Model Driven Resource Usage Simulation for Critical Embedded Systems

Michaël Lafaye, Laurent Pautet, Etienne Borde
LTCI
Telecom ParisTech
Paris, France
{lafaye, pautet, borde}@telecom-paristech.fr

Marc Gatti, David Faura
Centre Competence Calculateur
Thales Avionics
Meudon La Foret, France
{marc-j.gatti, david.faura}@fr.thalesgroup.com

Facing a growing complexity, embedded systems design relies on model-based approaches to ease the exploration of a design space. A key aspect of such exploration is performance evaluation, mainly depending on usage of the hardware resources. In model-driven engineering, hardware resources usage is often approximated by static properties. In this paper, we propose an extensible modeling framework, to describe with different levels of detail the hardware resource usage. Our method relies on the AADL to describe the whole system, and SystemC to refine the execution platform description. In this paper we expose how we generate and compose SystemC models from the execution platform model described in AADL. We also present promising experimental results obtained on an avionics use-case.

AADL, SystemC, mapping, early modeling, real-time systems

I. INTRODUCTION

In the design process of embedded systems, integration is a critical phase since it might alleviate performance issues related to the adequacy between the software application and its underlying execution platform. In order to reduce this risk, model driven engineering (MDE) allows exploring the design space by analyzing models of a system. Performance evaluation based on MDE methods approximate hardware components characteristics by sets of predefined properties corresponding to a general category of component, thus they lack of precision since their results are obtained independently of the usage context of these resources as opposed to the real execution. As a consequence, MDE methods dedicated to performances evaluation of embedded systems need improvements to be able to assess more precisely usage of the execution platform resources. Besides, to be used in an iterative design process, these methods must be extensible to provide more and more precise resources usage estimations.

Our objective is to bring that modeling part and complement those methods. In this paper, we propose a flexible code generation approach that produces a simulation environment from the model of an embedded system, allowing a detailed description of the execution platform resources usage. Our method aims at being complementary to high-level static modeling methods (giving more approximate results) and virtualization methods (focusing on functional evaluation).

We present here a prototype that produces SystemC [1] code from an AADL [2] specification. As a first step, we limited the scope of our prototype to the mapping of hardware AADL components into SystemC code. We made a first case study based on software architecture principles dedicated to the avionics domain. The results of these experiments are also presented in this paper.

The remainder of this paper is organized as follows: Section 2 gives an overview of the approach we propose. Section 3 describes the flexible modeling and simulation process that produces the simulation environment corresponding to the execution platform of an embedded system. Section 4 presents the simulation results obtained using this process on an avionic use-case. Finally we conclude this paper and present our future works.

II. APPROACH

Current MDE approaches are of great interest to anticipate on system performance. However they often approximate the hardware components description as black boxes with a few properties. Our objective is to complement such process by proposing an extensible modeling and simulation facility so as to explore the resources usage of an execution platform. The objective we pursue is represented on figure 1: from specification of the execution platform and a characterization of the software application, we propose to model the software application, its underlying execution platform and the deployment of software components onto hardware components thanks to the AADL.

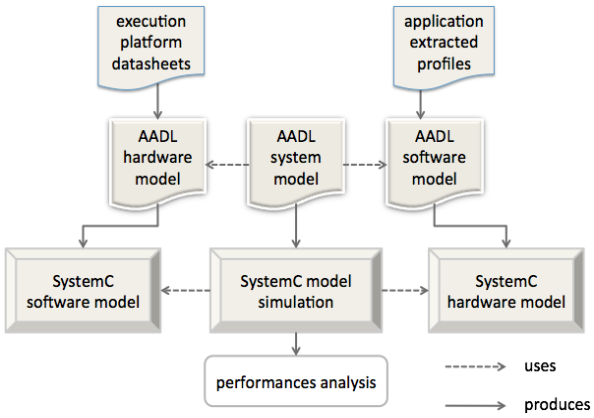

Figure 1. Modeling and simulation process

978-3-9810801-8-6/DATE12/©2012 EDAA

A. System Modeling with the AADL

AADL (Architecture and Analysis Description Language) [2] is an architecture description language developed by the SEI[1] and standardized by the SAE2. AADL was selected for this study since it allows to model both the software and the hardware architecture can be modeled, as well as the mapping of software components onto the execution platform. The language was used in several projects [4], taking advantages of the several annexes enriching the language. Especially, the ARINC653 annex [3] provides rules and properties to model partitioned systems. Indeed, to allow the execution of more than one application on the same execution platform while fulfilling safety requirements, avionics systems are developed following the ARINC653 standard. This standard specifies spatial and time partitioning concepts: each application is enclosed in one or several partitions (independent software module) and owns a part of the main memory (spatial partitioning). Moreover, each partition can access all the hardware resources during its execution window (time partitioning).

Thanks to the ARINC annex, those partitioning rules can be easily modeled as AADL processes (ARINC partitions) bounded to a part of the main memory component and scheduled by the main processor. As a consequence, AADL is thus particularly adapted for modeling partitioned real-time and embedded systems at high-level description.

B. System Description with SystemC

The target of our model-driven simulation framework is the SystemC, IEEE standard promoted by the OSCI. It is a hardware description language for system hardware/software co-design and validation through simulation thanks to the provided simulation kernel. SystemC is widely used in industry and supported by many tools that allow simulation and debug of a SystemC model. The language is defined as a set of modules that contain ports, methods and processes describing the component functionality, and communicate via channels.

SystemC can be used for system modeling at different levels of abstraction, and aims at simplifying systems conception by being a common language used from functional system description to detailed design conception. Among this abstraction levels, TLM [5] is becoming a de facto standard in industry for early exploration and verification processes [6]. It offers a good compromise between time development, description accuracy and simulation speed, 100 to 1000 times faster than CABA and RTL model simulation depending on the timing precision. Consequently, SystemC and its TLM library are particularly adapted for execution platform description and exploration through simulation thanks to its simulation kernel.

[1] Software Engineering Institute

[2] Society for Automotive Engineers

III. AADL TO SYSTEMC SIMULATION

We present in this section a method to produce the SystemC-TLM model corresponding to an AADL hardware model. We propose both a mapping for set of predefined AADL hardware components, and a method to integrate user-defined hardware components. Since our goal is to generate an execution platform, we focus on the following hardware components: memory, processor, bus and device. Generally, the mapping we propose relies on a database of SystemC configurable components. Each SystemC component defines a set of TLM sockets dedicated to the reception (respectively emission) of transactions from (*resp.* to) other components, and a priced state machine that describes the component's behavior upon reception of a transaction. In the remainder of this section, we present in more details the semantics of sockets and automata defining SystemC components.

A. Mapping Connections

In order to manage the communications between SystemC components, we generate for each component a set of TLM socket(s): from a hardware component instance in AADL system model, we retrieve the bus it is connected to and define an input/output TLM socket for the component and its bus, and connect them. Figure 2 illustrates this approach, including properties retrieving from the AADL model to configure the generated SystemC bus.

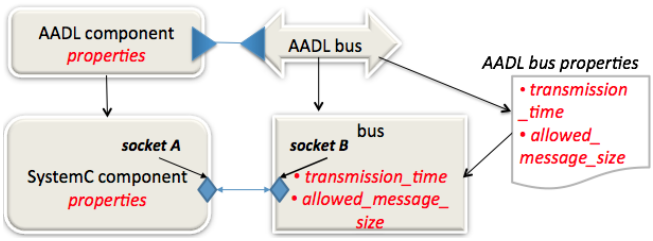

Figure 2. Connection Mapping from AADL to SystemC model

Each socket is bound to a generic SystemC method, *b_transport()*, which works on the initiator/target model. Initially written to simplify the communications by frame between initiator components such as processors and target components such as memory, we use this communication model and apply it on all the execution platform components we consider. The *b_transport()* method contains two parameters, *generic_payload* specifying the transactions arguments (command read/write/ignore, data address, data length and transaction status as "ok", "error"…) and *delay* , parameter we use as a simulation counter. We also add our own parameters to the communication model. As we said, components communicate by frames through the *b_transport()* method. When a component receives such frame, it decodes the address and if it matches its address, stimulates the state machine describing its behavior. For example, a memory component will decode the *command* parameter and executes a read or write operation before returning response status.

B. Mapping Hardwware Component into Automata

1) Automata Description

To describe the internal behavior of components, we use dedicated priced state machines. In these automata, each state represents an abstract action and the resources (time, power consumption…) consumed to realize it, and each transition defines the conditions that have to be verified to go from a source state to a target state. These conditions are specified over the content of a received transaction (in comparison to predefined constants retrieved from AADL properties). Finally, each automaton has at least two states: an idle state (corresponding to the situation when the component is not working) and one or some action states. The transitions represent conditions that have to be fulfilled to pass from one state to the next. Condition can be "when action is complete, i.e. when action time is totally consumed, go to the next state", or "when the signal S is set to 1, go to the next state".

The implementation of a SystemC automaton is defined in a generic SystemC-TLM method, *b_transport*, which simplifies the connections between components. Each time an instruction is received by the component, the *b_transport* method is called, decodes the frame properties and executes its behavior code. The goal of those state machines is to describe more precisely the component behavior and then introduce variability. For example, in a processor model, the execution time will be different for L1 cache hit or miss.

```
memory dram
features
        inport : requires bus access
main_bus;
end dram;

memory implementation dram.i
properties
        word_Size => 32 bits;
        write_time => (50ns .. 60 ns);
        read_time => (50ns .. 60 ns);
end dram.i;
```

```
If(refresh)
        delay+= sc_time(refresh_time,SC_NS;

switch(c->get_command())
 {
  case READ:
        delay += sc_time(read_time,SC_US);
        c->send_ok_response(trans,delay);
        break;
  case WRITE:
        delay += sc_time(write_time,SC_US);
        c->send_ok_response(trans,delay);
        break;
  case IGNORE:
        c->send_ok_response(trans,delay);
        break;
  default:
        c->send_error_response(trans,delay);
        break;
 }
}
```

Figure 3. Generated SystemC simple memory automata from AADL model

2) Automata Database

The database can be used in several ways, and is not limited to one. We present here two ways to configure the SystemC models: using simplified predefined automata, and using refined automata depending on the accuracy of the information contained in the AADL model. For instance, to describe a memory component, we can use the memory simple automata configured only with AADL properties *read_time, write_time, memory_protocol and word_size,* or using a refined component DRAM. In such component, the variability comes for example from the refresh operation, which does not occur at each reading or writing execution but comes periodically.

The goal of refined automata is to be more accurate in the component description, and then more accurate during the simulation and the performance analysis, and introduce variability, where other static based approach work with constant worst-case execution time value. To add a new component, we define a method allowing the user to define his component and adding it to the database. This method is composed of 3 steps:

- Adding the new automata in the SystemC database
- Adding the component kind in the corresponding *supported_<component_type>_kind*
- Adding the properties. In the AADL, components properties are defined in a specific file. To be consistent with the AADL, we propose, to add the corresponding property_set defining its properties.

For example, to describe a DRAM, we first add to the *memory_supported_kind* AADL property the "DRAM" type. Then we create a new property_set, *dram_properties*, in which we define the additional properties, for instance: *refresh_period* and *refresh_time*. These properties can be used together with the memory properties still defined (word size, read/write_time…). At least, we create and add in the database the corresponding DRAM SystemC automaton. Thus it is possible to elaborate an execution platform containing a DRAM component by declaring in the AADL model a memory with the *memory_kind* property set to *DRAM*. Figure 4 sums up how we generate the refined memory component thanks to the AADL adding properties:

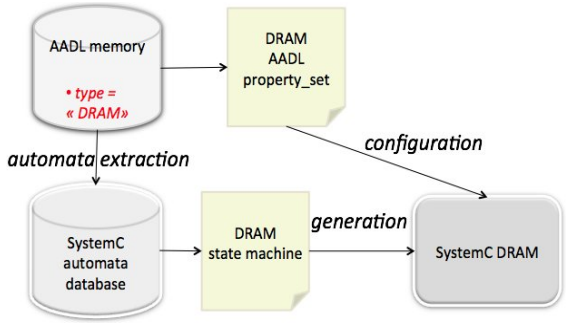

Figure 4. SystemC DRAM component generation

IV. CASE STUDY

Our main objective is to evaluate the usage of a proposed avionic execution platform according to a given set of applications stimuli. Then, to test our mapping and exploration process, we first model the avionic system with AADL using the ARINC653 annex, and generate with our mapping process the SystemC corresponding execution platform description. Then we stimulate it with a given test application. Both architecture and application are described in the next sub-sections.

A. Test Application

The application we use to test our process is an avionic communication application, which reads and sends data on input/output ports. It contains four partitions following the

temporal partitioning rule (each partition can access al the resources during its execution). Each partition mainly reads/writes data into memory (caches, dram) and targets an I/O, i.e sends read/write SystemC instruction to some hardware components.

B. Execution Platform

The execution platform is composed of two parts: the ARINC services connected to the hardware architecture. ARINC services are modeled as state machines targeting some hardware components. For example, "get_partition_status" service mainly reads data from the memory (operating mode, identifier, period, etc.) through the CPU. Hardware components are also modeled as interconnected state machines. In our example, the hardware architecture is composed of one CPU, one I/O controller, one main DRAM, four I/O interfaces and some buses (PCI, PCIe etc.).

C. Simulation and results

To explore the execution platform resources usage, we run the SystemC frames we extracted from the test application (ARINC653 services and calculation part, addition, affectation…). This extraction is currently "handmade", but automatic extraction is on development. Thanks to the SystemC simulation kernel, we run the stimuli and extract the execution platform response in term of resources consumed. In that example, we focus on the CPU usage during the first 200ms (figure 5). Values are computed each 50us. We also test our analyze process by setting a test requirement: CPU usage has to be under 80%. We can see in the figure 5 that the processor usage rate stays under this value, then the proposed architecture matches that requirement.

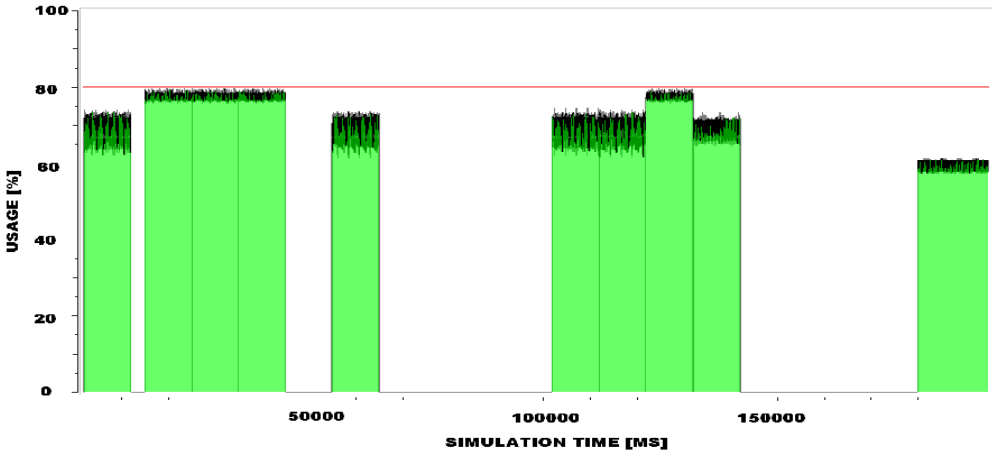

Figure 5. CPU usage of in each partition over the time

V. CONCLUSION AND FUTURE WORKS

In this paper, we have presented an approach to model an execution platform into a simulation environment that enables to evaluate its resources usage. This approach is extensible, allowing to model execution platforms with different levels of detail. The process takes advantage of two standardized languages, AADL for high-level system modeling, and SystemC-TLM for refined execution platform description and simulation thanks to its simulation kernel.

Our modular approach, based on a database of SystemC behavioral components, allows adding easily some new components to this database in order to refine SystemC execution platform description, and explore the hardware resources usage under different angles thanks to the viewpoints (timing, power consumption etc.). We saw our analyze process allow us to check the compliancy between a proposed execution platform architecture and given requirements.

We are currently improving our method by comparing our simulation results with real avionic applications performance. Another work in progress is to automate the extraction of software stimuli and their translation into SystemC frames. We are also integrating the modeling of the network connecting the processing modules in order to describe a whole avionic execution platform At least, we plan to target larger systems as systems of connected systems.

VI. REFERENCES

[1] Open SystemC Initiative. IEEE 1666: SystemC Language Reference Manual, 2005. www.systemc.org.

[2] S. A. E. (Society of Automotive Engineers) Architecture analysis and design language. SAE AS5506, SAE, 2009

[3] J. Delange, L. Pautet, A. Plantec, M. Kerboeuf, F. Singhoff, F. Kordon, "Validate, Simulate and Implement ARINC653 systems using the AADL", *CM SIGAda Ada Letters*, 2009.

[4] P. Dissaux, F. Signhoff, "the AADL as a Pivot Language for Analyzing Performances of Real Time Architectures", *4th European Congress ERTS Embedded Real Time Software,* 2008.

[5] L. Cai and D. Gajski, "Transaction Level Modeling: An Overview", *Center for Embedded Computer Systems, University of California*, 2003

[6] S. Swan, "SystemC TLM models and RTL verification", *DAC*, 2006