PNIF: AN INTERCHANGE FORMAT FOR SYSTEM SPECIFICATION WITH COLOURED PETRI NETS

A.M. KOELMANS
A.V. YAKOVLEV
Computing Science Department
University of Newcastle upon Tyne
NE1 7RU United Kingdom

D.J. KINNIMENT
Y. Xu
Department of Electrical and Electronic Engineering
University of Newcastle upon Tyne
NE1 7RU United Kingdom

Keywords Interchange formats, Petri Nets, EDIF, specification languages and paradigms, design modelling, formal techniques and methods for verification.

Abstract

Petri Nets are becoming increasingly popular as a means to formally specify and verify hardware systems at a high level. Many tools are being developed. Unfortunately, no common format is available that would allow these tools to interchange designs, for two main reasons: most tools only implement a subset of the Petri Net notation, and many tools do not allow the use of hierarchy. We present an interchange format for the description of hierarchical Coloured Petri Nets. It has many features in common with EDIF.

1. Introduction

Petri Nets are becoming increasingly attractive as a formalism for the design of hardware systems. The graphical nature of the Petri Net notation makes it more attractive to circuit designers than algebraic notations, which are much less intuitive. Petri nets are mathematically well founded, and can be used to check for potential hazards in circuits. They can be used as a modelling language to perform formal synthesis[2] and high level analysis of complex processor designs[8] and signal processing chips[10]. Petri Nets and their closely related notation, Signal Transition Graphs, are extensively used in the area of design of asynchronous circuits [5]. It is possible to translate Petri Nets to VHDL, and vice versa for subsets of VHDL [7, 9], making it possible to integrate Petri Net tools into existing design environments. Many researchers have proposed extensions to the Petri Net notation for the accurate modelling of circuit properties such as timing information (e.g., [6]).

As a result, an increasing number of researchers wish to use or develop Petri Net tools. There is currently a large number of such tools available, for example Cabernet, CPN/AMI, Design/CPN, Loopn, PNS, SURF–2, and XSimNet. A very comprehensive overview can be found in [4]. Some of these tools, such as Design/CPN, are commercial products and are not widely used. The vast majority of the tools are free. Unfortunately, since Petri Nets are normally only used through a graphical interface, the file format needed to save the nets is usually defined without consideration of the problem of interchange with other systems. As a result, there are as many file formats as there are tools, because different tools usually use different subsets of the full Petri Net notation. If researchers are to interchange designs, some common format will have to be found that is capable of describing the full Petri Net notation. Some form of standardisation is therefore urgently required in order to allow Petri Net designers to exchange designs without major difficulties. To the best of our knowledge, PNIF (Petri Net Interchange Format) represents a first attempt at such a format.

PNIF allows textual description of Coloured Petri Nets (CPNs) [3]. It is being developed as part of a joint project to synthesize synchronous and asynchronous controllers from CPNs. PNIF has a LISP–like syntax, and is heavily influenced by EDIF, the Electronic Design Interchange Format

[1] (version 2.0.0.). The basic structure of a PNIF description is

```
(keyword  parameter1 parameter2 ...... parametern)
```

The parameters in this construct may be atomic language components, or they may recursively consist of a keyword and a list of parameters. This particular kind of syntax is very suitable for the purpose of interchange, because it is simple (and therefore easy to parse and manipulate by means of software), easily extensible, and can be used to describe arbitrary complex structures. It is not intended for direct use by human beings.

The paper is organised as follows. We first discuss the major components of CPNs in general terms. We then discuss the major aspects of the interchange format: overall organization, definition of colours, variables and tokens, hierarchy, places, arcs and transitions, outputs, and geometrical aspects. We finish with a complete example.

2. Coloured Petri Nets

A Petri Net is a bipartite directed graph. It consists of *places* and *transitions* which are connected by *arcs.* Each net has a *marking* which is an assignment of *tokens* that circulate around the net. Net execution depends on rules that determine whether a transition is enabled or fired. Places, transitions and arcs have associated *conditions* and *actions* that determine these rules. They also have graphical properties, such as *labels* that are displayed on the graphical interface. Coloured Petri Nets have in addition the possibility to specify *colours*, which are in effect data structures, and functions and variables that operate on these data structures.

In order to allow the user to keep the complexity of the net within reasonable bounds, PNIF has provisions for *hierarchical* descriptions of nets, a feature frequently not implemented in Petri Net tools. Hierarchy is essential for reducing the complexity of net descriptions to manageable proportions. The net is modelled as a collection of small nets, which may instance each other. The semantics of hierarchy are discussed below.

All features of a net have *graphical* properties. For example, places are usually drawn as circles, and transitions as rectangles. These rectangles may differ in size. Arcs may follow irregular paths on the screen, in order to make the display more understandable. *Annotations* (i.e. text) may be inserted anywhere to improve readability. Text has, of course, a number of properties itself relating to font family, font size etc.

Once a net has been built, it is usually subjected to *analysis* or some form of *verification* in order to establish whether the net is free from hazards that would make it impossible to build a working circuit from it. There are many features that may be analysed. Some of these are discussed below. In some cases, the net may be subjected to *synthesis* if the analysis produced satisfactory results.

We do not intend to discuss the syntax of PNIF in any detail here. Rather, we discuss general and novel features, and present an example at the end of the paper to give the reader a good idea of what the format looks like. We divide the features into the following categories:

- basic PNIF constructs;
- overall design organisation;
- basic net elements, such as places and transitions;
- the use of hierarchy;
- advanced features such as analysis and verification;
- graphical properties.

3. Basic constructs

The basic types that are supported are boolean, (subranges of) integer, floating point, and enumerated types. PNIF supports a number of keywords to specify arithmetical and boolean expressions using these basic types. Programming language constructs are available through `if` and the `while` statements. A new colour (i.e. data type) for use in token expressions can be defined. It is of course also possible to declare variables, constants, arrays, and functions. Statements are available to assign values to variables. PNIF thus supports a full range of programming constructs.

4. Overall design organisation

A complete Petri Net description in PNIF may contain the following elements:

- data management features;

- global declarations of colours, variables, functions, etc.;
- the collection of subnets that make up the net.

The data management section is used to insert information used by data management tools. These include the PNIF version number, the version number of the Petri Net, the name of the program that created the net (optionally with its version number), the name of the site where the design was entered, the name of the author, a time stamp using Universal Time Coordinates, a title for the design, a possible annotation, and a list of external library files. All these features are similar to those found in EDIF.

5. Net structure

A net must have a generic name. It will normally have inputs and outputs, which must be identified as such. It will further consist of a collection of arcs, transitions, places, subnets, tokens, an initial marking, and a definition of the outputs generated by the underlying circuit implementation.

Places and transitions have associated predicates, actions and delays. A predicate is a boolean expression that describes under which condition(s) tokens resident within the place will be manipulated. If the predicate is to be displayed on the screen, it will have to be in the form of a string. Actions are performed as a result of the value of the predicate. A delay specifies minimum and maximum delays for transitions.

Place statements may also specify the minimum and maximum number of tokens that may be allowed to accumulate in the place during simulation. Transitions may also specify a name and an edge of a clock signal, with which the transition is to be synchronized. The clock edge is either a rising or falling edge. Arcs may have an associated action and delay, but no predicate. An arc statement also specifies the source and sink of the arc, i.e. from which place/transition to which transition/place the arc goes. The source or sink of an arc may be empty, allowing for 'dangling' arcs (which are used to prevent the graphical display from becoming too cluttered).

Each net declares the tokens initially available in the net. Tokens may of course have a particular type. If the number of tokens of a particular type is greater than 1, we have an array of tokens. The type must be a predefined type or a user defined colour.

Initial markings describe how many tokens of a particular type are initially present in a particular place.

One hardware design specific feature of PNIF is the definition of outputs of the underlying circuit. This takes the form of a number of *output equations* which will normally be in the form of boolean equations on the set of names of input variables.

6. Hierarchy

Hierarchy is essential in managing the complexity of large nets. Jensen[3] describes the issues in some detail. A lower level net (called a subnet in this paper) is called a *page.* Pages that act as a single transition are called *substitution transitions.* Similarly, there are *substitution places*. Both kinds have special semantics which are briefly explained below. In addition, one can regard subnets also as *macros*, which have no special semantics. Nets containing macros must be flattened before they can be analysed.

Each subnet has an *interface* to the outside world. This determines what internal parts of a net are accessible from the outside. The interface is similar to a list of formal procedure parameters in programming languages. It declares the *ports* of the design. Input and output ports will frequently have their own graphical representations. Port names must not clash with other internal names. They will normally be connected to internal net elements using appropriate arc statements.

Ports have different types dependent on how they are used. Ports of type `input`, `output` and `clock` indicate that the subnet has Jensen semantics. This means that a port will *merge* with the corresponding *sockets* in the surrounding net. All connecting nodes on the outside must be either places or transitions. If they are places, the subnet is a substitution transition, otherwise it is a substitution place. This enforces the same connection rules as in a flattened net. Enforcement of semantic rules will have to be done by design tools. The following port types are also available: `placeinput`, `placeoutput`, `transitioninput`, and `transitionoutput`. The use of such types indicates that the net is to be used as a macro. If a port has such a type, it is a proper part of the subnet (that is, it will not merge with any sockets in the outside net), and may only be connected to appropriate other parts of the outside net. For example, a port of type `placeinput` may only be connected to transitions in the outside net.

7. Advanced features

PNIF has a construct to indicate that the net has been analysed for a number of advanced properties, and that therefore analysis for these properties can be omitted, or otherwise be used by design tools to their advantage. The properties are displayed in the form of a list of strings.

There are many examples of such properties in the extensive Petri Net literature. Examples of *structural* properties include whether the net is marked, free choice, extended free choice, or whether the net describes a Finite State Machine. Examples of *behavioural* properties can be divided into those referring to *capacity* (i.e. safeness, k–boundedness, persistence) and *activity* (i.e. liveness and freedom of deadlocks and livelocks).

To the best of our knowledge, support for analysis and verification has not previously been used in file formats for Petri Net tools.

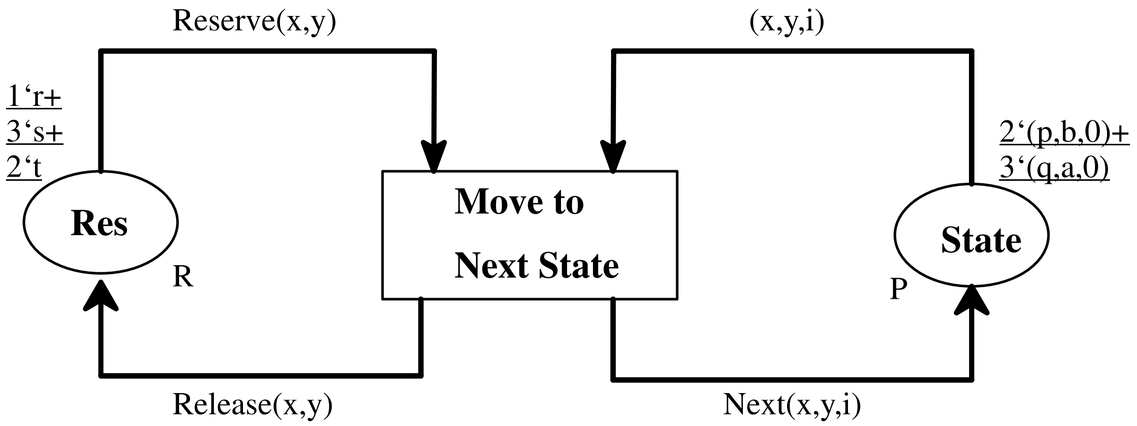
8. Graphical features

All net features have graphical properties. PNIF allows these to be set individually. They include annotations (i.e. text to be displayed with the object), font properties, and the thickness of the lines used to draw the objects. A rectangular canvas determines the overal extent of the coordinates of the objects. The following properties can also be specified: centre and radius of circles, length and width of rectangles, and start and finish points of arcs, as well as more complex curves.

In addition, arcs have associated information about the arrow heads and the arc type. Arrow heads, if present, may be at the start and/or finish of the arc. So called 'inhibitor' arcs have a special graphical appearance in order to distinguish them from ordinary arcs.

9. An example

The following example is based upon a net taken from Jensen [3]. It describes a simple resource allocation system. There are no complex actions or guards associated with the places and the transition.



The PNIF description of the example is as follows. Some definitions of functions, arcs and markings have been deleted for the sake of brevity. To demonstrate the use of hierarchy, the transition is shown as a supertransition (this is not the case in the Jensen example).

```
(pnif
   (info
      (title ”Resource Allocation”)
      (annotation ”Jensen, fig. 1.12”)
      (version ”4.3”)
      (program ”PNEditor”)
      (dataorigin ”University of Newcastle”)
      (author ”K. Jensen”)
      (timestamp 1994 6 5 13 45 10))
   (global
           (colour U (field a (enum p q)))
           (colour S (field b (enum a b c d e)))
           (colour I (field c (integer)))
           (colour P (field a (type U))
                     (field b (type S))
                     (field c (type I)))
           (colour R (field d (enum r s t)))
          (defun Succ (parameter y (type S))
             (body (if (equal y a) (then b)
                   (else (if (equal y b) (then c)
                   (else (if (equal y c) (then d)
                   (else (if (equal y d) (then e)
                   (else (if (equal y e) (then a)))))))))))
           (comment ”other functions go here”)
           (variable x (type U))
           (variable y (type S))
```

```
         (variable i (type I))
    )
  (net example
        (interface (port in (input))
                   (port out (output)))
        (transition t1
           (geometry (centre (pt 10 50)))
        )
        (arc a1
           (arrowhead ”end”)
           (arctype ”normal”)
           (path (placeref in) (transitionref t1))
        )
        (arc a2
           (arrowhead ”end”)
           (arctype ”normal”)
           (path (transitionref t1)(placeref out))
        )
    )
  (design resource
        (interface (port in (input))
                   (port out (output)))
        (canvas 200 100)
        (property ”safe”)
        (place p1
           (annotation ”Res”)
           (annotation ”R”)
           (geometry (centre (pt 10 50)))
         )
        (place p2
           (annotation ”State”)
           (annotation ”P”)
           (geometry (centre (pt 190 50)))
        )
        (subnet example a)
        (arc a1
           (arrowhead ”end”)
           (arctype ”normal”)
           (annotation ”Reserve(x,y)”)
           (path (placeref p1) (portaccess a in))
        )
        (comment ”other arcs are defined similarly”)
        (marking
          (atplace (placeref p1)
```

```
                    (tokenref tok1 1 (type R r)))
            (comment "other markings are defined similarly")
        )
    )
)
```

10. Conclusions

We have presented an interchange format for Coloured Petri Net descriptions. The format is currently in use as an exchange format for a project between two universities. It uses a LISP–like notation and is inspired by and closely related to EDIF. The notation allows hierarchy to be incorporated into designs. Development of the format is ongoing; in particular, geometrical information is still limited at the moment. The EDIF parser software was adapted to the PNIF grammar, and is available for interested parties, as well as conversion software from other formats.

11. Acknowledgements

The work was funded by EPSRC under contract J52327. The authors also wish to thank Artur Wrzyszcz of Bristol University for useful comments on the language. Special thanks go to Hilary Kahn of Manchester University for help and advice, and to Stephen Bevan of the Manchester EDIF centre for his help with adapting the EDIF parser to PNIF.

12. References

[1] EDIF Steering Committee, "EDIF Electronic Design Interchange Format Version 2 0 0", Electronic Industries Association, 1987.

[2] A.V. Yakovlev, A. Petrov. "Petri Nets and Parallel Bus Controller Design", Int. Conf. on App. and Th. of Petri Nets, pp. 344–364, Paris, 1990.

[3] K. Jensen, "Coloured Petri Nets", EATCS Monographs on Theoretical Computer Science, Springer Verlag, 1992.

[4] J.M. Goutal, R.K. Keller, "Petri.crim.ca – A densely populated, up–to–date Information Server on Petri Net Tools", Petri Net Newsletter, SIG on Petri Nets and Related System Models, pp. 59–63, German Computer Society, Oct. 1994.

[5] Yakovlev, A., .Koelmans, A.M., Lavagno, L. ”High level Modelling and Design of Asynchronous Interface Logic”, IEEE J. Design and Test of Computers, Special Issue on Asynchronous Logic Design, Spring Issue, 1995, pp. 32–40.

[6] G. Buonanno, S. Morasca, M. Pezze, K. Portman, D. Sciuto, ”A New Timed petri Net Model for Hardware Representation”, Proc. CHDL 91, pp. 281–300, North–Holland.

[7] J. Muller, h. Kramer, ”Analysis of Multi–Process VHDL Specifications with a Petri Net Model”, Proc. Euro–Dac 1993, pp. 474–479, IEEE Press.

[8] R. Razouk, ”The Use of Petri Nets for Modeling Pipelined Processors”, Proc. 25th DAC, 1988, pp. 548–553, IEEE Press.

[9] M. Minea, ”A VHDL Compiler for a High Level Synthesis System”, CADLAB Technical Report LiTH–IDA–R–93–23, Linkoping University, 1993.

[10] R.M. Shapiro, ”Validation of a VLSI chip using Hierarchical Colored Petri Nets”, J. Microelectron. Reliab., Vol 31, No. 4, pp. 607–625, 1991.