A Petri Net Meta-Model to Develop Software Components for Embedded Systems

Ricardo J. Machado
Dept. Sistemas de Informação
Escola de Engenharia
Universidade do Minho
Guimarães, Portugal
rmac@dsi.uminho.pt

João M. Fernandes
Dept. Informática
Escola de Engenharia
Universidade do Minho
Braga, Portugal
miguel@di.uminho.pt

Abstract

This paper presents a new Petri net (PN) meta-model, called shobi-PN v2.0, that can be used to specify the dynamic behaviour of concurrent systems, using object-oriented modelling concepts together with a generalised arc set capable of coping with the complexity of the current embedded systems. This new Petri net meta-model can also be used to support a component-based development approach in the design of generic and parametrisable control-oriented software components for embedded systems.

1. Introduction

Embedded systems can be qualified, in their vast majority, as control-dominated systems and usually their dynamic behaviour is specified with a state-oriented model, such as FSMs (Finite State Machines). However, real-time embedded systems are becoming nowadays hugely complex, meaning that a distinct approach is needed. The specification model has to fulfill several requirements, namely support for concurrency, timing constraints, hierarchy, data and control flow, and distributed computations. Thus, for modelling several perspectives of the systems (namely, their data and function views), it is mandatory to consider multiple-view models. It is also commonly agreed that designers can enhance their efficiency and productivity if there exists a common notation that everybody is able to understand. In this context, the authors recommend the use of some UML diagrams to specify and model embedded systems, because it is a notation that covers the most relevant modelling aspects of systems and it is a standard [1].

UML is a general purpose modelling language for specifying, visualising, constructing and documenting the artefacts of software systems, as well as for business modelling and other non-software systems [2]. At the moment UML is an OMG's (Object Management Group) standard for defining and designing software systems, but it is expected to become an ISO standard in the near future [3]. As a consequence of being a standard language, UML is being increasingly accepted as a language in industrial environments. UML is meant to be used universally for the modelling of systems, including automatic control applications with both hardware and software components, so the authors consider it as an adequate alternative for embedded systems. Several researchers have also selected UML as the notation for specifying embedded systems, which confirms the usefulness of UML for this engineering field and indicates that this notation is gaining widespread acceptance and usage within this community [4, 5, 6, 7, 8, 9].

For specifying the systems, the authors are using the main views captured by the following UML diagrams:
(1) *use case diagrams* are utilized to catch the functional aspects of the system as viewed by its users;
(2) *object diagrams* show the static configuration of the system, and the relations among the objects the system is composed of;
(3) *sequence diagrams* display scenarios of common interactions amongst the objects that constitute the system or the actors that interact with it;
(4) *class diagrams* show the information of ready-made components that can be used to construct systems and specify the hierarchical relationships among them.

Additionally, *Petri nets* (*shobi-PN v2.0*) are used to specify the dynamic behaviour of some objects/classes. It is important to note that, contrarily to our proposal, some authors use the UML state-oriented models to specify the algorithm of a specific method of an object, and not the entire life-cycle of that object [10]. Our proposal is different and assumes that the methods of the object are used as inputs or outputs of the PNs.

Although the OMG's Real-time Analysis and Design working group has not come yet with a final proposal for directly incorporating real-time concepts into the UML standard (namely in what concerns the syntax for the OCL language), the authors are using UML for dealing with hard real-time systems. Up to now, timed sequence diagrams and Oblog syntax (the language used by the authors to support the system's specification based on PNs) have been used for the specification of the canonical latency and duration constraints, which are viewed as composites for more accurate categories of timed requirements (for performance and safety constraints specification).

2. Petri Nets meta-models

State models can be specified for the system's components that possess a complex or interesting dynamic behaviour. UML has two different meta-models for this purpose: STATECHARTS and activity diagrams. Although these two meta-models present many important characteristics for reactive systems, namely concurrency and hierarchy, they do not allow an elegant treatment of the data path/plant resources management and the specification of dynamic parallelism. These are two crucial topics for complex, distributed and parallel embedded systems, since different parts of the system may require the simultaneous access to the same resource.

For digital systems and, more specifically, for control embedded systems, the application of Petri nets to the specification of the behavioural view can benefit from several research results. The designer can choose, among several PN meta-models, a specific one intentionally created to deal with the particularities of that kind of systems, like the ones referred in [11, 12, 13].

PNs constitute a mathematical meta-model that can be animated/simulated, formally analysed, and for which several implementation techniques are available (PLC, hardwired, PLD/FPGA, microprocessor-based). In this situation, for replacing UML's statecharts and activity diagrams, an extended PN meta-model, designated *shobi-PN*, is proposed, to specify the reactive behaviour of the system's components.

2.1. The shobi-PN v1.0

The traditional synchronous and interpreted PN meta-model (SIPN) [14] was developed, aiming at just the specification of the control part of the system: the data path/plant of the system can not be described with the mechanisms available on the meta-model. To overcome this limitation, the *shobi-PN v1.0* meta-model, which is an extension to the SIPN model, was developed [15]. The *shobi-PN v1.0* meta-model supports hierarchy and allows objects to be used for specifying the data path/plant resources.

The *shobi-PN v1.0* meta-model presents the same characteristics as the SIPN meta-model, in what concerns synchronism and interpretation, but adds new mechanisms by supporting object-oriented modelling ideas and new hierarchical constructs, in both the control unit and the data path/plant. This meta-model embodies concepts present in Synchronous PNs [16], Hierarchical PNs [17], Coloured PNs [18], and Object-Oriented PNs [19]. In the *shobi-PN v1.0* meta-model, the tokens represent objects that model data path/plant resources. The instance variables represent the information that is processed on the data path/plant and the methods are the interface between the control unit and the data path/plant. Each token models a structure of the data path/plant. A node (a transition or a place) invokes the tokens' methods, when the tokens arrive at that node. Each arc is associated with one or more colours which indicate the types of objects that are allowed to pass through that arc. This means that, for each data path/plant structure, there is a well-defined path on the PN.

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

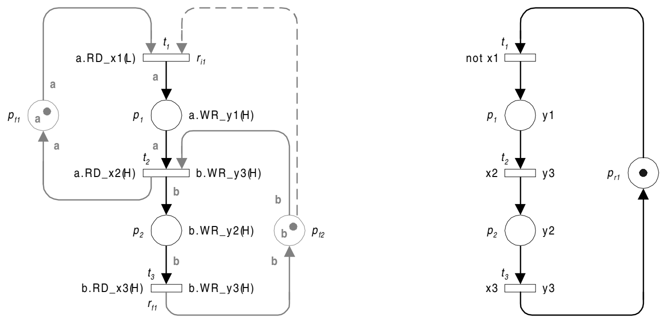


Figure 1. (left) shobi-PN for a simple control sequence and (right) its corresponding SIPN.

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. 1 to specify a simple control sequence, with two objects/tokens to model two structures of the data path.

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.

Hierarchy can be introduced in the specifications in two different ways:

(1) the control unit is modelled by the PN structure, and to introduce the hierarchy on the controller, macronodes (representing sub-PNs) may be used;

(2) the data path/plant 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.

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 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/plant structure, if the mechanism does not destroy the tokens' data structures consistency.

This *shobi-PN v1.0* meta-model has been exhaustively used in several application domains of medium complexity: industrial controllers [20], communication interfaces [21], and micro-architecture of processors [22].

2.2. The shobi-PN v2.0

The use of *shobi-PN v1.0* meta-model to specify the behaviour an industrial controller (the HIDRO production lines [23]) has revealed some semantic fragilities of that modelling approach, namely when it is mandatory to assure:

(1) the violation of levels of structural hierarchy by the introduction of tokens/objects in arbitrary zones of the PNs (this is very useful when, for some specific objects, it is crucial to bypass some levels of the controller's hierarchy);

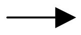
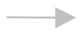
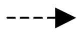
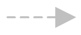
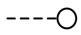
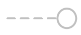
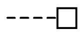
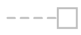
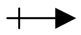
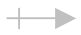
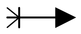
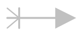
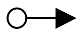
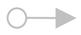
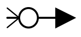
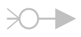
(2) the creation and destruction of objects for momentary reference of objects that are external to the system;

(3) the manipulation of the original (genuine) objects and not the eventual replica that the dynamic execution of the PNs can create (this is vital to deal with critical regions in the control of multiple accesses to shared resources - for instance, the elevators in the HIDRO lines case study).

To solve this three kinds of detected problems, the authors have extended the *shobi-PN v1.0* meta-model (which has originated the *shobi-PN v2.0* meta-model) by defining:

(1) a *generalised arc set* (*GAS*) which allows the use of 16 different types of arcs, each one with specific

Table 1. Generalised arc set of *shobi-PN v2.0* meta-model.

Arc	*Description*
	*control arc:* Arc that transports objects (and replica) and originates arcs in the SIPN meta-model.
	*closing arc:* Arc that transports objects (and replica) and does not originate arcs in the SIPN meta-model.
	*enabling control arc:* Control arc that does not transport objects (neither replica), it is solely used to enable transitions; it originates enabling arcs in the SIPN meta-model.
	*enabling closing arc:* Closing arc that does not transport objects (neither replica), it is solely used to enable transitions; it does not originate enabling arcs in the SIPN meta-model.
	*inhibitor control arc:* Control arc that does not transport objects (neither replica), it is solely used to inhibit transitions; it originates inhibitor arcs in the SIPN meta-model.
	*inhibitor closing arc:* Closing arc that does not transport objects (neither replica), it is solely used to inhibit transitions; it does not originate inhibitor arcs in the SIPN meta-model.
	*destructor control arc:* Control arc that does not transport objects (neither replica), it is solely used to destroy replica; it does not originate arcs in the SIPN meta-model.
	*destructor closing arc:* Closing arc that does not transport objects (neither replica), it is solely used to destroy replica; it does not originate arcs in the SIPN meta-model.
	*synchronous hierarchical control arc:* Control arc that transports pre-existent objects and it can violate levels of structural hierarchy; it does not transport replica.
	*synchronous hierarchical closing arc:* Closing arc that transports pre-existent objects and it can violate levels of structural hierarchy; it does not transport replica.
	*asynchronous hierarchical control arc:* Control arc that transports replica of pre-existent objects and it can violate levels of structural hierarchy.
	*asynchronous hierarchical closing arc:* Closing arc that transports replica of pre-existent objects and it can violate levels of structural hierarchy.
	*initialising synchronous hierarchical control arc:* Control arc that transports not yet existent objects, creating or destroying them; it can violate levels of structural hierarchy.
	*initialising synchronous hierarchical closing arc:* Closing arc that transports not yet existent objects, creating or destroying them; it can violate levels of structural hierarchy.
	*initialising asynchronous hierarchical control arc:* Control arc that transports replica of not yet existent objects, creating or destroying them; it can violate levels of structural hierarchy.
	*initialising asynchronous hierarchical closing arc:* Closing arc that transports replica of not yet existent objects, creating or destroying them; it can violate levels of structural hierarchy.

syntactic and semantic properties within the *shobi-PN v2.0* meta-model (table 1);
(2) the concept of *asynchronous macro-transition* (*AMT*) as an auxiliary mechanism to the *GAS*, to solve the specific problem of the violations of the structural hierarchy's levels.

In the 16 kinds of arcs there are some dual/complementary arc subsets.

2.2.1. Control vs. Closing. For each type of control arc, there exists a dual corresponding closing arc. Any

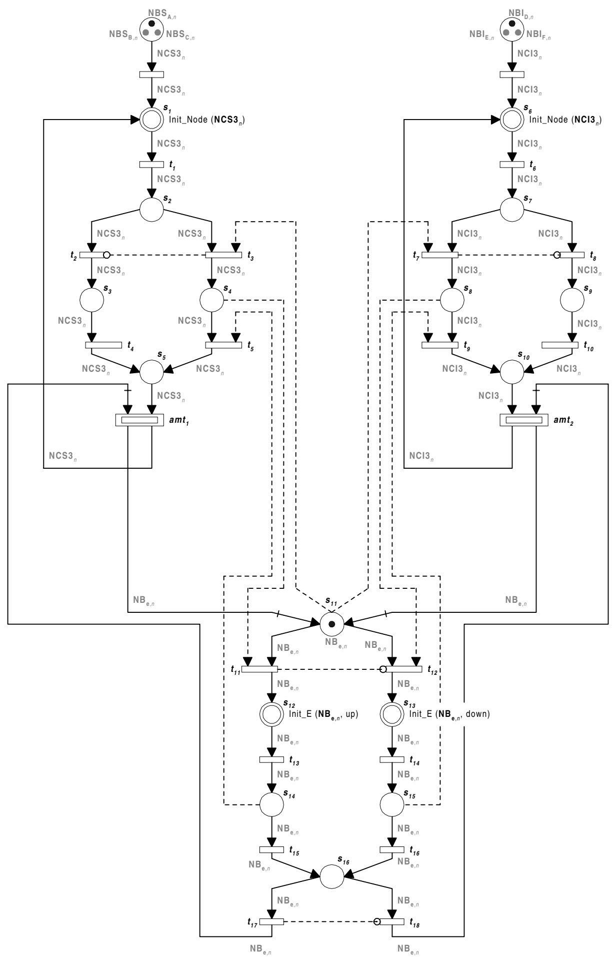


Figure 2. Upper level behaviour of an industrial controller specified with a shobi-PN v2.0.

control arc belongs to the skeleton (control path) of the net, which means that it must remain in the associated net for allowing the controller to be properly synthesised.

On the contrary, since the closing arcs do not directly contribute to the skeleton structure, they are withdrawn from the associated net. This duality allows the system's resources (data path resources) to be managed in an easy way, since the closing cycles may benefit from the use of as many types of closing arcs as types of control arcs.

2.2.2. Hierarchical vs. Non-hierarchical. All hierarchical arcs can violate the hierarchical levels of the net's structure. This hierarchical arcs' capacity permits the use of several modelling mechanisms, without reducing the flexibility in the communication with external objects among any level. Thus, the hierarchical levels can be violated (in a judicious and controlled way), whenever it is not advantageous to maintain those levels for interconnecting objects.

The need to allow some types of arcs to possess this capacity arises from the inexistence of equivalence between the controllers hierarchical levels and the data path ones. A non-hierarchical arc can not violate the hierarchy of the net's structure, i.e., it is unable to represent a flow transfer (transportation of objects or their replica), transversally to the hierarchical levels, since its scope is restricted to a single level.

2.2.3. Synchronous vs. Asynchronous. For each type of hierarchical synchronous arc, there exists a dual corresponding hierarchical asynchronous arc. The synchronous arcs transfer control flow when the *shobi-PN* meta-model firing rules are checked. Contrarily, the asynchronous arcs do not need to

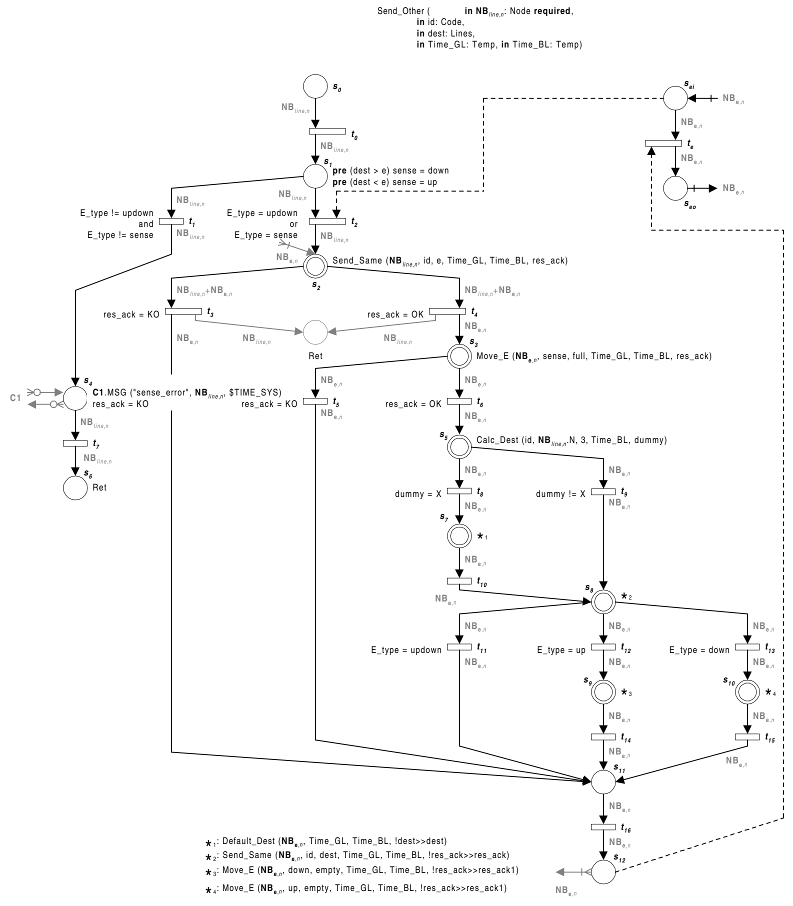

Send_Other (in NB_line,n: Node required,
in id: Code,
in dest: Lines,
in Time_GL: Temp, in Time_BL: Temp)
pre (dest > e) sense = down
pre (dest < e) sense = up
E_type != updown and E_type != sense
E_type = updown or E_type = sense
Send_Same (NB_line,n, id, e, Time_GL, Time_BL, res_ack)
res_ack = KO
res_ack = OK
Ret
Move_E (NB_e,n, sense, full, Time_GL, Time_BL, res_ack)
C1.MSG ("sense_error", NB_line,n, $TIME_SYS)
C1
Calc_Dest (id, NB_line,n.N, 3, Time_BL, dummy)
dummy = X
dummy != X
E_type = updown
E_type = up
E_type = down
*1: Default_Dest (NB_e,n, Time_GL, Time_BL, !dest>>dest)
*2: Send_Same (NB_e,n, id, dest, Time_GL, Time_BL, !res_ack>>res_ack)
*3: Move_E (NB_e,n, down, empty, Time_GL, Time_BL, !res_ack>>res_ack1)
*4: Move_E (NB_e,n, up, empty, Time_GL, Time_BL, !res_ack>>res_ack1)

Figure 3. Middle level behaviour of an industrial controller specified with a shobi-PN v2.0.

transfer flow in a synchronous way with the net evolution. This is the main reason why asynchronous arcs are only allowed to transport replicas and not objects. This limitation of the asynchronous arcs ensures the innocuity of its asynchronism. Even if the potential loss of control during the net evolution is taken into consideration (due to a subversive use of asynchronous arcs), the introduction of this asynchronism in the *shobi-PN v2.0* meta-model results from the need to complement the hierarchical levels violation mechanism, whenever the communication with the external objects is made without knowing its internal state.

The synchronous arcs are allowed to transport objects (and never replicas) as a way to support a precise control over the critical regions, on the management of multiple accesses of shared resources. This limitation of the synchronous arcs is crucial, due to the dynamic and parallel nature of PNs.

2.2.4. Initialising vs Non-initialising. Due to the asynchronism of invocations and also to the hierarchical levels violation, there is frequently the need to manipulate, at the arc level, objects not previously declared. Thus, the initialising arcs possess the ability to create or destroy an object (or the corresponding replica), respectively, immediately before or after its effective transport.

Since the *shobi-PNs* are conservative in relation to the data path objects, the usage of this type of arcs is restricted to the manipulation of objects external to the net under consideration.

In fig. 2 and fig. 3 there are two *shobi-PNs v2.0* that partially specify the behaviour of an industrial embedded controller.

The upper net (fig. 2) specifies the access control to the system's critical resources (the object $\boldsymbol{NB_{e,n}}$), while the middle net (fig. 3) specifies the algorithmic life-cycle execution of the $\boldsymbol{NB_{e,n}}$ object. These

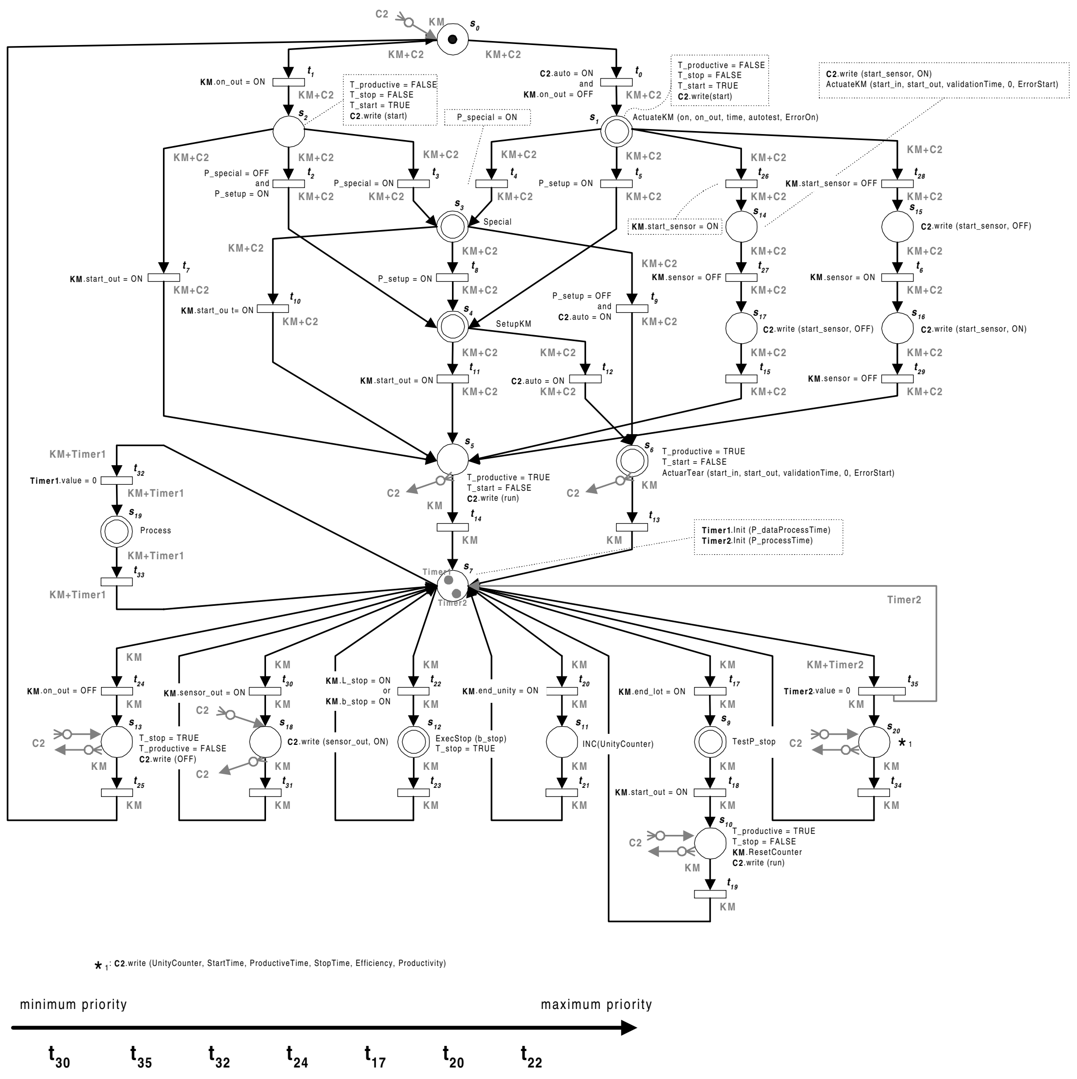


Figure 4. Generatrix shobi-PN v2.0 of an industrial knitting machine embedded supervisor.

shobi-PNs v2.0 are related with each other by the *AMTs* *amt1* and *amt2* of the upper net that allow the injection of the object $\boldsymbol{NB_{e,n}}$, using synchronous hierarchical control arcs, into the place $\boldsymbol{s_{ei}}$ of the middle net.

The *AMTs* allow a true hierarchical level violation, since the middle net is not the immediate sub-net of the upper net. The $\boldsymbol{NB_{e,n}}$ object located in place $\boldsymbol{s_{ei}}$ can be used inside the middle net when it is initialised with a $\boldsymbol{NB_{line,n}}$ object. In this situation, the replica of the $\boldsymbol{NB_{e,n}}$ object located in place $\boldsymbol{s_{ei}}$ is injected in the place $\boldsymbol{s_2}$ of the middle net by using an asynchronous hierarchical closing arc and the $\boldsymbol{NB_{e,n}}$ object is located in place $\boldsymbol{s_{eo}}$ by the firing of the $\boldsymbol{t_e}$ transition. When the replica of the $\boldsymbol{NB_{e,n}}$ object reaches the place $\boldsymbol{s_{12}}$ of the middle net, the $\boldsymbol{NB_{e,n}}$ object leaves the middle net and return to the upper net to place $\boldsymbol{s_{11}}$, passing throughout the *AMT*.

3. Software Components

Component based design (CBD) is strongly based on the reuse of previously designed components, avoiding the design from scratch of each system part. The use of the CBD approach demands:

(1) a cautious selection of the components that should be used in a specific situation, to assure a minimal integration and parameterisation effort;

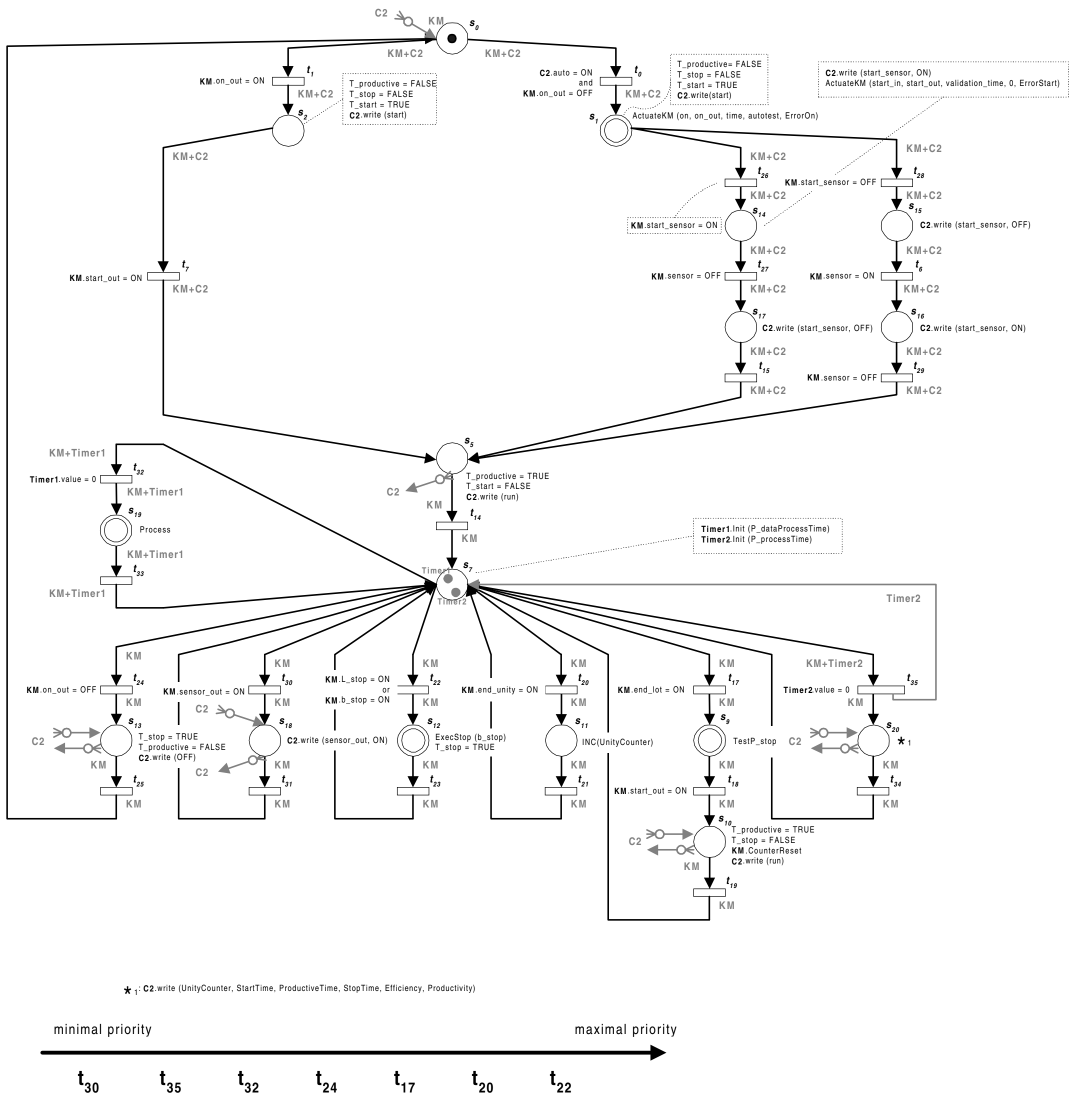


Figure 5. Parameterised shobi-PN v2.0 of an industrial knitting machine embedded supervisor.

(2) a correct component parameterisation to commit it to the final context;
(3) a careful interconnection with the other components to obtain the desired system.

One of the problems that arise from the CBD approach is related to the specification of a parametric and generic component capable of being instantiated within a set of well-characterised behavioural class. To solve this problem, in the context of the design of generic components for industrial embedded systems, the authors have used the *shobi-PN v2.0* meta-model to construct generatrix *shobi-PNs v2.0*. These generatrix PNs are viewed as architectural generalisations of all the possible *shobi-PNs v2.0* that can be obtained after parameterisation. Each generatrix *shobi-PN v2.0* has several Boolean parameters in transitions' propositions that allow a structural redefinition of the PN, to obtain a parameterised *shobi-PN v2.0*. Fig. 4 shows a generatrix *shobi-PN v2.0* of an industrial knitting machine embedded supervisor system with five parameters (*P_special*, *P_setup*, *P_program*, *P_wait* and *P_stop*). This means that, in a specific context, each parameter can be forced to a specific Boolean value imposing the elimination or preservation of control paths and generating a parameterised PN with a proper behaviour. Fig. 5 depicts a parameterised *shobi-PN v2.0* generated by the generatrix *shobi-PN v2.0* of fig. 4, to synthesise a controller to a specific knitting machine (the *LONATI 409/421*).

4. Oblog CASE Tool

From a pragmatic point of view, a methodology for developing systems can only be useful for its users if there exist tools supporting the development tasks. The authors' methodology is an on-going project, but there are already some tools available for the developers.

A software environment was developed to allow animation/simulation by generating UML sequence diagrams. These diagrams are built from the system specification and allow the designers to manually compare them with similar diagrams constructed previously in co-operation with the system's customers. This eases the methodology to follow the operational approach [24], which permits the customers to validate their requirements directly from the system specification (i.e. without having to fully develop a system prototype or even the system itself). If some errors are detected, the system specification can be modified prior to the implementation phase, which greatly reduces the development costs and increases the system's correctness.

Other tools were also developed, namely graphical editors (for specifying the systems) and compilers (for automatically generating *C* code for the *MCS-51* compatible microprocessors). There is already available a preliminary version of this tool-set, which is the successor of the tool previously developed by the authors [25]. It supports directly the *shobi-PN v2.0* meta-model, using the Oblog language/tool (www.oblog.com) as the implementation environment.

The Oblog tool allows the construction of object-oriented specifications and supports the automatic code generation using the RDL scripting language. To allow the use of the Oblog environment for supporting the PN-based embedded systems design, the authors have defined a collection of rules to overcome the inherent "limitations" (characteristics) of the Oblog language meta-model:
(1) Oblog does not directly allow a true state oriented approach, which has demanded the emulation of the state orientation approach over the Oblog meta-model;
(2) Oblog is essentially asynchronous with some points of synchronism, while the *shobi-PN v2.0* meta-model is mainly synchronous with some mechanisms to support asynchronous behaviours;
(3) the macronodes' structural hierarchy is not directly supported by Oblog, which has justified the construction of aggregations of sub-controllers to allow the definition of Oblog behavioural hierarchies equivalent to *shobi-PN* structural ones.

The referred rules are organised into three main groups:
(1) *Rules for the definition of an abstract class of parallel controllers*. This set of rules is used to specify abstract classes of parallel controllers in an object-oriented language, from which the concrete controllers (instances) can inherit the fundamental behaviour and refine the properties that must be specialised.
(2) *Rules for the state-orientation emulation*. This set of rules is used to emulate the state orientation approach, which allows an easy description of *shobi-PNs v2.0* in an object-oriented language not supporting the state orientation approach.
(3) *Rules for the construction of a collection of sub-machines*. This set of rules is used to construct the entire parallel controller in a hierarchical and incremental way directly supporting the *shobi-PNs v2.0* structure.

To obtain a true operational approach, in the use of Oblog to model *shobi-PN v2.0*-based software components for embedded systems, the authors have developed a simulation environment. This environment allows the designer to interact with the Oblog system's model and automatically generates sequence diagrams showing the interaction between the internal components of the system.

5. Conclusions

This paper presents the general characteristics of a PN meta-model, designated *shobi-PN v2.0*, that supports the design of complex embedded systems. The *shobi-PN* models are used within a multiple-view specification methodology that uses several UML diagrams for capturing different modelling perspectives of the systems. The authors defend the use of *shobi-PN* behavioural specifications, instead of using the UML *STATECHARTS* and activity diagrams, for specifying the objects' dynamic behaviour. In this context, the *shobi-PN v2.0* meta-model was generally explained, in what concerns the usefulness of its *GAS* and *AMT* concepts for allowing the easy violation of hierarchical levels, the creation and destruction of objects for momentary reference and the manipulation of the original objects and not their replicas. The *shobi-PN v2.0* meta-model can be used in a component-based development approach in the specification of software components as architectural generalisations of all the possible *shobi-PNs v2.0* that can be obtained after parameterisation. This paper also refers the existence of the software environment the authors have developed to directly support the design of complex embedded controllers.

6. References

[1] R.J. Machado, J.M. Fernandes, and H.D. Santos, "A Methodology for Complex Embedded Systems Design: Petri Nets within a UML Approach", *Architecture and Design of Distributed Embedded Systems*, B. Kleinjohann (*editor*), Kluwer A.P., Boston, U.S.A., May, 2001 (to be published).

[2] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*, Addison-Wesley, 1999.

[3] C. Kobryn, "UML 2001: A Standardization Odyssey." *Communications of the ACM*, no. 42. vol. 10, Oct., 1999, pp. 29-37.

[4] B. P. Douglass, *Real-Time UML: Developing Efficient Objects for Embedded Systems*, Addison-Wesley, 1998.

[5] A. Lyons, *UML for Real-Time Overview*, ObjecTime Limited, Apr., 1998.

[6] A. Lanusse, S. Gérard, and F. Terrier, "Real-Time Modeling with UML: The ACCORD Approach", *International Workshop on the Unified Modeling Language: Beyond the Notation*, Mulhouse, France, 1998.

[7] M. McLaughin, and A. Moore, "Real-Time Extensions to UML", *Dr. Dobb's Journal*, no. 292, Dec., 1998, pp. 82-93.

[8] L. Kabous, and W. Nebel, "Modeling Hard Real-Time Systems with UML: The OOHARTS Approach", *2nd International Conference on the Unified Modeling Language*, Fort Collins, U.S.A., Oct., 1999.

[9] R. Jigorea, S. Manolache, P. Eles, and Z. Peng, "Modeling of Real-Time Embedded Systems in an Object-Oriented Design Environment with UML", *3rd IEEE Int. Symp. on Object-Oriented Real-Time, Distributed Computing*, Newport Beach, U.S.A., March, 2000, pp. 210-203.

[10] W. Wolf, *Computers as Components: Principles of Embedded Computing System Design*, Morgan Kaufmann, 2001.

[11] M. Silva, "Logical Controllers", *IFAC World Congress*, vol. II, Milan, Italy, Nov., 1989, pp. 157-166.

[12] A. Semenov, A.M. Koelmans, L. Lloyd, and A. Yakovlev, "Designing an Asynchronous Processor using Petri Nets", *IEEE Micro*, no. 17, vol. 2, Mar./Apr., 1997, pp. 54-64.

[13] M. Sgroi, L. Lavagno, Y. Watanabe, and A. Sangiovanni-Vincentelli, "Synthesis of Embedded Software Using Free-Choice Petri Nets", In *Proc. of Design Automation Conference*, New Orleans, USA, June, 1999.

[14] J.M. Fernandes, M. Adamski, and A.J. Proença, "VHDL Generation from Hierarchical Petri Net Specifications of Parallel Controllers." *IEE Proc.: Computers and Digital Techniques*, no. 144, vol. 2, Mar., 1997, pp. 127-137.

[15] R.J. Machado, J.M. Fernandes, and H.D. Santos, "An Evolutionary Approach to the Use of Petri Net based Models: From Parallel Controllers to HW/SW Co-Design", *Hardware Design and Petri Nets*, chapter 11, A. Yakovlev, L. Gomes and L. Lavagno (*editors*), Kluwer A.P., 2000, pp. 205-222.

[16] R. David, and H. Alla, *Petri Nets & GRAFCET: Tools for Modelling Discrete Event Systems*, Prentice-Hall, 1992.

[17] R. Fehling, "A Concept of Hierarchical Petri Nets with Building Blocks", *Advances in Petri Nets 1993*, LNCS, vol. 674, Springer-Verlag, 1993, pp. 148-168.

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

[19] C. Lakos, "The Object Orientation in Object Petri Nets", *1st Workshop on Object-Oriented Programming and Models of Concurrency*, Torino, Italy, 1995.

[20] R.J. Machado, J.M. Fernandes, and A.J. Proença, "Specification of Industrial Digital Controllers with Object-Oriented Petri Nets", *IEEE Int. Symp. on Industrial Electronics*, vol. I, Guimarães, Portugal, July, 1997, pp. 78-83.

[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", *9th IFAC/IFIP Symp. on Information Control in Manufacturing*, vol. II, Nancy & Metz, France, June, 1998, pp. 269-274.

[22] R.J. Machado, J.M. Fernandes, and A.J. Proença,. "Hierarchical Mechanisms for High-level Modeling and Simulation of Digital Systems", *5th IEEE Int. Conf. on Electronics, Circuits and Systems*, vol. III, Lisbon, Portugal, Sep., 1998, pp. 229-232.

[23] J.M. Fernandes, R.J. Machado, and H.D. Santos, "Modeling Industrial Embedded Systems with UML", *8th IEEE/IFIP/ACM Int. Workshop on Hardware/Software Co-Design*, San Diego, U.S.A., May, ACM Press, 2000, pp. 18-22.

[24] P. Zave, "The Operational vs. The Conventional Approach", *Communications of the ACM*, no. 27, vol. 2, 1984, pp. 104-18.

[25] R.J. Machado, J.M. Fernandes, and A.J. Proença, "SOFHIA: A CAD Environment to Design Digital Control Systems", *Hardware Description Languages and Their Applications: Specification, Modelling, Verification and Synthesis of Microelectronic Systems*, chapter 10, C. Delgado Kloos and E. Cerny (*editors*), Chapman & Hall, 1997, pp. 86-88.