1 A METHODOLOGY FOR COMPLEX EMBEDDED SYSTEMS DESIGN
Petri Nets within a UML Approach

Ricardo J. Machado, João M. Fernandes, Henrique D. Santos

This paper focus mainly on the analysis phase, describing a UML-based approach for designing complex embedded systems, and specifically the usefulness of using shobi-PN v2.0 specifications, a Petri net extension, for modelling the dynamic behaviour. A relative complex case study is used to show the usefulness of the suggested specification approach.

1. Introduction

The vast majority of embedded systems are control-dominated systems and traditionally designers specify them using only a state-oriented model, such as FSMs. However, real-time embedded systems are getting quite complex, which implies that a different approach is necessary. The system specification has to fulfil several requirements, namely support for concurrency, timing constraints, hierarchy, data and control flow, and distributed computations.

Thus, for modelling more aspects of the systems (namely, data and function), it is important to consider genuine multiple-view models. There is also absolutely no doubt that IT organisations can improve efficiency and productivity if they share the same notation. In this context, the authors recommend the utilisation of some UML views to specify embedded systems, because it is a notation that covers the most relevant modelling aspects of systems and it is an OMG standard.

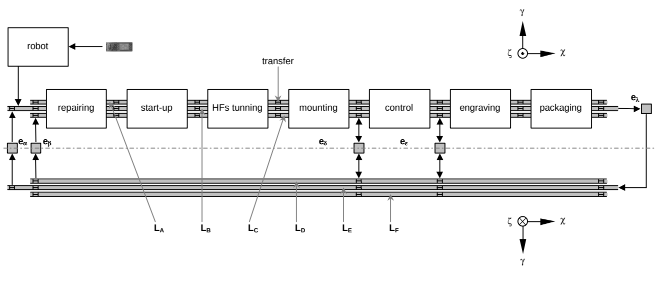


Figure 1. The HIDRO lines.

The diagrams shown in this paper are all relative to the Blaupunkt-Bosch’s car radios production lines (HIDRO lines) controller (fig. 1), which is one of the several complex embedded systems that the authors have used, as real case studies, to validate the proposed methodology.

2. UML

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 (Booch *et al.*, 1999). As a standard language for defining and designing software systems, UML is being progressively 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 believe that it is an appropriate choice for embedded systems. To confirm the usefulness of UML for this engineering field, several research teams (Douglass, 1998; Lyons, 1998; Lanusse *et al.*, 1998; McLaughin *et al.*, 1998; Kabous *et al.*, 1999; Jigorea *et al.*, 2000) have also adopted UML as the notation for specifying embedded systems, which means that this notation is gaining widespread acceptance and usage within this community.

The main views used by the authors for specifying the system are captured by the following diagrams: (1) *use case diagrams* are used to capture 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 that constitute the system; (3) *sequence diagrams* present scenarios of typical interactions among the objects that constitute the system or that interact with it; (4) *class diagrams* store the information of ready-made components that can be used to build systems and specify the hierarchical relationships among them; (5) *Petri nets* (*shobi-PN v2.0*) are used to specify the dynamic behaviour of some objects/classes.

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 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).

2.1. Use Case Diagrams

A use cases diagram is considered to be a powerful and useful technique for capturing the user's requirements. It is an easy-to-read diagram that divides the system in its functional points. A use case can be understood as a functionality or service that the system offers to its users.

The authors propose an extension to UML by adding a new property to use cases, that was designated reference. New properties are added in UML by tagged values, so each use case can have a reference that follows a numbering scheme similar to the traditional DFD numbering. Each use case at the system-level is assigned a reference (example: ref=2), and if this use case is refined by other sub-use cases, each of these will have a reference that uses the super-use case as a prefix (example: ref=2.3). This numbering scheme can be repeated to any depth and it helps those involved in the project to relate all use cases diagrams and will be also used during the transition from use cases to objects to ease the mapping between both models.

Fig. 2 shows the system-level (or top-level) use cases diagram of the HIDRO lines, where it is possible to visualise which actors perform which functionalities. Since use cases have different impact on the final system, they must be ranked taken into consideration their importance to the main functionality of the system. This allows the project to follow a risk-driven process, where the most important or complex functionalities of the system are first tackled, leaving the less important ones to be treated later.

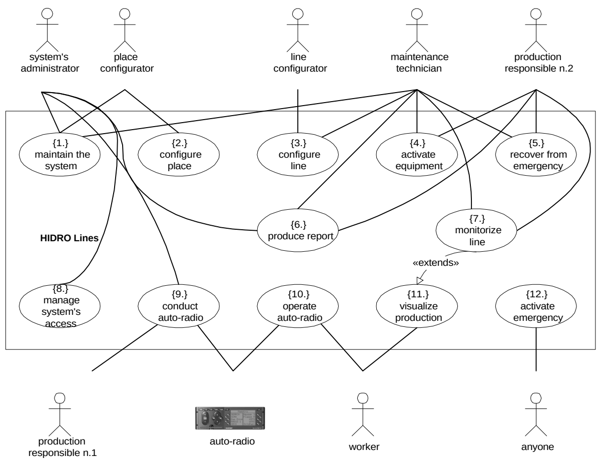


Figure 2. Use case diagram.

2.2. Object Diagrams

Object diagrams are also an important technique to show the components that constitute the system. Transforming the use cases that divide the system in a functional way into objects is a critical task, since usually there is no direct mapping from use cases to objects. Thus, a strategy, composed of some guidelines, is needed to guide the developers of the system on how to transform use cases into objects. There are some approaches for transforming the use cases diagrams into object diagrams, but the majority of them is based on personal feelings and some kind of magic. The authors have defined a systematic strategy, based on the object types (interface, entity and control) presented in (Jacobson *et al.*, 1992), for finding the objects of a given system based on its use cases. This strategy is called *4-step rule set* and has been published in (Fernandes *et al.*, 2000). Fig. 3 depicts the object diagram obtained by the application of the 4-step rule set strategy to the use case diagram of fig. 2.

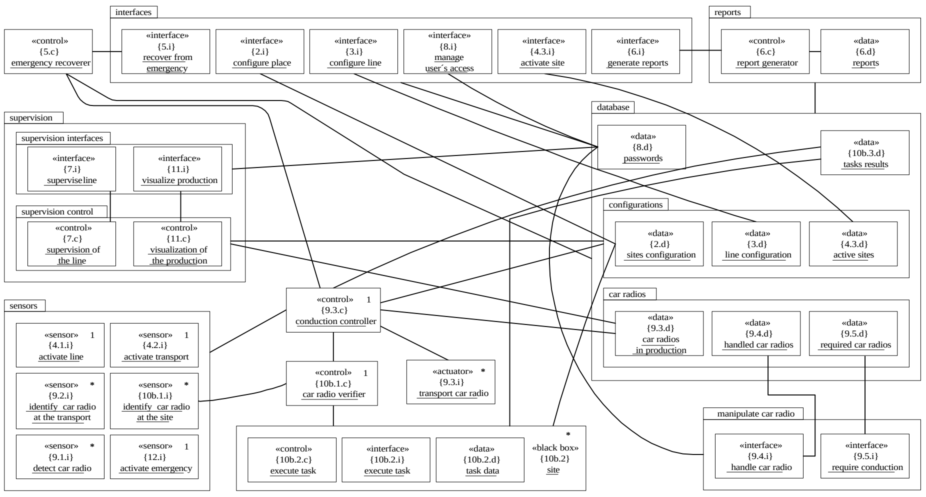
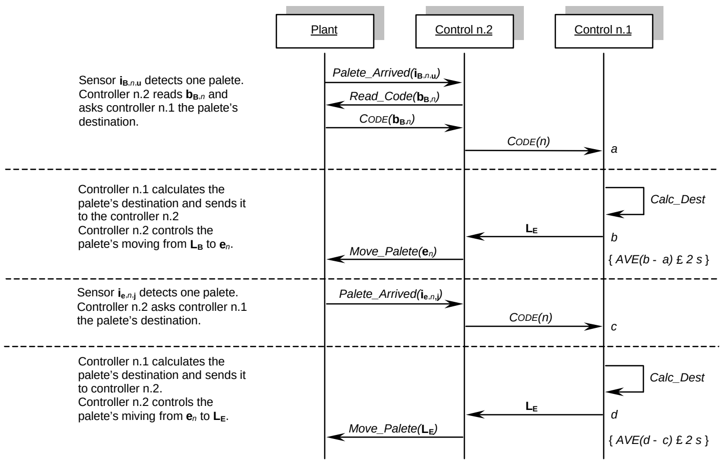

interfaces
reports
«control» {5.c} emergency recoverer
«interface» {5.i} recover from emergency
«interface» {2.i} configure place
«interface» {3.i} configure line
«interface» {8.i} manage user's access
«interface» {4.3.i} activate site
«interface» {6.i} generate reports
«control» {6.c} report generator
«data» {6.d} reports
supervision
supervision interfaces
«interface» {7.i} supervise line
«interface» {11.i} visualize production
supervision control
«control» {7.c} supervision of the line
«control» {11.c} visualization of the production
database
«data» {8.d} passwords
«data» {10b.3.d} tasks results
configurations
«data» {2.d} sites configuration
«data» {3.d} line configuration
«data» {4.3.d} active sites
car radios
«data» {9.3.d} car radios in production
«data» {9.4.d} handled car radios
«data» {9.5.d} required car radios
sensors
«sensor» 1 {4.1.i} activate line
«sensor» 1 {4.2.i} activate transport
«sensor» * {9.2.i} identify car radio at the transport
«sensor» * {10b.1.i} identify car radio at the site
«sensor» * {9.1.i} detect car radio
«sensor» 1 {12.i} activate emergency
«control» 1 {9.3.c} conduction controller
«control» 1 {10b.1.c} car radio verifier
«actuator» * {9.3.i} transport car radio
«control» {10b.2.c} execute task
«interface» {10b.2.i} execute task
«data» {10b.2.d} task data
* «black box» {10b.2} site
manipulate car radio
«interface» {9.4.i} handle car radio
«interface» {9.5.i} require conduction

Figure 3. Object diagram of the HIDRO lines system.

2.3. Sequence Diagrams

The proposed methodology uses sequence diagrams as an intermediate format to specify the system's dynamic behaviour. Sequence diagrams correlate several objects in a particular scenery, that corresponds to a part of each object's life cycle. Its is also possible to inscribe timing constrictions in the sequence diagram. Fig. 4 shows one sequence diagram of the HIDRO lines system.


Sensor $\mathbf{i}_{\mathbf{B},n,\mathbf{u}}$ detects one palete. Controller n.2 reads $\mathbf{b}_{\mathbf{B},n}$ and asks controller n.1 the palete's destination.

Controller n.1 calculates the palete's destination and sends it to the controller n.2 Controller n.2 controls the palete's moving from $\mathbf{L}_\mathbf{B}$ to $\mathbf{e}_n$.

Sensor $\mathbf{i}_{\mathbf{e},n,\mathbf{j}}$ detects one palete. Controller n.2 asks controller n.1 the palete's destination.

Controller n.1 calculates the palete's destination and sends it to controller n.2. Controller n.2 controls the palete's miving from $\mathbf{e}_n$ to $\mathbf{L}_\mathbf{E}$.

Plant
Control n.2
Control n.1
Palete_Arrived($\mathbf{i}_{\mathbf{B},n,\mathbf{u}}$)
Read_Code($\mathbf{b}_{\mathbf{B},n}$)
CODE($\mathbf{b}_{\mathbf{B},n}$)
CODE(n)
a
Calc_Dest
$\mathbf{L}_\mathbf{E}$
b
Move_Palete($\mathbf{e}_n$)
{ AVE(b - a) £ 2 s }
Palete_Arrived($\mathbf{i}_{\mathbf{e},n,\mathbf{j}}$)
CODE(n)
c
Calc_Dest
$\mathbf{L}_\mathbf{E}$
d
Move_Palete($\mathbf{L}_\mathbf{E}$)
{ AVE(d - c) £ 2 s }

Figure 4: Sequence diagram.

The authors' methodology allows the use of UML non-standard sequence diagrams, that are called *scenery diagrams*. These diagrams are very useful when the pictorial representations of the data path/plant's control sub-sequences are relevant for a thoroughly understanding of the controller's behaviour. In fig. 5 it is possible to observe a scenery diagram of the HIDRO lines.

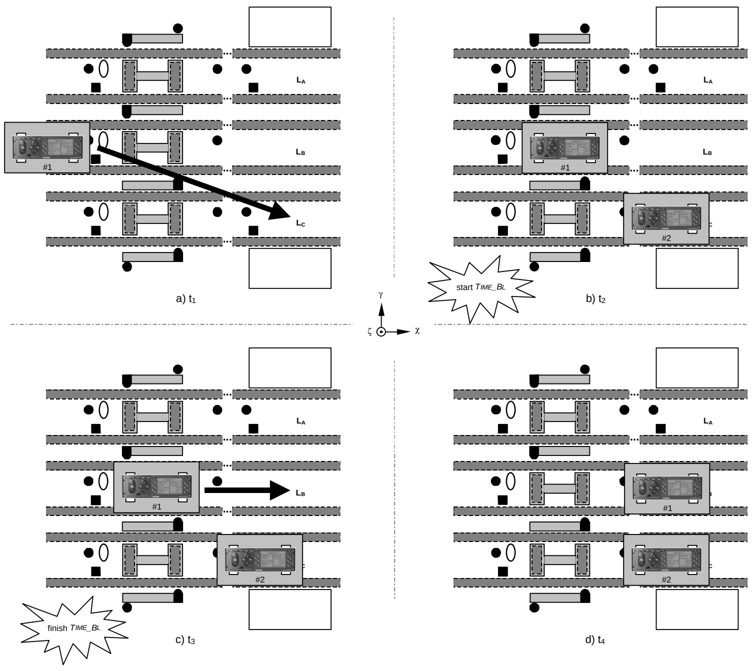


Figure 5: Scenery diagram.

The methodology also proposes the use of non-standard data path/plant diagrams for data path/plant's resources static specification, since UML does not define any diagram for that. These diagrams are needed to understand the pictorial representation of the data path/plant's resources involved in the scenery diagrams.

3. Petri Nets

For the system' components that have a complex or interesting dynamic behaviour, a state model can be specified. 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 necessities for complex, distributed and parallel embedded systems, since different parts of the system may try to access simultaneously the same resource. *SPECCHARTS* are also another interesting state-oriented meta-model for specifying and designing embedded systems (Gajski *et al.*, 1994).

For control embedded systems, the application of Petri nets (PNs) to the specification of the behavioural view can benefit from a huge amount of available research. The designer can choose, among several PN meta-models, a specific one intentionally developed to deal with the semantical specificities of that kind of systems, like the ones referred in (Kleinjohann *et al.*, 1997; Sgroi *et al.*, 1998).

PN is a mathematical meta-model that can be formally analysed and for which several implementation techniques are available. In this context, the authors have developed an extended PN meta-model, designated *shobi-PN*, to specify the reactive behaviour of the system' components instead of using the "*StateCharts* + activity diagrams" UML proposal.

3.1. The shobi-PN v1.0

The traditional synchronous and interpreted PN meta-model (SIPN) was developed, aiming 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 (Machado *et al.*, 2000). 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 (David *et al.*, 1992), Hierarchical PNs (Fehling, 1993), Coloured PNs (Jensen, 1992), and Object-Oriented PNs (Lakos, 1995). 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.

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 exhaustive used in several application domains of medium complexity: industrial controllers (Machado *et al.*, 1997b), communication interfaces (Machado *et al.*, 1998a), and micro-architecture of processors (Machado *et al.*, 1998b).

3.2. The shobi-PN v2.0

The use of *shobi-PN v1.0* meta-model to specify the behaviour of the level 2 controller of the HIDRO production lines has revealed some semantical 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 syntactic and semantic properties within the *shobi-PN v2.0* meta-model; (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. Fig. 6 shows one *shobi-PN v2.0* specification net of the HIDRO production lines level 2 controller.

The tokens/objects that must appear in the *shobi-PN* nets are found by calculating the system high-level object diagram, obtained by applying to the global object diagram (fig. 3) one filtering and collapsing technique, also developed by the authors (see fig. 7).

4. Tools

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 proposed methodology is an on-going project, but there are already some tools available for the developers.

A graphical environment was developed to allow animation/simulation which generates UML sequence diagrams. These diagrams are built from the system specification and allow the designers to compare them with similar diagrams constructed previously in co-operation with the system's customers.

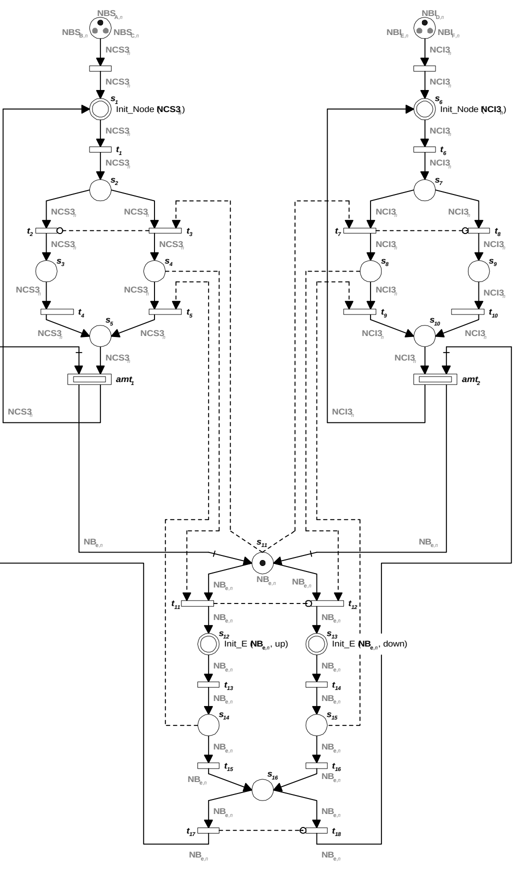


Figure 6: A *shobi-PN v2.0* specification net.

This eases the methodology to follow the operational approach (Zave, 1984), 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 are under development, namely graphical editors (for specifying the systems) and compilers (for automatically generating C code for the MCS-51 compatible microprocessors). A preliminary version of all this tool-set is expected to be available very soon. This tool-set is the descendant of the one authors have presented in (Machado *et al.*, 1997a), i.e. it supports directly the *shobi-PN* meta-model, using the Oblog language (www.oblog.com) as the implementation support.

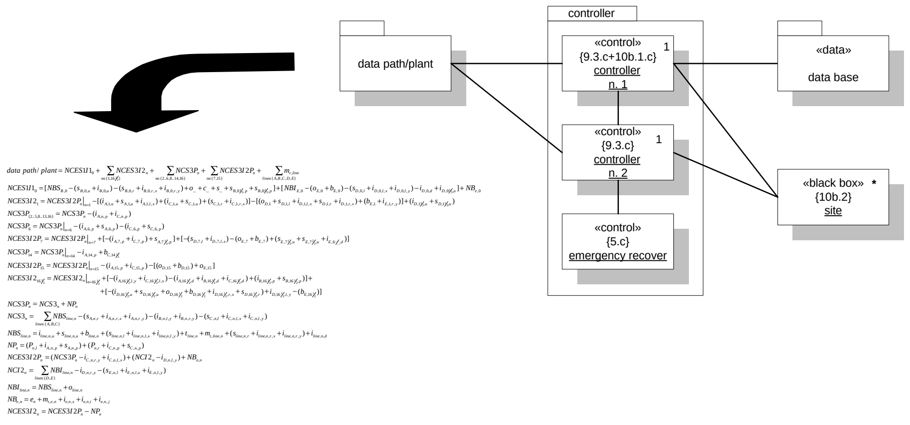


Figure 7: High-level object diagram of the HIDRO lines.

5. Conclusions

This paper presents the general characteristics of a UML-based methodology to support the design of complex embedded systems. The authors defend the use of PN-based behavioural specifications, instead of using the UML *STATECHARTS* and activity diagrams. In this context, the *shobi-PN v2.0* meta-model was generally explained, in what concerns the usefulness of its *GAS* and *AMT* concepts. This paper also shows the simulation environment the authors have developed to directly support the design of complex embedded controllers. All the UML standard and non-standard diagrams shown in this paper are relative to the Blaupunkt-Bosch's auto radios production lines. As a typical example of a complex embedded system, the controller for this production line has been used by the authors to validate their methodology.

References

Booch, G., Rumbaugh, J., Jacobson, I. (1999). *The Unified Modeling Language User Guide*, Addison-Wesley.

David, R., Alla, H. (1992). *Petri Nets & GRAFCET: Tools for Modelling Discrete Event Systems*, Prentice-Hall.

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

Fehling, R. (1993). "A Concept of Hierarchical Petri Nets with Building Blocks." *Advances in Petri Nets 1993*, Lecture Notes in Computer Science, vol. 674, pp. 148-168, Springer-Verlag.

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

Gajski, D., Vahid, F., Narayan, S. (1994). *Specification and Design of Embedded Systems*, Prentice-Hall.

Jacobson, I., Christerson, M., Jonsson, P., Övergaard, G. (1992). *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley.

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

Jigorea, R., Manolache, S., Eles, P., Peng, Z. (2000). "Modeling of Real-Time Embedded Systems in an Object-Oriented Design Environment with UML." *3rd IEEE International Symposium on Object-Oriented Real-Time, Distributed Computing - ISORC'2000*, pp. 210-203, Newport Beach, U.S.A., March.

Kabous, L., Nebel, W. (1999). "Modeling Hard Real-Time Systems with UML: The OOHARTS Approach." *2nd International Conference on the Unified Modeling Language - UML'99*, Fort Collins, U.S.A., October.

Kleinjohann, B., Tacken, J., Tahedl, C. (1997). "Towards a Complete Design Method for Embedded Systems Using Predicate/Transition-Nets." *Hardware Description Languages and Their Applications: Specification, Modelling, Verification and Synthesis of Microelectronic Systems*, chapter 1, pp. 4-23, C. Delgado Kloos e E. Cerny (*editors*), Chapman & Hall.

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

Lanusse, A., Gérard, S., Terrier, F. (1998). "Real-Time Modeling with UML: The ACCORD Approach." *International Workshop on the Unified Modeling Language: Beyond the Notation - UML'98*, Mulhouse, France.

Lyons, A. (1998). *UML for Real-Time Overview*, ObjecTime Limited, April.

Machado, R. J., Fernandes, J. M., Proença, A. J. (1997a). "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, pp. 86-88, C. Delgado Kloos e E. Cerny (*editors*), Chapman & Hall.

Machado, R. J., Fernandes, J. M., Proença, A. J. (1997b). "Specification of Industrial Digital Controllers with Object-Oriented Petri Nets." *IEEE International Symposium on Industrial Electronics - ISIE'97*, vol. I, pp. 78-83, Guimarães, Portugal, July.

Machado, R. J., Fernandes, J. M., Proença, A. J. (1998a). "An Object-Oriented Model for Rapid-Prototyping of Data Path/Control Systems - A Case Study." *9th IFAC/IFIP Symposium on Information Control in Manufacturing - INCOM'98*, vol. II, pp. 269-274, Nancy & Metz, France, June.

Machado, R. J., Fernandes, J. M., Proença, A. J. (1998b). "Hierarchical Mechanisms for High-level Modeling and Simulation of Digital Systems." *5th IEEE International Conference on Electronics, Circuits and Systems - ICECS'98*, vol. III, pp. 229-232, Lisbon, Portugal, September.

Machado, R. J., Fernandes, J. M., Esteves, A. J., Santos, H. D. (2000). "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, pp. 205-222, A. Yakovlev, L. Gomes e L. Lavagno (*editors*), Kluwer Academic Publishers.

McLaughin, M., Moore, A. (1998). "Real-Time Extensions to UML." *Dr. Dobb's Journal*, (292),82-93, December.

Sgroi, M., Lavagno, L., Watanabe, Y., Sangiovanni-Vincentelli, A. (1998). "Quasi-Static Scheduling of Embedded Software Using Free-Choice Petri Nets." *1st Workshop on Hardware Design and Petri Nets - HWPN'98*, Lisbon, Portugal, June.

Zave, P. (1984). "The Operational vs. the Conventional Approach." *Communications of the ACM*, 27(2):104-18, 1984.