Properties Preservation in Distributed Execution of Petri Nets Models

Anikó Costa[1], Paulo Barbosa[2], Luís Gomes[1], Franklin Ramalho[2], Jorge Figueiredo[2], and Antônio Junior[2]

[1] Universidade Nova de Lisboa, Portugal
{akc,lugo}@uninova.pt
[2] Universidade Federal de Campina Grande, Brasil
{paulo,franklin,abrantes,antonio}@dsc.ufcg.edu.br

Abstract. Model-based development for embedded system design has been used to support the increase of system's complexity. Several modeling formalisms are well matched for usage within this area. One of the goals of this work is to contribute to the usage of Petri nets as system specification language within model-based development of embedded systems having MDA proposals as a reference for the development flow. Distributed execution of the Petri net model is achieved through model partitioning into sub-modules. System decomposition is obtained through net splitting operation. Two types of implementation platforms were considered: compliant and non-compliant with zero time delay for communication between modules. Using model-checking techniques, properties associated with the execution of the distributed models in both types of platforms were compared with the execution of the initial (centralized) Petri net model.

Keywords: Petri net, model decomposition, distributed execution, model checking.

1 Introduction

The sustained growing of system's complexity has not been followed by the increase of the designer's productivity due to the lack of adequate methods and tools. The Model-based development approach can adequately support improvements in this area. In the past few years the research on embedded system design has been increasing. Most of the works use UML (Unified Modeling Language) as system design language. However, UML can be used for system specification, but not for analysis and syntheses. For that purposes is need to add specific profiles and included them into the tools which uses UML for system development. For example in [1] is proposed an UML profile for Modeling and Analysis of Real-Time and Embedded. Another work [2] is based on high-level UML design of system components and use SystemC for system behavior's validation.

Using Petri nets [3] [4] as system specification language for modeling the control part of the embedded system, the development process can be improved. Petri nets are

L.M. Camarinha-Matos, P. Pereira, and L. Ribeiro (Eds.): DoCEIS 2010, IFIP AICT 314, pp. 241–250, 2010.

an adequate formalism for modeling concurrency, parallelism and synchronization. These characteristics are commonly found when dealing with complex systems.

When one faces a complex system model, it is natural to want to decompose it into several sub-models due to the growing role that distributed systems are playing nowadays. The net splitting operation was introduced for this purpose [3], allowing to divide a Petri net model into several sub-nets. By using formal verification techniques we can obtain a reliable way to ensure that the execution of the original model and the parallel execution of the several sub-nets that were produced by the net splitting operation produce the same behavior, considering a selected set of properties (examples of selected properties include liveness, boundness, and relevant traces). In this way, we should get distributed models that can be implemented using different approaches:

- Implementing the global system model as a centralized controller;
- Using model decomposition to obtain several components, considered as independent controllers and implementing them in the same platform using synchronous communication channels for the communication between the components with zero time delay in the communication;
- Implementing the above referred components on different platforms where considering zero time delay for the communication becomes unfeasible (as they are in regions having different time domains, for instance).

This paper intends to prove that a specific set of properties associated with different distributed implementation models are preserved after the referred splitting transformations be implemented according to the Model Driven Architecture (MDA) approach [6] [7] [8]. The selected set of properties (liveness, boundness, and relevant traces) will be evaluated considering the execution behavior when different types of implementation platforms are considered.

The structure of the paper is the following. Section 2 discusses the main contributions on technological innovation issues. In the Section 3, the methodology overview of the MDA approach is presented. In Section 4, a running example is used to illustrate the usability of the proposed methodology. In Section 5, verification issues are addressed, concluding that all the proposed implementation models have the same behavior. Finally, in Section 6 conclusions are presented.

2 Contribution to Technological Innovation

The ultimate goal of the works where the presented one is included is to contribute to the usage of Petri nets as a system level specification language for embedded system design. In particular, the innovative contribution is to use the net splitting operation to formally support a technique to start from global system Petri net model and be able to obtain parallel components, which would be executed in heterogeneous platforms. As will be shown in the following sections, a set of properties of the global system model will be preserved after the described transformation, even when different types of platforms are considered for implementation.

The contribution of this paper is to introduce a MDA based methodology for embedded system design, when distributed execution of the model is a concern and

where Petri nets are used as system specification language. The main focus is given to the properties verification of the transformed models.

3 Methodology Overview and Objectives

Our reference methodology for embedded system design using hardware-software co-design techniques starts with the description of the system's functionalities through UML (Unified Modeling Language) use cases. Afterwards, these use cases are manually translated into a set of operational formal models, where Petri nets are considered as the reference model formalism. These models are amenable to support property verification and to be automatically translated into code, after being partitioned into components and mapped into specific implementation platforms.

Our intention is to improve that methodology introducing the MDA philosophy of development. In this approach, the effort is focused on models, metamodels, and model transformations. In MDA, an application can be characterized by several models at different levels of abstraction. The relations between those models are defined as transformations between models.

The highest considered abstraction level is the Platform Independent Model (PIM). In such level, the models represent the system requirements and are independent of any specific implementation platform. The MDA approach emphasizes that a correct model at the PIM level should maintain its correctness independently of selecting different technology mappings afterwards.

The next level of abstraction is the Platform Specific Model (PSM). At this level, the models reflect the specific system characteristics. In this level, specific constructs of each platform are considered, although, as emphasized before, the behavior of the original PIM model must be preserved.

Finally, as the lowest level of abstraction, we are considering the implementation code, reflecting the concrete syntax of a specific implementation platform. This artifact does not insert any specific abstraction to the models, just being a straightforward representation of the PSM concepts.

In this work, the emphasis is on the PSM level. To illustrate these different abstraction levels and the model transformations between them, we use Petri nets (see Fig. 1), as we want to emphasize the usage of Petri nets as a system level specification formalism. However, from the point of view of the proposed methodology flow, Petri nets can be replaced by any other behavioral formalism with similar characteristics, without loss of generality of the proposed methodology flow.

Using Petri net as system level specification language, in the first level of abstraction we apply a transformation in order to decompose the model into several sub-nets using the net splitting operation [5]. These operations are introduced in our MDA approach by transformation rules that define how to transform a global Petri net model into partitioned Petri net sub-models (subnets). Afterwards, PSMs are generated from these subnets. In the case of a decomposed PIM model, we can consider two types of PSMs to be generated. One of them is where the communication between

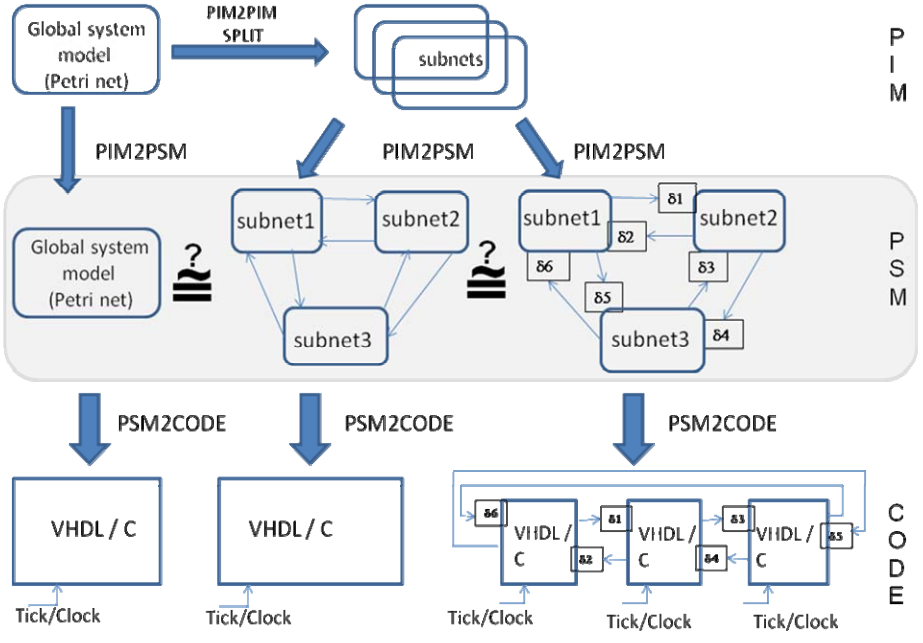

Fig. 1. From PIMs to code: an MDA-based approach

the models is made using synchronous communication channels [9] and the zero delay time paradigm can be applied. On the other hand, it is also possible to consider the case where the zero delay time paradigm does not hold and the communication between the models is made using a random time delay on the communication channels.

As the lowest level of abstraction those models are mapped into implementation code for a specific platform. Two types of implementation platforms for the distributed model are considered: those having a global clock/tick and where a synchronous communication between the components is possible according with the zero delay time paradigm, and those having different time regions (heterogeneous platforms) and where the synchrony paradigm does not apply and the communication is modeled using a delay between the components.

The objective of this paper is to present the way to demonstrate that the generated PSM models associated with different platforms have the same execution behavior.

The expected benefits of using an MDA approach for generating the models and respective code of the partitioned components are twofold: The first one is to lower the cost of deploying a given component and corresponding code on multiple platforms through reuse; The second one is to reach a higher level of automation in the code generation process, through the specification of transformations among those models.

In the next section we will present a running example to illustrate the application of the proposed methodology.

4 Running Example

We consider a simple example of application introduced by [10] with two cars going back and forth, as shown on the Fig. 2

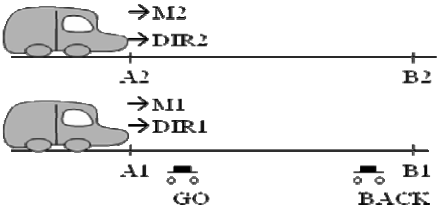

Fig. 2. Application example

Their movements are synchronized at the end points (in A[i] and B[i], respectively). The cars start moving when both are in the initial position (in A1 and A2) and the button GO is activated. They stop when reach the end point B1 and B2, respectively. To go back to the initial position both cars have to be in the respective positions B[i] and the button BACK has to be activated. A possible Petri net PIM model for this simple control problem is presented in Fig. 3 a). As PIM2PIM model transformation we can use the net splitting operation [5] to decompose the Petri net and obtain a set of distributed controllers (to be deployed one for each car). This can be done by choosing a set of nodes (transitions GO and BACK for the example) from

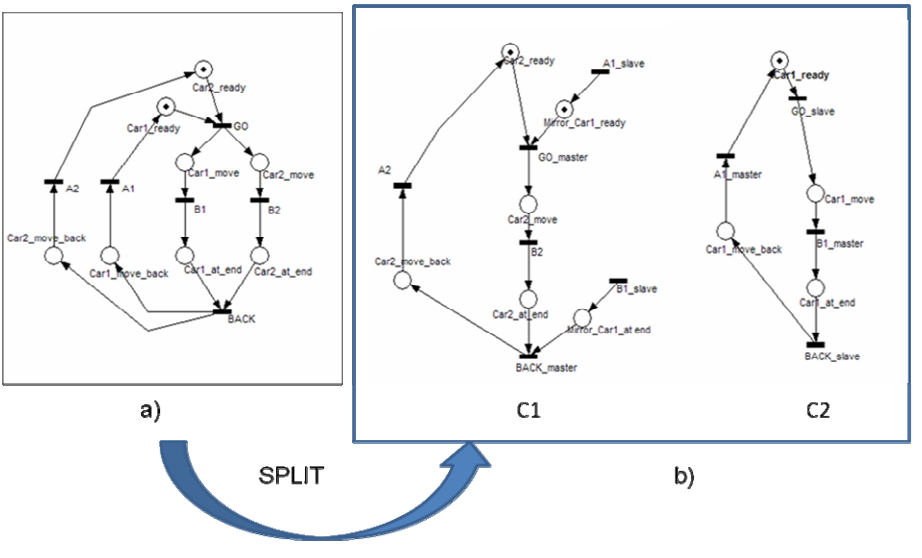

Fig. 3. Petri net models for two cars system: a) global controller; b) distributed controllers

where the net will be splitted. As the result of this operation we obtain four sub-nets, corresponding to the control of the movement in each direction of the cars. To obtain the model of the controller for each car we have to map two sub-nets to each controller (which are combined using the net addition operation [11]). The resulted models are shown in the Fig. 3 b).

As PSM model of the global system for the controller of our system is shown on Fig. 4 where the input and the output signals of the system are represented (dependencies on output signals are not explicitly represented, but they are associated with "Cari_move" places of the Petri net model).

When we consider a distributed model instead of a centralized model, as shown on Fig. 3 b) we can consider different behaviors associated with the PIM2PSM transformation. One of them is when the communication between the components is represented by synchronous communication channels. This means that pairs of transitions belonging to different sub-nets (one with attribute master and other one with attribute slave, as presented in Fig. 3b)) are synchronized through a synchronous communication channel and will fire at the same time, considering the same tick. The corresponding model is represented in Fig. 5. This model is composed of two sub-models. The sub-model where the transition with the attribute master is included generates an output event which is read by the other sub-model where a transition with the same name and attribute slave is included. Execution of both sub-models will satisfy the synchrony paradigm.

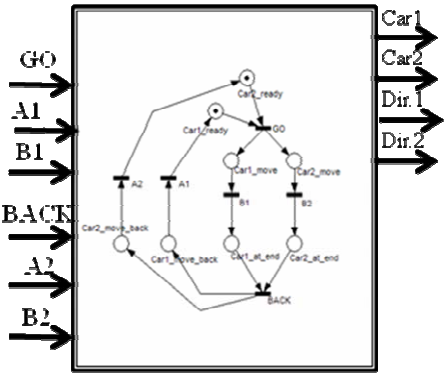

Fig. 4. Global system model

Another possible modeling attitude in the case of the PIM2PSM transformation is to consider firing in different instants for transitions involved in the same communication channel, resulting in a non-zero time delay associated with the communication between the two components. Considering that our communication channel, even when considering synchronous firing, is directed (which means that an output event is

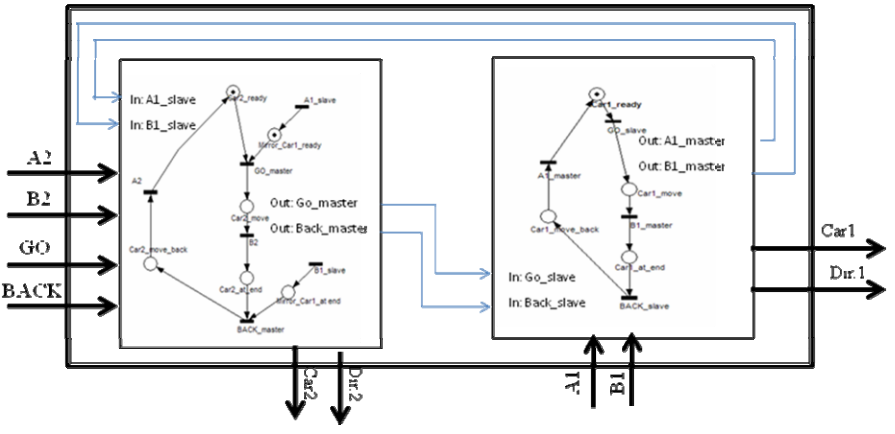

Fig. 5. Distributed system model implementation considering zero time delay

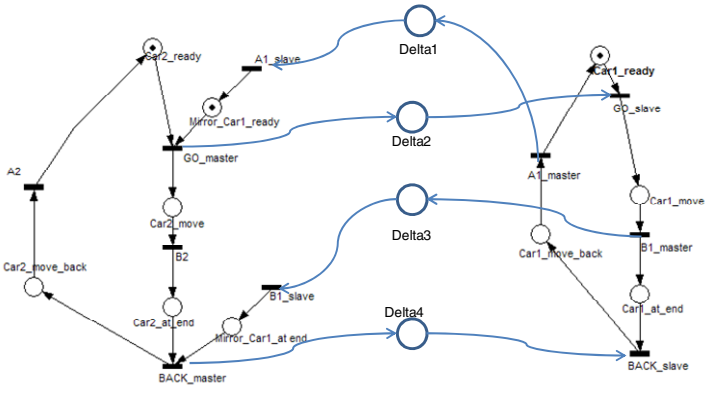

Fig. 6. Distributed system model including the communication between the components

generated by the master transition firing, which is read as an input event by the slave transition), it is possible to explicitly include this dependency as a delay δ into the model, and represent the delay as a place between the transition master and transition slave, as in the Petri net model of Fig. 6. It is easy to verify that, for this new model, those places are safe (limited to only one token).

The model of Fig. 6 corresponds to the system implementation where each sub-model is in execution in a different platform with different execution ticks (different time regions). In this way, we can not guarantee the synchronous firing of transitions belonging to different components. In this case the component with the master transition generates an output event which will be connected to the component to which the respective slave transition belongs as an input signal associated to that transition. This view of the model is represented in Fig. 7.

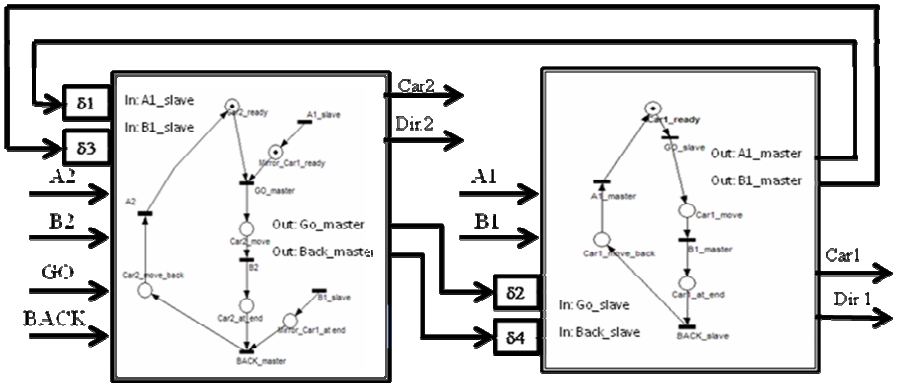

Fig. 7. Distributed system model implementation considering different platforms

5 Verification

For verification purposes, we are using rewriting logic and model checking techniques, supported by the Maude framework [12] [13] and Linear Temporal Logic (LTL) language [14], relying on its specification patterns to define properties verification.

Due to space restriction we choose to include here analysis on the verification only on the following question goals: *if an event P happens between two events Q and R.* The choice for this class of properties allows us to show that two models involved in proposed transformations can preserve, at least the partial order, the same behavior *P* bounded by the limits of the events *Q* and *R*.

For each Petri net presented in the figures Fig. 3 a), Fig. 3 b), and Fig. 5, the equations associated with the events *Q*, *R* and *P* are defined in a way that *Q* is equivalent to firing of transition GO, *P* is equivalent to firing of transition BACK , and *P* is equivalent to firing of transitions B[i] or A[i].

By applying the specification patterns in this scenario, we used the following temporal logic formula:

```
[](Q and !R -> (!R W (P and !R)))
```

In this sense, our objective is to demonstrate that a specific firing sequence associated with the initial model of the global system is verified for all cases of the different implementation models. One possible trace is the following:

$$\text{Go} \rightarrow (\text{B1 and B2}) \rightarrow \text{Back} \rightarrow (\text{A1 and A2}) \rightarrow \text{Go}$$

The semantics of this trace states that the order of firing of B1 and B2 is indifferent, the same way as the order of firing of A1 and A2. The important constraint to be checked is the firing of those transitions before firing transitions BACK and Go respectively.

By using the Maude 2.4 toolset, where we have defined the Petri nets presented in the figures Fig. 3 a) and b) and Fig. 5 and the equations needed for the verification purposes, the result in the above presented case is TRUE for all execution models as was defined in the previous section.

In this sense, preservation of this trace propriety is preserved along presented transformations.

On the other hand, in the case when the event *P* is defined as a non-observable sequence, the result of the verification is a counterexample. One of the non-observable sequences is Go → (B1) → Back, (considering no occurrence of B2). For this case we obtained for all our nets a result as a counterexample.

The associated codes and results are available in the institution's web site at http://www.uninova.pt/~mda-veritas.

6 Conclusions

The paper presents an MDA approach for verification of properties preservation of the transformed models. As reference modeling formalism was used Petri net and the net splitting operation for model decomposition. It was demonstrated using model checking techniques that all models associated with our running example keep the same behavior when mapped into implementations platforms with different constraints.

Complementary the different models were coded in VHDL and deployed into Spartan 3 FPGA platforms. It was validated that the different physical implementations keep the same behavior for relevant situations.

Acknowledgments. The work presented was partially supported by the collaborative project Verificação Semântica em Transformações MDA Envolvendo Modelos de Redes de Petri (MDA-VERITAS - Semantic Verification in MDA Transformation using Petri nets models), funded by FCT - Fundação para a Ciência e a Tecnologia (Portugal) and CAPES - Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (Brazil) (http://www.uninova.pt/~mda-veritas).

References

1. Brisolara, L.C.B., Kreutz, M.E., Carro, L.: UML as Front-End Language for Embedded Systems Design. In: Gomes, L., Fernandes, J.M. (eds.) Behavioral Modeling for Embedded Systems and Technologies: Application for Design and Implementation, pp. 1–23. Information Science Reference, Hershey (2009)
2. Gargantini, A., Riccobene, E., Scandurra, P.: Model-Driven Design and ASM Validation of Embedded Systems. In: Gomes, L., Fernandes, J.M. (eds.) Behavioral Modeling for Embedded Systems and Technologies: Application for Design and Implementation, pp. 24–54. Information Science Reference, Hershey (2009)
3. Girault, C., Valk, R.: Petri nets for systems engineering: A Guide to Modeling. Verification and Applicatons. Springer, Heidelberg (2003)
4. Reisig, W.: Petri nets: An Introduction. Springer, New York (1985)
5. Costa, A., Gomes, L.: Petri net Partitioning Using net Splitting Operation. In: 7th IEEE International Conference on Industrial Informatics (2009)
6. OMG: Object Management Group (2009), http://www.omg.org
7. Miller, J., Mukerji, J.: Mda guide version 1.0.1. Object Management Group, OMG (2003)

8. Barbosa, P., Ramalho, F., Figueiredo, J., Junior, A., Costa, A., Gomes, L.: Checking Semantics Equivalence of MDA Transformations in Concurrent Systems. Journal of Universal Computer Science (JUCS) 15(11), 2196–2224 (2009), http://www.jucs.org/jucs_15_11/checking_semantics_equivalence_of
9. Christensen, S., Hansen, N.D.: Coloured Petri Nets Extended with Channels for Synchronous Communication. In: Valette, R. (ed.) ICATPN 1994. LNCS, vol. 815, pp. 159–178. Springer, Heidelberg (1994)
10. Silva, M.: Las Redes de Petri: En la Automática y la Informática. Editorial AC, Madrid (1985)
11. Barros, J.P., Gomes, L.: Net Model Composition and Modification by Net Operations: A Pragmatic Approach. In: 2nd IEEE International Conference on Industrial Informatics, Berlin, Germany, June 24-26 (2004)
12. Maude system and tools, http://maude.cs.uiuc.edu/maude1/tutorial/
13. Clavel, M., Durán, F., Eker, S., Lincoln, P.: Martí -Oliet, N., Meseguer, J., Quesada, J.F.: Maude: Specification and programming in rewriting logic. Theoretical Computer Science (2001)
14. Specification patterns for temporal logic model-checking. SAnTos Laboratories, http://patterns.projects.cis.ksu.edu/documentation/patterns/ltl.shtml