

A Formal Approach for Component Based Embedded Software Modelling and Analysis

Hyggo Oliveira Almeida*, Leandro Dias da Silva*, Elthon Oliveira† Angelo Perkusich*

*Federal University of Campina Grande/Department of Electrical Engineering, Campina Grande, Paraiba, Brazil
{leandro,hyggo,perkusic@dee.ufcg.edu.br}

†Federal University of Campina Grande/Graduate Program in Computer Science, Campina Grande, Paraiba, Brazil
elthon@dsc.ufcg.edu.br

Abstract— To develop software for embedded systems the designer must take into account different kinds of problems and complexities. The main issues are related to late integration with the target hardware and the separation of the development environments from target execution platforms. Due to the fact that in most situations they are developed using low-level programming languages, approaches to develop complex software systems, such as component based software development, cannot be used at all. In this work we introduce a component based development process that integrates colored Petri nets and a component model in order to introduce better abstraction mechanisms in the modelling, analysis and verification processes for embedded software systems.

I. INTRODUCTION

Embedded systems software has been increased in complexity in the last years. Component based software development (CBSD) is a key approach to deal with such complexity. However, most of the component models are targeted to desktop computers, with abundant hardware resources managed by high level operating systems, making available a high level abstraction programming model. On the other hand, embedded software has specific problems and complexities to be considered, such as late integration with the target hardware and separate environments for development and target execution. Therefore, it is necessary to define and adopt abstraction techniques, theories, and tools to develop embedded software. Also, the life-cycle to develop software for embedded systems must be shorter than for desktop applications, and the development process must address the constant changes in hardware in a safe, and controlled way, but still considering the time to market. In order to deal with these constraints, concepts such as product lines [1], software architectures [2], and components [3], have been applied to promote the adoption of the development processes to allow the assembly of systems based on a framework and a set of components. Thus, common components of entities or functionalities previously developed can be reused, and a cheaper and faster development process can be used, thus resulting in more dependable embedded software.

On the other hand, to ensure high availability, the development process for embedded software must address quality requirements. In this context, validation techniques, such as formal modelling, simulation and model checking, are becoming essential [4].

The use of formal methods to model systems aggregates

several advantages such as, for instance, automatic simulation, and proof of properties. In the context of this work, Hierarchical Colored Petri Nets (HCPN) [5] are used as a formal description language. Colored Petri Nets have been extensively used in many different applications [6], and the modelling activities are supported by a set of computational tools named Design/CPN [7]. In [8], a component-based development process based on HCPN modelling and verification was introduced.

The component development process has two main development phases: architecture definition and component integration based on a framework. These phases are time consuming and their application can be effectively improved if a middleware to compose and integrated models for embedded components are provided.

In this paper, it is presented the integration of the component model defined for the COMPOR infrastructure [9], called COMPOR-CM, and the component based development process previously mentioned. COMPOR-CM is used as a middleware for composition and integration, reducing both development effort and time.

The remaining of this paper is organized as follows. In Section II, the component-based development process is presented. In Section III, the integration of COMPOR-CM with the development process is described. In Section IV, it is presented the embedded system case study. In Section V, the application of the approach to the case study is shown. In Section VI, some related works are discussed. Finally, in Section VII, the conclusions and suggestion for future works are presented.

II. COMPONENT-BASED DEVELOPMENT PROCESS

Using the component approach a complex system can be developed using existing components. The main problem in this case is to define how components are connected. One possible solution for this problem is to explicitly define a software architecture.

In Figure 1, the life cycle for the systematic development of systems based on components defined in [8] is illustrated. The modelling phase is present in almost all the steps.

Once the requirements are specified, an architecture for the system is defined. The specification of the architecture is in terms of the functionalities defined in the requirements analysis, and the components satisfying the functionalities can be identified. The definition of the

architecture and the identification of the functionalities are considered in the modelling.

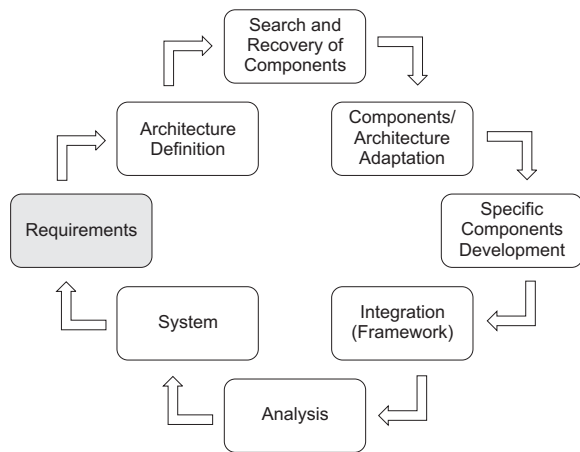


Fig. 1. Life cycle for component-based development

A search is performed to identify possible candidates. It can be the case that it is difficult to find component satisfying all the requirements. It can be necessary to adapt the model before reusing it in the current design. It can be possible that some functionalities do not have components representing them, or even it can be the one that aggregates value to the design. In that last case the functionalities may be the major difference of the system being developed compared to other systems or, even, a commercial secret of the system, and they should be developed locally.

The integration step consists of using a framework for the composition of the system based on the components. In the framework, it is defined the part that does not change in a product line. In the case of component-based systems, the framework is responsible for the management of the interactions among the components. The recovery, adaptation, development, and integration steps are explicitly modelled.

The last step is the analysis and it may consist of several methods as, for instance, simulation, proof of properties, and tests. Even if the system is validated, changes in the requirements are possible along the time. Therefore we have a cycle. In other words, it is considered that a system always evolves, and this evolution is always treated by the modelling of the system.

III. INTEGRATING COMPONENTS WITH THE DEVELOPMENT PROCESS

As mentioned before, the two main phases of the development process are the architecture definition and the component integration through framework conception. It occurs because such phases require effort to define and construct the framework model which integrates the components. In [10], for example, a framework for embedded system had to be constructed from the scratch.

In order to reduce effort and time consuming for applying these process phases, the COMPOR-CM model can be used as a generic framework which manages the component integration.

A. An Overview of COMPOR-CM

In the context of CBD, inserting, removing, and changing application components are directly related to the coupling among such components. In the case that components are tightly coupled, it is more difficult to adapt the application when changes occur. Even when interfaces are well defined, the explicit reference among components does not allow dynamic runtime changes in the functionalities' providers.

In COMPOR-CM, it is insured that there are no explicit references among functionalities' providers for a given application. This feature promotes a significant increase in the flexibility for inserting, removing and changing such providers, even at runtime.

The architecture of COMPOR-CM is composed by two kind of entities: containers and functional components. Functional components implement the application functionalities, making them available by means of services. A functional component is not composed by other components, that is, it has no child components. On the other hand, containers implement no functionalities. The function of a container is to manage the access to the services provided by its child components.

In Figure 2 the interaction process is illustrated and the steps to promote the interaction are detailed in what follows.

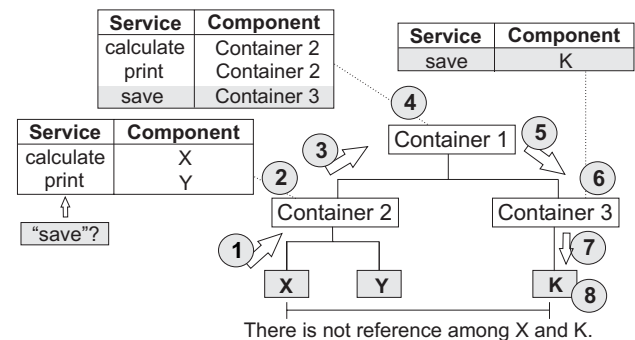


Fig. 2. Service based interaction: localization and execution without explicit references

- 1) Component X requests the execution of service "save" to its parent container.
- 2) Based on its service table, Container 2 verifies that no child components implement the "save" service.
- 3) The Container 2 forwards the request to its parent container, in this case Container 1.
- 4) The Container 1, according to its service table, verifies that one of its children implements the "save" service, Container 3 in this case. The Container 1 sees Container 3 as the component that implements the requested service.
- 5) The Container 1 then forwards the service request to Container 3.
- 6) The Container 3 does not implement the service but it has a reference to the component that implements the requested service, and forwards the request to it, in this case component K.

- 7) The component K executes the “save” and returns the result, if it is defined, following the reverse path back to the requester.

It is important to point out that there are no references between the service requester component (“X”) and the service provider component (“K”). Thus, it is possible to change the component that provides the “save” service without modify the rest of structure. In COMPOR-CM, it is also defined an event based interaction model, but the process is very similar and it is omitted from this paper.

B. HCPN Model for COMPOR-CM

In order to construct a formal model that represents the COMPOR-CM specification, a Hierarchical Colored Petri Net (HCPN) model has been constructed regardless of the containers and components structure. Therefore, to construct a specific structure, it is only necessary to define the value of a CPN/ML [5] variable named *System*. The Petri net model captures the control of the interaction among components and containers based on this variable. After the definition of the *System* variable, simulation and analysis can be performed.

The HCPN model for COMPOR-CM is divided into three submodels or pages. Each of these pages are described next.

1) *Environment Model*: For the environment model page, shown in Figure 3, the high level component model interaction is modelled. The page is divided into three parts: *Interface*, *Containers*, and *Components*. The *Interface* part represents the input and output of messages from components and containers. The *Containers* and *Components* parts represent the sending of a message to a container and to a component, respectively.

As can be seen in Figure 3, when a message token (service request or response) is located in the *Container-Level* place, it can only be forwarded to the *ToParent* place or to *ToChild* place. This represents the fact that, for any component, a service request or response can only be sent to the parent container or to one of the child components. Thus, functional components do not interact directly.

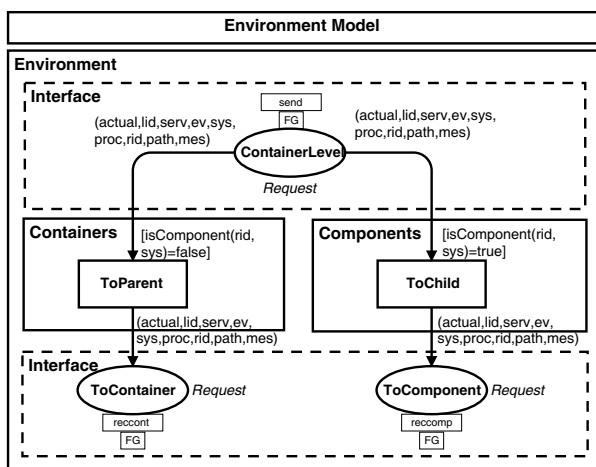


Fig. 3. CPN model for the environment

- 2) *Container Model*: In this page, shown in Figure 4, it is modelled the container behavior for two different situations: when a service request or response is received; and, when an event is received.

This page is divided into four parts: *Interface*, *Service*, *Event*, and *Error*. The *Interfaces* part represents the communication between the *Container* and the *Environment* pages.

The *Error* part represents the situation when there is no parent container for the current container. Also, if the message is a service request, it represents that there is no functional component that implements the service requested.

The *Service* part represents the container when the message is a service request. In this case, the container must look into his children to find one that implements the request. If there is no children implementing the request the container must forward the message to his parent. If there is no parent, the *error* part deals with this message.

The *Event* part represents the container behavior when the message is an event. In this case the container must look into his children to see if they are interested in this message. Moreover, the container forward the message to his parent to continue the broadcast. Again, if there is no parent, the *error* part deals with it.

3) *Component Model*: This page of the model captures the behavior of functional component for two different situations: i) when a service response/request or the announcement of an event message arrives; ii) when a service response/request is sent. This page is graphically divided into four parts: *Interface*, *Service*, *Event* and *Message*.

The *Interface* part represents the communication between the functional component and its parent container. The *Message* part represents the initial state of a component. The *Event* part represents two possible situations for a component: when the component announces an event; and when the component receives an event announcement.

The *Service* part represents the behavior of a component when either a service request or response arrives or when it performs a service request. This part of the model is used to verify if the component implements or not the required service. Also, it models the sending of service requests performed by the component to its parent container.

IV. CASE STUDY: A COMPONENT BASED EMBEDDED AUTOMATION AND CONTROL SYSTEM

In Figure 6 it is presented the architecture for the embedded system, which is based on the framework structure introduced in [11]. The entities shown in this figure are: sensors and actuators, client applications (CAs), the embedded module and the real-time server. In the context of this paper we consider the embedded module and the real-time server.

A. Embedded Module

The embedded module is a real-time system that interacts with the physical environment by means of the sensors and actuators, as well as distributed devices connected

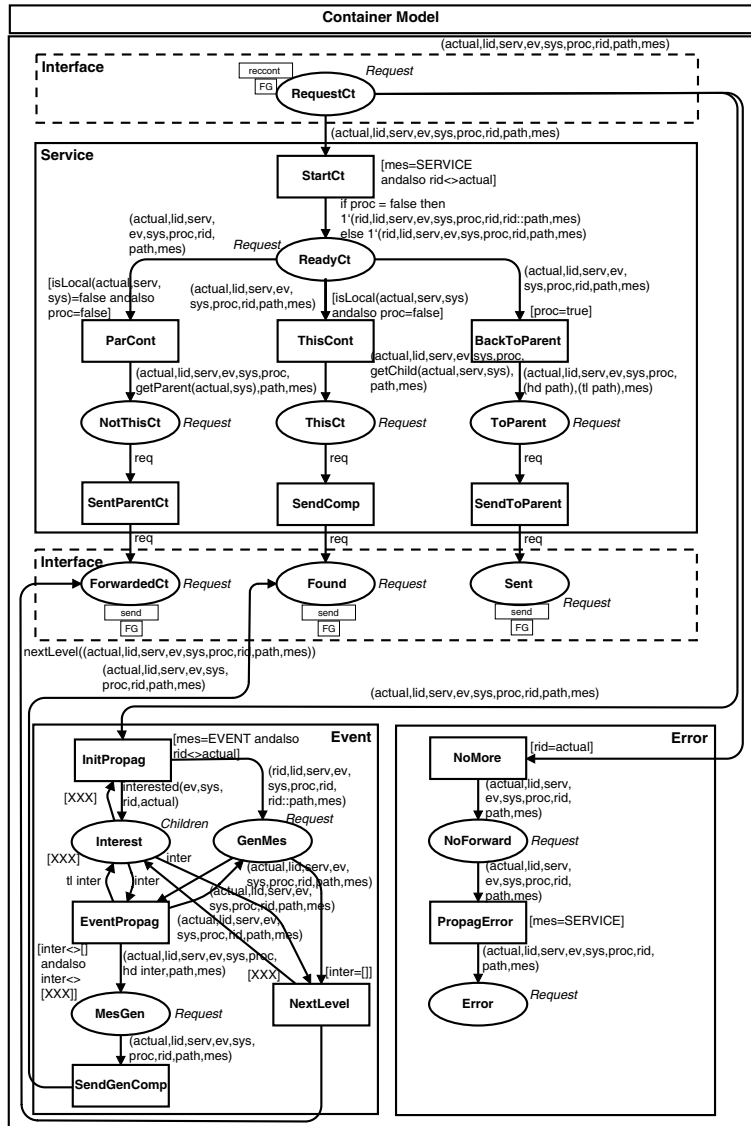


Fig. 4. CPN model for the Container

by a real-time TCP/IP network. The hardware components of the module receive events or alarms from sensors and write information in a shared data area. Then, software components can read such alarm and event information, convert them to an adequate format, and send them to the real-time server.

On the other hand, the real-time server can request for an actuator through the software components that write the request in the shared data area. Thus, hardware components can read the requests from shared area and send them to the specific actuators. The internal architecture for the embedded module is shown in Figure 7. COMPOR-CM is applied in the software components infrastructure, which is depicted in Figure 8.

The I/O interpreter instantiates objects for the information on the shared area. It also receives objects from the system and write the data into the shared area. The data converter converts the data instantiated as objects to the format required by the applications. The synchronizer is used for communication between the embedded module

and the real-time server. The component device controller is used for controlling tasks for the devices such as initialization and tuning.

B. Real-time server

The internal architecture of the real-time server is shown in Figure 9.

The synchronizer component of the real-time server is identical to the embedded module. The data controller is used to route data to the applications and to receive data from them. The user interface module provides the interface between the applications and the system.

V. APPLYING THE APPROACH TO THE CASE STUDY

In this section it is described the application of the approach to the component based real-time embedded automation and control system described in Section IV. Following the development cycle, the requirements phase can be considered the system description previously presented. Thus, the next phase is the architecture definition. According to COMPOR-CM, the embedded system case

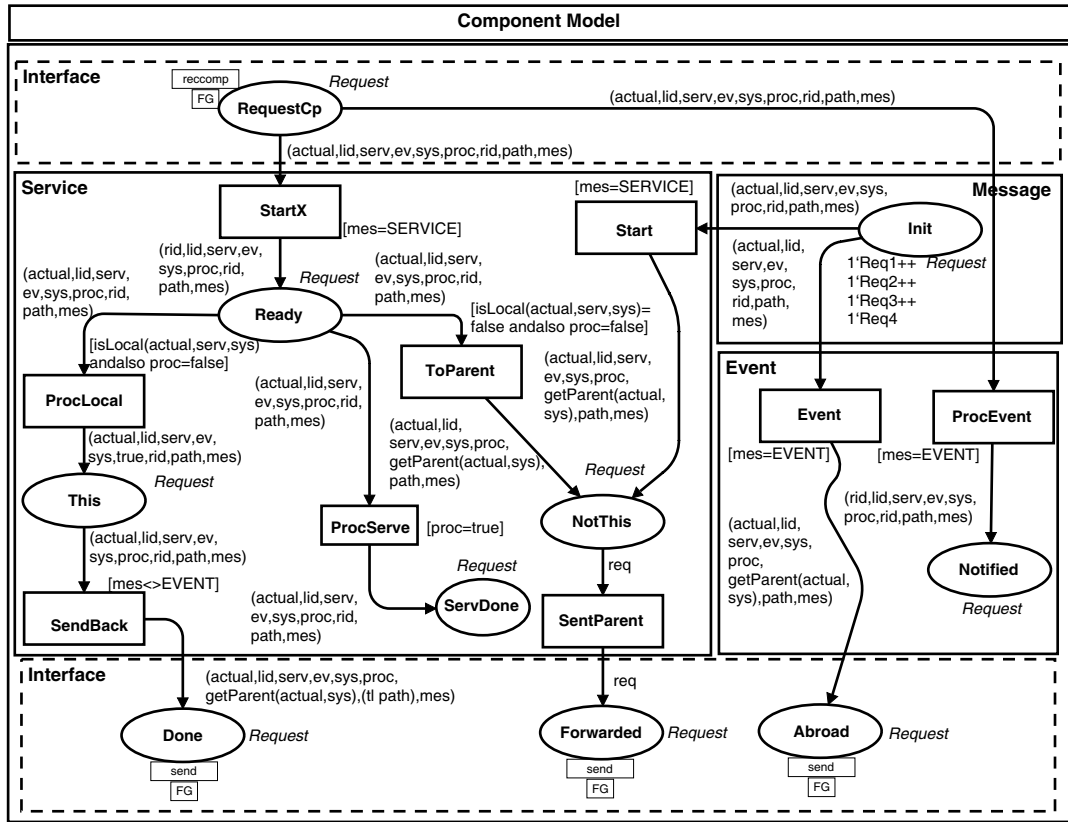


Fig. 5. CPN model for the Functional Component

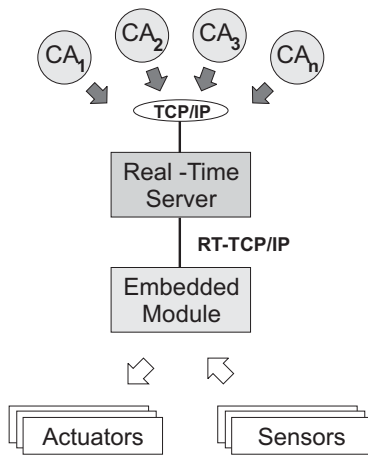


Fig. 6. Architecture of the real-time embedded automation and control application

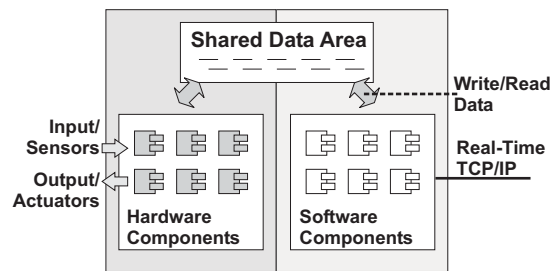


Fig. 7. Internal architecture of the embedded module

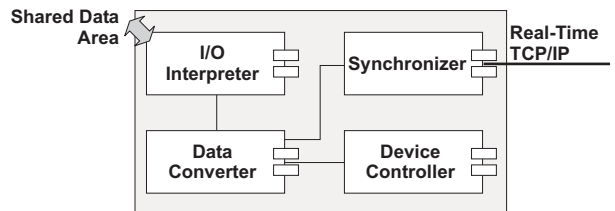


Fig. 8. Software components infrastructure for the embedded module

study can be defined by containers for both embedded and real-time modules. Such containers compose the *System Container*, as illustrated in Figure 10. Functional components are also illustrated.

The *IO Interpreter*, *Data Converter*, *Device Controller* and *Synchronizer* components belong to the embedded module container. In turn, the *UI Module*, *Data Controller*, and *Synchronizer* components belong to the real-time module container. The functionalities previously described for each component are mapped to services. Such services are registered in the containers when the components are deployed. According to the COMPOR-CM specification,

there are no references among the components. Thus, it is possible to define Petri net models for the components regardless the structure. The *System CPN/ML* variable value for the embedded system structure is described below. Services(svcs) and events(avs) descriptions are omitted for clear.

```
System = [
  (IM, [svcs], EMC, [], [avs]),
  (DCM, [svcs], EMC, [], [avs]),
  (DVM, [svcs], EMC, [], [avs]),
```

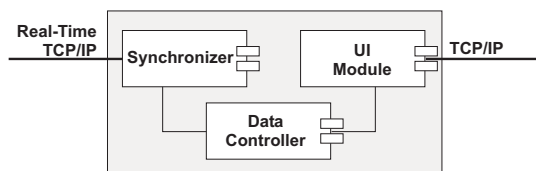


Fig. 9. Software components infrastructure for the real-time server

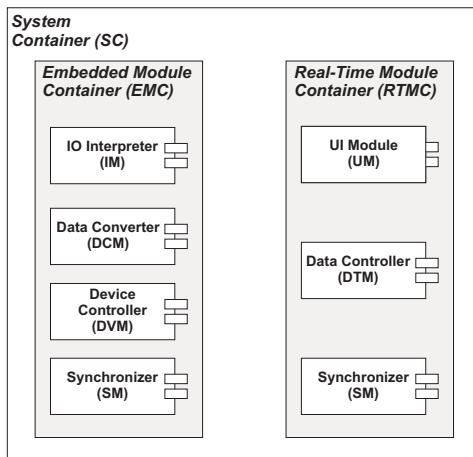


Fig. 10. Embedded system architecture according to COMPOR-CM

```
(SM, [svcs], EMC, [], [evs]),
(UM, [svcs], RTMC, [], [evs]),
(DTM, [svcs], RTMC, [], [evs]),
(SM, [svcs], RTMC, [], [evs]),
(EMC, [svcs], SC, [IM,DCM,DVM,SM], [evs]),
(RTMC, [svcs], SC, [UM,DTM,SM], [evs]),
(SC, [svcs], SC, [EMC,RTMC], [evs])
]
```

After defining this variable value, it is possible to connect Petri net model page instances for each functional component. Such models are the result of the application of the search, adaptation, and component development phases. CPN models for the embedded system components are presented in [10]. The integration of the components are done by the HCPN COMPOR-CM. Thus, it is possible to reduce the effort and time applied to the integration development phase.

VI. RELATED WORK

Some works related to the use of Petri nets and the component based approach have been also proposed. In [12] it is introduced a component based Petri net model for the specification and validation of cooperative information systems. More specifically, it is proposed a framework, called CO-NETS, that integrates object-oriented structuring mechanisms, modularity concepts and some constructions from semantical data modelling into an appropriate variety of algebraic Petri nets.

In [13] it is proposed a formal specification language for modelling distributed systems, called CO-OPN (Concurrent Object Oriented Petri Net). It is based on coordinated algebraic Petri nets. It is described a method for generating an executable prototype from a CO-OPN specification. Moreover, it is provided the generation of Java code that fulfills the *Java Beans* component architecture.

VII. CONCLUSIONS

In this paper the integration of a component based development process and a component model, named COMPOR-CM, for embedded software modelling has been presented. The COMPOR-CM is used as a middleware to integrate components, reducing both development effort and time in the architecture definition and framework conception phases.

The development process, which is based on Hierarchical Colored Petri Net models, was described highlighting the phases that are related with the COMPOR-CM model. Also, the COMPOR-CM Petri Net models are presented, focusing on the mechanisms that implement design flexibilities, such as configurable component architecture.

Moreover, the design of an application based on the introduced process resulting in a software architecture according to the COMPOR-CM model has been presented. The proposed approach is then applied to a component based embedded automation and control system.

As a future work we are developing an architecture and computational tools for specifying, composing, verifying and deploying embedded software components developed in the context of the COMPOR infrastructure.

REFERENCES

- [1] P. C. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*, ser. SEI Series in Software Engineering. Addison-Wesley, Aug. 2001.
- [2] M. Shaw and D. Garlan, *Software architecture: perspectives on an emerging discipline*. Prentice-Hall, Inc., 1996.
- [3] C. Szyperski, *Component Software: Beyond Object-Oriented Programming*. Addison Wesley, 1999.
- [4] W. Damm and M. Cohen, "Advanced Validation Techniques Meet Complexity Challenge in Embedded Software Development," I-Logix White Paper - <http://www.ilogix.com/>, 2004.
- [5] K. Jensen, *Coloured Petri Nets: Basic Concepts, Analysis, Methods and Practical Use*, ser. EACTS - Monographs on Theoretical Computer Science. Springer-Verlag, 1992.
- [6] K. Jensen, Ed., *Fourth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, Aarhus, Denmark, August 28-30. DAIMI, 2002.
- [7] K. Jensen and e. al, "Design/CPN" 4.0, Meta Software Corporation and Department of Computer Science, University of Aarhus, Denmark, 1999, on-line version:<http://www.daimi.aau.dk/designCPN/>.
- [8] L. D. da Silva and A. Perkusich, "Formal verification of component-based software systems," in *Proceedings of The First International Workshop on Verification and Validation of Enterprise Information Systems VVEIS-2003*, Angers, France, Apr. 2003.
- [9] H. O. Almeida, A. Perkusich, E. B. Costa, and R. B. Paes, "COMPOR: a Methodology, a Component Model, a Component based Framework and Tools to Build Multiagent Systems," *CLEI Electronic Journal*, vol. 7, no. 1, 2004.
- [10] L. D. da Silva and A. Perkusich, "A model-based approach to formal specification and verification of embedded systems using coloured petri nets," in *Embedded System Development with Components*. University of Mannheim and Fraunhofer Institute for Experimental Software Engineering, 2004, to appear.
- [11] A. Perkusich, H. O. Almeida, and D. H. Araujo, "A Software Framework for Real-Time Embedded Automation and Control Systems," in *Proceedings of 9th IEEE International Conference on Emerging Technologies and Factory Automation*. Lisboa, Portugal: NJ:IEEE: Piscatway, 2003, pp. 181-184.
- [12] N. Aoumeur and G. Saake, "A component-based Petri net model for specifying and validating cooperative information systems," *Data Knowl. Eng.*, vol. 42, no. 2, pp. 143-187, 2002.
- [13] S. Chachkov and D. Buchs, "From formal specifications to ready-to-use software components: The concurrent object-oriented petri net approach," in *International Conference on Application of Concurrency to System Design, Newcastle*. IEEE Computer Society Press, June 2001, pp. 99 - 110.