s_2).

6.2 Prompts and control activities

On the level of transformation schemata, six basic structures are distinguished with respect to "input prompts". For the sake of visualization, the net diagrams show only those parts of an assumed complete net which define the reactions of a control activity to incoming prompts. For the other dynamic aspects see the preceding subsection.

Notice that the net models of prompted control activities conceive the transitions defined by a transformation specification as forming a sequential process. Concurrency is possible *between* control activities rather than within a control activity. Each PN diagram shows an initial marking which models a possible initial state of the activity.

It should be emphasized that the PN models were designed to show the basic functional components of input prompt processing in control activities. This is to say that certainly more streamlined or more elegant PN models can be constructed for specific ESML diagrams.

Standing control activities

The ESML diagram of Figure 6.11 (a transformation schema) characterizes a control activity without any input prompts. Its meaning is specified by the PN diagram of Figure 6.12.

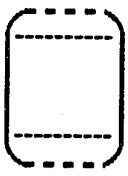


Figure 6.11: control activity without input prompts

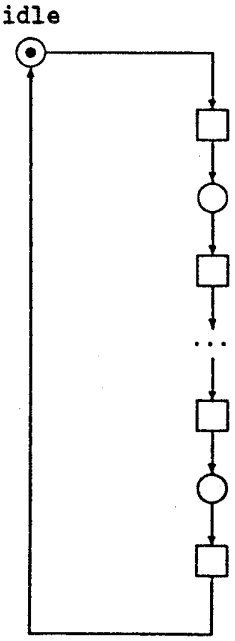


Figure 6.12: PND corresponding Figure 6.11

The PN model shows that a standing control activity without input prompts is always enabled to be performed. The state represented by the token in place `idle` can not be distinguished from a control point of view. It simply indicates a kind of internal home state of the activity.

Adding a suspend and a resume prompt to a standing control activity we get the ESML diagram of Figure 6.13. Its semantics is formally specified by the PN diagram of Figure 6.14.

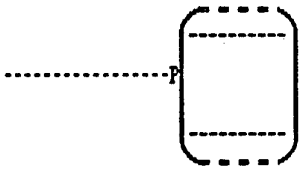


Figure 6.13: control activity with a pause (S/R) input prompt

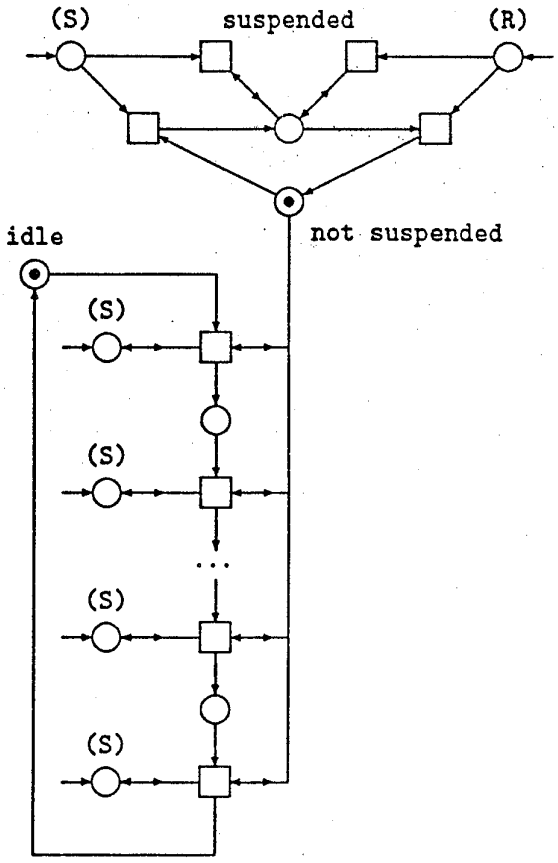


Figure 6.14: PND corresponding Figure 6.13

The PN model results from adding a suspend/resume "module" to the previous model. For the sake of readability, place (S) is represented several times. Multiple representation of the same net element (place or link) is merely a graphical convenience. An equivalent diagram is obtained by drawing only one symbol with usually many arcs. At any instance, a suspend prompt (token in place (S), i.e. in *all* symbols representing (S)) and/or a resume prompt (token in place (R)) can arrive. This is indicated by the dangling arcs entering the place symbols. The two places suspended and not suspended in the middle model the situations of being suspended or not suspended. Initially, the PN model has a token in place not suspended and in an arbitrary place of the cycle.

The model shows how a suspend – resume sequence works, in particular, how a resume prompt causes the suspended activity to continue from where it left off. It also shows that a suspend prompt disappears if the activity is already suspended and a resume prompt disappears if the activity is not suspended. Note that the places suspended and not suspended are complementary to each other.

Triggered control activities

The ESML diagram of Figure 6.15 denotes a control activity with a trigger prompt. Its semantics is specified by the PN diagram of Figure 6.16.

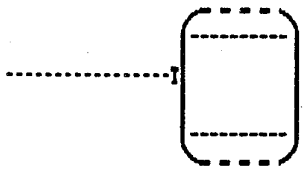


Figure 6.15: control activity with a trigger input prompt

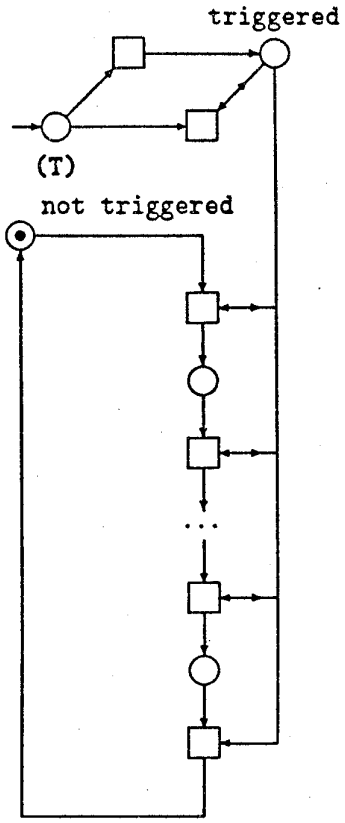


Figure 6.16: PND corresponding Figure 6.15

The PN model shows that a triggered control activity has three special states: place `not triggered` models the "home state" of the activity, place `(T)` the arrival of a trigger prompt and place `triggered` the state after having responded to the trigger. At any instance, a trigger prompt can arrive (token appears in `(T)`). A trigger prompt is discarded if the activity is already triggered and has not yet reached its natural termination (token in `not triggered`). Each step within the activity requires the token in `triggered`. Only the last step causes the token in `triggered` to disappear.

A more complex structure arises if we consider a control activity with a trigger, a suspend and a resume prompt as specified by the ESML diagram of Figure 6.17. Its semantic is specified by the PN diagram of Figure 6.18.

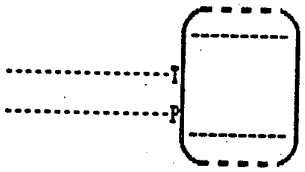


Figure 6.17: control activity with a trigger and a pause input prompt

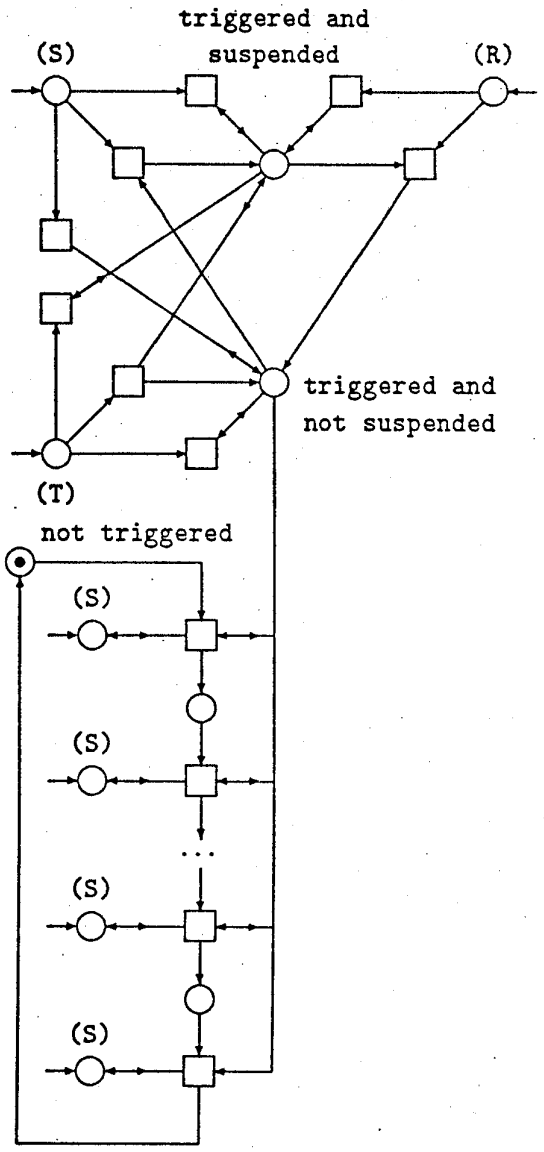


Figure 6.18: PND corresponding Figure 6.17

The PN model results from adding the already known suspend/resume module to the previous model of a triggered activity. As there is some interference between responses to a trigger prompt and responses to suspend/resume prompts, two additional links and three link extensions (additional branches) are necessary to reflect exactly and completely the possible interferences. For example, a suspend prompt reaching an activity which is not triggered will simply disappear (see link with the two places `(S)` and `triggered and not suspended`). The same happens if a trigger prompt reaches the activity in the state `triggered and suspended`. If a triggered activity is suspended, a resume prompt allows it to continue exactly where it was suspended. In order to visualize the dynamics captured by the model the reader is invited to "animate" the model by playing the token game with real pieces.

Enabled/disabled control activities

A control activity with an enable and a disable prompt is specified by the ESML diagram of Figure 6.19. Its semantics is defined by the PN diagram of Figure 6.20.

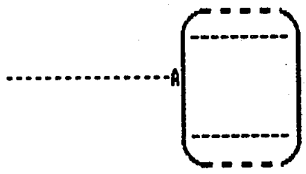


Figure 6.19: control activity with an activate (E/D) input prompt

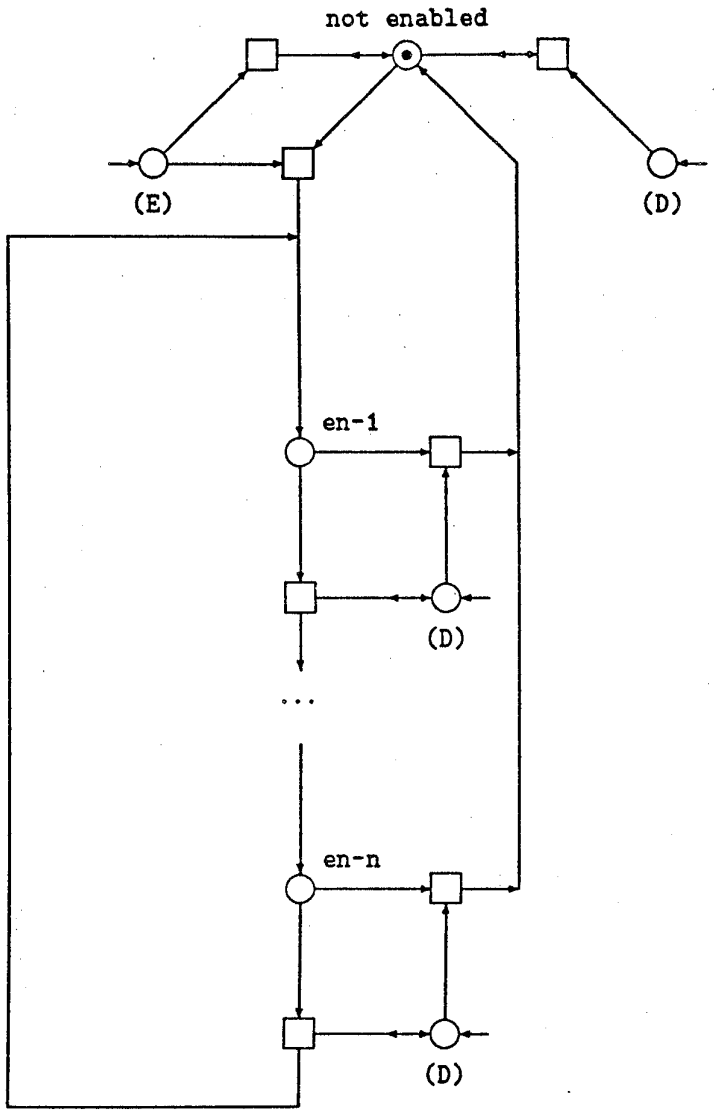


Figure 6.20: PND corresponding Figure 6.19

The PN model shows that there are three states of special interest: place `not enabled` models the initial state and the state resulting from responding to a disable prompt, place `(E)` models arrival of an enable prompt, and place `(D)` of a disable prompt. For the sake of readability, place `(D)` is represented several times. At any instant, an enable prompt and/or a disable prompt can arrive (token appears in `(E)` and/or `(D)`). A token in one of the places named `en-`... ("enabled") means that the activity is enabled. For the sake of simplicity the current states of an enabled activity have been concentrated in single places `en-1`, `en-2`, ... `en-n` without loss of generality.

When playing the token game recall that arrival of a disable prompt is reflected in the model in that a token appears in all place symbols with name `(D)`. However, only one of the

changes defined by the links which connect a place `en-...`, the place `(D)` and the place `not enabled` will be enabled and will remove the token from "its" place `en-...` and `(D)`. The next enable prompt will then enable the change which causes a token to appear in place `en-1`.

The most complex structure arises when the two "binary" prompts are combined. The ESML diagram of Figure 6.21 specifies a control activity with an enable, a disable, a suspend and a resume prompt. Its semantics is defined by the PN diagram of Figure 6.22.

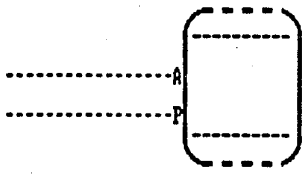


Figure 6.21: control activity with an activate and a pause input prompt

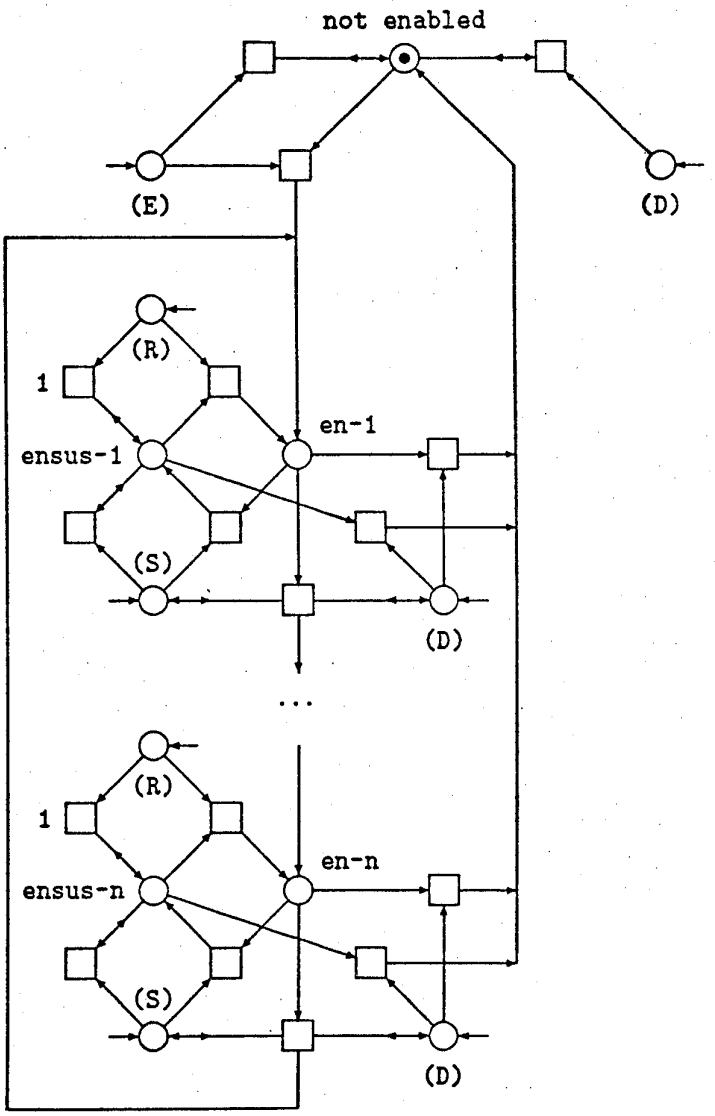


Figure 6.22: PND corresponding Figure 6.21

The PN model results from extending the previous one. This time, however, embedding the suspend/resume module requires an individual complementary place to each place with `en-...` in order to allow for a later continuation of execution due to a resume prompt. These complementary places are named with `ensus-...` ("enabled and suspended"). For the sake of readability, places `(S)` and `(R)` are represented several times. This is also true for link `1` which links `(R)` and *all* places `ensus-...` in order to discard a resume prompt arriving when the activity is not suspended. At any instance, any combination of the four prompts

can arrive (token appears in one or more places of the set (E), (D), (S) and (R)). If there is a conflict, e.g. a suspend and a disable prompt arrive simultaneously and the activity is enabled, the model does not prescribe a decision concerning its resolution. Whether the next state of the activity is "enabled and suspended" or "not enabled" is not determined by the model. The reader is encouraged to "run" the model by manual simulation with physical tokens in order to explore the various effects resulting from concurrent prompts. Four dangling arcs indicate that a token can arrive at any moment unless there is still a token in the place.

7 Operational activities

This section illustrates the use of PN diagrams for modeling some basic structures of operational activities by means of examples. They are all drawn from Volume 1 of [12]. We do not include examples with suspend/resume prompts since they would not add new aspects compared with the foregoing section.

Standing operational activities

Figure 9.1 of [12] shows a model of a standing operational activity with intermittently available value-bearing input and output flows *Aircraft Number* and *Aircraft Status* and a non-depletable input store *Airspace Status*. It can be translated into the PN diagram of Figure 7.1.

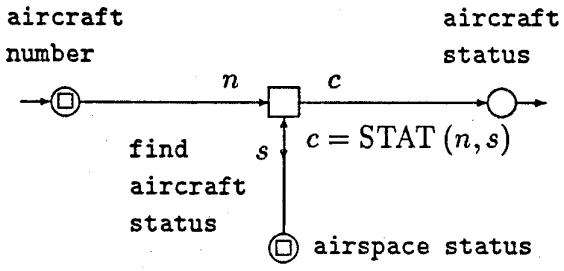


Figure 7.1: PND for a standing operational activity

In this PN diagram, `aircraft number`, `airspace status` and `aircraft status` are compact places. The operator STAT is assumed to return the status of an aircraft with number n from the airspace status s. The annotated link `find aircraft status` specifies all legal combinations of n, s and c, i.e. all combinations which yield *True* (recall that the values of n, s and c are drawn from the domains of the respective places). Each legal combination represents a change defined by the link. In this and the following examples, all compact places are supposed to have their domains suitably defined. Thus no formal annotations (terms) are given with the place symbols.

The arrival of an aircraft number enables the activity to produce an aircraft status. However, due to our restriction with respect to the number of values in a place, an aircraft number or an aircraft status can only appear in a place if the place is empty.

Triggered operational activities

Figure 9.5-A of [12] shows a model of a triggered operational activity with a continuously available value-bearing input flow *Temperature* and a non-depletable store *Temperature History*. It can be translated into the PN diagram of Figure 7.2.

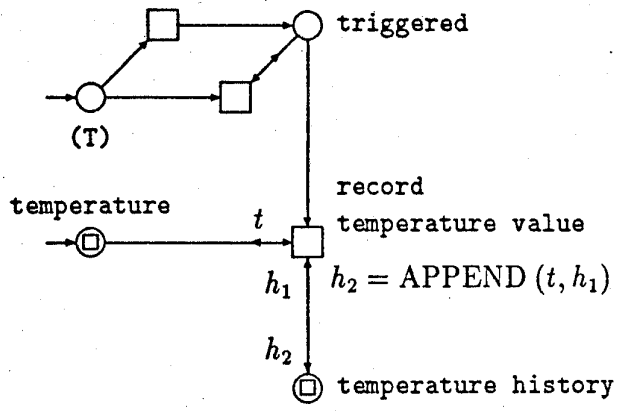


Figure 7.2: PND for a triggered operational activity

In this PN diagram, places (T) and triggered are elementary places as in Figures 6.16 and 6.18, while temperature and temperature history are compact places. The expansion of the double-headed arrow with two annotations is given in Section 5, Figure 5.3. The current temperature value is continuously available in temperature (except, of course, at the moment of its change). Output (a new temperature history) can be produced not until after the activity was triggered. The new temperature history h_2 is defined to be the result of applying the operator APPEND on t and the current temperature history h_1 ("appends" t to h_1 in some way). Recall that the arrival of a trigger prompt is reflected by a token in place (T). This enables the activity to enter the state triggered and subsequently to record the (current) temperature value in the temperature history. When this occurs, the token in triggered turns absent.

Figure 9.5-B of [12] shows a model of a triggered operational activity with a non-depletable input store *Airspace Status* and an intermittently available value-bearing output flow *Airspace Status Report*. It can be translated into the PN diagram of Figure 7.3.

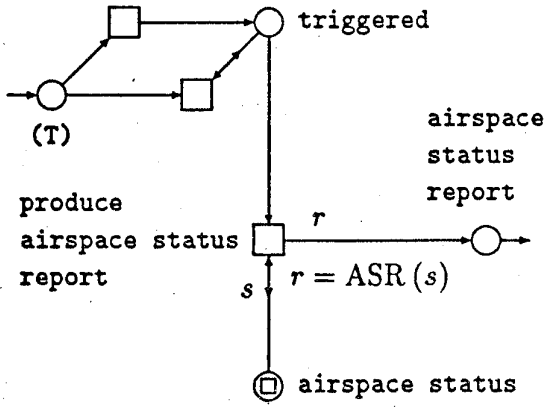


Figure 7.3: PND for a triggered operational activity

This diagram, with compact places `airspace status` and `airspace status report`, shows an activity modeled by the link `produce airspace status report`. Once triggered, it can produce a report r which is defined to be the result of applying the operator ASR ("airspace status report generator") on the airspace status s.

Enabled/disabled operational activities

The following three examples show typical configurations of connections with enabled/disabled operational activities without suspend/resume.

Figures 9.2, 9.3 and 9.5-C of [12] show a model of an enabled/disabled operational activity with intermittently available value-bearing input and output flows *Aircraft Number* and *Aircraft Status* and a non-depletable store connection *Airspace Status*. It can be translated into the PN diagram of Figure 7.4.

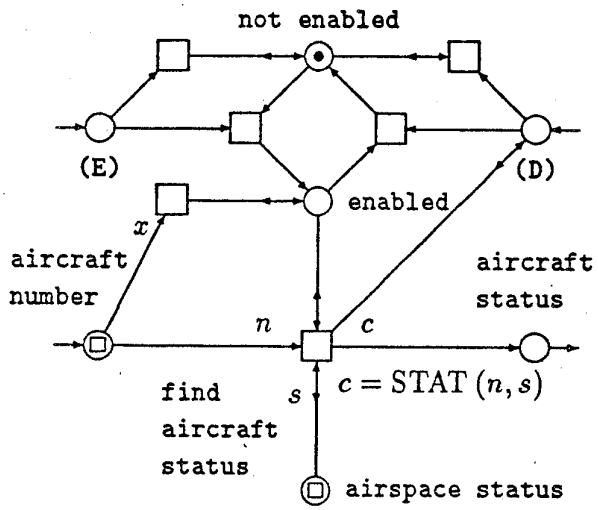


Figure 7.4: PND for an enabled/disabled operational activity

This PN diagram shows three compact places `aircraft number`, `airspace status` and `aircraft status`. The places `(E)`, `not enabled`, `(D)` and `enabled` are elementary places as in Figures 6.20 and 6.22. The dynamics expressed by Figure 7.4 includes the following processes: A token appears in `(E)` and causes the activity to become enabled (token in place `enabled`) or, as expressed in [12], "the processing of the prompt causes the token on the Enable flow to be removed, and a token to be placed on the transformation itself to indicate that it is currently enabled"). If the aircraft number (value in `aircraft number`) had arrived before the token was in `enabled`, it would have been discarded. As long as the activity is enabled, placing a token on `(D)` results in the removal of both tokens (viz. from `(D)` and `enabled`). The resulting marking of the activity model (token in `not enabled` only) shows it is currently disabled. Notice that for an occurrence of `find aircraft status` it is not only required that `enabled` is marked but also that a disable prompt has not arrived (see restoring exit branch to `(D)`). The aircraft status is defined to be the result of applying STAT on the airspace status s and the aircraft number n.

Figure 9.4 of [12] shows a model of an enabled/disabled operational activity with a continuously available value-bearing input flow *Temperature* and output flow *Heater Level*, which can be translated into the PN diagram of Figure 7.5.

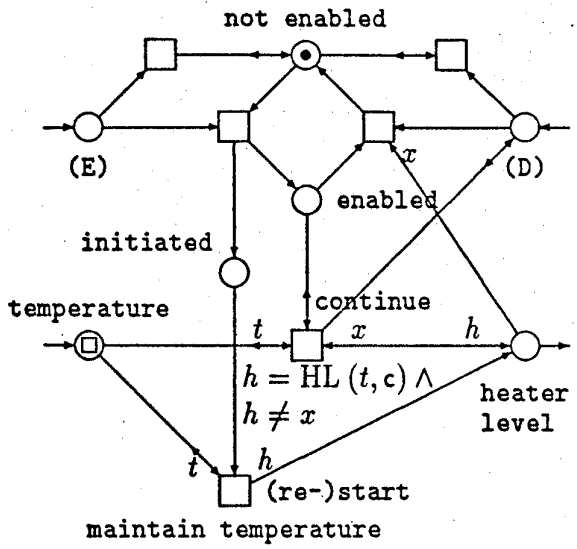


Figure 7.5: PND for an enabled/disabled operational activity

Since the links `continue` and `maintain temperature` have the same annotation, the formula

$$h = \mathrm{HL}\,(t, \mathrm{c}) \wedge\ h \neq x$$

is not repeated in the diagram and applies to both links (but nevertheless should be evaluated independently for each link).

This PN has the two compact places `temperature` and `heater level` as well as five elementary places `(E)`, `not enabled`, `(D)`, `enabled` and `initiated`. A continuously available value-bearing output flow is produced while a token is present in place `enabled` and a disable prompt has not arrived (see restoring exit branch to `(D)`). In Figure 7.5, the special problem is, that "output is prohibited as long as the transformation is disabled" [12]. This requires a construct for removing the value from the place `heater level` together with the token from `enabled` and a construct which models that "when the transformation is enabled ... it may begin producing output values". The latter is specified by the link `(re-)start`. The new value of the output flow `heater level` h is defined to be the result of applying HL on the (new) current temperature t and some constant c. The previous value of the output flow x is used for specifying that a new value must be different from the old one.

Figure 9.5-D of [12] shows a model of an enabled/disabled operational activity with a continuously available value-bearing input flow *Temperature* as the only value-bearing flow connection and a non-value bearing flow *Overrun*. It can be translated into the PN diagram of Figure 7.6.

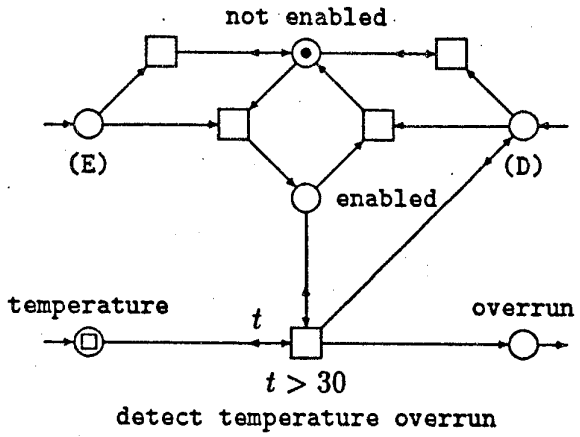


Figure 7.6: PND for an enabled/disabled operational activity

Figure 7.6 shows one compact place `temperature` and the five elementary places `(E)`, `not enabled`, `(D)`, `enabled` and `overrun`. Place `overrun` is also said to model a "signal". The model is quite simplistic as has been stated in Section 5.1. The present model allows to produce a signal whenever the token has been removed from `overrun` and the temperature continues to be more than 30. With a further link and additional places one could specify that a (new) overrun can only be signalled after the temperature had dropped below the level of 30 and exceeded it anew.

8 Conclusions

The complexity of real-time systems and the degree of comprehensibility required for constructing dependable implementations call for rigorous modeling tools when informal and semi-formal models can no longer provide the desired additional insight and confidence in a system under development. Formal languages can provide the necessary rigour. Can they also support intuition? High-level system models are or should be developed, discussed and analysed by professionals of the application area rather than of computer science. A graphics-based modeling language with a rigorous interpretation could provide a formalism which stimulates comprehension of a model rather than to hinder it.

The results reported in this paper are expected to supply evidence that a modeling language like ESML should be complemented with a formalism for static and dynamic system modeling which at the same time supports intuition. Petri nets, tailored to the applications addressed with ESML, have proved useful as a formal and graphical language for this purpose. Their use is demonstrated through examples taken from ESML and TS literature.

Not surprisingly, the graphical symbols for functional units in ESML models and PN models contribute considerably to visualization of system structure and system behavior. In particular, when concurrency of system events cannot or should not be excluded, a controllable simulation of the system model becomes of utmost importance, since it is usually this way that a designer can explore the interaction of partially independent subsystems which constitute the system at hand – provided that the model doesn't exceed a manageable size. Applications showed that for this purpose the essential behavior of the object system, i.e.,

the system to be monitored and controlled, has to be modeled together with the real-time control system.

Increasing size of the models requires modularization. Hierarchically structured models consisting of hundreds of "blueprints" have to be edited and updated consistently, while they must also be amenable to execution through all levels of the hierarchy. Software packages for ESML models and for PN models are commercially available which meet these requirements. Obviously, executability of models has become one of the principal motivations behind future attempts to progress formal system modeling for engineering applications.

Further research will have to include a variety of methodological efforts among which the following seem to be the most relevant with respect to engineering applications. Petri nets offer a number of high-level modeling constructs which have not been shown in this exercise, e.g., places with several items, several copies of the same item in a place, state constraints, transition constraints, synchronic distance, etc.. An in-depth analysis of typical real-time system modeling needs could reveal the potential benefits of those constructs to modeling system structures in engineering environments. In a next step, a collection of basic PN constructs for real-time control and PN macros for frequently occurring ESML patterns could be developed along with drawing and lettering conventions similar to other graphics-based modeling languages. For the practitioners who have to solve problems rather than to develop tools, a collection of "recipies" would be instrumental for translating from ESML into PN, for improving the expressive power of "raw translations", for analysing the obtained PN models or for drawing conclusions in order to feed them back into the ESML models. All these endeavors should, in the long-term, contribute to an integrated methodology which covers the stages from informal to formal system modeling and analysis. In addition, such a collection could provide heuristics for assessing the relationship between a system model and the real world counterpart.

References

[1] W. Bruyn, R. Jensen, D. Keskar and P. T. Ward: *ESML: An Extended Systems Modeling Language Based on the Data Flow Diagram.* ACM SIGSOFT Software Engineering Notes vol. 13, no. 1, pp. 58–67 (January 1988)

[2] H. J. Genrich and K. Lautenbach: *System Modelling with High-Level Petri Nets.* Theoretical Computer Science 13 (1981), pp. 109–136

[3] D. J. Hatley and I. A. Pirbhai: *Strategies for Real-Time Specifications.* Dorset House Pub., New York 1987

[4] K. M. van Hee, G.-J. Houben and J. L. G. Dietz: *Modeling of Discrete Dynamic Systems — Framework and Examples.* Information Systems, vol.14, no. 4, pp. 277–289 (November 1989)

[5] E. L. Miranda: *Specifying Control Transformations through Petri Nets.* ACM SIGSOFT Software Engineering Notes vol. 14, no. 2, pp. 45–48 (April 1989)

[6] W. Reisig: *Petri Nets. An Introduction.* Springer 1985, 161 pages

[7] G. Richter: *Clocks and their use for time modeling.* In A. Sernadas, J. Bubenko, Jr. and A. Olivé (eds.), Information Systems: Theoretical and Formal Aspects. North-Holland 1985. pp. 49–66

[8] G. Richter: *A note on side-conditions and inhibitor arcs.* Newsletter 21 of GI-SIG Petri Nets and Related System Models, June 1985. pp. 29–37

[9] P. T. Ward: *The Transformation Schema: An Extension of the Data Flow Diagram to Represent Control and Timing.* IEEE Transactions on Software Engineering, Vol. SE-12, No. 2, pp.198–210 (February 1986)

[10] P. T. Ward: *Embedded Behavior Pattern Languages: A Contribution to a Taxonomy of Case Languages.* The Journal of Systems and Software 9, pp. 109–128 (1989)

[11] P. T. Ward and D. Keskar: *A Comparison of the Ward/Mellor and Boeing/Hatley Real-Time Methods.* Proceedings, Twelfth Structured Methods Conference, Chicago 1987, pp. 356–366

[12] P. T. Ward and S. J. Mellor: *Structured Development for Real-Time Systems.* 3 volumes. Yourdon Press - Prentice-Hall, Englewood Cliffs 1985