Hierarchical Mechanisms for High-level Modeling and Simulation of Digital Systems

Ricardo Jorge Machado, João Miguel Fernandes, Alberto José Proença
Department of Informatics, School of Engineering, University of Minho
4709 Braga codex, Portugal
rmac@di.uminho.pt

***Abstract* - The main purpose of this paper is to present an useful set of hierarchical mechanisms of specification to model and simulate digital systems. The shobi-PN model (Synchronous, Hierarchical, Object-Oriented and Interpreted Petri Net), was developed to support the use of hierarchy to model both the control unit and the data path of the systems. Modeling of a pipelined architecture unit of a microprocessor is considered as a case study to illustrate the capabilities of the hierarchical mechanisms proposed.**

***Keywords*: High-level Modeling, Digital Design, Petri Nets, Object-Orientation.**

I. Introduction

The design of complex digital systems is conceptually divided in two parts: the control unit and the data path. To specify a digital system, both the control unit and the data path should be addressed by the specification model and the CAD environment. The complexity of the design task grows when the controller behavior presents parallel activities.

Among the existing modeling paradigms, the PN-based one allows an easy specification of cooperative systems. PNs are a graphical language, easy to understand and systems modeled with PNs benefit from a mathematical theory to formally validate their properties [1].

The application of PN to hardware design has resulted in several well-known models [2, 3, 4, 5]. These models have proved to be useful and efficient to specify controllers, but none of them exploits the benefits of object-orientation to deal with the complexity of systems.

The use of object-oriented principles (hierarchy, encapsulation, inheritance, polymorphism, modularity, etc.) provides powerful mechanisms to specify hardware systems, for high-level modeling [6, 7]. The object-oriented paradigm is widely explored in the construction of a hierarchy of classes for modeling the data path and in the invocation of methods to specify the actions assigned to the parallel control sequences. These techniques also hold promise in the hardware/software co-design.

II. Hierarchical Mechanisms

A new PN model, shobi-PN (Synchronous, Hierarchical, Object-Oriented and Interpreted Petri Net), was developed to support the use of hierarchy and to model the control unit and the data path of digital systems [8]. It has been shown that the nucleus of the shobi-PN model can also be used as a kernel for distributed control systems [9].

The shobi-PN model presents the same characteristics as the SIPN (Synchronous and Interpreted PN) model [10, 11], in what concerns synchronism and interpretation, and adds functionalities by supporting object-oriented modeling approaches and hierarchical mechanisms, in both the control unit and the data path. This model embodies concepts present in synchronous PNs [12], hierarchical PNs [13], colored PNs [14], and object-oriented PNs [15].

In the shobi-PN model, the tokens represent objects that model data path resources. The instance variables represent the information that is processed on the data path and the methods are the interface between the control unit and the data path. The tokens may be considered as colored, if SIPN tokens are viewed as uncolored (the SIPN places are safe). Each token models a structure of the data path.

A node (transition or place) invokes the tokens' methods, when the tokens arrive at that node. However, only the methods that have a direct relation with the hardware control signals are directly invoked in the PN. There are additional methods available at the objects' interface that are not used by the PN. These methods are invoked by the simulation software to visualize the contents of a data path structure in any state of the PN.

Each arc is associated with one or more colors which indicates the type of objects that are allowed to pass through that arc. This means that there is a well-defined path on the PN to be traversed by each data path structure. This requirement simplifies the PN and limits the capacity of some places, since it is not needed that objects, that are not invoked, unnecessarily traverse the PN.

```
CLASS: register_1bit
VARIABLES:
   BOOL: bit
METHODS:
   BOOL RD_BIT (BOOL level)      VOID WR_BIT (BOOL level)
   { if (level == H)             { if (level == H)
     then return (bit)             then bit = H
     else return (NOT bit)         else bit = L
   }                             }
```

Figure 1. The class *register_1bit.*

Hierarchy can be introduced in the specifications in two different ways. The control unit is modeled by the PN structure, and to introduce the hierarchy on it, macronodes (representing sub-PNs) may be used. The data path resources are represented by the internal structure of the tokens, and the hierarchy can be introduced by *aggregation* (composition) of several objects inside one single token (a macrotoken) or by using the *inheritance* of methods and data structures.

If a 1-bit register is to be modeled by an object, the corresponding class should be declared with a Boolean variable and methods to read and to write that variable [16]. Fig.1. shows that class written in a generic description language.

0-7803-5008-1/98/$10.00©1998 IEEE.

Synthesis of the Controller

For simulation purposes, the shobi-PN specification can be used directly, but to synthesize the control unit, the control part of a shobi-PN must be transformed into an SIPN. This mapping is possible if the following points are considered: (1) structural compatibility in the control unit representation; (2) PN re-initializations; and (3) simultaneity on the invocation of different methods on the same node.

To ensure the structural compatibility of the control unit representation between the SIPN model and the shobi-PN model, it is imposed that the skeleton of the shobi-PN must be structurally equivalent to an SIPN without re-initializations. The following concepts used in relation to shobi-PNs are defined: (1) *Control Net*: set of contiguous nodes and arcs of a shobi-PN that structurally corresponds to a SIPN without re-initializations; (2) *Control Track*: path defined by one token in a control net; (3) *Control Nodes*: nodes (places or transitions) of a control net; (4) *Control Arcs*: arcs of a control net; (5) *Closing Track*: path defined by a token outside a control net; (6) *Closing Nodes*: nodes of a closing track; (7) *Closing Arcs*: arcs of a closing track; (8) *Closing Cycle*: path defined by the movement of one token in a shobi-PN which is composed of a control track and also, if applicable, a closing track (it can be identified by the tracking of the color associated with all the arcs of the cycle); (9) *Associated Net*: SIPN structurally equivalent to a control net after the introduction of the re-initializations for the uncolored tokens.

These concepts can be more easily understood by using the shobi-PN in Fig.2.(a) to specify a simple control sequence, with two objects/tokens to model two structures of the data path.

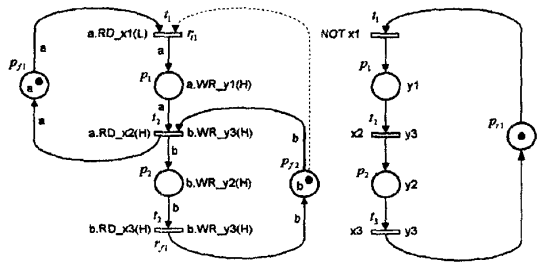

Figure 2. (left) shobi-PN for a simple control sequence and (right) its corresponding SIPN.

In this example, the control net is composed by the following set of nodes {*t1*, *p1*, *t2*, *p2*, *t3*} and by the arcs that directly link them. It defines the skeleton of the shobi-PN. The control track for the token *a* consists of {*t1*, *p1*, *t2*}, while the control track for the token *b* consists of {*t2*, *p2*, *t3*}. The closing track for the token *a* consists of {*t2*, *pf1*, *t1*} and for token *b* consists of {*t3*, *pf2*, *t2*}. The closing cycles for tokens *a* and *b* are {*t1*, *p1*, *t2*, *pf1*} and {*t2*, *p2*, *t3*, *pf2*}, respectively .

Mapping the shobi-PN into the associated net is made by transforming *k*-limited places into safe places. A safe place generates in the SIPN, whenever marked, all the control signals associated to the methods invoked in the corresponding place in the shobi-PN. As an example, consider the SIPN in Fig.2.(b).

The closing tracks allow the shobi-PN to be reinitialized, since they drive to closing nodes [*pf1*, *pf2* in Fig.2.(a)] the tokens from the control net, which enable control nodes (*t1*, *t2*). Nonetheless, after the extraction of the closing tracks from the shobi-PN, the control net obtained is not by itself cyclic because it is not closed (when there are closing tracks).

The nodes that close the control net after removing the closing tracks, must be identified in the shobi-PN, since the automatic definition of these nodes may not be unique: some systems may possess some parts of the PN for set-up purposes (L1-live PN [1]).

In the control net, the appropriate nodes (in practice transitions) are tagged as initial and final nodes so that the associated net is obtained by connecting one marked place with each pair of initial and final nodes. Each marked place is an input to the initial node and an output to the final node. In Fig.2.(a), transitions *t1* and *t3* were tagged as an initial node (tag *ri1*) and final node (tag *rf1*), respectively.

Initially marked control places may exist, which is another possibility to reinitialize the net, because these places are directly mapped into marked places in the associated net. In some extreme situations, initial and final nodes may not exist at all, which means that the control net is closed: all closing cycles are exclusively composed by control tracks, with no need to define closing tracks. Hybrid situations are also possible.

The base shobi-PN model allows PNs to have nodes that invoke methods to different tokens, in different markings. This possibility of the model must be avoided for digital systems, because it does not allow the extraction of the control net of an associated net. If this imposition is not considered non-determinism occurs; it becomes impossible to distinguish which colored tokens in the shobi-PN correspond to the uncolored tokens in the SIPN (associated net).

There should exist an unique mapping from the colored tokens into uncolored ones. This means that there is a semantical loss during the transformation of a shobi-PN into a SIPN, since the colored tokens represent objects (with data structures and methods), while the uncolored tokens just represent a Boolean variable.

If the behavior of the shobi-PN does not ensure the simultaneity in the execution of the invoked methods in each node, that requirement must be structurally imposed by an implicit synchronization.

In fig.2.(a), transition *t2* invokes methods to different objects: method RD_x2(H) to object *a* and method WR_y3(H) to object *b*. Nevertheless, these invocations are made simultaneously, due to the synchronism imposed by the enabling of transition *t2*, that depends on the markings of its input places (*p1*, *pf2*). Otherwise, if it were possible to fire transition *t2* without the presence of a token *b* in place *pf2*, it would not be possible to guarantee that the Mealy output signal *y3* will be activated, whenever transition *t2* fires. In these situations, the reading and writing of hardware signals would depend not only on the transitions' firings and on the PN marking, but also on a complex and dynamic management of the data path resources. This flexibility of the base shobi-PN model is conceptually powerful, but difficults the compatibility between the shobi-PN and SIPN models, because the unique mapping of colored tokens into uncolored ones is lost.

Replica Mechanism

Whenever several methods that use the same data structures are concurrently invoked to a given token in different nodes, it is necessary to support a replica mechanism. This mechanism allows a token to be replicated as many times as needed, so that it is structurally possible to concurrently invoke methods to

the same token, but in distinct areas of the PN. This mechanism can be used as an elegant solution for a complex problem (the multiple-sourcing) that could be alternatively, but inefficiently, solved at the algorithmic level, by changing the PN structure.

This mechanism becomes indispensable when the modeling of the data path by hierarchical aggregation is not possible. The replica are the only solution to ensure the parallelism inherent to the data path structure, if the mechanism does not violate the consistency of the tokens' data structures.

Using the shobi-PN model, it is possible to easily model complex behaviors of hardware systems (data path and controller) by decomposing the global model, even if the sub-structures have a parallel time-evolution.

The Design Flow

A design flow which incorporates the mechanisms presented in this section was conceived, allowing designers to model digital systems, to validate their properties and to simulate their behavior [17]. It also allows the automatic generation of VHDL code to synthesize the digital controller [11].

III. Example: The Pipeline Unit

The shobi-PN model was already used for digital control systems design in other projects in different application areas [18, 19]. In this section, to clarify the concepts introduced in this article, a detailed example is presented: a general-purpose pipeline unit of a microprocessor [20]. This example completely specifies the control sequences and the correct management of the data items and the data path units. The data operations are intentionally left unspecified because they are not important for showing the relevant characteristics of the shobi-PN model.

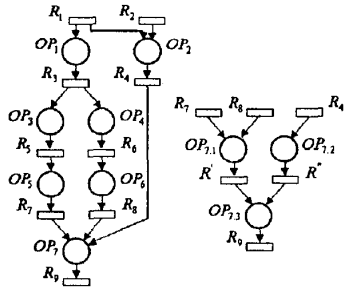

Figure 3. Data flow diagrams for (left) the pipeline unit and (right) operation OP_7.

Pipeline systems possess a set of operating units that process information simultaneously (but using different data items) allowing temporal overlapping of several instructions' execution.

Each register, excluding the first and the last ones, is simultaneously an input for one or more operators and an output for another one. To memorize the result of operator OP_i in its register R_{i+1}, the operation OP_{i+1} in the previous cycle must already be finished and its result already stored in register R_{i+2}. The operation OP_i can only be started whenever register R_i is already loaded with the results from operation OP_{i-1}.

Before starting to describe the shobi-PN, the pipeline unit's data path resources need to be modeled, by identifying the objects and defining its variables and methods. After a careful analysis of the system statement [Fig.3.(a)], three different classes are identified to model the data path: the register class, the operator class, and the macro-operator class. According to this selection, the corresponding classes to the identified objects are declared and coded in a proper description language.

The control unit is then specified with a shobi-PN, using as tokens instances of the previously defined classes. Each token is invoked by methods existing in its interface during its travelling along the PN.

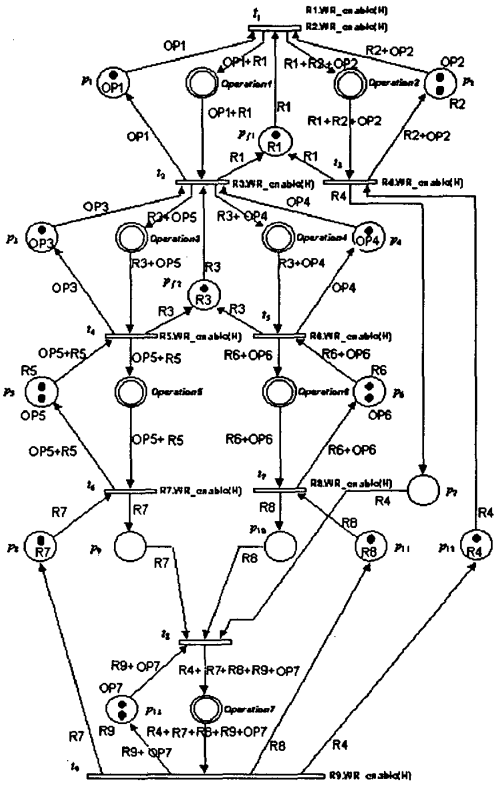

Figure 4. shobi-PN for the pipeline unit.

The pipeline unit can be specified by the shobi-PN in Fig.4. This shobi-PN possesses few closing paths, because it only needs three closing places $\{pf1, pf2, pf3$[Fig.5]$\}$. However, there are four control places $\{p2, p5, p6, p13\}$ with two tokens: one control token and one closing token. *Closing Tokens* do not originate marked places in the associated net, whether *Control Tokens* give origin to control places. This situation implies the initial existence of non-safe control places at the shobi-PN, but this should not be seen as a problem, because one of the tokens is a closing one.

The shobi-PN presents a regular structure, because the pipeline unit data path is also regular, both in structural and behavioral terms. Each operation, basic or complex, can be refined in a lower level of the hierarchy, since macronodes are used at the highest level of the PN to represent operations. For the OP_7 operation, its data flow diagram is refined as presented in Fig.3.(b). Its corresponding macronode is presented in Fig.5. where it is possible to observe the hierarchical decomposition of the macrotoken *OP7* in its sub-tokens *OP7.1*, *OP7.2* and *OP7.3*. It is also considered that the registers *R'* and *R''* are sub-tokens of the macro-token *OP7*.

Registers *R1* and *R3* need the existence of the replica mechanism, since each one is concurrently invoked with itself. In this case, it is mandatory to use replica because the concurrently-invoked methods access the same object's data structure, which does not allow the object to be decomposed in several ones. To ensure the unification of the replica in the top shobi-PN, two closing places {*pf1*, *pf2*} were added to the closing cycles.

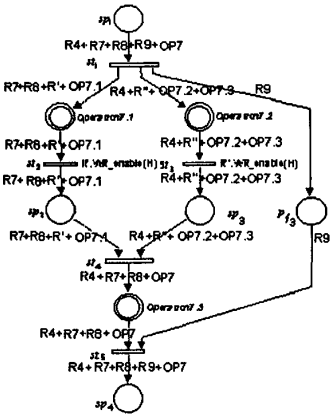

Figure 5. Operation 7 refinement in shobi-PN.

IV. Conclusions

This article shows that the shobi-PN model is an useful modeling tool to specify digital control systems. This model is one of the few known formalisms using object-oriented PNs to specify both the parallel control unit and the data path of a hardware system in an integrated and modular way. The shobi-PN model presents synchronous behavior, object-oriented approaches, and hierarchical mechanisms for specifying digital systems. As a consequence, this new model directly supports hierarchical structures in both the control unit and the data path, allowing the specification of digital parallel control systems in a modular, hierarchical and incremental way.

The use of object-oriented principles (hierarchy, inheritance, modularity, etc.) provides new ways to specify and model digital systems. Since classes are built with data structures and operations, it is relatively easy to apply the object-oriented paradigm in the hardware domain. These techniques also hold promise in the hardware/software co-design.

V. References

[1] T. Murata. Petri Nets: Properties, Analysis and Applications. *Proc. of the IEEE*, 77(4):541-80, Apr/89.

[2] M. Augin, F. Boeri, C. André. New Design using PLAs and Petri Nets. *Int. Symp. on Measurement and Control (MECO'78)*, pp. 864-9, Greece, 1978.

[3] Z. Peng, K. Kuchcinski, B. Lyles. CAMAD: A Unified Data Path / Control Synthesis Environment. *IFIP Conf. on Design Methodologies for VLSI and Computer Architecture*, Pisa, Italy, Sep/88.

[4] K. Bilinski, M. Adamski, J. Saul, E.L. Dagless, Petri-Net-Based Algorithms for Parallel--Controller Synthesis. *IEE Proc.: Computers and Digital Techniques*, 141(6):405-12, Nov/94.

[5] L. Gomes, A. Steiger-Garção, Programmable Controller Design Based on a Synchronized Coloured Petri Net Model and Integrating Fuzzy Reasoning. *16th Int. Conf. on Application and Theory of Petri Nets*, Torino, Italy, Jun/95.

[6] S. Swamy, A. Molin, B.M. Covnot. OO-VHDL: Object-Oriented Extensions to VHDL. *IEEE Computer*, pp. 18-26, Oct/95.

[7] K. Agsteiner, D. Monjau, S. Schulze. Object-Oriented High-Level Modelling of System Components for the Generation of VHDL. *EURO-DAC'95 with EURO-VHDL'95*, pp. 436-41, Brighton, UK, Sep/95.

[8] R.J. Machado. Hierarchy in Object-Oriented Petri Nets for the Specification of Digital Systems (in Portuguese). M.Sc. thesis, Dep. Informatics, U.Minho, Braga, Portugal, Nov/96.

[9] A. Pina, J.M. Fernandes, R.J. Machado. Genetic regulatory mechanisms by means of Extended Interactive Petri Nets. *IEEE Int. Conf. on Systems, Man, and Cybernetics (SMC'97)*, Orlando, Florida, USA, Oct/97.

[10] J.M. Fernandes, A.M. Pina, A.J. Proença. Concurrent Execution of Petri Nets based on Agents. *1st Workshop on Object-Oriented Programming and Models of Concurrency*, Torino, Italy, Jun/95.

[11] J.M. Fernandes, M. Adamski, A.J. Proença. VHDL Generation from Hierarchical Petri Net Specifications of Parallel Controller. *IEE Proc.: Computers and Digital Techniques*, 144(2):127-37, Mar/97.

[12] R. David, H. Alla. *Petri Nets & Grafcet: Tools for Modelling Discrete Event Systems.* Prentice-Hall, UK, 1992.

[13] R. Fehling. A Concept of Hierarchical Petri Nets with Building Blocks. *Advances in Petri Nets 1993*, vol. 674 of LNCS, pp. 148-68. Springer-Verlag, 1993.

[14] K. Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*, vol. I. Springer-Verlag, Berlin, 1992.

[15] C. Lakos. The Object Orientation in Object Petri Nets. *1st Workshop on Object-Oriented Programming and Models of Concurrency*, Torino, Italy, Jun/95.

[16] S. Kumar, J. Aylor, B. Johnson, W. Wulf. Object-Oriented Techniques in Hardware Design. *IEEE Computer*, pp. 64-70, Jun/94.

[17] R.J. Machado, J.M. Fernandes, A.J. Proença. SOFHIA: A CAD Environment to Design Digital Control Systems. *XIII IFIP Conf. on Computer Hardware Description Languages and Their Applications (CHDL'97)*, Toledo, Spain, pp. 86-8, Apr/97.

[18] R.J. Machado, J.M. Fernandes, A.J. Proença. An Object-Oriented Model for Rapid Prototyping of Data Path/Control Systems - A Case Study. *9th IFAC Symp. on Information Control in Manufacturing (INCOM'98)*, Nancy and Metz, France, Jun/98. (accepted for publication)

[19] R.J. Machado, J.M. Fernandes, A.J. Proença. Specification of Industrial Digital Controllers with Object-Oriented Petri Nets. *IEEE Int. Symp. on Industrial Electronics (ISIE'97)*, Guimarães, Portugal, Jul/97.

[20] J. Tomé. *Control of Digital Systems* (in Portuguese). Colecção Informática e Computadores. Editorial Presença/INESC, Lisbon, Portugal, 1989.