Towards Automatic Performance Modelling Using the GENERICA Component Model

Nabila Salmi
MOVEP Laboratory, USTHB
Algiers, Algeria,
LISTIC Laboratory, Université de Savoie
Annecy le Vieux, France
Email: nsalmi@usthb.dz

Malika Ioualalen
MOVEP laboratory
USTHB University
Algiers, Algeria
Email: mioualalen@usthb.dz

Mehdi Sliem
MOVEP laboratory
USTHB University
Algiers, Algeria,
Email: msliem@usthb.dz

***Abstract*—Software designers are often interested in predicting performances of their designed applications, especially for component-based software design where high quality is targeted. In this context, several technics have been proposed. However, none of these approaches has gained widespread industrial use, and automatic tools supporting component-based systems analysis are needed. In this objective, we propose, in this paper, a novel general component model, called *GENERICA*, enabling the description of component-based systems unifying software and hardware components, as well as their deployment and runtime environments and performance characteristics. The aim of this new model is to help designers in deriving automatically performance models, allowing thus automatic qualitative and quantitative analysis of component-based applications, basing on architecture descriptions and component behaviours. The Architecture Description Language (ADL) of GENERICA combines software and hardware components, and allows to describe component-based configurations with performance annotations. Targeted generated performance models consist of Stochastic Petri Nets (SPN) and Stochastic Well-formed Nets (SWN).**

***Keywords*-Component-Based Systems; software component; hardware component; performance annotations; performance modelling.**

I. INTRODUCTION

Component-based design of systems is more and more applied for building modern complex hardware and software systems. In this approach, precompiled elementary components, with explicitly defined provided and required interfaces, are being assembled together [1]. Such systems are known as Component-Based Systems (CBS). The main goals are to improve software quality and to reach reduced cost and easy maintaining and upgrade. Several academic and industrial component models have been developed, such as Fractal [2], EJB [3], CCM [4], AADL [5], Palladio [6], Koala [7], etc.

Very often, designers are interested in predicting performances of their designed systems (such as response times, throughput, etc.), to avoid performance problems after implementation, which can lead to re-designing substantial costs. It would be helpful if the designer can automatically perform an "a priori" analysis of his/her systems. This requires to generate automatic component modelling. As component performance depends not only on implementation, but also on the context the component is deployed in, it would be beneficial if we can get deployment and runtime environment information directly from the system architecture description or from some other component specification tools, to automatically build a performance model for a system. Indeed, we need two information categories: on one side, the runtime environment nature (e.g., hardware components, middleware components, etc.); on the other side, information about context performance (e.g., processor rate, memory space, number of component threads, etc.). Despite the numerous proposed component models, only few of them offer such information or some related specifications, such as Palladio [6] and Procom [8].

On this behalf, we attempt, in this work, to provide a component model with necessary information enabling automatic component performance modelling; we propose a general component model with its Architecture Description Language (ADL) allowing to describe component properties, as well as deployment and runtime environment and performance characteristics. These specifications are used to derive from architecture descriptions and component behaviours an automatic mapping into performance models, without additional modelling. So, we describe here the *GENERICA* model, developed for this aim. Targeted generated performance models are Stochastic Petri Nets (SPN) and Stochastic Well-formed Nets (SWN) [9], which are well-known for their expressiveness and existing performance analysis methods and tools, such as GreatSPN [10].

The paper is organized as follows. Section II discusses related work. Then, Section III presents main requirements of a general component model. We detail, in Section IV, our proposal, the GENERICA component model. Corresponding generated performance models are given in Section V. Section VI introduces the GenTools prototype that we developed to support compilation of architecture descriptions of Generic systems and model generation, and illustrates component modelling with an application example. Section VII concludes the paper.

II. RELATED WORK

Over the last decades, several component models have been proposed, They have been applied to a large spectrum of application domains. Several classifications and surveys have been also achieved, attempting to identify key features of component software approaches [11][12][13][14][15]. According to the classification done by Crnkovic et al. [12], two kinds of component models are distinguished: general-purpose and specialized models. General-purpose models have similar solution patterns, whereas specialized ones have specific domain characteristics Hence, many component characteristics are not always included in existing component model, and no complete or generic component model gathers all component features, except UML/MARTE [16], which is a quite generic model capturing a large number of systems, even if it is hardly used because of its complexity.

Besides, providing deployment, performance and runtime properties in a component model is uncommon. These properties are extra-functional and their specification is still not widespread. In this context, resource usage and some other performance properties have been modelled by few models such as component models are Palladio, SaveCCM, ProCom, Pin and CompoNETs, as given in some surveys [12][13].

The Palladio component model (PCM) [6] is a domain-specific component model, designed to enable early performance predictions for component-based business software architectures. For that purpose, deployment environment and resource allocation to components are specified using a proper domain-specific modelling language. Then, PCM models are created using an integrated modelling environment, called PCM-bench, and performance metrics are derived from these models using analytical techniques and simulation.

SaveCCM [17] is also a domain-specific component model designed for embedded control automative applications, targeting to provide predictable vehicular systems. It considers resource usage and analysability of the dependability and real-time properties. Component behaviour modelling is done using timed automata extended with tasks. Analysis is then performed at design time using a model checker.

Procom [8] is a component model for control-intensive distributed embedded systems. An extra-functional component behaviour is described in a dense time state-based hierarchical modelling language. This behaviour consists namely in timing, resource consumption, component allocations, etc. Pin [18] is a simple component technology, used also in Prediction-Enabled Component Technologies (PECT). It supports prediction of average latency in assemblies and in stochastic tasks, and formal verification of temporal safety and liveness. Finally, CompoNETs [19] is a general-purpose component model, based on CCM, where, additionally, the internal behaviour of a software component and intercomponent communication are specified by Petri Nets. A mapping from the constructs of the component models to Petri Nets is defined.

From these model descriptions, we deduce that some component models are domain-specific, missing genericity, and others support some behavioural or performance specifications, requiring sometimes deployers or experts to provide such information on the designed application. We want to provide designers with tools allowing them to perform "a-priori" analysis of their systems or applications, basing on automatic generated performance models and without requiring an expert intervention. To do so, the component dimension (which deals with general component and assembly properties) and the performance behavioural dimension should be gathered in the same model, to enable automatic component performance modelling and analysis at design time. However, no model includes at the same time the two kinds of properties (component and performance), and if such model exists, often an expert performs this modelling task for performance analysis of designed systems.

Hence, we introduce in this paper a general component model, the GENERICA model, which fares better along the two dimensions, inspired from two models: the Fractal model [2] and the AADL model [5]. These two models comprise many generic features, as well as UML/MARTE, which make them interesting to use, however they lack performance characteristics. Our component model is a combination of common component/assembly features, runtime environment and performance features: It includes genericity features of Fractal, and hardware component features from AADL, but also allows designers to describe runtime and performance characteristics, as attributes of software and hardware components. Thus, specific software and hardware systems, such as embedded systems, can be modelled using GENERICA. Moreover, the specification of all these features with performance annotations is useful to derive directly from architecture descriptions and component behaviours an automatic mapping into performance models, without additional effort modelling, and hence useful to conduct an automatic a priori qualitative and performance analysis of designed systems.

III. Requirements for a general component model

As defined by Crnkovic et al. [12], a component model defines standards for properties that individual components must satisfy, and methods for composing components. Component properties are commonly known as being functional properties and extra-functional specifications (like quality of service attributes). These properties are exposed by means of interfaces, whereas composing components includes mechanisms for component interaction. These mechanisms are mainly bindings defining connections between interfaces. Besides, modern applications generally run in a multi-layered environment. An application is deployed on an application server, which, in turn, runs on some virtual machine (e.g., Java virtual machine, .NET, etc.). The virtual machine works on an operating system (OS), which utilizes some hardware resources. Such configurations highlight several factors, which may influence performances of an application and particularly performances of a component-based application:

- Number of execution flows (threads) of software components,
- Processing rates of hardware components,
- Processing rates of operating systems and middleware under which the application is running,
- Amount of space memory required during execution,
- Amount of necessary resources allocated to component, and
- Parallel applications running under the same operating system.

Consequently, we identify the following main elements that should be allowed by a general component model:

- Software components, which may be primitive or composite, and whose exposed interfaces may be of any kind (service invocation interface or event-based interface),
- Component bindings, being synchronous (invocation service) or asynchronous (event-based) connections.
- Hardware components composing the deployment and runtime environment, and
- Deployment and performance component features (service rates, used memory, required resources, threads, etc.).

These requirements have led to the GENERICA model, which gathers all these characteristics.

IV. The GENERICA Component model

Our model is defined around common component concepts that are components, interfaces and interactions. To be generic, we allow the definition of software components, hardware components and system configurations describing a component-based application deployed on a running environment (as it is shown in Figure 1). Finally, performance annotations are added to describe performance properties. To describe component architectures following our model, we defined the GENERICA ADL, based on a textual XML syntax.

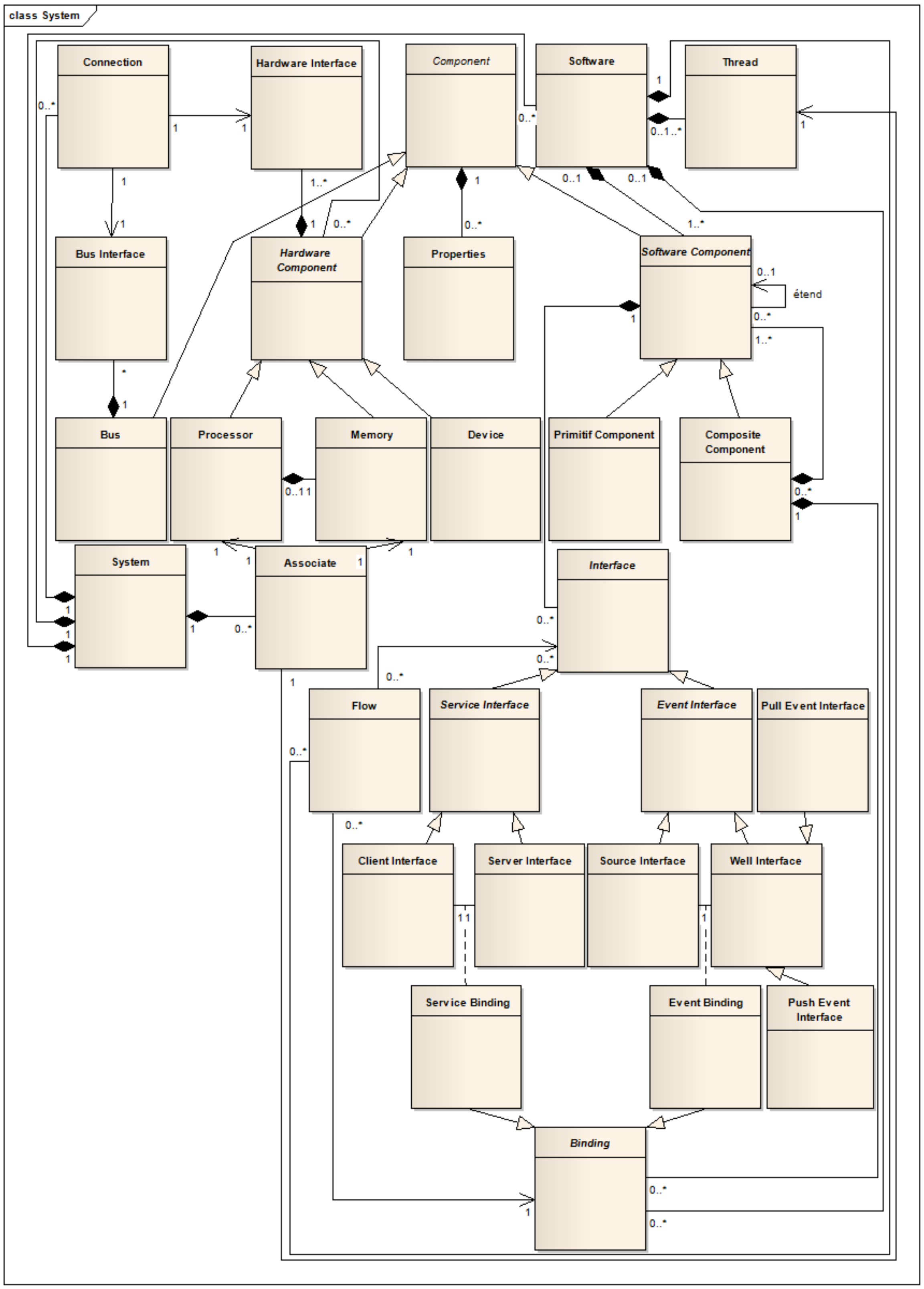


Fig. 2: GENERICA Metamodel

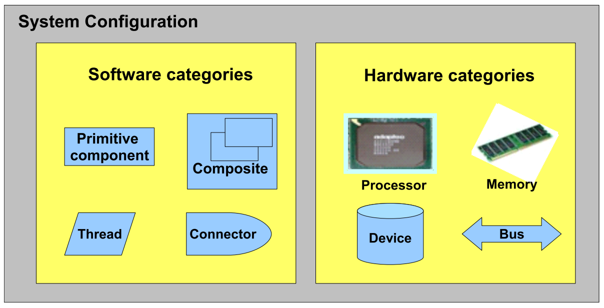


Fig. 1: Software and hardware categories

A. Software components

As in other component models, a software component is made of a content and a set of access points, called interfaces, used for interaction with its environment. The content is either composite, composed of a finite number of other components, allowing components to be nested at an arbitrary level, or be primitive (source code) at the lowest level.

An interface can be a functional interface describing component functionalities, or a control interface for non functional properties such as monitoring and control over execution. To be as generic as possible, two sorts of functional interfaces are defined in GENERICA:

• Service invocation interfaces enabling synchronous communications. Two kinds of interfaces are used: a client interface requesting a service, and a server interface providing the service.

• Event based interfaces, resulting in asynchronous communications. In this case, an event source interface generates events and an event sink interface receives event notifications. The reception of a notification causes the acknowledgment of the reception and execution of a specified handler.

To communicate, components are connected by relating their corresponding interfaces through a binding.

Other characteristics can be described by GENERICA: generalization, component inheritance, connectors, sharing, etc.

B. Hardware components

The interest in adding hardware components to our model is to allow descriptions of the runtime environment and hardware systems as well. These descriptions enable to do a detailed performance modelling, to be used for qualitative and quantitative system analysis while considering the running platform influences. Four kinds of hardware components are defined in GENERICA:

• Processor: models a processor associated to a minimal OS.
• Memory: represents a storage device.
• Bus: acts for all kind of networks or bus communication.
• Device: defines peripheral or resource elements whose internal structure is ignored. Hardware components interact through bus components, instead of using interfaces.

C. Threads

Software components are linked to hardware components by defining component running states being executed on hardware elements. These running states are represented by threads. To describe that in our ADL, one or several threads are first associated to software components; then, these threads are linked to hardware components. At least one thread must be associated to a component application. The main role of thread description is to allow multithreading definition in the generated performance models, so that to enable computation of multithreading impact and performances on the analyzed system.

D. System configurations

A GENERICA system configuration consists of a software application mapped to a hardware platform. The mapping is made by describing a semantic connection (or association) between component threads (defined for the software) and hardware components. Consequently, three parts form the system configuration (see the example below):

• A "software" part, where are described software components,
• A "hardware" part, defining hardware components, and
• An "association" part relating component threads to hardware components.

If the designer wants to describe only a software application, it is possible to omit the hardware part description.

E. Data flows

Sometimes, when invoking a component service, the called component invokes itself a service from another component, which in turn may call another service, etc. So, data cross several components until executing the first requested service. This case corresponds to a data flow, defined as data routing across the system architecture or dependencies between several requests being service invocations or events notifications. For instance, receiving an event notification on a sink interface of a given component may cause service invocation to another component. Data flows are useful to build a complete detailed knowledge about the studied system, which helps in generating a correct modelling. So, it is important to highlight data flows in an architecture description of a system. For that purpose, GENERICA allows to describe data flows as a dependency between a server interface and a client or source interface of the same component, or between a sink interface and a client or source interface.

F. Performance annotations

One of the main contributions of GENERICA is the definition of performance annotations, which will enable later to map components into formal performance models, namely Stochastic Petri Net (SPN) and Stochastic Well-formed Net (SWN) models. For this purpose, we need to specify some information:

• To assess multithreading impact, the number of threads of components and the system configuration is necessary.

• To evaluate service or event processing performances (such as response times or throughput), we need to know the processing rate as well as code size or an estimated execution time of a service or event processing.

• When interest is given to storage size, we need the storage capacity or speed, the data bus speed and dataflow size.

So, we distinguish the following annotations appearing as attributes added to corresponding elements:

• Four annotations for hardware components: data bus speed, processing rate, storage capacity and processor scheduling strategy. Note that this information is useful for assessing software components running even on the same or heterogenous systems with different processor families for instance.

• Four annotations for software components: estimated number of thread instructions, estimated execution time of a service method, dataflow size, and request arrival rate.

All described core concepts of the GENERICA component model are gathered in its metamodel shown in Figure 2.

V. MODELLING WITH SPN/SWN

From the main characteristics of the GENERICA model, we derive a generic approach for automatic model generation of GENERICA systems, inspired from previous work [20]. The proposed modelling is based on the Stochastic Petri Net (SPN) and Stochastic Well-Formed Petri Net (SWN) models; the SWN model being a high level (coloured) model of Petri Nets with probabilistic extensions for performance analysis [9]. These formalisms are state based models, well known for being able to model complex systems with concurrency and conflicts, and widely used for qualitative and performance analysis. In particular, the SWN model is well suited for behavioral symmetries of system's entities.

Let be a GENERICA system defined through an ADL description and a set of Java classes corresponding to primitive components. To generate a model for this system, we first model each primitive component. For this purpose, as a GENERICA component may be made of a local behaviour (set of internal actions) and a set of interfaces, we defined basic SPN/SWN models for interfaces and internal component behaviour. Using these basic models, each primitive component is modelled. Finally, we generate the GENERICA system global model, using previously generated SPN/SWN models. This global model highlights components and component communication, and hence, bottlenecks can be detected within this model. More details can be found in [20].

VI. ILLUSTRATION

A first tool helping in building an GENERICA architecture description has been developed: the GENERICA toolbox.

A. The GENERICA tool prototype

The GENERICA component model ADL has been implemented into a Java prototype GenTools, using the Java language. This prototype provides an editor for introducing an ADL system description, a compiler to check syntactical and semantical errors and an SPN/SWN model generator for primitive components and for the whole application. To do a qualitative or/and performance analysis of generated models for a given application, we need to use existing SPN/SWN analysis tools such as the GreatSPN tool [10]. It would be interesting to have such analysis automated after model generation. This is one of our future work. The user interface of the GENERICA toolbox is depicted in Figure 3, showing an application example with its generated model.

B. Running example

To illustrate the description of a component-based application using the GENERICA model, we use a typical industrial application (Figure 4), the stock quoter system, which is an extended version of an application presented in [21]. This application is a system managing a stock information database, chosen mainly for its components exposing, at the same time, service invocation and event-based interfaces. When the values of particular stocks change, a StockDistributor component sends an event message that contains the stock name to two StockBroker components. If the first StockBroker component is interested in the stock, it can obtain more information about it, by invoking a service operation offered by an Executor component. This latter processes the received request, generates data and invokes itself a service request from a persistence server component to save its results. Besides, if the second StockBroker component is interested in the stock, it processes locally the event. Figure 4 shows the interactions between the different components.

```
<system name="Stock Quoter">
 <!-- Hardware architecture -->
 <hardware>
  <components>
    <processor name="processor1">  <interface name="IntP"/> </processor>
    <memory name="memory1">  <interface name="IntM"/> </memory>
     ...
   </components>
  <connections>
    ...
  </connections>
 </hardware>
 <!-- Software architecture -->
 <software name="Stock application">
  <components>
   <component name="StockDistributor1">
     <interfaces> <event name="newst" role="source"/>  </interfaces>
   </component>
   <component name="StockBroker1">
     <interfaces> <event name="rec" role="sink"/>  <service name="rqst" role="client"/>
     </interfaces>
     ...
  </component>
   .. .
  <component name="Executor">
     <interfaces> <service name="prc" role="service"/> <service name="save" role="client"/>
     </interfaces>
     ...
  </component>
   ...
  </components>
  <bindings>
   ...
  </bindings>
  <flows>
   <flow name="fl1" in-component="StockBroker1" for-interface="rqst" for-dependency=
     "execute" take-binding="BrokExec"/>
     ...
  </flows>
  <threads><thread name="thread1" /></threads>
 </software>
 <!--Hardware&Software Associations-->
 <associations>
   <associate name="ass1" processor="processor1" memory="memory1" thread="thread1"/>
 </associations>
</system>
```

Fig. 5: Part of the GENERICA ADL of the application example

Figure 5 shows a part of the architecture description of the application using the GENERICA ADL.

The generated SPN global model is depicted on the user interface of Figure 3.

VII. Conclusion and Future work

This paper presented the GENERICA component model, a new general model, which deals with two dimensions: the component dimension describing general component and assembly characteristics, and the performance behavioural dimension related to deployment, runtime environment and performance properties. An ADL language has been also proposed for GENERICA, as well as a corresponding performance modelling approach based on SPN and SWN models. The long-term objective of introducing such a general component model, is to enable automatic performance component modelling and hence automatic a priori qualitative and performance analysis of component based systems. Even if introduction of a generic component model may lead to a complex specification, we think of its usefulness for several design fields such as embedded systems. This component model has been implemented into a Java toolbox prototype, the GenTools toolbox, supporting compilation of ADL descriptions and model generation. The tool has been experimented on several GENERICA applications.

However, still more research work is required in several directions, such as integrating the GENERICA toolbox in a global modelling and analysis tool, starting from the ADL description and automatic modelling, and resulting in performance computations, given specification of performance indexes of interest. We also target to use the automated modelling of primitive components in a compositional analysis step, based on components models, to have time and memory savings during models analysis. This can be done thanks to our previous work [20], which defined a structured performance analysis method for analysing a CBS in an efficient way allowing, to reduce computation times and memory usage (basing on primitive component models rather than the global net). Finally, we are working on modelling reconfiguration features of GENERICA CBSs and verification of their behaviours.

References

[1] C. Szyperski, *Component software*, 2002, vol. 2nd Edition.

[2] E. Bruneton, T. Coupaye, and J. Stefani, "The fractal component model, version 2.0-3," http://fractal.ow2.org/specification/ (October 2013), Tech. Rep., Feb 2004.

[3] Sun Microsystems, "EJB 3.0 specification," http://www.oracle.com/technetwork/java/index.html, Jul 2007.

[4] Object Management Group, "CORBA component model specification. version 4.0," http://www.omg.org/spec/CCM/4.0/ (October 2013), Apr. 2006.

[5] SAE, "Architecture analysis et design language (aadl)," SAE Standards AS550, Tech. Rep., November 2004.

[6] S. Becker, H. Koziolek, and R. Reussner, "Model-based Performance Prediction with the Palladio Component Model," in *WOSP2007*. Buenos Aires, Argentina: ACM Sigsoft, 2007.

[7] R. van Ommering, F. van der Linden, J. Kramer, and J. Magee, "The Koala component model for consumer electronics software," *IEEE Computer*, vol. 33, no. 3, pp. 78–85, Mar. 2000.

[8] S. Sentilles, A. Vulgarakis, T. Bures, J. Carlson, and I. Crnkovic, "A component model for control-intensive distributed embedded systems," in *CBSE*, 2008, pp. 310–317.

[9] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad, "Stochastic well-formed colored nets and symmetric modeling applications," *IEEE Trans. on Comp.*, vol. 42, no. 11, pp. 1343–1360, Nov 1993.

[10] Perf. Eval. Group, "GreatSPN home page: http://www.di.unito.it/~greatspn," Torino, Italy, 2002.

[11] H. Aris and S. S. Salim, "State of component models usage: justifying the need for a component model selection framework," *Int. Arab J. Inf. Technol.*, vol. 8, no. 3, pp. 310–317, 2011.

[12] I. Crnkovic, S. Sentilles, A. Vulgarakis, and M. R. Chaudron, "A classification framework for software component models," *IEEE Transactions on Software Engineering*, vol. 37, pp. 593–615, 2011.

[13] J. Feljan, L. Lednicki, J. Maras, A. Petricic, and I. Crnkovic, "Classification and survey of component models," Målardalen University, Technical Report ISSN 1404-3041 ISRN MDH-MRTC-242/2009-1-SE, December 2009.

[14] K.-K. Lau and Z. Wang, "Software component models," *IEEE Trans. on Software Engineering*, vol. 33, no. 10, pp. 709–724, October 2007.

[15] N. Medvidović and R. N. Taylor, "A classification and comparison framework for software architecture description languages," in *IEEE Trans. On Soft. Eng.*, vol. 26, no. 1, 2000, pp. 70–93.

[16] S. Taha, A. Radermacher, S. Gérard, and J.-L. Dekeyser, "Marte: Uml-based hardware design from modelling to simulation," in *FDL*, 2007, pp. 274–279.

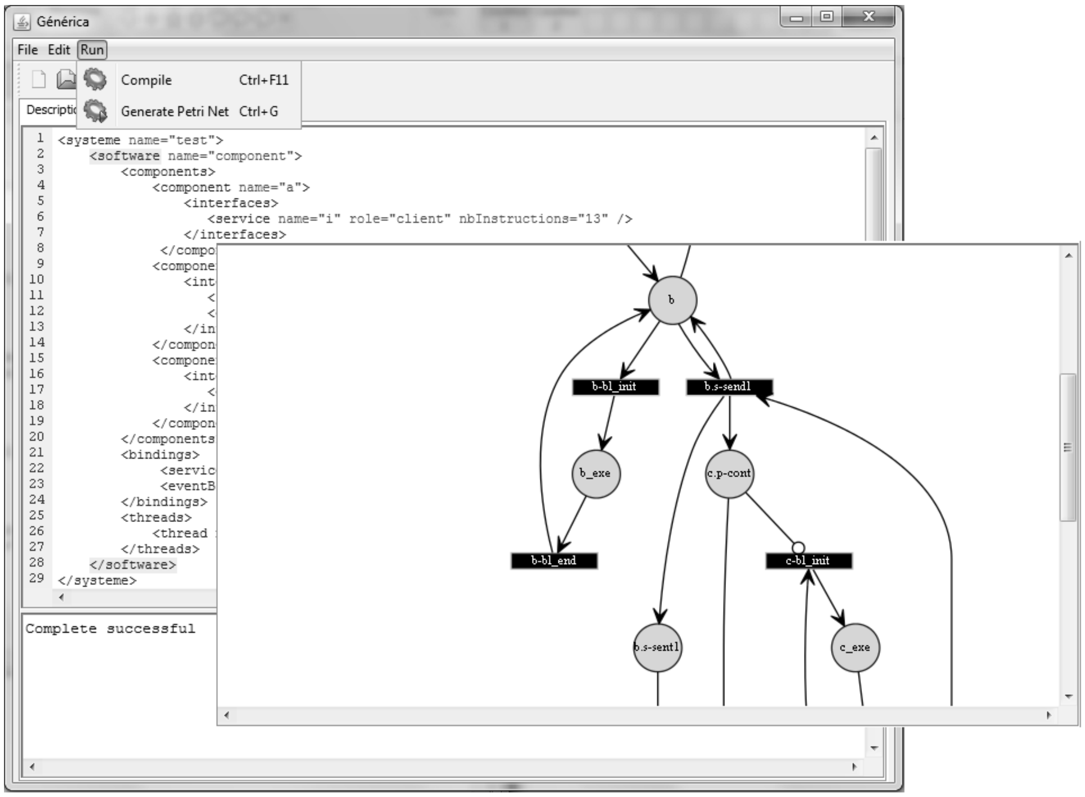
[17] M. kerholm, J. Carlson, J. Fredriksson, H. Hansson, J. Håkansson, A. Möller, P. Pettersson, and M. Tivoli, "The save approach to component-based development of vehicular systems," *J. Syst. Softw.*, vol. 80, no. 5, pp. 655–667, May 2007.

[18] S. Hissam, *Pin Component Technology (V1.0) and Its C Interface*, ser. Technical note. Carnegie Mellon University, 2005.

[19] R. Bastide and E. Barboni, "Component-Based Behavioural Modelling with High-Level Petri Nets," in *MOCA '04- Third Workshop on Modelling of Objects, Components and Agents , Aahrus, Denmark , 11/10/04-13/10/04*. DAIMI, October 2004, pp. 37–46.

[20] N. Salmi, P. Moreaux, and M. Ioualalen, "Structured performance analysis for component based systems," *International Journal of Critical Computer-Based Systems (IJCCBS)- Part II - Issue 1/2*, vol. 3, no. 1, pp. 96–131, 2012.

[21] D. Schmidt and S. Vinoski, "Object interconnections: The CORBA component model: Part 2, defining components with the IDL 3.x types," *C/C++ Users Journal*, April 2004.



```
<systeme name="test">
    <software name="component">
        <components>
            <component name="a">
                <interfaces>
                   <service name="i" role="client" nbInstructions="13" />
                </interfaces>
             </compo
            <compone
                <int
                   <
                   <
                </in
            </compon
            <compone
                <int
                   <
                </in
            </compon
        </components
        <bindings>
             <servic
             <eventB
        </bindings>
        <threads>
            <thread
        </threads>
    </software>
</systeme>
```

```
Complete successful
```

Fig. 3: User interface of the the GenTools tool

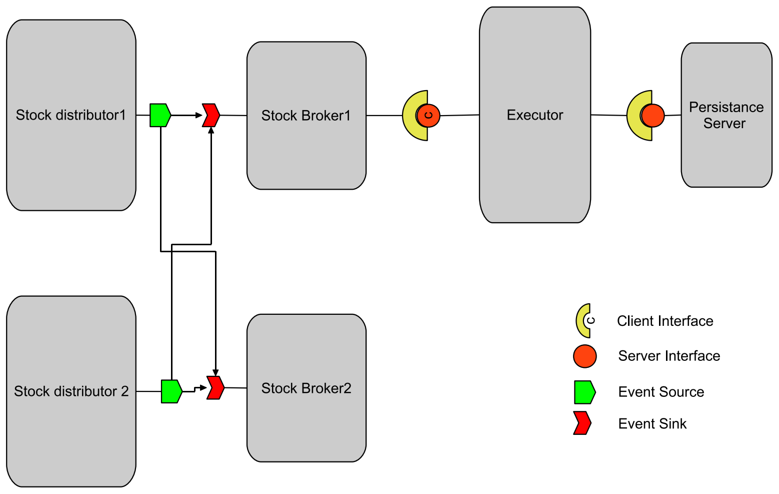


Fig. 4: Application example