Control and Cybernetics

vol. **24** (1995) No. 2

PN-tools: environment for the design and analysis of Petri nets

by

Zbigniew Suraj

Institute of Mathematics
Pedagogical University
Rejtana 16a, 35-310 Rzeszów
Poland

Petri nets are now widely used in both theoretical analysis and practical modeling of concurrent systems. The practical use of Petri nets is strongly dependent upon the existence of adequate computer tools – helping the user to handle all the details of a large and complex description. For Petri nets one needs editors as well as analysis programs. Graphical work stations provide an opportunity to work directly with the graphical representations of Petri nets.

This paper describes the integrated graphical Petri net tools (in the following called PN-tools) for construction of nets (also hierarchical nets), as well as modification and analysis. PN-tools allows us to work with different classes of Petri nets. Several analysis tools are available for each of these classes. PN-tools is running on IBM PC microcomputers under MS-DOS operating system.

Keywords: Petri net tools, computer based tools

1. Informal introduction to Petri nets

Petri nets, Petri (1966), are a promising tool for describing and studying information processing systems that are characterized as being concurrent, asynchronous, nondeterministic, distributed, parallel, and/or stochastic.

Petri nets have been proposed for a very wide variety of applications, Brauer, Reisig, Rozenberg (1987a, 1987b), Jensen, Rozenberg (1991), Murata (1989), Peterson (1981), Proc. Int. Workshop Timed Petri Nets (1985, 1987), Reisig (1985), Voss, Genrich, Rozenberg (1987).

The use of computer aided tools is a necessity for practical applications of Petri nets. Most Petri net research groups have their own software packages and tools to assist the drawing, analysis, and/or simulation of various applications. The papers by Feldbrugge (1986, 1990), Feldbrugge, Jensen (1987, 1991) provide a good overview of typical Petri net tools. Some of these tools and their

applications are discussed in details in Billington, Wheeler, Wilbur-Ham (1988), Chiola (1985), Jensen et al. (1990), and Starke (1985).

The rest of this paper consists of the following topics. First Petri nets are introduced by means of a small example and an informal definition of their structure, behaviour and extensions is presented. Then we describe the hierarchical Petri nets. Next we discuss how to analyse Petri nets, how to support them by various computer tools, and we also describe shortly an implementation environment. Finally, some conclusions concerning, in particular, future plans for PN-tools are presented.

1.1. A simple example of Petri net

The Petri net, Reisig (1985), presented in Figure 1 describes a system of reader and writer processes of an operating system. There is a set of places (drawn as circles) and a set of transitions (drawn as rectangles). The places and their tokens represent states, while the transitions represent state changes. However, each place may contain several tokens.

Now let us take a closer view of the Petri net in this figure. It consists of two different parts: the **net structure** and the **net inscriptions**.

The net structure is a directed graph with two kinds of nodes, **places** and **transitions**, interconnected by **arcs** – in such a way that each arc connects two different kinds of nodes (i.e. a place and a transition). Such a graph is called a bipartite directed graph.

Each net inscription is attached to a place, transition or arc. In Figure 1 places have two different kinds of inscriptions: **names** and **initial markings** (which define initial states of Petri nets), while transitions and arcs only have one kind of inscription: **names** and **arc weights**, respectively. All net inscriptions are positioned next to the corresponding net element, markings are represented by dots (or cardinals), and can be thought to reside in the places of a Petri net, while arc weights are represented as cardinals. The arc weight '1' is omitted.

Names of nodes have no formal meaning. They can be omitted and one can use the same name for several nodes (although this may create confusion). Names are used in the feedback information from PN-tools to the user, e.g. in the textual representations of a net. To make the feedback unambiguous, it is recommended to keep names unique, but this is not enforced. In this paper we use the capital letters for names of nodes. As explained above each place must have an initial marking and this determines tokens which may reside on that place. By convention we omit initial markings which are equal zero.

1.2. Dynamic behaviour of Petri nets

One of the most important properties of Petri nets is that they have a well-defined semantics which defines the behaviour of the system. The ideas behind

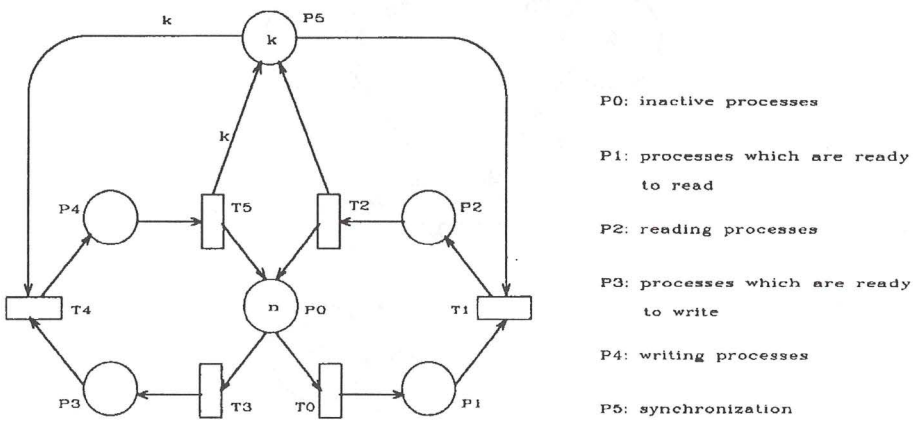

Figure 1. A system of reader and writer processes of an operating system

the semantics, we shall demonstrate by means of Figure 2 – which contains one of the transitions from Figure 1.

At first, we check whether the transition is **enabled** in current marking. This is done by evaluating all the input arc weights: In the present case the two arc weights are equal to 1 and k, respectively. Thus we conclude that the transition is enabled – because each the input places contains a sufficient number of tokens which the corresponding arc weight evaluates (one token on P3 and k tokens on P5). When a transition is enabled it may **occur** and it then removes tokens from its input places and adds tokens to its output places. The number of removed/added tokens are determined by the value of the corresponding arc weights. An occurrence of the transition removes one token from the place P3, removes k tokens from P5 and adds one token to P4.

A distribution of tokens (on places) is called a **marking**. The **initial marking** is the marking determined by the initial state of a system. Now we can ask whether a transition T4 is enabled in a given marking M – and when this is the case we can speak about the marking M' which is **reached** by the occurrence of T4 in M. It should be noticed that several transitions may be enabled in the same marking. In that case there are two different possibilities: Either there are enough tokens (so that each transition can get its own share) or there are too few tokens (so that several transitions have to compete for the same input tokens). In the first case the transitions are said to be **concurrently enabled**. They can occur in the same **step** and they each remove their own input tokens and produce their own output tokens. In the second case the transitions are said to be in **conflict** with each other and they cannot occur in the same step.

In the initial marking of Figure 1 we observe that the transition, for example, T0 is concurrently enabled with T3. This means that we can have a step where

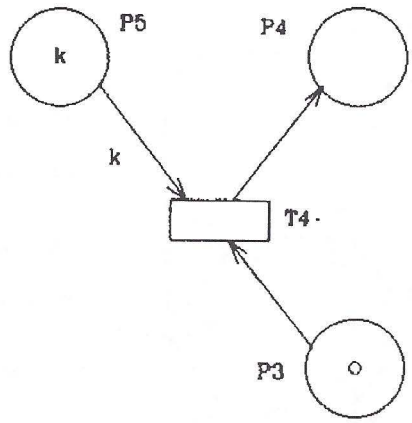

Figure 2. A transition from the system of reader and writer processes

both T0 and T3 occur. When it occurs a token is moved from P0 to P1 and a token from P0 to P3. It should be noticed that the effect of this step is the same as when the two transitions occur after each other in an arbitrary order.

For more information about Petri nets see Peterson (1981), Reisig (1985), or Starke (1990).

1.3. Extensions of Petri nets

In this section, we give the examples of extensions to the Petri net model which have been implemented in our PN-tools. The simplest extension to Petri nets are **inhibitor arcs** (IA-nets), Hack (1975). An inhibitor arc leads from a place P to a transition T and inhibits the firing of T if the token load of P is not less than its multiplicity w. If $w > 1$, then, additionally, an ordinary arc from P to T with multiplicity less than w is allowed.

Figure 3 shows the net with inhibitor arc. Note that the inhibitor arc (P1,T1) is identified by a small circle at one end. Thus in the Petri net of Figure 3, transition T1 can fire only if there is a token in P2 and P3 and zero tokens in P1.

Petri nets with inhibitor arcs are intuitively the most direct approach to increasing the modeling power of Petri nets. All other extensions to Petri nets which are defined in this section are in fact equivalent to Petri nets with inhibitor arcs.

Another extension of Petri nets are the so-called **self-modifying**, nets (SM-nets) introduced by Valk (1978). The only difference to PT-nets is that the multiplicity of an arc f can be the name P of a place. In this case, the multiplicity of the arc f changes with the marking of the place P. At the given marking M

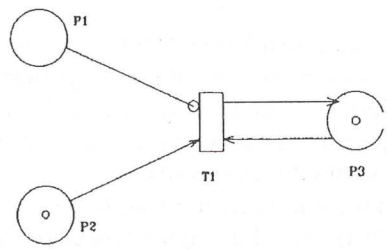

Figure 3. An example of a net with inhibitor arc

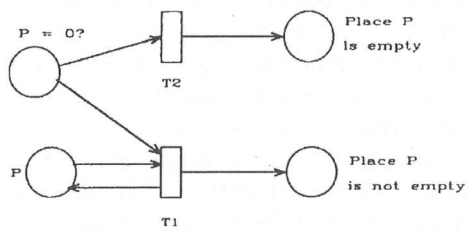

Figure 4. Using priorities to test if the marking of a place P is zero or nonzero. Transition T1 has priority over transition T2

the multiplicity of f equals M(P), hence it is modified by the firing of transitions.

Petri nets with **priorities** (P-nets) have been suggested by Hack (1975). Priorities can be associated with the transition such that if T and T' are both enabled, then the transition with the highest priority will fire first.

In the case of priorities, we can easily test if a place P is zero. This is shown in Figure 4. If we put a token into place P=0? and define the priority of transition T1 to be higher than the priority of transition T2, then we will get a token in one of the two places at the right depending on the marking of place P. This results from the fact that transition T1 can fire only if it is enabled, and it is enabled only if place P has a token. If T1 cannot fire because P is empty, then, and only then, will transition T2 fire.

1.4. Hierarchical Petri nets

In this small section we give some recommendations for hierarchical nets – the use of which makes it possible to relate a number of individual nets to each other in a formal way (i.e. in a way which has a well-defined semantics and thus allows formal analysis).

The basic idea behind hierarchical Petri nets is to allow the modeller to construct a large model by combining a number of small nets into a larger net.

It is also possible to translate a hierarchical Petri nets into a non-hierarchical Petri net. This means that the theoretical modeling powers of these two classes of nets are the same. However, from a practical point of view, the net classes have very different properties. To cope with large systems we need to develop strong structuring and abstraction concepts.

Section 3.3.2 describes some Petri net tool to support the construction and analysis of hierarchical Petri nets built upon work by Valette (1978).

2. Analysis of Petri nets

The methods for analysing Petri nets can be roughly divided into several categories: analysis by means of simulation, study of reachability set, transformation by homomorphism, and invariants. The most straightforward kind of analysis is simulation – which is very useful for the understanding and debugging of a system, in particular in the design phase and the early validation phases. There are, however, also more formal kinds of analysis – by which it is possible to prove that a given system has a set of desired properties (e.g. absence of deadlock, the possibility to return to the initial state, and an upper bound on the number of tokens). This chapter contains a brief introduction to the main ideas behind the most important analysis methods and it contains references to papers in which the technical details of these methods can be found.

2.1. Analysis by means of simulation

Simulation can be supported by a computer too or it can be totally manual (e.g. performed on a blackboard or in the head of the modeller). Simulation can reveal errors, but in practice never be sufficient to prove the correctness of a system. Simulation is often used in the design phases and the early investigation of a system design (while the more formal analysis methods are used for the final validation of the design). In Section 3.4 we give a detailed description of an existing graphical Petri net simulator.

2.2. Analysis by means of reachability graphs

The basic idea behind reachability graphs is to construct a graph which contains a node for each reachable state and an arc for each possible change of state. Obviously such a graph may, even for small Petri nets, become very large – and sometimes infinite. Thus it is necessary to construct and analyse the graph using automated methods – it is desirable to develop techniques which make it possible to work with reduced reachability graphs without losing too much information. One of the possibilities is reduction by means of covering markings. This method looks for transition sequences leading from a system state to a larger system state (one with additional tokens) and the method guarantees that the reduced reachability graph always becomes finite. The method has,

however, some drawbacks. First of all it only gives a reduction for unbounded systems (and most practical systems are bounded). Secondly, so much information is lost by the reduction that several important properties (e.g. liveness and reachability) are no longer decidable. For more information see Finkel (1990) and Karp, Miller (1969).

A reachability graph can be used to prove properties about the modelled system. For bounded systems a large number of questions can be answered. Deadlocks, mutual exclusion, reachability and marking bounds can be decided by a simple search through the nodes of the reachability graph, while liveness and home markings can be decided by constructing and inspecting the strongly connected components.

As described above, the reachability graph method can be totally automated – and this means that the modeller can use the method, and interpret the results, without having much knowledge about the underlying mathematics. In Section 3.5.2 we describe some Petri net tool to support the calculation and analysis of reachability graphs.

2.3. Analysis by means of invariants

Invariant analysis allows logical properties of Petri nets to be investigated in a formal way. There are two dual classes of invariants. A place invariant (P-invariant) characterises the conservation of a weighted set of tokens, while a transition invariant (T-invariant) characterises a set of transition sequences having no effect, i.e. with identical start and end markings.

The main advantages of invariant analysis are the low computational complexity (in particular, compared to the method of reachability graphs described in Section 2.2) and easy parametrization with respect to system parameters (through the definition of different initial markings).

The main drawback is the difficulty to automate the interpretation of invariants and the incompleteness (in the sense that it is usually only possible to calculate either necessary or sufficient conditions, for a given property).

It is desirable to be able to make an automatic computation of invariants. This can be done by solving a matrix equation. The matrix equation can be solved by standard Gauss elimination. It is sufficient to find a basis from which all invariants can be constructed (as linear combinations). By means of invariants it is possible to investigate many different kinds of system properties, e.g. deadlocks, mutual exclusion and marking bounds. For more details about the calculation of invariants, see Memmi, Vautherin (1987). Above, we have discussed how to calculate invariants by solving a matrix equation. The problem is, however, often of a different nature – because we already have a set of vectors and just want to verify that these are invariants. This task is much easier and it can be done totally automatically.

As described above, transition invariants are the duals of place invariants. Transition invariants can be calculated in a similar way as place invariants.

Transition invariants are found by solving a matrix equation (obtained by transposing the matrix used to find place invariants). Each transition invariant is a solution to the matrix equation. The opposite is, however, not always true. Transition invariants are used for similar purposes as place invariants (i.e. to investigate the behavioural properties of Petri nets).

In Section 3.7.2 we describe some Petri net tools to support the computation and the analysis of invariants.

2.4. Analysis by means of reductions

Petri nets can also be analysed by means of reductions. The basic idea behind this method is to modify a Petri net – without changing a selected set of properties, e.g. liveness, marking bounds, deadlocks and reachability. The modification of the net is performed by means of a set of transformations rules and may be carried out manually, automatically or interactively. In the latter case the strategy is decided by a person, while the detailed computations and checks are made by a computer.

The purpose of the transformation is to obtain a small and simple net for which it is easy to investigate the given properties. A serious problem with reduction methods is that they often are non-constructive (because the absence of a property in the reduced net, usually, does not tell much about why the original net does not have the property. An exception is the reduction method to calculate place or transition invariants, mentioned in Section 2.3. In this case it is, from the reduced net, possible to determine a set of the invariants for the original net – and this means that the analysis results can be interpreted in terms of the original net. In Section 3.6 a Petri net tool to support the execution of net reduction based on transformation rules developed by Berthelot (1987) is described.

2.5. Other methods of analysis

For ordinary Petri nets several kinds of analysis methods are known. One of the methods uses structural properties, it means properties which can be formulated without considering the behaviour (i.e. transition sequences) of Petri net to deduce behavioural properties. For more information see Best (1987).

In Section 3.5.1 we describe a Petri net tool to support the analysis of structural properties, which is implemented on the basis of the method as mentioned above.

3. Computer tools for the users of Petri nets

3.1. Why do we need computer tools for Petri nets?

The practical use of Petri nets is strongly dependent upon the existence of adequate computer tools – helping the user to handle all the details of a large and

complex description. For Petri nets one needs editors (supporting construction and modification of Petri nets) as well as analysis programs (supporting different analysis methods), Jensen (1987). Graphical work stations provide an opportunity to work directly with graphical representations of Petri nets (and reachability graphs). The most important advantages of using computerized Petri net tools are, Jensen (1987):

- the possibility to obtain better results,
- the possibility to create faster results,
- the possibility to make interactive presentations of the analysis results,
- the possibility of 'hiding' technical aspects of the Petri net theory inside the tools,
- the possibility of producing fast results of good quality – without requiring too deep knowledge of the Petri net theory.

Furthermore it is important to be able to use Petri nets together with other specification/implementation languages.

3.2. Which tools do we have in PN-tools?

This section contains a general characteristic of PN-tools. As mentioned above, our integrated computer system PN-tools may be used to help the system designer in proving the correctness of his design. PN-tools consists of four logical parts:

1. EDITOR/SIMULATOR is a window-based graphical and textual editor/simulator for entering, exiting, constructing, simulating, and editing Petri nets.
2. ANALYSER is a set of programs by means of which basic structural and dynamic properties (see Reisig (1985), Starke (1990)) of nets can be checked. The results of such analysis allow to detect syntactic (sometimes even semantic) design errors.
   For certain subclasses of PT-nets the structural properties can be used to deduce dynamic properties.
3. REDUCER is a program to reduce the size of a net (and of its reachability graph) preserving liveness and boundedness. The use of such program is necessary if the storage capacity of a given computer system is not sufficient to analyse a given net. Reduced nets are thus made accessible for treatment by ANALYSER.
4. VERIFIER is a subset of programs which calculate the invariants and other structural information (e.g. state machine components) from a given net which reflect certain structural properties of the modelled system. Invariant analysis can be done by computing generator sets of all P-/T-invariants and of all nonnegative invariants. Vectors can be tested for invariant properties.

3.3. Editors

To be able to work with practical applications of Petri nets – in an effective way – the user needs all the tools mentioned above. In particular, for Petri nets there is a need for tools supporting construction of nets, as well as modification. Graphical work stations provide the opportunity to work – not only with textual representations of Petri nets – but also directly with the graphical representations.

This section describes the textual Petri net editor (TPN-editor) and the graphical Petri net editor (GPN-editor) belonging to our PN-tools.

Generally, these editors are a computer aid developed for creation, manipulation and simulation of Petri nets. They are based on place/transition nets, self-modifying nets, priority nets, and nets with inhibitor arcs.

3.3.1. Textual Petri net editor

The user needs textual editors to be able to construct and modify Petri nets, working directly with their mathematical representation. This representation can be expressed by flow relations, incidence matrices, functions, etc.

The TPN-editor is a program belonging to the part EDITOR/ SIMULATOR. This program realizes Petri nets the size of which is only restricted by the main storage capacity of the given computer. It allows the user to construct, combine, refine, modify and syntactically check Petri nets and their extensions considered in this paper. The nets can be read-in from a file or the terminal. The generated nets can be stored in files.

The TPN-editor enables us, in particular, to execute the following operations:

- reading a net from a file;
- deleting the places or transitions;
- deleting the arcs;
- changing a net number;
- changing multiplicities;
- output of a net to the terminal;
- writing a net to a file;
- changing a marking;
- changing a place number or a transition number;
- raising all the place numbers (transition numbers respectively) by a span (to make two place/transition sets disjoint);
- gluing together two nets at their common transitions;
- merging two nets by merging common nodes and arcs;
- refining nodes by nets.

Moreover, the TPN-editor has many of the facilities provided by normal word processing systems.

3.3.2. Graphical Petri net editor

General information A nice aspect of Petri nets is the fact that they can be graphically represented and that one can visualize the states reached during an execution. From users' experience a graphical editor is an absolute necessity if people are to effectively use Petri nets. Increasingly, graphic description techniques play a significant role in the fields of analysis, system specification, and documentation. However, creation of graphics is quite expensive without relevant computer support and proves very inflexible in the case of modifications. Moreover it seems natural to use a dialogue, which is close to the operations performed, if the net was drawn by hand on a piece of paper.

A Petri net constructed by means of the GPN-editor is called a **PN-graph** and it consists of several types of graphical objects. In this paper, the word *graph* denotes the mathematical concept of a graph (i.e. a structure which consists of a set of nodes interconnected by a set of edges). Each object is either a **node**, a **connector** (between two nodes) or a **description** (i.e. a subordinate of another object). Places and transitions are nodes, arcs are connectors, while all the net inscriptions are descriptions. In addition to the PN-**objects** (e.g. places, transitions, arcs and net inscriptions), which are formal parts of the model there may also be **auxiliary objects** which have no formal meaning but play the role of comments.

It is possible for the user to determine, in great detail, how he wants the PN-graph to look. One of the most attracting features of Petri nets is the very appealing graphical representation. In the GPN-editor each object has its own set of **attributes** which determine e.g. the position, shape, size and line colour. When a new object is constructed the attributes are determined by a set of **defaults** (each object type has its own set of defaults). At any time the user can change one or more attributes for each individual object. All options in the GPN-editor have defaults and these can be changed by the user.

The GPN-editor supports hierarchical Petri nets (for the moment it supports substitution transitions and places in the sense of Valette (1978)) and this means that each PN-graph can contain a number of pages. Each page is displayed in its own screen. The page objects can be moved and modified in exactly the same way as other types objects, and this means that the user can determine how the page hierarchy looks.

The hierarchies in a PN-graph can be constructed in many different ways – ranging from a pure top-down approach to a pure bottom-up: Part of a page can by a single editor operation be moved to a new subpage: The user selects the nodes to be moved and invokes the operation, then the editor checks the legality of the selection, creates the new page, moves the subnet, and creates a hierarchy inscription for it.

There is also an editor operation to turn an existing node into a supernode (by relating it to a new page). The user invokes the operation, then the editor makes the hierarchy page active and enters a mode in which the user, by means

of the mouse or the keyboard, can construct the desired subpage. To destroy the hierarchical relationship between a supernode and a subpage the user simply deletes the corresponding page.

The user works with a high-resolution raster graphical screen and a mouse. For the moment PN-tools has been implemented on IBM PC machines – and they can easily be moved to other machines running DOS (e.g. Apollo 700 work stations). It is recommended, but not necessary, to have a large colour screen. The PN-graph under construction can be seen in a number of screens (where it looks as close as possible to the final output obtained by a printer or a plotter). The editor is menu driven and has self-explanatory option names. The user moves and resizes the objects by direct manipulation – i.e. by means of the mouse (instead of typing coordinates and object identification numbers on the keyboard). This also applies to the pages which can be opened, closed, scrolled, scaled and deleted. When the user deletes a page connector the corresponding hierarchical relationship is destroyed (and thus the corresponding supernodes become ordinary nodes).

One important difference between the GPN-editor and many other drawing programs is the possibility to work with **layers** (groups) of objects. This means that the user is able to select a set of objects and simultaneously change the attributes, delete the objects, copy them, or move them. The user can select layers in many different ways (e.g. by dragging the mouse over a rectangular area or by pressing a key while he points to a sequence of objects). The GPN-editor allows the user to perform operations on layers in exactly the same way as they can be performed on individual objects (there are only very few operations which do not make sense for layers) – and this has the same effect as when the corresponding operation is performed on each layer member one at time. All members of a layer have to belong to the same page.

In the design of the GPN-editor it has been important for us to make it as flexible as possible. As described above, this means that it is possible to construct PN-graphs which look very differently. However, it also means that each graph can be created in many different ways. One example of this principle is the variety of ways in which the page hierarchy can be constructed. Another example is the fact that the GPN-editor allows the user to construct various objects in many different orders: Some users prefer first to construct the net structure (i.e. the places, transitions and arcs). Later they add the net inscriptions. Other users prefer to create templates. Then they create the graph by copying the appropriate templates to the desired positions and modifying the text (if necessary). Finally, most users work in a way which is a mixture of the possibilities described above. We think that this kind of flexibility – where the user controls the detailed planning of the editing process – is extremely important for a good tool. Thus the GPN-editor has been designed to allow most operations to be performed in several different ways.

A PN-graph contains several different kinds of information and this means that the individual pages very easily become cluttered. To avoid this the user

is allowed to make some information invisible (without changing the semantics of the objects to which that information are attached). As an example the user may hide the capacities of all places.

It should be noticed that the generality of the GPN-editor means that the user can create various graphs, diagrams, and it also is possible to produce bad nets. We do not believe it is sensible to try to construct a tool which makes it impossible to produce such objects. Such a tool will, in our opinion, inevitably be far too rigid and inflexible. However, we do of course believe that the tool should make it easy for the user to make good nets.

There are many other facilities in the GPN-editor: operations to open, close, save and print graphs. It is also the intention to allow the user to save part of a graph and later load it into another graph. In this way it will be possible to create libraries of reusable submodels. There is an operation to check the syntax of the PN-graph, see further on, operations which assist the user to select the object, move objects to another positions (on the same page), change object size, merge a group of nodes into a single node, duplicate a node (by using the command on a group of nodes, it is possible to get a subnet which is identical to an existing subnet), hide and show objects and change the graphical layering of the objects, operations to redraw the page hierarchy – when this has become too cluttered, operations to select groups, and other ones which make it easy to create arcs with right angles and vertical/horizontal segments.

Any operation preserves the syntactical correctness of the net. For instance, deleting a place also deletes the adjacent arcs. The nets can be output in a terse text format, a verbose text format suitable for careful checking, and graphically.

The GPN-editor can be used at many different skill levels. Casual and novice users only have to learn and apply a rather small subset of total facilities. The more frequent and experienced users gradually learn how to use the editor more efficiently: All the more commonly used commands can be invoked by means of key shortcuts.

Finally, it should be mentioned that the GPN-editor is designed to work with large PN-graphs – i.e. graphs the size of which is practically only restricted by the main storage capacity of the given computer.

In particular, the GPN-editor has the following main editor functions (cf. Jensen (1987):

- add, delete place, transition and arc,
- edit marking and capacity of places,
- create nodes with refinements, i.e. they may be made to represent a subnet (thus, a net may be structured hierarchically by the user),
- rescale nodes,
- name places and transitions,
- reposition nodes (if a node is repositioned, all its arcs are automatically adjusted too),
- add, delete and reposition text comments,

- associate text comment with places, transitions, arcs and the net,
- move, copy, delete, save, load subparts of the net,
- rotate and symmetry subparts of the net,
- rescale the entire net or a subnet (this changes both the view and the final product),
- merge subnets into single net,
- refine net nodes,
- create, modify and draw net layers,
- set up attributes of net layers,
- name and select views of the net,
- redraw the net,
- produce output at different quality and speed (it also is possible to output only part of a net),
- use grid for aligning items,
- save/load file to/from disk,
- construct the different textual representations of a net.

Syntax check The GPN-editor is syntax directed – in the sense that it recognizes the structure of Petri nets and prevents the user from making many kinds of syntax errors. This is done by means of a large number of built-in syntax restrictions. All the built-in restrictions deal with the net structure and hierarchical relationships. As examples, it is impossible to make an arc between two transitions (or between two places), and to create an illegal structure in the substitution hierarchy. These restrictions are necessary in order to guarantee that the PN-graph has a well-defined semantics – and thus they must be fulfilled before a simulation (and other kinds of behavioural analysis) is performed.

All the net structure checking is done by the GPN-editor and it is the error messages of this editor which is presented to the user. These messages are easy to understand and use Petri net terminology.

The GPN-editor allows the user to give each page, transition and place a name (i.e. a text string) and a number. It should, however, be understood that these names have no semantic meaning. Names are used in the feedback information from the editor to the user. To make this information unambiguous it is recommended to keep names unique. Many users have a large number of transitions and places with an empty name (and this is no problem, as long as the current net is not used in making a simulation).

The possibility of performing an automatic syntax check means that the user has a much better chance of getting a consistent and error-free PN-graph. This is very useful – also in situations where the user is not interested in making a simulation (or other kinds of machine assisted behavioural analysis).

3.4. Graphical Petri net simulator

The GPN-editor and the graphical Petri net simulator (the GPN-simulator) are two different parts of the same program and they are closely integrated with each other. The GPN-simulator is able to work with large Petri nets, e.g. Petri nets with 25 screens of a monitor.

The GPN-simulator, during the execution of a simulation step, goes through three different phases: First it makes a selection (according to a choice of a strategy) between enabled transitions, then it removes and adds tokens at the input/output places of the occurring transitions, and finally it calculates the new enabling.

The user must be able to follow the on-going simulation – and it is obvious that no screen (or set of screens) will be able simultaneously to display all page instances of a large model. Like the editor, the GPN-simulator uses a screen for each hierarchy page and on this screen the simulator displays the subnet of one of the corresponding hierarchy page.

When the transitions (a transition) occur(s) the simulator automatically displays the corresponding page screen (if necessary), brings it on top of the subnet, and scrolls the screen so that the transition(s) becomes (become) visible. The user can, however, tell that he does not want to observe all page hierarchies. In that case the simulator still executes the transitions of the non-observed page hierarchies but this cannot be seen by the user. The user can also work in this way that he asks the simulator to pause after each simulation step. At each breakpoint the user can investigate the system state (and decide whether he wants to continue or cancel the remaining part of the simulation).

It is possible to perform both **manual** and **automatic** simulations. In a manual simulation the simulator calculates and displays the enabling, the user chooses the occurrence transitions to be executed and finally the simulator calculates the effect of the chosen step. During the construction of a step, the simulator assists the user in many different ways, e.g. the simulator always shows the current enabling (and updates it each time a new occurrence transition is added/removed at the step). In an automatic simulation the simulator chooses among the enabled occurrence transitions like by means of a random number generator. It is possible to specify how large each step should be: It may contain a single occurrence transition or as many as possible.

The user can, at any time during a simulation, change between manual and automatic simulation. It is usual to apply more of manual simulation modes early in a project (e.g. when a design is being created and investigated) while the automatic modes are more used in the later phases (e.g. when the design is being validated). There are many other facilities in the GPN-simulator: An operation that proposes a step (which can be inspected and modified by the user before it is executed); operations to return to the initial marking of the PN-graph and to change the current marking of an arbitrary place (this means that it often is possible to continue a simulation in the case where a minor modelling

error is encountered); an operation to save system states. Moreover, the earlier comments about different skill levels and a consistent and self-explanatory user interface also apply to the GPN-simulator.

Finally, it should be mentioned that many modellers use simulation during the construction of PN-graphs. It is thus very important that it is reasonably fast to shift between the editor and the simulator (and that it is possible to simulate selected parts of a large model).

3.5. Analysers

The proof of the correctness of a system specification in a Petri net based language usually is done in two substeps. First, the underlying net is analysed with respect to basic dynamic and structural properties such as safeness, boundedness, liveness, conflicts, resetability, deadlock and livelock avoidance, holding of facts and the resetability or coverability of certain (wanted or unwanted) states, moreover, some structural properties such as coverability by invariants or special type subnets (like strongly connected state machines) are tested. Next, using the results of such an analysis the correctness of the complete model is verified. It appears, that it is rather difficult, if not impossible, to do this in a systematic way. But, the more information has been collected and the more design errors have been detected during the analysis and simulation of the underlying Petri net, the easier the second substep is to carry out.

This part of PN-tools is to analyse Petri nets with respect to the structural and dynamic properties.

3.5.1. Structure checking and liveness

For testing structural properties from which, in case of ordinary Petri nets, liveness properties follow, the modules APROPE, ALVDTP and ASMCTE are available.

If one is given an unknown net, using the program APROPE one can obtain information on elementary net properties such as: the number of places (transitions), the minimal (maximal) number of the net nodes, as well as the maximal entrance (exit) degree of the net nodes. One can check basic structural properties. Additionally, it is tested, whether the read-in net considered as an undirected graph, is connected.

If a net is an ordinary one the program ALVDTP checks first whether it is a state machine, a free-choice net, an extended free-choice, and an extended simple net. These properties are related with the liveness via the deadlock-trap-property (see Reisig, 1985). Then the minimal deadlocks are computed. If there exists a clean deadlock, the net is not live. Next, the possible conclusion connected with the liveness of a net follows:

1. If a given net is an extended simple net and the deadlock-trap-property holds, then a net is live Holt, et al. (1974).

2. If a given net is an extended free-choice and the deadlock-trap-property does not hold, then a net is not live Commoner, (1972).
3. If for a given net the deadlock-trap-property holds, then no dead marking can be reached.

Moreover, the module ALVDTP checks whether the net is a state machine decomposable (which implies that it is bounded under any initial marking) and, in this case, whether it is a state machine allocable (from which we can conclude that it admits a live marking). If the net is a state machine allocable the initial marking is examined whether it marks any strongly connected state-machine component, in this case, the net is live.

The module ASMCTE decides whether a given net is a state machine coverable, i.e. coverable by components which are state machines.

3.5.2. Reachability graph analysis

In the modules described above possible conclusions to the dynamic properties are drawn from the structural properties on the basis of an initial marking. If there are no information for such conclusions, we have to investigate the reachability graph.

The module ACGDTM computes the coverability graph, Karp, Miller (1969), which is identical with the reachability graph in the case of a bounded net. There by, this module provides full information on the boundedness and coverability properties of a given net. Besides, ACGDTM shows the dead transitions and markings.

The module AREACT tests a marking which has to be initially input from the terminal for reachability. If a given marking is reachable, then a corresponding path is output.

Liveness, conflicts and resetability of bounded Petri nets can be examined by using the module ALCTRE. First, this module builds the reachability graph. If a dead marking is found, ALCTRE ends with a corresponding indication, otherwise it is checked whether there are the dead transitions. If no dead markings can be reached, the resetability is tested. If a net is not resetable, a list of nonresetable markings is output. Then, the module checks at each reachable marking at which several transitions have concession whether one of these transitions takes the concession of another upon firing. The obtained conflicts are printed in a table.

The module ARGRTE computes the reachability graph for bounded nets. It also lists the dead transitions at an initial marking as well as dead markings. The module writes the reachability graph to a file whereby dead markings and dead transitions are indexed.

If a given net is not bounded, the number of an unbounded place is output.

The module APRIOR treats the priority nets. The priorities are requested in the beginning and can be read from the terminal. This module works (under

the priority firing rule) like ARGRTE. The test for boundedness is omitted, because this property is undecidable for the priority nets.

3.6. Reducer

The module REDUC can be used to reduce the size of the given net, so that it becomes analysable by modules from ANALYSER subsystem, and it can be used to find an equivalent small net with known properties. This module implements the most essential local reduction steps known from the literature, e.g.

- merging of nodes which share all predecessors and successors,
- fusion of equivalent places,
- reduction of different kinds of place/transition chains.

Various reduction steps of a net are offered in the menu of the program REDUC.

3.7. Verifiers

This subsystem provides tools for the verification of Petri net models. By verification we mean the proof that certain desired system properties hold in the Petri net model.

The verification method is as follows. First, we compute a basis for the space of all (nonnegative) place (transition) invariants. Next, from this we can derive information on boundedness, liveness and livelock properties, moreover, in general invariants have an interpretation in terms of the modeled system which can be useful for its verification. It is worth to reflect that several conclusions derived from invariants are related only to nonnegative invariants (e.g. net coverability by P-invariants).

This subsystem consists of the modules VCOMPO, VINALL, and VINTES.

3.7.1. Components of a net

If a net model is generated by synchronizing subnets modeling e.g. sequential subprocesses, then the investigation whether all subprocesses participate in the total process and whether they enter it structurally (as a component) is part of the verification of the total system. This verification problem is supported by VCOMPO.

3.7.2. Computation of invariants

A basis for the set of all place (transition) invariants is computed by the module VINALL. The set of all invariants is the set of vectors generated by arbitrary linear combinations from the computed set. This module also computes a basis for the set of nonnegative place (transition) invariants. Correspondingly, the set of all nonnegative invariants is the set of vectors that can be generated from the computed set by means of linear combinations with nonnegative coefficients.

A system of generators for the set of nonnegative subinvariants (surinvariants) of places and transitions, respectively, can also be computed by the module VINALL. In this case, too, the set of all invariants of the type considered is the set of vectors that can be generated by linear combinations with nonnegative coefficients. In order to save time, the set computed here might be not minimal. On the other hand, restriction of storage may lead to doing without the completeness of the computed set (coverable invariants are deleted). However, in this case each of the invariants being of interest is coverable by one of the invariants that can be generated from the computed set.

3.7.3. Invariant test

The module VINTES permits to check the properties of place and transition vectors, respectively. It shows whether the vector investigated is an invariant, a subinvariant or a surinvariant. Moreover, it can be tested which values the single equations yield for this vector. The process of computation of these equations is written out.

First, for a given net to be read-in it has to be indicated. Then the user can select whether place or transition vectors will be tested. As for VINALL the incidence matrix (at a transition vector) and its transposed matrix (at a place vector), respectively, is built up.

3.8. Extensions to the net analysing modules

When working with our Petri net analyser (PN-analyser) we have found it difficult to model complex systems with PT-nets, because the nets are often large and it is not quite easy to see all the interactions between the nodes of the net. Due to that we have found it necessary to use extended nets for our modeling purposes; but then we need an analyser for those nets.

The fastest way to construct an analyser is to use existing programs as much as possible. Therefore our first step to enable the analysis of extended nets is to translate them into PT-nets and to use our PN-analyser. All we need is an automatic extended Petri net – to the Petri net translator (the PN-translator). The PN-translator is implemented first for: self-modifying nets, priority nets, nets with inhibitor arcs and some subclass of PT-nets. The PN-translator has the following tasks:

- it forms a PT-net corresponding to a given SM-net (IA-net, P-net),
- it maps the initial marking of the SM-net (IA-net, P-net) into a marking of the PT-net,
- it maps a marking of the PT-net (e.g. a deadlock marking found by the PN-analyser) into a marking of the SM-net (IA-net, P-net).

Using the PN-translator and the PN-analyser will be only a temporary way in analysing extended nets (SM-nets, IA-nets, P-nets). But it will be a reality

very soon. And after that we can concentrate on implementing an extended analyser.

4. Implementation environment

PN-tools has been developed under a DOS environment running on IBM PC computers. The access to the particular programs can take place from PN-tools level. The interaction between the user and the system PN-tools consists only of the selection of commands in menus. To each net, we associate the graphic representation, two textual representations and several textual files which include an addition information about a given net; these all files are managed in a directory whose name is created in an installation stage of PN-tools. For monochromatic graphics the Hercules card is proposed. If colour graphics are to be used, then SVGA is preferred. Black and white graphics hardcopy can be obtained using e.g. STAR printer. It is also possible to use e.g. ROLAND plotter as terminal for obtaining colour graphics hardcopy. Moreover, this yields much better typographical quality.

5. Future plans for PN-tools

5.1. Extensions of PN-tools

The GPN-editor/-simulator is being extended to handle timed Petri nets, the extension of ordinary (classical) Petri nets making it easy to describe systems which are time-driven. It will then be possible to use the same net model to analyse both logical correctness and time performance of a system.

The implementation of timed Petri nets will be finished during the second half of 1995. Later we will also extend the GPN-editor to allow the user to construct and modify Petri nets by means of a set of behaviour preserving transformation rules (for more information see Berthelot (1987)). We will also extend the GPN-simulator to handle code segments written in languages such as: Pascal and C++ and we will extend the GPN-editor/-simulator to handle the other hierarchy constructs and different extensions of Petri nets (e.g. FIFO-nets, timed and high-level Petri nets).

5.2. Additional PN-tools

Additional Petri net tool will be created to support reachability graph analysis. The tool will construct in a graphical mode reachability graphs for Petri nets and their extensions considered in this paper. It will also assist the user in the analysis of the constructed graphs. As described in Section 2.2, a large number of system properties can be automatically determined from the reachability graph (by inspection of individual markings and from strongly connected components). There is, however, also a need to develop more complex search systems by which the user can perform an interactive inspection of a large reachability graph. The

Petri net reachability graph tool will be able to handle Petri nets mentioned above and it will be tightly integrated into the existing PN-tools. It will e.g. be possible to ask the GPN-simulator to execute an occurrence sequence which is found in the reachability graph – or as the reachability graph analyser to search for markings which are identical to or larger than the current marking of the GPN-simulator.

To keep the size of reachability graphs manageable it will be necessary to create reachability graphs for selected parts of a large model. The first version of the reachability graph tool will be available during 1995. It is, however, obvious that this, among other things, will depend upon the priority given to the improvement of the new reachability graph tool (and other extensions of existing PN-tools).

Finally we want to develop PN-tools to support reduction and translation methods and the analysis of special subclasses and/or extensions of Petri net – e.g. as described in Starke (1987). Such tools have, however, lower priority than those described above.

6. Conclusions

We have presented PN-tools, collecting some of the different kinds of computer tools which are needed in the Petri net area. These tools support the user in construction of nets (also hierarchical nets), as well as modification and analysis in a natural and effective way. Moreover, PN-tools provides the opportunity to work not only with textual representations of Petri nets but also directly with graphical representations.

The main point of PN-tools is its extensibility: it is easy to connect other tools to the system. PN-tools allows to describe and analyse different kinds of nets, owing to a flexible textual and graphical representation of the nets. Integrating several tools for designing nets, checking structural and dynamic properties, etc., PN-tools provides an environment for design and verification of nets.

There is a large number of different groups who work with the development of Petri net tools. However, many of the tools are still research prototypes. Only few tools handle many kinds of nets and very few handle hierarchical nets. For use in industrial environments, there are only few tools that are powerful enough, sufficiently robust, and have the necessary documentation and support.

It seems that our system presented in the paper is a professional tool which is well maintained and offers a stable platform for extensions.

By using PN-tools, organizations and management consultants obtain a method and a tool with which operational procedures and organizational structures can be analysed quickly and described in a presentable way.

Analysts, system designers and everyone who, in the framework of project development, has to describe coherently and vividly complex procedures of system engineering on the basis of a theoretical method, are able to carry out their

task in a more economical and time-saving way.

Acknowledgements

The author is grateful to the Departmental Program RP I.09 for financial support for the Petri net research and to Tadeusz Gąsior, Urszula Niedziałek, Bogumił Komarek, Paweł Hałys and Stanisław Wasilewski for their high quality programming.

References

BERTHELOT G. (1987) *Transformations and Decompositions of Nets.* In: W. Brauer, W. Reisig and G. Rozenberg (eds.): Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986 Part I, Lecture Notes in Comput. Sci., vol. 254, Springer-Verlag, 359-376.

BEST E. (1987) *Structure theory of Petri nets: the free choice hiatus*, In: Lecture Notes in Comput. Sci., vol. 254, Springer-Verlag, 168-205.

BILLINGTON J., WHEELER G., WILBUR-HAM M. (1988) *PROTEAN: a high-level Petri net tool for the specification and verification of communication protocols*, IEEE Transactions on Software Engineering, Special Issue on Tools for Computer Communication Systems, SE-14(3), 301-316.

BRAUER W., REISIG W., ROZENBERG G. **(eds.) (1987a)** *Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986 Part I*, Lecture Notes in Comput. Sci., vol. 254, Springer-Verlag.

BRAUER W., REISIG W., ROZENBERG G. **(eds.) (1987b)** *Petri Nets: Applications and Relationships to Other Models of Concurrency, Advances in Petri Nets 1986 Part II*, Lecture Notes in Comput. Sci., vol. 255, Springer-Verlag.

CHIOLA G. (1985) *A software package for the analysis of generalized stochastic Petri net models.* In: Proc. Int. Workshop Timed Petri Nets, Torino, Italy, July 1-3, 1985, 136-143.

COMMONER F. (1972) *Deadlocks in Petri nets*, Appl. Data Res. Inc. RR CA-7206-2311, Wakefield.

FELDBRUGGE F.H.J. (1986) *Petri Net Tools.* In: G. Rozenberg (ed.), Advances in Petri nets 1985, Lecture Notes in Comput. Sci., vol. 222, Springer-Verlag, 203-223.

FELDBRUGGE F.H.J, JENSEN K. (1987) *Petri net tool overview 1986.* In: W. Brauer, W. Reisig and G. Rozenberg (eds.): Petri Nets: Applications and Relationships to Other Models of Concurrency, Advances in Petri Nets 1986 Part II, Lecture Notes in Comput. Sci., vol. 255, Springer-Verlag, 20-61.

FELDBRUGGE F.H.J. (1990) *Petri net tool overview 1989.* In: G. Rozenberg (ed.): Advances in Petri Nets 1989, Lecture Notes in Comput. Sci., vol. 424, Springer-Verlag, 151-178.

FELDBRUGGE F., JENSEN K. (1991) *Computer tools for high-level Petri nets.* In: K. Jensen, G. Rozenberg (eds.), High-level Petri Nets. Theory and Application, Springer-Verlag, 691-717.

FINKEL A. (1990) A *minimal coverability graph for Petri nets.* Proc. of the 11th Int. Conference on Application and Theory of Petri Nets, Paris, 1-21.

HACK M. (1975) *Decidability questions for Petri nets*, Ph.D. dissertation, Department of Electrical Engineering, Massachusetts Institute of Technology, Cambridge.

HOLT A.W. ET AL. (1974) *Final R. for the Proj. "Development of the Theoretical Foundations for Description and Analysis of Discrete Information Systems"*, vol. I: Semantics, vol. II: Mathematics. Mass. Computer Associates, Inc., Wakefield.

JENSEN K. (1987) *Computer Tools for Construction, Modification and Analysis of Petri Nets.* In: W. Brauer, W. Reisig and G. Rozenberg (eds.): Petri Nets: Applications and Relationships to Other Models of Concurrency, Advances in Petri Nets 1986. Part II, Lecture Notes in Comput. Sci., vol. 255, Springer-Verlag, 4-19.

JENSEN K. ET AL. (1990) *Design/CPN extensions.* Meta Software Corporation, 150 Cambridge Park Drive, Cambridge MA 02140, USA.

JENSEN K., ROZENBERG G. **(eds.) (1991)** *High-level Petri Nets. Theory and Application.* Springer-Verlag.

KARP R.M., MILLER R.E. (1969) *Parallel program schemata.* Journal of Computer and System Sciences, vol. 3, 147-195.

MEMMI G., VAUTHERIN J. (1987) *Analysing nets by the invariant method.* In: W. Brauer, W. Reisig and G. Rozenberg (eds.): Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986 Part I, Lecture Notes in Comput. Sci., vol. 254, Springer-Verlag, 300-336.

MURATA T. (1989) *Petri nets: Properties, Analysis and Applications*, Proc. of the IEEE, vol. 77, no. 4, 541-580.

PETERSON J.L. (1981) *Petri Net Theory and the Modeling of Systems.* Prentice-Hall, Inc., Englewood Cliffs, N.J.

PETRI C.A. (1966) *Kommunikation mit Automaten*, Bonn: Institut für Instrumentelle Mathematik, Schriften des IIM Nr.3, 1962. Also, English translation, Communication with Automata. New York: Griffiss Air Force Base. Tech. Rep. RADC-Tr-65-377, vol. 1, Suppl. 1.

Proc. Int. Workshop Timed Petri Nets (1985) Torino, Italy, July 1-3, 1985.

Proc. Int. Workshop Petri Nets and Performance Models (1987) Madison, WI, August 24-26, 1987.

REISIG W. (1985) *Petri Nets.* EATCS Monographs on Theoretical Computer Science, vol. 4, Springer Publ. Company.

STARKE P.H. (1985) *Petri-Netz-Maschine – A Software Tool for Analysis and Validation of Petri Nets. Systems Analysis and Simulation*, Proc. of the 2nd Int. Symp., Berlin 1985. – Oxford: Pergamon, 474–475.

STARKE P.H. (1987) *On the mutual simulatability of different types of Petri nets.* In: K. Voss, H.J.Genrich, G. Rozenberg (eds.), Concurrency and Nets, Springer-Verlag, 481-495.

STARKE P.H. (1990) *Analyse von Petri-Netz-Modellen.* B.G. Teubner, Stuttgart.

SURAJ Z. (1990) *GRAPH: A graphical system for Petri net design and simulation.* Petri Net Newsletter no. 35, 32-36.

SURAJ Z. (1993) *A System for the Design and Analysis of Petri Nets.* ICS Research Report 3/93, Warsaw University of Technology.

VALETTE R. (1978) *Analysis of Petri nets by stepwise refinements.* Journal of Computer and System Sciences, vol. 18, no. 1, Academic Press, Inc., New York and London, 35-46.

VALK R. (1978) *Self-Modifying Nets: a Natural Extension of Petri Nets.* In: G. Ausiello, C. Bohm (eds.), Automata, Languages and Programming, Lecture Notes in Comput. Sci., vol. 62, Springer-Verlag, 464-476.

VOSS K., GENRICH H.J., ROZENBERG G. **(eds.) (1987)** *Concurrency and Nets.* Advances in Petri Nets, Springer-Verlag.