AADS: AADL Simulation and Performance Analysis in SystemC

Roberto Varona-Gómez, Eugenio Villar

{roberto, villar}@teisa.unican.es

GIM - TEISA - University of Cantabria – Spain

http://www.teisa.unican.es/

Abstract

AADS supports the performance analysis of AADL specifications throughout the refinement process from the initial system architecture till the complete application and execution platform are developed.

1. Introduction

Nowadays, embedded systems must support the deployment of heterogeneous applications within heterogeneous architectures. In most cases, the execution platform is not fixed and must be designed and optimized in conjunction with the application software. Therefore, early estimation of the system performance on the executive platform, under real-time constraints, is desirable. Such analysis requires a unified model of the application and the architecture, and an effective means to define the mapping of application functions onto architecture resources and services. AADL [1] provides such a modelling framework.

There is a commonly recognized need for new development frameworks that allow designers to perform efficient exploration of design alternatives and analyze system properties throughout the design cycle. Some system properties can be obtained by static analysis. Many other properties can only be obtained through simulation. In any case, system simulation is needed for performance analysis under real execution conditions. System simulation enables the correct dimensioning of the system, detection of locks, missed deadlines and other potential problems raised by the complex interaction among components that can be found in a real system. The earlier all those problems are detected, the less the associated cost of correcting them [2].

SystemC has become the standard language for modelling and simulation of HW/SW embedded systems [3].

In this paper, AADS, an AADL simulation and performance analysis framework, is presented. The tool can support prototype-based design allowing the functional and non-functional verification of the system while it is being refined until the final implementation. Based on SystemC, the framework supports the seamless integration of any HW component and an easy optimization of the executive platform.

2. AADS

AADS [4] is written in Java and it has been developed as a plug-in [5] of Eclipse [6].

AADS enables the modelling of a subset of AADL for purposes of implementation and simulation. The starting point of the simulator is a functional AADL specification without detailed code. For each component, the corresponding timing constraints are defined. This initial AADL specification supports the verification of the global performance constraints of the system based on the specific timing constraints of the different components. The AADL model is parsed using AADS and a model suitable to be simulated with SCoPE [7] is produced, in order to check if the AADL constraints are fulfilled.

As the design process advances and, on the one hand, the actual functionality is attached to the software components using the corresponding source code and, on the other, the functionality is mapped onto specific platform resources, a more accurate performance estimation is performed. These refined properties will be added to the AADL model and a new model is generated by AADS. By comparing (e.g. using assertions) the initial timing constraints with these refined timing estimations, it is possible to verify the non functional correctness of the design process in any refinement step. The corresponding methodology is shown in Fig. 1.

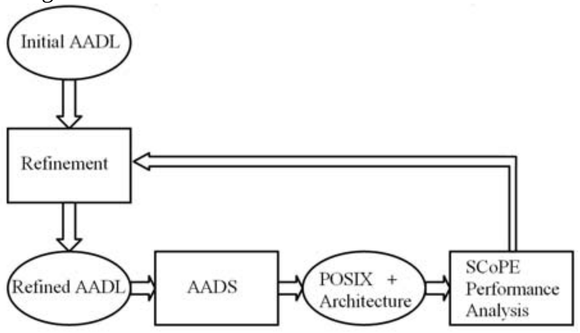

Figure 1: Refinement of AADL

3. Translation from AADL

AADL enables the specification of both the architecture and functionality of an embedded real-time system. AADS translates both to SystemC (see Fig. 2). It parses the AADL model so the functionality is translated to an equivalent POSIX [8] model and the architecture is represented in XML.

The functional elements are translated as follows:

Threads. An AADL thread translates seamlessly into a POSIX thread.

Periodic threads. *Dispatch_Protocol* and *Period* are set. The source code of the thread is put into an infinite loop. It waits to repeat the loop for exactly the time specified.

Port connections translate into message queues, signals and global variables:

Message queues. An AADL event data port connection between threads translates into a POSIX message queue between threads. Properties *Queue_Size* and *Queue_Processing_Protocol* are used.

Signals. An AADL event port connection between threads translates into a sending of POSIX signals between threads. The signals used are the user-definable real-time signals.

Global variables. An AADL data port connection between threads translates into a global variable between threads. This translation is suitable for immediate data port connections.

The AADL properties are translated as follows:

Scheduling_Policy and Priority of threads. An AADL property set called *UC* with two properties *POSIX_Scheduling_Policy* and *Priority* has been defined.

Compute_Execution_Time (min, max). The minimum causes the call to a function that consumes that time. The maximum uses a timer whose expiry triggers one of the last real-time signals to be sent and a function to be called that lowers the priority of the thread, and waits for a while before restoring the initial priority.

Names. Properties *Activate_Entrypoint* and *Source_Text* are used.

Initialize / Finalize_Entrypoint. These properties determine the routine called at the start/end of the start routine of the corresponding thread.

Initialize / Finalize_Execution_Time (min, max). The minimum causes the call to a function that consumes that time. It checks the maximum time, to see if this amount of time has elapsed and returns if it has been.

The issues related to the subprograms are the following:

Subprogram. An AADL subprogram translates into a routine.

Subprogram calls. Local calls and remote client-server calls translate into calls from one routine to another. *Actual_Subprogram_Call* is used.

Subprogram parameters. AADL parameters translate into parameters of the subprogram by value or reference.

AADL data are managed as follows:

Data type. Simple independent AADL data give rise to a data type. *Source_Data_Size* is used.

Simple Data. A simple AADL data subcomponent of a thread or a process gives rise to a simple global variable.

Composite Data. This data generate a C++ class of data with its methods and/or member data. The composite data subcomponents of a thread or a process give rise to a global variable

The hardware architecture is structured through the XML file generated by AADS. It is used as part of the configuration parameters of SCoPE and is divided into:

HW_Platform. Any AADL implementation of a processor, memory, bus or device must be specified in the *HW_Components* subsection. Properties *Assign_Byte_Time, Read_Time, Write_Time, Word_Coun, Word_Size* and *Memory_Protocol* are used. The *HW_Architecture* and *Computing_groups* subsections use *Base_Address* and *requires bus access.*

SW_Platform. This section has two subsections: *SW_Components* and *SW_Architecture*.

Application. This section has two subsections: *Functionality* and *Allocation. Activate_Entrypoint, Source_Text* and *Actual_Processor_Binding* are used.

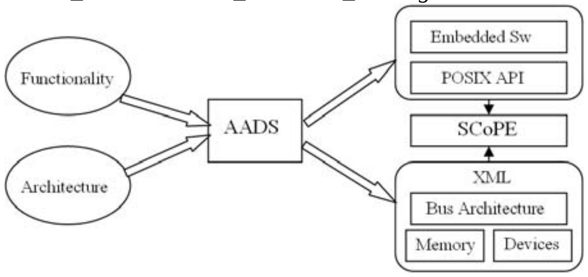

Figure 2: Translation with AADS.

4. Conclusion

In this project, we have developed AADS, an AADL SystemC simulation tool. AADS supports the refinement of AADL models through performance analysis done with SCoPE, after translating those models.

The generation of the SystemC model from the AADL specification is not straightforward. Nevertheless, the SystemC model generated by AADS is able to capture the fundamental dynamic properties of the initial system specification. In this way, AADS supports design space exploration by refinement of the AADL functionality and its implementation on an optimized platform.

Future work includes incorporation of AADS features that appear in the annex behaviour specification and in V2.0 of the AADL standard.

5. References

[1] SAE: AADL. June 2006, document AS5506/1. www.sae.org/technical/standards/AS5506/1.

[2] A.D. Pimentel et al.: "A systematic approach to exploring embedded system architectures at multiple abstraction levels", IEEE Transactions on Computers, 2006.

[3] H. Posadas et al.: RTOS modeling in SystemC for real-time embedded SW simulation: A POSIX model. Design Automation for Embedded Systems. Springer. 2005.

[4] AADS V1.2 UC 2008. www.teisa.unican.es/AADS

[5] P. H. Feiler, A. Greenhouse: OSATE Plug-in Development Guide. CMU. Pittsburgh. (2006).

[6] The Eclipse Foundation 2009. www.eclipse.org

[7] SCoPE V1.1.0 UC 2009. www.teisa.unican.es/scope

[8] M. González: POSIX tiempo real. UC, Santander 2004.

Acknowledgement

The authors would like to thank their colleagues in the GIM in the University of Cantabria and their colleagues in the SPICES project for their assistance.