CARAT: a Toolkit for Design and Performance Analysis of Component-Based Embedded Systems

Egor Bondarev, Michel Chaudron
Eindhoven University of Technology
5600 MB, Eindhoven, The Netherlands
e.bondarev@tue.nl

Peter H.N. de With
LogicaCMG / Eindhoven Univ. of Tech.
5605 JB, Eindhoven, The Netherlands
p.h.n.de.with@tue.nl

Abstract

Solid frameworks and toolkits for design and analysis of embedded systems are of high importance, since they enable early reasoning about critical properties of a system. This paper presents a software toolkit that supports the design and performance analysis of real-time component-based software architectures deployed on heterogeneous multi-processor platforms. The tooling environment contains a set of integrated tools for (a) component storage and retrieval, (b) graphics-based design of software and hardware architectures, (c) performance analysis of the designed architectures and, (d) automated code generation. The cornerstone of the toolkit is a performance analysis framework that automates composition of the individual component models into a system executable model, allows simulation of the system model and gives design-time predictions of key performance properties like response time, data throughput, and usage of hardware resources. We illustrate the efficiency of this toolkit on a Car Radio Navigation benchmark system.

1. Introduction

Design and development of current software-intensive systems requires support from powerful Computer-Aided Software Engineering (CASE) tools. Software tools, like Rational Rose [2], have proven to be efficient for design phases of the software development. They provide a broad functionality from a diagram consistency checking to state-chart simulation and code generation.

However, for the development of real-time embedded system, the above software tools often lack a decent function for performance verification. Real-time systems are characterized by their strict end-to-end response-time, throughput, and robustness requirements. Early verification of these constraining requirements already at the design phase reduces technical risks and production costs. Software tools (like VTune [4] and HProf [5]) exist that provide full-fledged functionality for all kinds of performance verification and optimization, but they require source code of an application and cannot be used at the design phase.

The software tools providing both design and performance-analysis facilities can be divided into two categories: commercially available and academia-based ones. The LinuxLink [6] toolkit from TimeSys is an example of a commercial tooling environment that enables design, performance assessment and optimization of software products on Linux platforms. The performance assessment is provided by an embedded simulator based on Rate-Monotonic Analysis (RMA), which limits its applicability when a heterogeneous hardware platform is used. Academia-based tools provide various analysis techniques, ranging from formal methods to simulation-based techniques. The Sesame environment [14] features modeling and simulation methods and tools for the efficient design of heterogeneous embedded multimedia systems. The Real-Time Calculus Toolbox [7] deploys an efficient analytical approach with algebra operators for resource load and events curves.

Presently, a trend is noticeable to build complex software according to principles of component-based software engineering (CBSE). CBSE brings a number of valuable benefits for an embedded system designer. It enhances the design modularity through the component encapsulation and explicit interface descriptions, and it allows the reuse of software entities, which reduces production cost. However, none of the above-mentioned tools support design and performance analysis of component-based systems. The amount of available CASE tools for component-based real-time systems is rather limited. CB-SPE Tool [9] features graphical design and performance assessment of systems built from conventional software components. It adopts RT-UML profile annotations for components and composes these annotations into a system Queuing Network (QN) model at the component-assembly phase. Analysis of the QN model leads to the predicted system performance. The SEESCOA Tool [8] enables designing software systems from components with specified timing contracts and run-time monitoring of these contracts. Both of the tools provide solid performance-assessment functionality. However, these tools lack design-support features, like repository, large-scale visualization and automated code generation. The SAAM tool [1] based on the powerful SAAM method, provides architecture evaluation of various extra-

978-3-9810801-2-4/DATE07 © 2007 EDAA

functional properties. Unfortunately, the tool does not address performance analysis at sufficient level of detail and accuracy. The DARPA Evolutionary Design of Complex Software (EDCS) program [11] focuses on the development and integration of tools to support architecture composition and evaluation of CORBA-based systems. Rapide is a primary example of an EDCS tool. It allows to browse through and animate complex sequences of events generated by an architecture simulation. The tool is efficient for understanding and validating the complex behavior of distributed component-based architectures.

Contribution. In this paper we present the CARAT (Component Architectures Analysis Tool) toolkit for design and performance analysis of real-time component-based software systems. The toolkit supports the complete design cycle and consists of the following integrated tools: a repository, graphical editors, a performance analyzer, a visualizer and code generator. The underlying methodology for performance analysis (PA) is based on *composable component models* and *simulation* of multitasking hardware resources.

The rest of this paper is as follows. Section 2 explains the overall PA methodology. Section 3 describes the architecture of the toolkit. Section 4 discusses the benefits and limitations of the toolkit.

2. Performance-Analysis Methodology

This section briefly describes our methodology, proposed earlier in [12], for design-time performance analysis of CB architectures on multiprocessor platforms. Fig. 1 shows an overview of the methodology, which consists of three (partly) iterative design phases. The *Modeling* phase involves software component and hardware IP (Intellectual Property) blocks development. It results in composable software and hardware component models, representing abstract specification of the component behaviour and resources provision/requirements. The *System Design* phase aims at the software and hardware composition and deploys rules for composition of the above-mentioned models into an executable system model. The *Performance Analysis* phase includes a number of techniques enabling prediction of the system behaviour and performance by simulation of the obtained executable system model.

The component models are the cornerstone of the framework. The models of the same type are composable to enable an automated composition of the system model. The models are stored as XML files in a component package.

For software components, the framework introduces three types of models: resource, behaviour and process models. Typical models for hardware IP blocks are memory, communication and processor performance models. The *resource model* specifies resource requirements (e.g. number of claimed CPU instructions) of each accessible individual operation of a component. The *behaviour model* describes the operation's underlying calls to operations of other interfaces of other components. The *process model* specifies the processes activated and running within an active component. Note that all these component models should be supplied by the component provider.

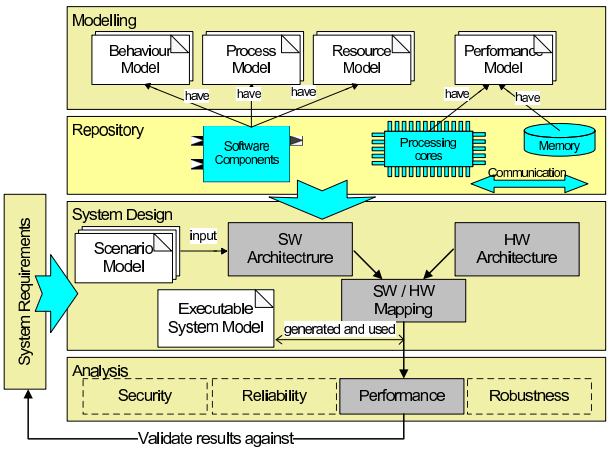

Figure 1. The performance analysis methodology.

The PA framework introduces *performance models* for hardware IP blocks. For instance, a performance model for a processing core defines its instruction type (RISC, CISC or VLIW) and execution frequency. The data for performance models can be obtained from supplier data sheets.

As input for the system-design phase, the designer has system requirements and various third-party hardware and software components stored with their corresponding models in a repository. The designer selects the software components that together satisfy functional requirements and *may* satisfy extra-functional requirements. The designer specifies component composition by tailoring, instantiating and connecting the selected services. A hardware-architecture specification can be done in parallel. In most of the cases, a hardware platform is pre-specified. If not, the designer selects hardware components from a repository and chooses a specific topology, number of processing nodes, types of memory, communication means and scheduling policies.

Once the software and hardware architecture are specified, the mapping of the software components on the hardware nodes can be made. The mapping defines for a component on which processor it will be executed.

The model-composition rules, defined by the framework, are used to synthesize all involved component and hardware models into an *executable system model*. Briefly, the system model represents a set of tasks (running in a system) with a detailed description of: (a) synchronization constraints between the tasks, (b) a sequence of operations and interactions executed by each task, (c) execution time and communication load imposed by each operation within a task, and (d) task period, jitter, offset and deadline.

The obtained executable system model is used for performance analysis. The PA results in predicted system properties (throughput, task latencies, number of missed deadlines, utilization of HW resources) that should be validated against the specified system requirements. If there is a mismatch between them, the system-design and analysis phases should be re-iterated. During the iteration, the designer

may: (a) try out other available software components and hardware IP blocks, (b) make a different SW/HW mapping or (c) apply different scheduling policies.

3. Architecture of the CARAT Toolkit

The CARAT toolkit is implemented in Java and realized as a number of Eclipse plug-ins. This enables easy installation and high portability. The data specification and exchange between these plug-ins are organized by XML-based model structures. The architecture of the toolkit, which is depicted in Fig. 2, consists of the following modules: Repository, Graphical Designer, Performance Analyzer, Visualizer and Code Generator. A brief description of these modules is given in the following paragraph.

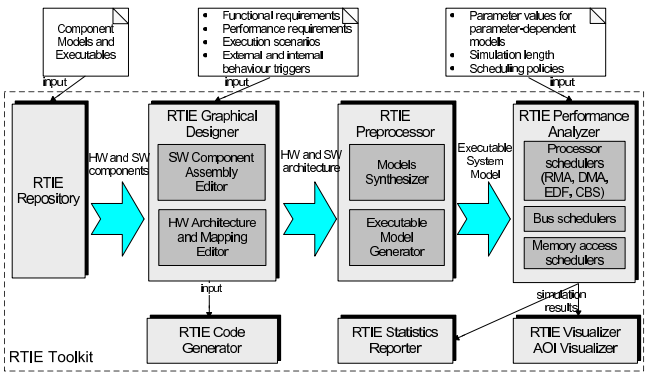

Figure 2. Architecture of the CARAT toolkit.

The *Repository* provides storage and retrieval of executables of software components and various models of software components and hardware IP (Intellectual Property) blocks. The *Graphical Designer* contains two editors for constructing (a) software component assemblies, and (b) hardware resource topologies with assigned deployment (mapping) of the software components. The *Preprocessor* takes the data from the Repository (model specifications) and from the Graphical Designer (defined architectures), and synthesizes the models into an executable system model, containing description of the tasks running in the system. The *Performance Analyzer* uses this model as an input for virtual scheduling of the tasks on the corresponding hardware resources. The Performance Analyzer outputs an execution timeline (behaviour) for each task on every hardware resource (processor, memory bank and bus). The predicted timelines are interactively drawn by the *CARAT Visualizer*. The *Statistics Reporter* provides the designer with a broad range of data on the predicted performance properties. If the predicted performance satisfies the performance requirements, the *Code Generator* can generate the application "glue code" that instantiates and binds the software component executables according to the specified component assembly.

CARAT Repository. This tool provides remote storage of third-party component executables and their corresponding models. It allows the designer to search for components satisfying input criteria and view the component specifications. The Repository is accessible from the Graphical Designer, so that the components instantiation on the graphical canvas can be performed with a simple drag-and-drop procedure.

CARAT Graphical Designer. This module enables specification of SW component assemblies and HW architectures (see Fig. 3).

In the SW Assembly Editor, the designer instantiates and binds the provides and requires interfaces of the components together, thereby specifying the static structure and communication topology of the system. The designer may also specify external stimuli (environmental/user events or interrupts) for the software assembly, which will be used at the performance-analysis phase.

Fig. 3 depicts a snapshot of the design process of a car radio navigation (CRN) system [13], on which we have validated our PA methodology and tested this CARAT toolkit. Among the functional requirements, we had the following performance requirements for the CRN system.

RTR1: The response time of the operation "change the sound volume" is less than 200 ms (per one knob grade, the knob has 32 grades).

RTR2: The response time of the operation "find and retrieve an address specified by the user" is less than 200 ms (minimal inter-arrival time of the entry is 1000 ms).

RTR3: The response time of the operation "receive and handle Traffic-Message-Channel message" is less than 350 ms (minimal inter-arrival time of the messages is 3000 ms).

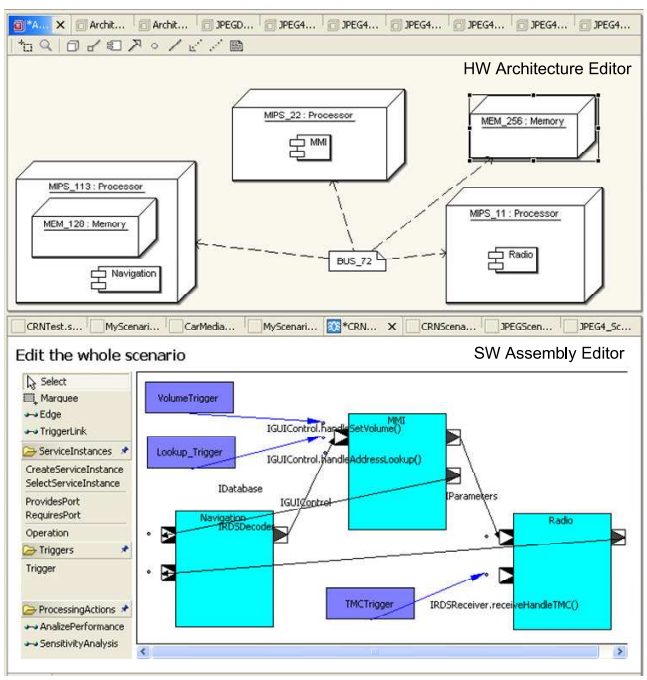

Figure 3. Graphical Designer consisting of HW Architecture Editor and SW Assembly Editor.

In the SW Assembly Editor we instantiated the following three Robocop software components [3] in order to satisfy functional system requirements.

- The Man-Machine Interface (*MMI*) component, which takes care of all interactions with the end-user, such as handling key inputs and graphical display output.
- The *Navigation* component, which is responsible for destination entry, route planning and turn-by-turn route guidance giving the driver visual advices. The navigation functionality relies on the availability of a map database and positioning information.
- The *Radio* component, which is responsible for tuner and volume control as well as handling of TMC traffic information services.

The MMI component provides its functionality via the IGUIControl interface and requires to be bound to IParameters and IDatabase interfaces. The Navigation component provides IDatabase and IRDSDecoder interfaces and requires IGUIControl interface. The Radio component provides IParameters and IRDSReceiver interfaces and requires an IRDSDecoder interface. By binding the interfaces we define possible component communication.

These three components are not active (i.e. they have no active processes implemented inside the components). The system behaviour can be triggered by the user or environmental events. To emulate these events, we created and parameterized the three following stimuli (dark-grey boxes in the SW Assembly Editor) that actually trigger the behaviour of the CRN system.

The VolumeTrigger emulates the user "change the sound volume" event. The deadline for the system response time on this event was set to 200 ms, according to the requirement RTR1. The minimal inter-arrival time of this event entering the system has been calculated as follows. We assumed that the user can make the complete turn of the sound knob within 1 second. The knob has 32 grades. The min. inter-arrival time for one grade is 1 sec / 32 grades = 31 ms.

The LookupTrigger emulates the user event "find and retrieve an address". According to the requirement RTR2, the minimal inter-arrival time and response deadline for these events were set to 1000 ms and 200 ms, respectively.

The TMCTrigger emulates the TMC messages that arrive to the Radio component from a TMC station. According to the requirement RTR3, the minimal inter-arrival time and response deadline for these messages were set to 3000 ms and 350 ms, respectively.

In order to activate the above-mentioned stimuli, we connected each of them to the component operations that they trigger first, once they occur.

We have observed that these three events may occur in parallel. Therefore, we set all three behaviour triggers to form a critical execution scenario. We call the execution scenario critical, when it can impose a resource overload or create a hazard for fulfilling any performance requirement. Note that many scenarios are possible for one software architecture. Every scenario is represented in a separate page of the SW Assembly Editor. Once the scenarios are defined, the hardware architecture and SW/HW mapping can be specified in the HW Architecture Editor.

The HW Architecture Editor is a graphical canvas located on top in Fig. 3. It allows (a) selection of the processors, memory blocks and communication lines from the repository, (b) creation of an arbitrary topology from the selected elements and (c) mapping of the involved software components onto the hardware elements. The processors and memory blocks can be connected by any type of communication lines in various styles, like star, token ring and mixed forms. A memory block can be specified as a local (in processor) or global memory. The executable components can be mapped onto processing nodes, while components representing buffers can be mapped onto memory blocks.

Preprocessor and Performance Analyzer. This CARAT module is a computationally complex part of the toolkit. As input, it takes both the specified software and hardware architectures, and the supplementary models of the involved software components and hardware blocks. The tool synthesizes this set of data into an executable system model. This automated synthesis is possible because of the compositional nature of the component behaviour, process and resource models. The system model, outlined in Section 2.2, represents tasks running in the system.

For the CRN system scenario, described above, the Preprocessor synthesized three tasks (according to the number of triggers). Each task is periodically initiated by a corresponding stimulus. The periodicity and deadline parameters of a task were assigned in accordance to the causing trigger parameters. The sequence of component operations involved in each task has been constructed from the behaviour models of participating components. The processor, communication and memory load imposed by each task have been extracted from the component resource models.

The *CARAT Performance Analyzer* provides wide set of schedulers for processors and communication lines and enables discrete-event simulation of the system model.

The designer specifies simulation period, operation execution times for worst, best and average cases, and scheduling algorithms for hardware resources (see Fig. 4.a). The simulation unit can be set to millisecond or microsecond.

Once the configuration settings are defined, the executable model can be simulated (virtually scheduled) on the specified hardware nodes. The simulation results in a task-execution timeline for each processing node, timeline of data occupancy for busses and buffers. Besides, the results include the found maximum latency and resource load for tasks, data throughput and total resources utilization.

The resulting performance analysis of the CRN system is given in the next paragraphs.

CARAT Visualizer. This tool shows two types of data:

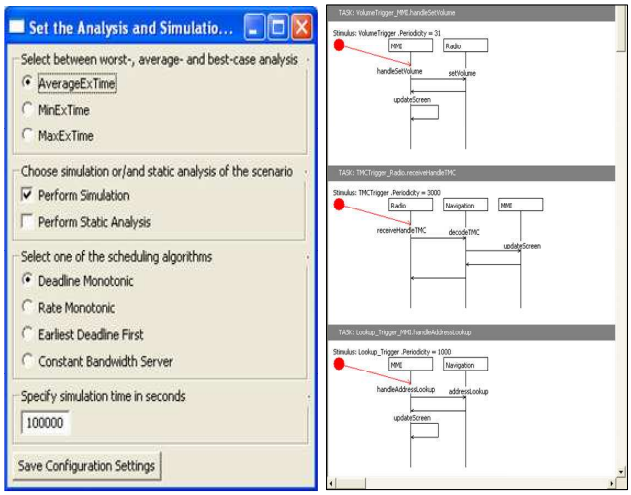

Figure 4. a) Simulation settings window; b) Synthesized MSCs for the three CRN tasks.

synthesized message-sequence chart (MSC) for a task, and task-execution timeline of the involved hardware resources.

For the above-specified CRN system scenario, the Visualizer has depicted three MSC diagrams (see Fig. 4.b). For example, the second MSC shows that the task instance is triggered by TMCTrigger every 3000 ms and consists of the three following operations: receiveHandleTMC(), decodeTMC() and updateScreen().

The predicted task-execution timelines for the three processors and the bus load in the CRN system are given in Fig. 5. For example, the timeline of the processor MIPS_22 shows that the three tasks share this resource and interleave with each other. For each task instance, the timeline shows the start, execution and completion times together with the task deadline. Fig. 5 shows that the first instance (job) of the TMCTrigger_Radio task failed to meet the deadline. The analysis of the timelines led to the conclusion that this happens due to the high MIPS_22 processor-load which is caused by the higher-priority task VolumeTrigger_MMI.

The bus-load timeline shows the available bus bandwidth at each moment. The numbers on top of the bus-usage peaks specify IDs of the tasks transferring data over the bus.

Statistics Reporter. This tool gathers all relevant information from the simulation, processes it and stores the data in a text file. The performance statistics resulting from the simulation of the CRN system are depicted in Fig. 6. The PA of the CRN system showed that the task initiated by VolumeTrigger takes 90.04% of the MIPS_22 processor. The total load of this processor equals to 93.76%. Similar type of data for other processors is also shown in Fig. 6.

The designer of the CRN system is primarily interested in the data related to the response-time requirements RTR1-3. The data represented in the lower part of the statistics file shows worst- and best-case response times and number of missed deadlines for each of the task. The analysis of the

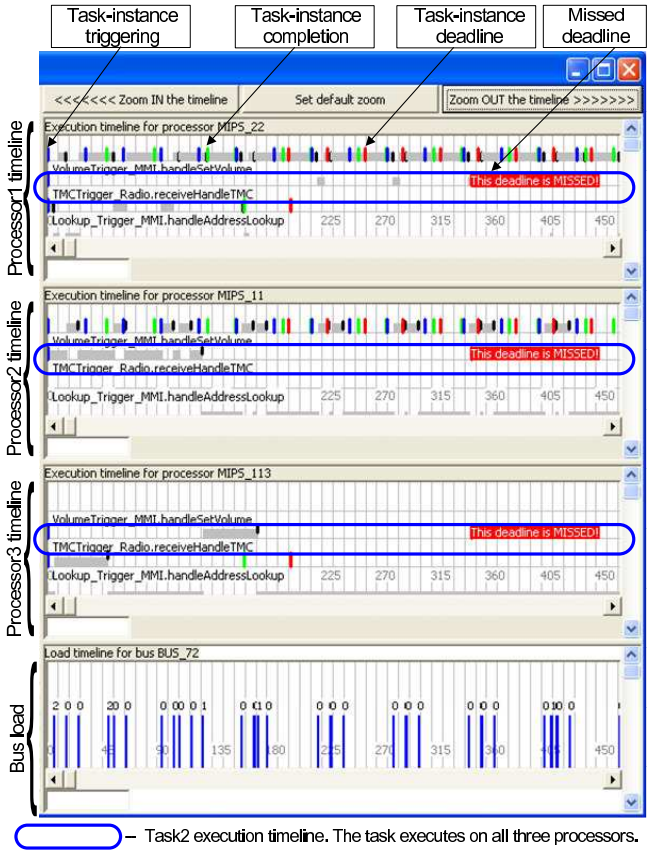

Figure 5. Processor- and bus- load timelines.

data leads to the following conclusions. The task initiated by VolumeTrigger (with correspondent requirement RTR1 = 200 ms) has the worst response time = 69 ms. All the 3235 instances of the task met the specified deadline. The task fired by LookupTrigger (with correspondent RTR2 = 200 ms) has the worst response time = 310 ms. Two task instances out of 105 instances missed the deadline. The task initiated by TMCTrigger (with correspondent RTR3 = 350 ms) has the worst completion time = 551 ms. All 48 jobs of the task missed the deadline. Further analysis led us to the conclusion that the processor MIPS_22 is over-utilized, while the powerful processor MIPS_113 is over-specified (load of 6.16%). The suggestion has been made to re-map the MMI component onto this powerful processor.

4. The Toolkit Properties

Completeness and Consistency Checks. The CARAT toolkit provides completeness and consistency checks between various design diagrams. It identifies (a) missing component models in the repository, (b) erroneous bindings between provides and requires interfaces, (c) absence of software components on the correspondent hardware-mapping diagram, and (d) missing hardware connections for software component bindings.

Robustness and Efficiency. We have validated the toolkit by designing and deploying a number of embed-

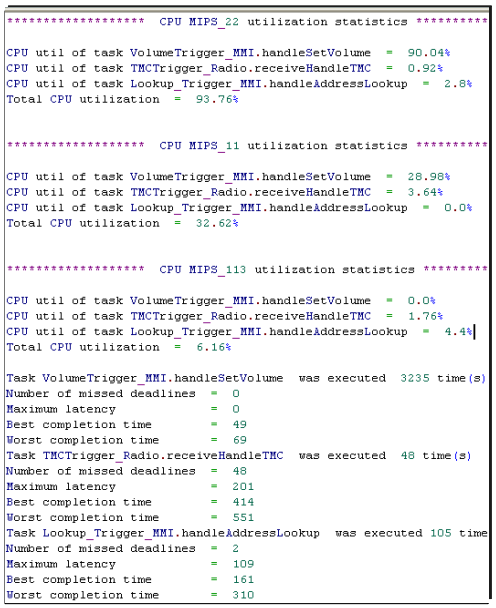

Figure 6. Statistics on performance properties.

ded systems: a PDA-based complex MPEG-4 application, JPEG2000 decoder and the car radio navigation system. The simulation speed of the performance analyzer was the following. For three-four tasks simulated on three processors for 10 mln milliseconds, it takes 2-10 minutes for the analyzer to complete. We expect linear increase of the simulation time for more complex systems.

We have also identified the *prediction accuracy* of the analyzer, comparing the predicted data against the measured data obtained from the implemented applications. For simple applications with one processor and stable operation execution times, the accuracy is close to 100%. For CRN system, the accuracy stays within 70-130%.

Limitations. The toolkit and analysis methodology impose a number of limitations that needs further study. Firstly, it provides only static mapping of software component onto hardware resources. Secondly, it does not enable analysis of the components whose execution times are specified by probability distribution curves. Finally, the simulation techniques do not guarantee finding boundary conditions: worst- and best-case response times.

5. Conclusion

We have presented a toolkit for design and PA of component-based systems deployed on heterogeneous multiprocessor platforms. The toolkit supports the complete design cycle and enables performance analysis at the early design phase, when the source code is not available. The underlying PA methodology is based on modeling of individual components and automated composition of those models. The composition results in an executable system model, representing detailed information about processes (tasks) executing in a system and jobs fired by these processes. The executable model is a subject for simulation-based performance evaluation.

The advantages of the toolkit and PA framework occur in multiple ways. Firstly, the framework and toolkit are generic with respect to application domains and architectural styles. For instance, the toolkit can be applied to systems designed in “pipes-and-filters”, “blackboard” or “client-server” architectural styles. Secondly, the toolkit enables performance predictions for heterogeneous multiprocessor platforms by (a) provision of a wide set of protocols and virtual schedulers for simulation, (b) supporting active and passive components, and (c) multi-processor specification of component resource requirements.

In future plans, we focus on design-optimization algorithms as a further support for the above framework.

References

[1] R. Kazman, “Tool support for architecture analysis and design”, *Proc. SIGSOFT’06 workshop*, 1996.

[2] “IBM Rational Rose Software”, *Public website* http://www.ibm.com/rational

[3] “Robocop: Robust Open Component Based Software Architecture”, *Public deliverables*, http://www.hitech-projects.com/euprojects/robocop/deliverables.htm

[4] “Intel VTune Performance Analyzer”, *White paper*, http://www.intel.com/support/performancetools/vtune/sb/cs-009650.htm

[5] “PerfAnal: A Performance Analysis Tool”, *White paper* http://java.sun.com/developer/technicalArticles/Programming

[6] “LinuxLink by TimesysTM”, *Public website*, http://www.timesys.com/products/what.htm

[7] “Modular Performance Analysis with Real-Time Calculus”, *Public website*, http://www.mpa.ethz.ch/Rtctoolbox/Overview

[8] D. Urting *et al*., “A Tool for Component Based Design of Embedded Software”, *Proc. TOOLS Pacific*, Sydney, 2002.

[9] A. Bertolino, R. Mirandola, “CB-SPE Tool: Putting Component-Based Performance Engineering into Practice”. *Proc. 7th Int. Symposium on CBSE*, Edinburgh, 2004.

[10] B. Kienhuis *et al.*, “A Methodology to Design Programmable Embedded Systems - The Y-chart Approach”. *In Proc. of SAMOS Symposium*, Germany, 2002.

[11] R. Creps, P. Kogut, “Using DARPA Software Architecture Technologies to Design and Construct CORBA-based Systems”, *Proc. OMG-DARPA Workshop*, USA, 1998

[12] E. Bondarev *et al.*, “Compositional Performance Analysis of Component-Based Systems on Heterogeneous Multiprocessor Platforms”, *In Proc. of 32th Euromicro Conference; CBSE Track*, Croatia, September 2006.

[13] E.Bondarev, M.Chaudron, P. de With, “A Process for Resolving Performance Trade-Offs in Component-Based Architectures”, *Proc. 9th Int. CBSE Symposium*, Sweden, June, 2006.

[14] J. Coffland and A. Pimentel, “A Software Framework for Efficient System-level Performance Evaluation of Embedded Systems”, *Proc. of the ACM SAC*, 2003.