Hardware Design and Petri Nets

A workshop within the 19th International Conference on Applications and Theory of Petri Nets, Lisbon, Portugal, Tuesday, June 23 1998

Title:
An Evolutionary Approach to the Use of Petri Net based Models: From Parallel Controllers to HW/SW Co-Design

Authors and affiliation:
Ricardo Jorge Machado, João Miguel Fernandes, António Joaquim Esteves, Henrique Dinis Santos
Dept. de Informática, Universidade do Minho

Mailing address, phone and fax numbers, and e-mail address:

Address Dep. of Informatics
School of Engineering
University of Minho
4709 Braga codex
PORTUGAL
Phone: 351-53-604470
Fax: 351-53-604471
Email: rmac@di.uminho.pt

Abstract:

The main purpose of this article is to present how Petri Nets (PNs) have been used for hardware design at our research laboratory. We describe the use of PN models to specify synchronous parallel controllers and how PN specifications can be extended to include the behavioural description of the data path, by using object-oriented concepts. Some hierarchical mechanisms which deal with the specification of complex digital systems are highlighted. It is described a design flow that includes, among others, the automatic generation of VHDL code to synthesize the control unit of the system. The use of PNs as part of a multiple-view model within an object-oriented methodology for hardware/software codesign is debated. The EDgAR-2 platform is considered as the reconfigurable target architecture for implementing the systems and its main characteristics are shown.

Keywords:

Petri Nets, Hardware Design, Reconfigurable Architecture, HW/SW Co-Design

An Evolutionary Approach to the Use of Petri Net based Models: From Parallel Controllers to HW/SW Co-Design

Ricardo J. Machado, João M. Fernandes,
António J. Esteves, Henrique D. Santos
Dept. of Informatics, School of Engineering, University of Minho
Braga, Portugal

Abstract *The main purpose of this article is to present how Petri Nets (PNs) have been used for hardware design at our research laboratory. We describe the use of PN models to specify synchronous parallel controllers and how PN specifications can be extended to include the behavioural description of the data path, by using object-oriented concepts. Some hierarchical mechanisms which deal with the specification of complex digital systems are highlighted. It is described a design flow that includes, among others, the automatic generation of VHDL code to synthesize the control unit of the system. The use of PNs as part of a multiple-view model within an object-oriented methodology for hardware/software codesign is debated. The EDgAR-2 platform is considered as the reconfigurable target architecture for implementing the systems and its main characteristics are shown.*

Keywords: Petri Nets, Hardware Design, Reconfigurable Architecture, HW/SW Co-Design

1 Introduction

The control unit and the data path must both be considered by the specification model and the CAD environment, in order to fully specify a digital control system. The behaviour of the control unit of a digital system is usually described with an Finite State Machine (FSM). Whenever the system functionality presents concurrent activities, the specification of the sistem becomes more problematic and awkward, if we only use FSM-based techniques.

To specify a parallel controller using FSM techniques, there are, at least, two alternatives: (1) serially-linked controllers can be obtained by identifying sub-routines in the specification, or (2) concurrently-linked controllers should be connected with semaphore bits or common lines [1]. These solutions are generally awkward to apply, and can result in inefficient implementations due to the pre-partitioning, which limits the concurrency to the number of FSMs used. It is also hard to verify for parallel synchronization problems, such as deadlocks or multiple-sourcing.

Among the existing modelling paradigms, the PN-based one allows an easy specification of cooperative subsystems [2]. PNs are associated with a graphical notation, which is easy to understand and a system modelled with a PN might benefict from a mathematical theory to formally check its properties [3].

2 Parallel Controllers

A development methodology for digital systems must provide tools for system specification, modelling and implementation. System specification includes the description of the expected behaviour (functionality) of the digital system. Modelling involves constructing a mathematical formalism embodying the specified system behaviour. This formalism can be manipulated and analysed to determine properties of the system which are not necessarily apparent from the initial problem statement, usually written in a natural language, such as english. The modelling formalism may also be adopted for the system specification, when there is a close correspondence between the system model and its specification.

Several researchers have shown that PNs are a powerful formalism to model the behaviour of parallel systems, namely, parallel digital controllers. A marking of the PN is equivalent to a global state of the modelled system (node in the reachability graph) and a change of the marking corresponds to a state transition (edge in the reachability graph). A detailed analysis of the model, based on a set of well established methods, allows the detection of a large number of design errors prior to the system implementation.

Several types of PNs were proposed to model digital systems, either by imposing restrictions to a basic formalism, or by adding extensions to it. PN-based controllers can be best modelled by safe PNs, which can be viewed as a natural extension to FSMs, providing an easy migration path from FSM to PN-based specifications. To effectively model parallel controllers using safe Place/Transition nets [2], the following modifications were proposed [4]: (1) logic expressions are assigned to transitions (the guards); (2) Moore type output signals are associated to places, while Mealy type output signals are related to transitions, to represent the controller actions; (3) transitions firing are synchronized with the active edge of a (global) clock; and (4) enabling and inhibitor arcs are supported.

The resulting PN type is called Synchronous Interpreted PN (SIPN), which can be used to specify the control unit of synchronous digital systems. Some PN formalisms, namely STGs (Signal Transition Graphs) [5], were also proposed to specify asynchronous digital circuits [6]. To execute an SIPN, all the enabled transitions at a given moment wait for a clock pulse and then all fire to produce a new marking. Fig. 1 presents an SIPN example, where `xi` are input signals and `yi` are output signals.

A software framework was developed at our laboratory [4], to accept an SIPN-based controller specification, written in the ConPar language, in order to validate the properties of the controller and to allow the PN model animation. Validation of

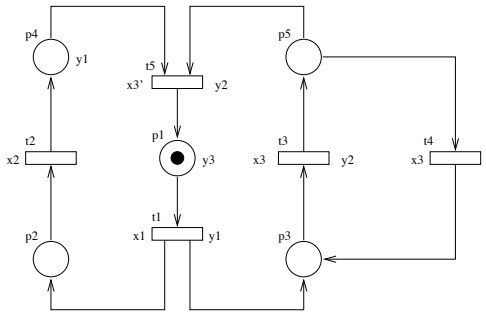

Figure 1. An SIPN specification of a controller.

models is important if they are to evolve into implementations, so a compiler was also included in the framework to generate the corresponding VHDL code [7]. This code can feed standard ECAD packages for simulation and synthesis purposes.

2.1 The ConPar description language

PNs can also be viewed as formal models for logic rule-based specifications They make the straightforward link between algebraic numerical methods and the symbolic mathematical logic based methods of specification, optimisation, verification and synthesis. The rule-based form of specification can be considered as an alternative textual form of timing diagram description. The causality among signals is explicitly given in terms of local, relevant inputs, outputs and state changes.

The ConPar description language, which is an extension to a previously defined language called PNSF [8], was developed to specify SIPN-based controllers, supporting macroplaces. In ConPar notation, a transition is described as a conditional rule:

<label> **:** <PreConditions> **|-** <PostConditions> **;**

The precondition and postcondition are respectively formed from input and output place symbols. When the preconditions of a rule are satisfied (hold), the postconditions are made true (they will hold). Logical conjunction of all related discrete states is assumed when the precondition contains more than one discrete state symbol.

For example, the transition `t1` in Fig. 1 has input place `p1`, output places `p2` and `p3`, it is guarded by input `x1` and the output signal `y1` is activated when the transition is enabled. In ConPar notation, this transition is described as follows:

```
t1: p1 * x1 |- p2 * p3 * y1;
```

2.2 The VHDL compiler

The ConPar description language corresponds to an intermediate representation that links the SIPN model to the corresponding VHDL description. This transformation (from ConPar to VHDL) can be obtained automatically by using a VHDL compiler already developed.

To obtain an efficient implementation, the PN is directly mapped into boolean equations without explicitly enumerating all possible global states and global state changes [9]. The specification is given in terms of the local states changes (local transitions) and one-hot code state assignment is used [1].

A VHDL textual PN description of parallel controllers was proposed in [1], which describes a VHDL template with `ASSERT` statements to enable the syntatic and semantic correctness of the model to be tested. Experimental results, developed at Inmos in a practical design, achieved a 50% area reduction and a 40% speed improvement over the best FSM synthesis. In this work, this VHDL template was adopted for automatic code generation.

The VHDL code generated by our compiler is more readable than the one created with the CAMAD approach [10], where the VHDL code is not directly related to the original PN specification. This may cause some implementation inefficiency, since the PN is transformed into an FSM (which is built with the same algorithm as the reachability graph) and then translated into VHDL code using a `CASE` statement inside a `PROCESS`.

Examples on the use of SIPNs, the ConPar language and the CAD envirnonment can be found in [11, 12, 13]. The complete grammar for the ConPar is described in [7]

3 Parallel Controllers + Data Path

The SIPN model was developed, aiming just the specification of the control part of the digital system: the data path of the system can not be described with the mechanisms available on the model. In some situations, this is considered to be a severe limitation, since it does not allow the integrated development of the whole hardware part of the system. For instance, the simulation task may become difficult because the information on the data path may have to be obtained from different simulation environments.

To overcome this limitation, the shobi-PN model [14], which is an extension to the SIPN model, was developed. The shobi-PN model supports hierarchy and allows objects to be used for specifying the data path resources. A full digital system can be specified and tested, following a structured and incremental approach.

The shobi-PN model presents the same characteristics as the SIPN model, in what concerns synchronism and interpretation. It also adds new functionalities by supporting object-oriented modelling approaches and new hierarchical mechanisms,

in both the control unit and the data path. This model embodies concepts present in Synchronous PNs [15], Hierarchical PNs [16], Coloured PNs [17], and Object-Oriented PNs [18].

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 coloured, if SIPN tokens are viewed as uncoloured (the SIPN places are safe). Each token models a structure of the data path.

A node (a transition or a place) invokes the tokens' methods, when the tokens arrive at that node. Nevertheless, 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 colours which indicates the type of objects that are allowed to pass through that arc. This means that, for each data path structure, there is a well-defined path on the PN. 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.

Hierarchy can be introduced in the specifications in two different ways. The control unit is modelled by the PN structure, and to introduce the hierarchy on the controller, 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.

3.1 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 is transformed into an SIPN. This mapping is possible if it ensured that there is a structural compatibility in the control unit representation. Other topics, such as the PN reinitializations and the simultaneity on the invocation of different methods on the same node, are also important for the mapping but they are not considered in this paper: for details please refer to [14].

To ensure the structural compatibility of the control unit representation in the SIPN and shobi-PN models, it is imposed that the skeleton of the shobi-PN is structurally equivalent to a SIPN without reinitializations. The following concepts used for shobi-PNs are introduced: (1) Control Net: set of contiguous nodes and arcs of the shobi-PN that structurally corresponds to the SIPN without reinitilizations; (2) Control Track: path defined by a token in the Control Net; (3) Control Nodes: nodes (places or transitions) of the Control Net; (4) Control Arcs: arcs

of the Control Net; (5) Closing Track: path defined by a token outside the 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 a token in the shobi-PN. It is composed by a Control Track and also, if applicable, by a Closing Track. It can be identified by the tracking of the colour associated with all the arcs of the cycle; (9) Associated Net: SIPN structurally equivalent to the Control Net after the introduction of the reinitilizations for the uncoloured 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.

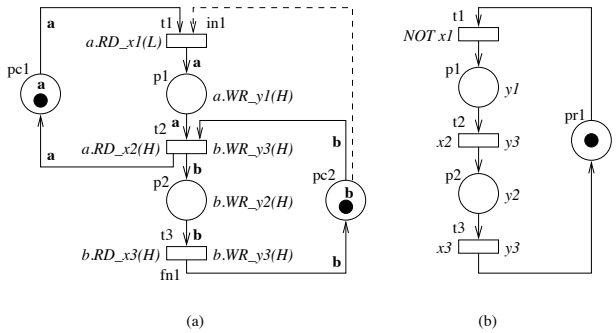

Figure 2. (a) shobi-PN for a simple control sequence and (b) its corresponding SIPN.

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 token a consists of $\{t1, p1, t2\}$, while the control track for token b consists of $\{t2, p2, t3\}$. The closing track for the token a consists of $\{t2, pc1, t1\}$ and for token b consists of $\{t3, pc2, t2\}$. The closing cycles for tokens a and b are $\{t1, p1, t2, pc1\}$ and $\{t2, p2, t3, pc2\}$, respectively.

Mapping the shobi-PN into the associated net is made by transforming k-limited places in safe places. A safe place generates, 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).

3.2 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 innefficiently, solved at the algorithmic level, by changing the PN structure.

This mechanism becomes indispensable when the modelling 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 destroy the tokens' data structures consistency.

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

SOFHIA (Software for Hierarchical Architectures), a CAD environment that covers all the design phases, was also developed to directly support the shobi-PN model [19]. Examples on the use of shobi-PNs, the SOFHIA CAD environment can be found in [20, 21]

4 Hardware/Software Codesign Methodology

A methodology to system development based on the operational approach is essential to guarantee that complex systems can be addressed [22]. The main idea of this approach is based on an executable specification that evolves through transformational refinements to obtain the final implementation.

Object-oriented models (usually multiple view models covering the system's object, dynamic, and functional perspectives) are expected to fully address the above requirements, since they allow the easy refinement of application-domain objects during the whole process. However, there is no established methodology for hardware/software codesign that exploits the benefits of object-oriented system modelling techniques.

MOOSE is a graphical/textual method which is geared towards the development of embedded computer systems and leads to codesign after the system as a whole has been investigated through the use of abstract and executable models [23]. MOOSE uses a multiple view model for specifying the systems: OIDs (Object Interaction Diagrams) which are DFD-like diagrams for functional modelling, Domain Model for object modelling, and STD (State Transition Diagrams) to model the dynamic behaviour of the system.

Our methodology is based on MOOSE, with the following modifications: (1) STDs are replaced by shobi-PNs, which allow an easy handling of concurrency within the dynamic management of the system's objects; (2) the MOOSE paradigm follows essentially the waterfall process model, whilst we proposed a more iterative approach in the definition of the committed and the platform architectures,

since we embed the HDL/HLL generation in the partitioning phase (Fig. 3); (3) an umbrella testbed is included to cover all the development phases, to allow behavioural co-simulation, partitioning co-simulation and implementation co-verification; (4) MOOSE bases partitioning on the designer experience and intuition, while we use an automatic partitioning based on heuristics and an iterative refinement through resources area and time estimation; and (5) a target architecture (EDgAR-2) is used for the hardware parts implementation.

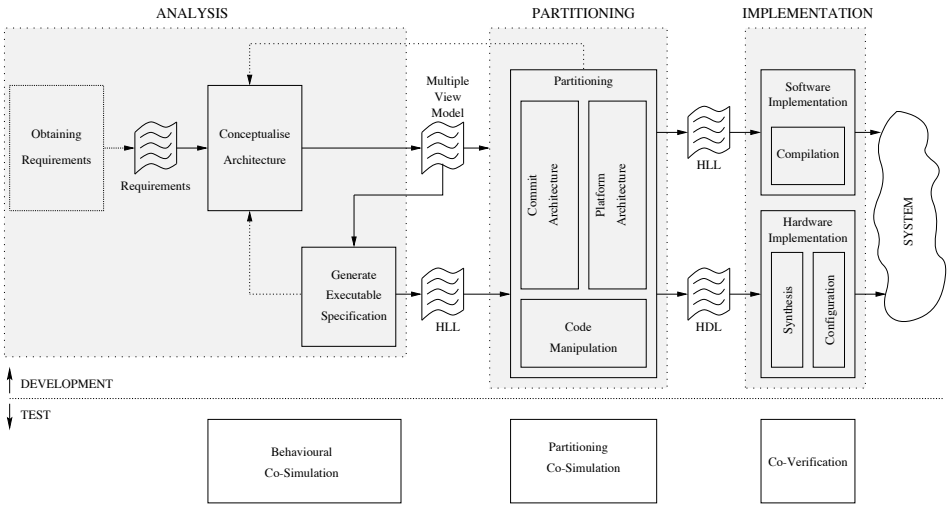

Figure 3. The proposed methodology.

The parallel capabilities of PNs are essentially used during the partitioning activities, which refine the executable specifications towards an equivalent CFSMD model (specified by an HDL) to map it into the hardware reconfigurable components (section 5).

5 EDgAR-2: the reconfigurable target architecture

5.1 The architecture

EDgAR-2 is an FPGA/CPLD based system used for hardware/software codesign and rapid system prototyping. The EDgAR-2 is the successor of the EDgAR. Both systems were developed at the Informatics Department of the University of Minho. The EDgAR was first conceived as part of a stand alone emulation tool for digital systems [24]. The EDgAR-2 is an enhancement architecture, including updated and powerful devices, with In System Programmable (ISP) property and a PCI bus interface, which allow it to be a reconfigurable hardware block in a host PC,

implementing a codesign machine or even a high versatile prototype tool for digital design.

Programmable logic devices (PLDs) can be divided into two classes: one based on coarse grain two level logic blocks, with guaranteed time delay (CPLDs), typically used for control paths or time critical circuits; the other based on fine grain multi level logic blocks (FPGAs), typically used for data paths or space critical circuits [25].

Since each PLD class is suitable to implement complementary parts in a typical digital system, the EDgAR-2 includes devices of both types. The basic architecture element (Fig. 4) is a module composed of an array of 4 Processor Modules (PMs), including each one a control unit and a data path unit. The PMs are interconnected in a linear way with dedicated buses, forming a PM pipeline. Both sides of the array are available to interconnect several EDgAR-2 boards, in a larger array or pipeline. Each PM is implemented with a Xilinx 4010E FPGA [26] — data path — and a 211SP MACH [27] - control path. In what concerns the host PC, each PM is linked to one byte from the 32 bits PCI data bus. In this way, the software module in a codesign realisation can access all the 4 PMs during the same bus cycle, assuming it is possible to manage the common address space in that way. The PCI bus is also used to (re)programme the FPGAs, using the same connectivity, while the CPLDs, for that purpose, use a dedicated independent bus based in a parallel port.

The main EDgAR-2 characteristics can be summarised as follows: (1) fully in system programmable; (2) flexible clock schemes (respecting to frequency and source), with a limited support for asynchronous problems; (3) polling and interrupt mechanism for communication with the host system; (4) PCI burst mode support; (5) pipeline structure; and (6) scalable architecture.

Because EDgAR-2 is full in system programmable, it supports the reconfigurable paradigm. Together with a real time operating system (RTOS) and a custom PCI based computer architecture, the machine obtained is a powerful tool to solve time critical problems. However, during the initial phase and for validation purposes, the prototype operates only under control of the Windows NT (tm) operating system, using a proprietary device driver.

5.2 The computational model

The EdgAR-2 architecture was designed to directly accommodate a finite state machine with data path (FSMD) model [28]. This model is basically an extension to the well known FSM model, with a data path to support a higher level of data abstraction, including primitive variable objects and the associated logical and arithmetic operators. From the structural point of view, and tanking in account the respective properties, the FSMD model is composed of 2 components: an FSM controller and a data path. This (FSM controller, data path) pair can be called a Processor Element (PE) and is directly mapped into an EDgAR's PM. Since the architecture can support several FSMDs, both at the PM level - PE cluster without

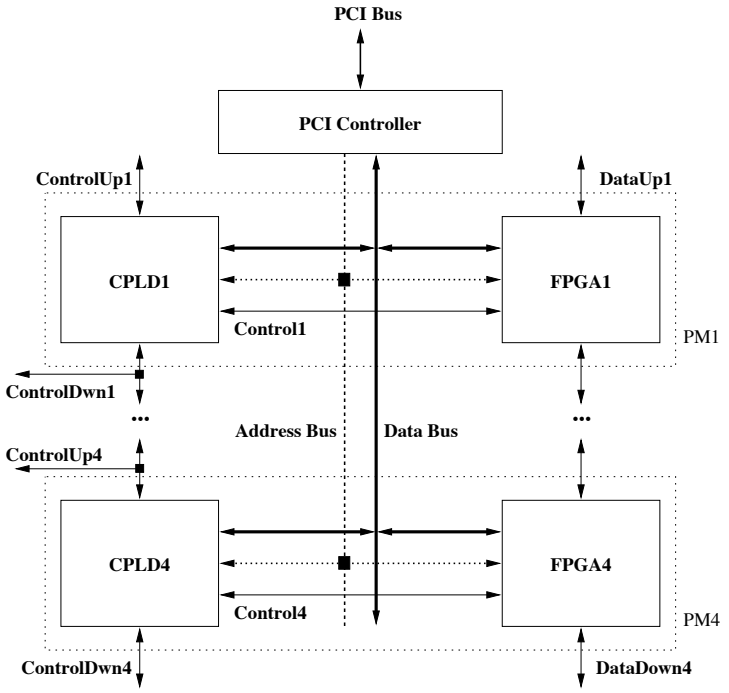

Figure 4. The EDgAR-2 architecture.

connection restrictions - and the board level - one dimensional PE cluster array -, the model naturally includes concurrency, and it can be renamed as concurrent FSMD (CFSMD).

Furthermore, if the development system supports hierarchy, it is possible to work at an architecture level [29] using models such as the Hierarchical Concurrent FSMDs (HCFSMD) or even the Program State Machine (PSM) [30]. These last two models can also be used for modelling at the system level. However, because PLD devices waste a lot of space implementing the (re)programmability feature, the logical resources become critical, which is a restriction to use higher abstraction levels. This issue around the level of abstraction and the resources consumed is essentially the same we find in the software domain concerning the high level languages vs. assembly languages. From this point of view, EDgAR-2 can be compared with a computer with a few kbytes of memory.

From the above discussion, and despite EDgAR-2 model supports higher level of abstractions, the CFSMD model seems to be the best choice, allowing concurrent descriptions at the RTL level. This decision affects the complexity of the development tools, and can impose some restrictions to the implementation of codesign methodologies, having EDgAR-2 as the target hardware.

As stated before, pipeline is also supported by the computational model of the

architecture. At the outer level EDgAR-2 can be defined as a pipeline of PE clusters, with each cluster formed by arbitrary concurrent PEs. This architecture is well suited, for instances, to a production line with several interdependent machines, each with a set of concurrent processes. At the PE level pipeline is not explicit in the architecture, but especially the FPGAs easily allow the addition of pipeline stages to the data path. To what concerns the control path, typically it is not profitable to introduce pipeline stages, because of the faster operation of the 2-level logic CPLDs.

To summarise, the target hardware defines an elementary PE architecture, comprising: (1) an FSM controller block (CPLDs) - supporting registers and two-level logic. This block does not include enough memory to be micro-programmed. The micro-programmability would allow a microprocessor model, which is not needed because the host has a processor. (2) a data path block (FPGAs) - supporting logic and arithmetic operators up to 32 bits wide, data structures with low complexity (up to $4 * 1$kbyte) and the common logic structures (MUXs, decoders, registers, ALUs up to 32). A system is composed of an arbitrary number of concurrent PEs, forming clusters, eventually linked in a pipeline structure.

5.3 The communication model

At a high level of abstraction two communication class mechanisms can be identified: message passing and shared memory. Some of the operations typically found in a message passing mechanism are: point-to-point communication (one to one relation), broadcasting (one to all relation), scatter (one node sends a distinct messages for each node), gather (one node gets a distinct messages from each node) [31]. Shared memory evolves the coordinated access of all processes to a common memory space, requiring the definition of arbitration rules. EDgAR-2 does not include a large common memory, and so it imposes severe restrictions to the implementation of shared memory mechanisms.

At a lower level of abstraction, a physical topology imposes more or less restrictions to the implementation of the above communication mechanisms. Some of the topologies commonly used on systems like the EDgAR-2 are: (1) chain - all nodes, except the two end nodes, communicates with two neighbours, building a chain; (2) ring - a chain topology, where the two end nodes are connected; (3) n-dimensional mesh - every node communicates with its adjacent ones, building a n-dimensional cube; (4) centralised or star - a topology where one single node (concentrator) connects to all the others; (5) hierarchical or tree - the connections between nodes present an hierarchical or tree-like structure; (6) complete or fully connected topology - every node communicates to all the others; and (7) an irregular topology.

These topologies can be explicitly defined in the architecture design, or implemented in software or even trough programmable routing devices, allowing topology changes.

To what concerns EDgAR-2, it is necessary to distinguish the communication

between processes running on hardware, on software and on both hardware and software. The processes running on hardware must conform to the CFSMD computational model proposed. If this model is not restricted, it requires a fully connected topology at the PE level. Both the CPLDs and FPGAs programmable structures are very rich in terms of in chip interconnections, allowing, for most applications, a fully connected topology at the PE cluster level. To implement the same topology at the PM level it is necessary to use larger chips with much more pins, which imposes a higher cost and a higher complexity of the PCB routing.

The analysis of the CFSMD model for several problems - (embedded) control systems domain - showed that only a small number of systems will justify a fully connected topology. So it was decided to provide the architecture of the EDgAR-2 with the following combined topology: a fully connected or centralised topology at the PE cluster level, with a chain (or optionally a ring) topology at the PM or system level.

The processes running on software naturally use the mechanisms supported by the host operating system. This issue has no influence in the EDgAR-2's model and will not be more detailed here. Finally, in a codesign environment, there will be hardware processes communicating with software processes. As described before, the host can access all the PMs through a PCI bus. For complexity reasons and because typically EDgAR-2 works as a coprocessor, the PCI interface does not implement Bus Master operations. This decision implicitly give to the software processes the control over hardware processes. This corresponds to the centralised mechanism defined above. Fig. 5 shows the communication model just described.

Any communication involving EDgAR-2 can be synchronous or asynchronous. The later requires some type of handshake, while the former just requires a signal from the sender to the receiver. Besides, there will be some type of protocol, which is naturally limited by the amount of resources available. Typically, a communication should be restricted to a register transfer with a request/acknowledge handshake. If a more elaborated protocol and/or handshake is required, its specification must be described, making it in an additional PE.

Using the communication topology described, higher level communication models can be implemented. The decision of which model to use will be done by the development methodology (however, it is not difficult to realise a complex communication protocol which could waste most of the EDgAR-2 resources). Some guidelines for this decision are described here. First, communication models based on large memory utilisation are not supported because the lack of memory on the EDgAR-2 board. Second, the broadcasting operation used on the message-passing model is not suitably supported to other levels rather than PE cluster level, because it will require the architecture to allow simultaneous write operations to all devices, which is not the case - to execute a message broadcasting a time overhead is imposed by the required sequence of point-to-point communications. Third, it is necessary to use low complexity communication protocols, in order to achieve a good balance between communication resources and processing resources.

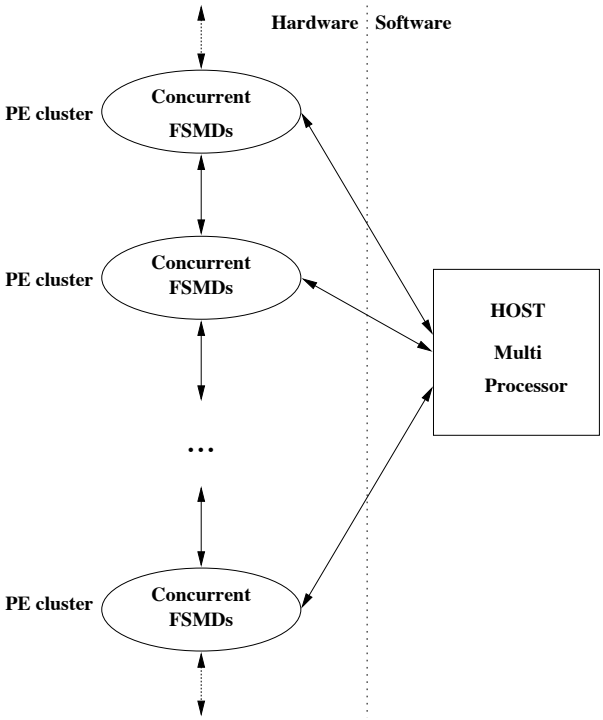

Figure 5. The EDgAR-2 communication model.

6 Conclusions

This paper has presented the evolution of a family of PN-based models to allow the specification of sequential and parallel controllers, with or without data path, and has also shown how is it possible to use them within a wider methodology for hardware/software co-design. PNs are viewed as a fundamental component of a multiple-view model to deal with parallel and concurrent behavioural specification at system-level design. The whole methodology is object-oriented and follows the operational approach to allow the generation of executable specifications and transformational refinements to obtain the hardware and software solutions for system implementation. The co-design target architecture is also presented, which includes a CPLD/FPGA-based ISP board and a host PC with real-time multiprocessing capabilities. The target architecture proposed offers an interesting low cost environment to research on co-design methodologies and on hardware rapid-prototyping for a broad range of time critical problems.

References

1. J. Pardey and M. Bolton. Logic Synthesis of Synchronous Parallel Controllers. *Proc. of the IEEE Intern. Conf. on Computer Design*, pp. 454–7, 1991.
2. T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–80, April 1989.
3. R. Valette, M. Courvoisier, J.M. Bigou, and J. Albukerque. A Petri Net Based Programmable Logic Controller. In *IFIP First Intern. Conf. on Computer Applications in Production and Engineering*, April 1983.
4. J. M. Fernandes. Petri Nets and VHDL in the Specification of Parallel Controllers. MSc thesis, Dept. Informatics, University of Minho, Braga, Portugal, July 1994. (in portuguese)
5. A. Yakovlev, L. Lavagno, and A. Sangiovanni-Vincentelli. A unified signal transition graph model for asynchronous control circuit synthesis. *Formal Methods in System Design*, (9):139–188, 1996.
6. A. Yakovlev, A.M. Koelmans, A. Semenov, and D.J. Kinniment. Modelling, Analysis and Synthesis of Asynchronous Control Circuits Using Petri Nets. *INTEGRATION: the VLSI Journal*, (21):143–70, 1996.
7. J. M. Fernandes, M. Adamski, and A. J. Proença. VHDL Generation from Hierarchical Petri Net Specifications of Parallel Controller. *IEE Proceedings: Computers and Digital Techniques*, 144:127–37, March 1997.
8. T. Kozlowski. Petri-net-based CAD tools for parallel controller synthesis. MSc thesis, University of Bristol, England, March 1993.
9. M. Adamski. Parallel Controller Implementation using Standard PLD Software. In W. R. Moore & W. Luk, editor, *FPGAs*. Abingdon EE&CS Books, 1991.
10. Z. Peng. Digital System Simulation with VHDL in a High-level Synthesis System. *Microprocessing and Microprogramming*, 35:263–70, 1992.
11. J. M. Fernandes and A. J. Proença. Petri Nets in the Specification and Validation of Parallel Controllers. In *1o. Encontro Nacional do Colégio de Engenharia Electrotécnica*, pp. 113–8, Ordem dos Engenheiros, Lisboa, Portugal, May 1994. (in portuguese).
12. J. M. Fernandes, A. M. Pina, and A. J. Proença. Concurrent Execution of Petri Nets based on Agents. In *1st Workshop on Object-Oriented Programming and Models of Concurrency within the XVI Intern. Conf. on Applications and Theory of Petri Nets*, Torino, Italy, June 1995.
13. J. M. Fernandes, A. M. Pina, and A. J. Proença. Simulation and Synthesis of Parallel Controllers based on Petri Nets. In *VII Simpósio Brasileiro de Arquitetura de Computadores — Processamento de Alto Desempenho (SBAC-PAD'95)*, pp. 481–92, Canela, Brazil, July 1995. (in portuguese).
14. R. J. Machado. Hierarchy in Object-Oriented Petri Nets in the Specification of Digital Systems. MSc thesis, Dept. Informatics, University of Minho, Braga, Portugal, November 1996. (in portuguese).
15. R. David and H. Alla. *Petri Nets & Grafcet; Tools for modelling discrete event systems*. Prentice-Hall, UK, 1992.
16. R. Fehling. A Concept of Hierarchical Petri Nets with Building Blocks. In G. Rozenberg, editor, *Advances in Petri Nets 1993*, vol. 674 of *LNCS*, pp. 148–68. Springer-Verlag, 1993.
17. K. Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*, vol. I. Springer-Verlag, Berlin, Germany, 1992.

18. C. Lakos. The Object Orientation in Object Petri Nets. In *1st Workshop on Object-Oriented Programming and Models of Concurrency within the 16th Intern. Conf. on Applications and Theory of Petri Nets*, Torino, Italy, June 1995.
19. R. J. Machado, J. M. Fernandes, and A. J. Proença. SOFHIA: A CAD Environment to Design Digital Control Systems. In C. Delgado Kloos and E. Cerny, editors, *XIII IFIP Conf. on Computer Hardware Description Languages and Their Applications (CHDL'97)*, pp. 86–8, Toledo, Spain, April 1997. Chapman & Hall.
20. R. J. Machado, J. M. Fernandes, and A. J. Proença. Specification of Industrial Digital Controllers with Object-Oriented Petri Nets. In *IEEE Intern. Symp. on Industrial Electronics (ISIE'97)*, Guimarães, Portugal, vol. 1, pp. 78–83, July 1997.
21. R. J. Machado, J. M. Fernandes, and A. J. Proença. An Object-Oriented Model for Rapid Prototyping of Data Path/Control Systems — A Case Study. In *9th IFAC Symp. on Information Control in Manufacturing (INCOM'98)*, Nancy and Metz, France, June 1998. (accepted for publication).
22. P. Zave. The Operational versus the Conventional Approach to Software Development. *Communications of the ACM*, 27(2):104–18, February 1984.
23. D. Morris, G. Evans, P. Green, and C. Theaker. *Object-Oriented Computer Systems Engineering*. Applied Computing. Springer-Verlag, London, U.K., 1996.
24. A. J. Esteves, J. M. Fernandes, and A. J. Proença. EDgAR: A Platform for Hardware/Software Codesign. In *Embedded System Applications*, Ed. C.Baron, J.-C. Geffroy and G.Motet, Kluwer Academic Publishers, Boston, USA, pp. 19–32, June 1997.
25. H. D. Santos. *Specification Methodologies and Analysis of Digital Systems: development of an APA (GLiTCH) controller*. PhD thesis, Dept. Informatics, University of Minho, Braga, Portugal, July 1996. (in portuguese).
26. Xilinx. *The Programmable Logic Data Book*. Xilinx, 1996.
27. AMD. *MACH 1, 2, 3 and 4 Family Data Book*. Advanced Micro Devices, 1995.
28. D. D. Gajski, N. D. Dutt, and A. C.-H. Wu. *High-Level Synthesis: Introduction to Chip and System Design*. Kluwer Academic Publishers, 3th edition, 1994.
29. D. Gajski and R. Kuhn. Guest's editors introduction: New VLSI Tools. *IEEE Computer*, 16:11–4, 1983.
30. D. Gajski, G. Marchioro, and J. Zhu. *Essential Issues in Codesign*, pp. 1–45. Kluwer Academic Publishers, 1997.
31. Z. Xu and K. Hwang. Modelling Communication Overhead: MPI and MPL Performance on the IBM SP2. *IEEE Parallel & Distributed Technology*, pp. 9–23, Spring 1996.