AADL Simulation and Performance Analysis in SystemC

Fourth IEEE International workshop UML and AADL

2nd June 2009 Potsdam, Germany

Roberto Varona Gómez
Eugenio Villar
{roberto, evillar}@teisa.unican.es
University of Cantabria, Santander, Spain.

This work has been partially supported by the Spanish Ministry of Industry, Tourism and Trade through the ITEA 05015 SPICES Project and the TEC2008-04107 project.

Index

State of the Art

- Simulation and performance analysis of AADL models represent an important stage in Model Driven Engineering. Different approaches:
- Ocarina [5] is a tool suite that uses code generation facilities in Ada and C to analyze the AADL model.
- ADeS is a simulation tool that uses the event-driven approach as simulation engine technique [7].
- Cheddar [8] is a set of Ada packages that enables the design of a new scheduler and direct interpretation using the Cheddar environment.
- The Furness toolset [9] translates models into the real-time process algebra ACSR to explore the state space looking for violations of timing requirements.
- M. Yassin Chkouri et al. propose in [10] a translation from AADL models into BIP models to allow simulation.

State of the Art

- ADAPT [11] translates an AADL architectural model into a dependability evaluation model in the form of a Generalized Stochastic Petri Net (GSPN).
- T. Abdoul et al. [12] produce an IF timed automata model which is the entry point of the validation process, processing it with the IFx framework.
- E. Jahier et al. [13] translate the architecture into a non-deterministic synchronous model to which the SW components in Scade or Lustre can be integrated, to simulate it with Lurette.
- S. Gui et al. [14] use the linear hybrid automata in the design phase statically to abstract the semantics of the SW components of AADL explicitly.
- M. Brun et al. [15] translate to OIL configuration code and to C code which is compatible with the OSEK/VDX RTOS.

State of the Art

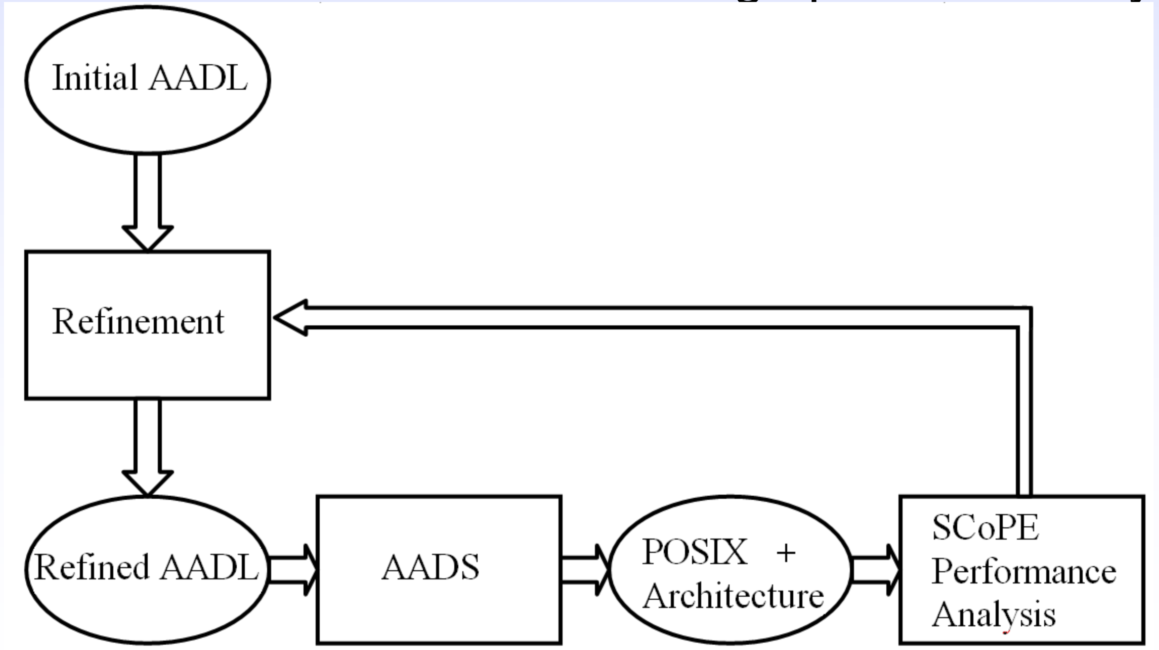
- Annex D of the AADL standard gives guidelines to translate AADL SW components into source code (C, Ada).
- After analyzing the state of the art, it appears that no approach uses SystemC, which is the recognized standard for modeling HW/SW platforms, with its great potential for processors, buses, memories and specific platform HW integration.
- SCoPE is a C++ library that extends the standard language SystemC without modifying it. It simulates C/C++ SW code based on two different operating system interfaces (POSIX and MicroC/OS). Moreover, it co-simulates these pieces of code with HW described in SystemC.
- AADS supports AADL simulation in SystemC, thus allowing modeling the HW platform and permitting HW/SW co-design. The AADL model is based on POSIX, therefore supporting many different RTOS.

AADS

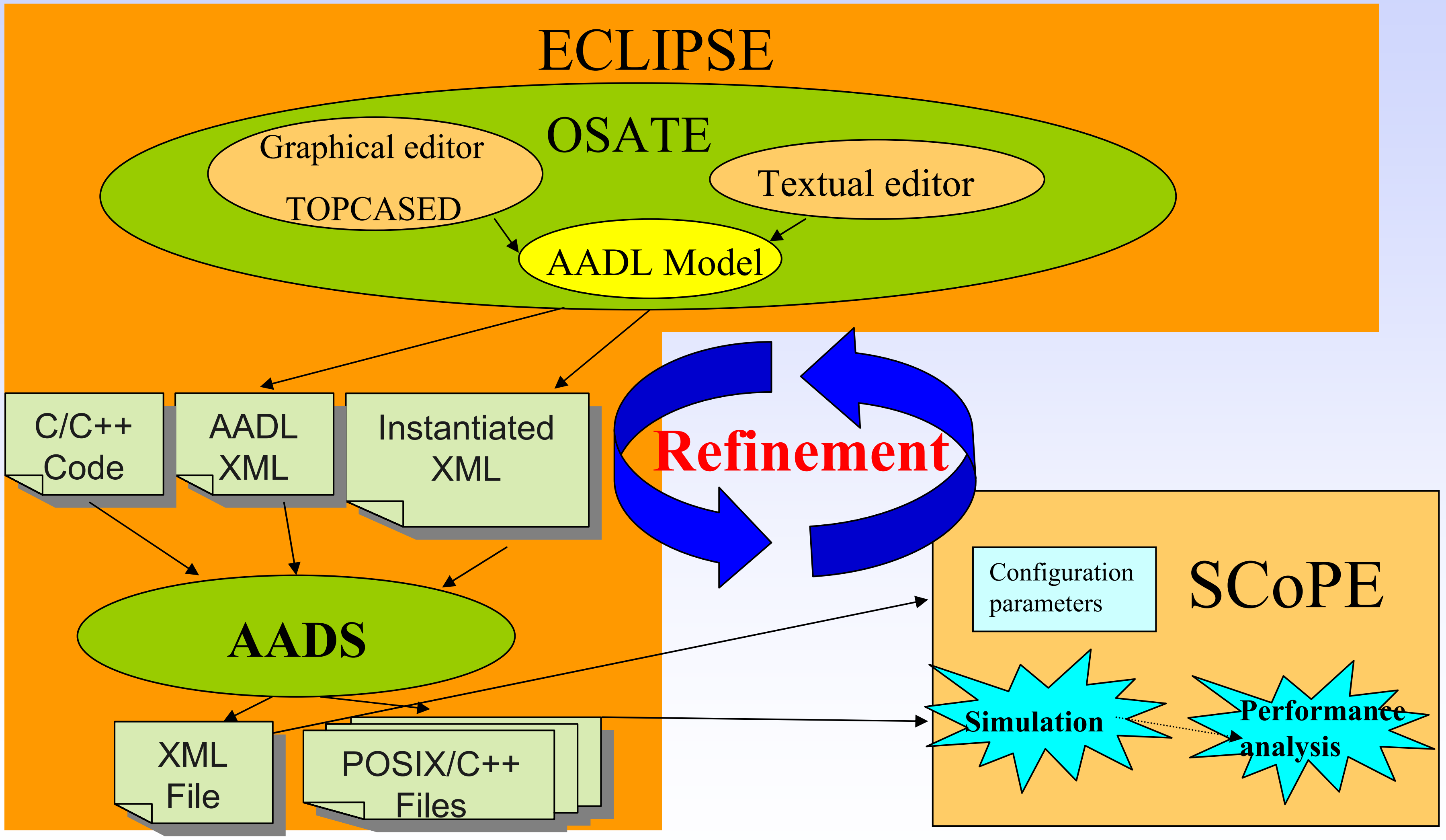
- AADS is written in Java and it was developed as a plug-in of Eclipse for Windows.
- AADS enables the modeling of a subset of AADL for purposes of implementation and simulation. The starting point of the simulator is a functional AADL specification without detailed code. For each component, the corresponding 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 for simulation with SCoPE is produced, in order to check if the AADL constraints are fulfilled.

AADS

- As the design process advances and, on the one hand, the actual functionality is attached to the SW 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 the initial timing constraints with these refined, timing estimations, it is possible to verify the non functional correctness of the design process at any refinement step.



AADS



ECLIPSE
OSATE
Graphical editor
TOPCASED
Textual editor
AADL Model
C/C++ Code
AADL XML
Instantiated XML
Refinement
AADS
XML File
POSIX/C++ Files
SCoPE
Configuration parameters
Simulation
Performance analysis

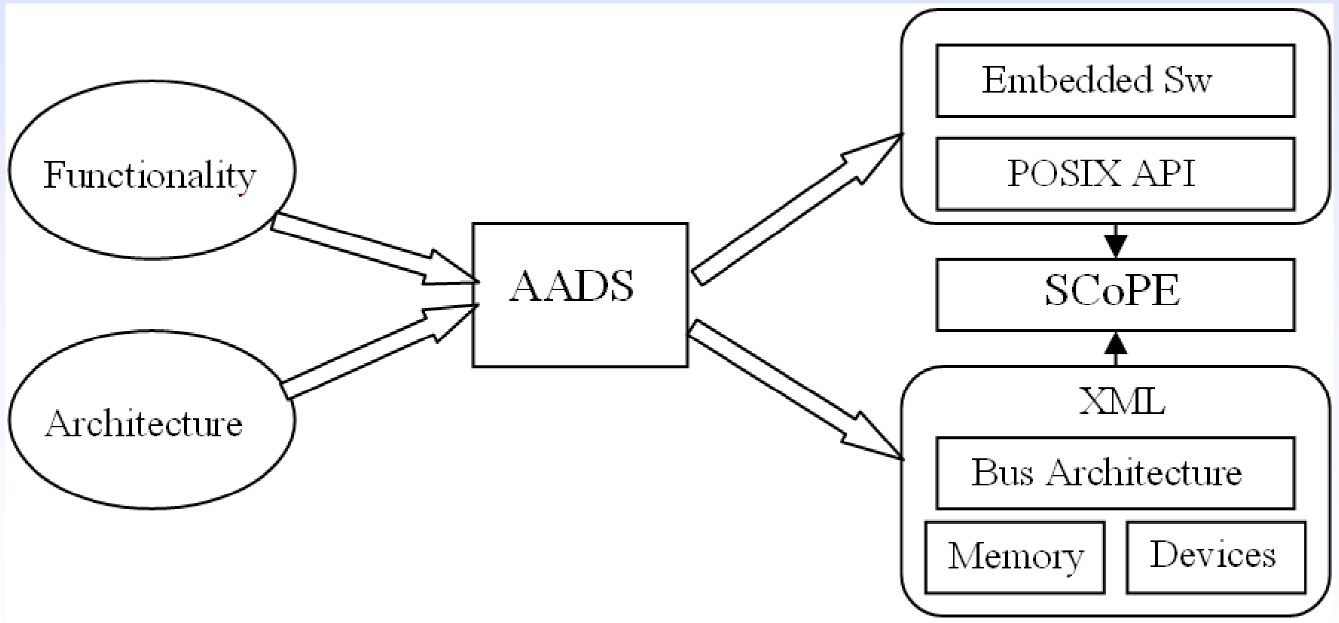
AADS

www.teisa.unican.es/AADS

www.teisa.unican.es/scope

Translation from AADL

- AADL enables the specification of both the architecture and functionality of an embedded real-time system. AADS translates both, it parses the AADL model so the functionality is translated to an equivalent POSIX model and the architecture is represented in XML.



Translation from AADL

- The functional elements are translated as follows:
- **Threads**. An AADL thread is a concurrent schedulable unit of sequential execution through source code and multiple threads represent concurrent execution paths. A POSIX thread is an execution thread in a program and an application can have multiple execution threads running concurrently. An AADL thread translates seamlessly into a POSIX thread.
- In POSIX, a thread attribute object must be defined and initialized with the default value for all of the individual attributes used by a given implementation. AADS determines how the other scheduling attributes of the created thread are to be set, that is that the scheduling policy and associated attributes are to be set to the corresponding values. Thus AADS can now call the POSIX function to create a new thread with the specified attributes. The specified routine is then launched as a starting routine.

Translation from AADL

- **Periodic threads**. A thread is periodic if repeated dispatches occur during a specific time interval. An AADL periodic thread has its *Dispatch_Protocol* property set to *Periodic* and its *Period* property set, for example, to 20 ms.
- These two properties are translated putting the source code of the POSIX thread into an infinite loop. At the beginning of the loop the current time is obtained. At the end of the loop the current thread is suspended until either the time value of the clock reaches the absolute time specified (the current time plus the period), or a signal is delivered to the calling thread and its action is to invoke a signal-catching function, or the thread is terminated. By doing this it waits to repeat the loop for exactly the time specified in the *Period* property.

Translation from AADL

- Port connections translate into global variables, message queues and signals:
- **Global variables.** An AADL data port interfaces for typed state data transmission among components without queuing. Data ports are represented by typed variables in source text. A global variable is a variable that is accessible in every scope. Global variables are used extensively to pass information between sections of code that do not share a caller/called relation like concurrent threads. An AADL data port connection between threads translates into a global variable between threads.
- The data type of this global variable is derived from the type of ports connected. The source/destination thread that corresponds to the AADL source/destination thread of the data port connection, can write/read a value in/from that global variable.

Translation from AADL

- **Message queues.** An AADL event data port models message communication with queuing of messages at the recipient. Message arrival may cause dispatch of the recipient and allow the recipient to process one or more messages. POSIX message queues allow threads to exchange data in the form of messages. Messages placed in the queue are stored until the recipient retrieves them. An AADL event data port connection between threads translates into a POSIX message queue between threads.
- The attributes of the message queue must be set. The value of the maximum number of messages is taken from the AADL property *Queue_Size* of the destination port if it exists. The AADL property *Queue_Processing_Protocol* is set to *FIFO* as corresponds to a message queue. The message queue is created to both send and receive messages in non-blocking mode. The thread corresponding to the AADL source/destination thread of the event data port connection should add/receive a message of the specified length to/from the message queue specified with the priority indicated.

Translation from AADL

- **Signals.** An AADL event port interfaces for the communication of events raised by subprograms, threads, etc. that may be queued. An example of use of an event port includes alarm communications that may be queued at the recipient, where the recipient may process the queue content. A signal is a limited form of inter-thread communication used in POSIX-compliant operating systems. It is an asynchronous notification sent to a thread in order to notify it of an event that occurred. When a signal is sent to a thread, the operating system interrupts the thread's normal flow of execution. If the thread has previously registered a signal handler, that routine is executed. Otherwise, the default signal handler is executed. An AADL event port connection between threads translates into a sending of POSIX signals between threads.
- Signals used are user-definable real-time signals. The structure type of an object used to represent sets of signals must be used with the POSIX functions that initialize and empty a signal set, add a signal to a signal set and examine and change blocked signals before creating the thread. The source/destination POSIX thread that corresponds to the AADL source/destination thread of the event port connection sends/waits for the signal (zero timeout for no blocking if there is no signal received).

Translation from AADL

- The AADL properties are translated as followed:
- **Scheduling_Policy and Priority of threads.** An AADL property set called *UC* with two properties *POSIX_Scheduling_Policy* and *Priority* has been defined. The first is an enumeration of the values *SCHED_FIFO, SCHED_RR, SCHED_SPORADIC* and *SCHED_OTHER*, and the second is an integer from 0 to 32. The first is obviously used to set the scheduling policy of the treads. The second is used with the appropriate minimum value for the scheduling policy specified to set the scheduling parameter attributes of the threads.

Translation from AADL

- ***Compute_Execution_Time (min, max).*** The minimum time causes a call to a function that consumes that processing time to assure that at least that time is consumed. This function is adjusted at the beginning of the application to assure that the exact time is consumed. Thus the minimum execution time is the time established by this property for this thread.
- The maximum time requires the creation of a timer that is set with this time until the next expiration of the timer. Therefore, the timer expires in a maximum time nanoseconds from when the call is made. When this timer expires, one of the last real-time signals is sent and a function called. This function lowers the priority of the thread and waits for a while before restoring the initial priority of the thread using the same method. When the priority of the thread is low, the scheduler avoids executing the thread and other threads can be processed. Thus we assure that the maximum time of execution is the one of this property for this thread.

Translation from AADL

- **Names.** Property *Activate_Entrypoint* of a thread is the name of the C++ function that contains the source code of that thread. Thus, this is the name of the function executed as a starting routine when creating the thread. *Source_Text* of a thread is the name of the C++ file containing the source code of that thread.
- ***Initialize / Finalize_Entrypoint*.** The name of the routine called at the start/end of the start routine of the corresponding thread is derived from this property.
- ***Initialize / Finalize_Execution_Time (min, max).*** The minimum time causes the call to a function that consumes that processing time to assure that at least that time is consumed. It checks the maximum time, to see if this amount of time has elapsed and return if it has been.

Translation from AADL

- The issues related to the subprograms are the following:
- **Subprogram**. An AADL subprogram component abstraction represents sequentially executable source text, a callable component, with or without parameters, that operates on data or provides server functions to components that call it. A routine is a portion of code within a larger program, which performs a specific task and is relatively independent of the remaining code. An AADL subprogram translates into a routine.
- **Subprogram calls**. Two types: call sequences and remote calls. The local call from a thread or from another subprogram within the same thread to a subprogram is made in AADL through the sub-clause call and is translated into direct calls from the thread start routine or from the routine respectively.
- The remote client-server call from a subprogram in a thread to another subprogram in another thread is made through the sub-clause call and the property *Actual_Subprogram_Call*. This remote call translates into a call from one routine to another.

Translation from AADL

- **Subprogram parameters.** A parameter represents call and return data values or references to data passed into and out of a subprogram, so it can be by value or by reference. In AADL the data values are in or out parameters and references are requires data access. Connections must be established between the ports of the thread (or the subprogram) and the ports of the subprogram. The data type of the AADL out parameter, if any, determines the data type of the routine; if there is no out parameter the type is void. Thus, the AADL parameters translate into parameters of the subprogram by value or reference. The translation permits data exchange among subprograms.

Translation from AADL

- AADL data are managed as follows:
- **Data type**. The AADL data abstraction represents static data and data types. Data component declarations are used to represent: application data types, the substructure of data types via data subcomponents within data implementation declarations and data instances. In general, a data type defines a set of values and the allowable operations on those values. Simple independent AADL data gives rise to a data type. These data types will be used later to define the type of a global variable, a message, etc. The name of the data type can be inferred from the name of the AADL data. This translation takes into account the property *Source_Data_Size*. In the case of data types, it specifies the maximum size required to hold a value of an instance of the data type.

Translation from AADL

- **Simple Data.** A simple AADL data subcomponent of a thread or a process gives rise to a simple global variable. The name and type can be inferred from the name and the AADL data type.
- **Composite Data.** AADL composite data is data that has one or more subprograms as features and/or one or more datum as subcomponents. This data generates a C++ class of data with its methods and/or member data. The name of the class can be inferred from the name of the AADL data. The names and types of the methods and members can be inferred from the AADL subprograms and data. The composite data subcomponents of a thread or a process give rise to a global variable whose type is that class. The name can be inferred from the name of the AADL data.

Translation from AADL

- The HW 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, SW_Platform and Application.*
- **HW_Platform.** Any AADL implementation of a processor, memory, bus or device must be specified with its *category* and *name* in the *HW_Components* subsection of *HW_Platform*. The AADL property *Assign_Byte_Time* is used to set the *frequency* parameter in the XML file. For memories we use the properties *Read_Time* and *Write_Time*. These properties have their values in time units (ns, ms and so on) and must be transformed into MHz. To know the *mem_size* of a memory, both *Word_Count* and *Word_Size* AADL properties are required. Finally the *mem_type* of a memory is derived from *Memory_Protocol* in the AADL model. If the component is a processor *proc_type* must be specified.

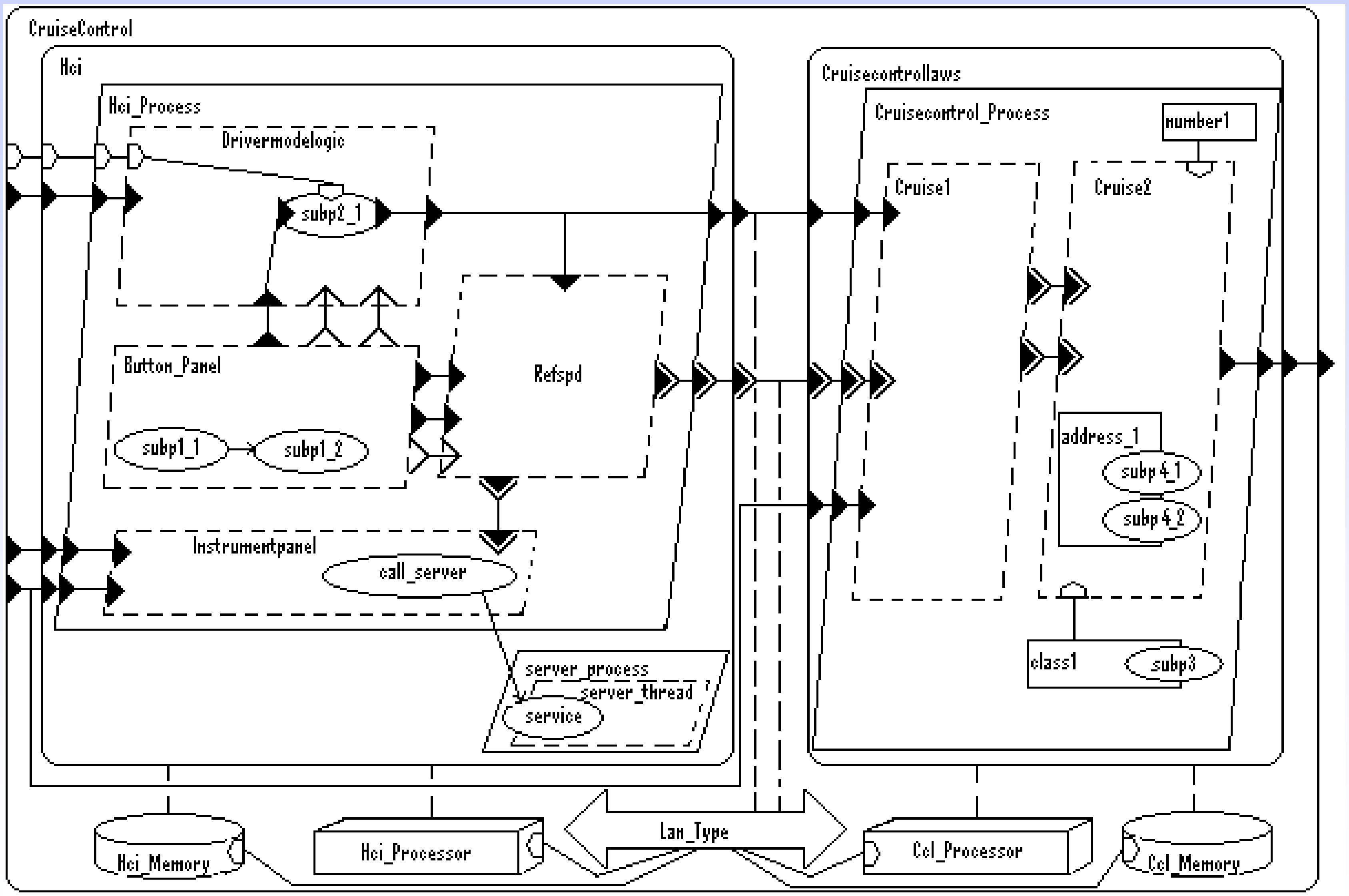
Translation from AADL

- The *HW_Architecture* and *Computing_groups* subsections of *HW_Platform* are the next of the XML file. To know the *start_addr* of a memory we take the AADL property *Base_Address.* The *component* and *name* are inferred from the AADL model. HW components are grouped by buses as they are connected to them in AADL through the connections *bus access* and the features *requires bus access.*
- **SW_Platform.** This section has two subsections: *SW_Components* and *SW_Architecture*. This section takes into account the buses that are defined to make the equivalent nodes. In this section the operating systems are specified.
- **Application.** This section has two subsections: *Functionality* and *Allocation.* Filling the *Functionality* section is straightforward from the AADL model using the property of a thread *Activate_Entrypoint* for the *function* and *Source_Text* for the *file*. The *name* is the same as the one of the thread. For the *Allocation* section we need to know the property of a thread *Actual_Processor_Binding*, and find out which bus the processor is bound to and then find out which node that bus corresponds to.

Case Study

- The proposed method implemented in AADS has been tested in a typical case study, the cruise control, to assure the feasibility of the translation. Cruise control is a system that automatically controls the velocity of a motor vehicle. The driver sets a speed and the system will take over the throttle to maintain it.

Case Study



CruiseControl
Hci
Hci_Process
Drivermodelogic
subp2_1
Button_Panel
subp1_1
subp1_2
Refspd
Instrumentpanel
call_server
server_process
server_thread
service
Cruisecontrollaws
Cruisecontrol_Process
number1
Cruise1
Cruise2
address_1
subp4_1
subp4_2
class1
subp3
Hci_Memory
Hci_Processor
Lan_Type
Ccl_Processor
Ccl_Memory

Case Study

- The figure shows an AADL model of a cruise control system, borrowed from the collection of AADL examples in the OSATE release, but modified to add some subcomponents. The system component contains two processors and two memories connected by a bus, and two SW subsystems. Each of the subsystems is bound to a separate processor and to a separate memory. Threads communicate via data ports, event ports and event data ports. Some data access connections can be seen too. There are some subprograms within threads and within data subcomponents and the call sequences (local and remote) between them are shown. The parameter connections between subprograms are shown too. One subsystem has two processes, one with four threads and the other with one. The other subsystem contains one process, with two threads.

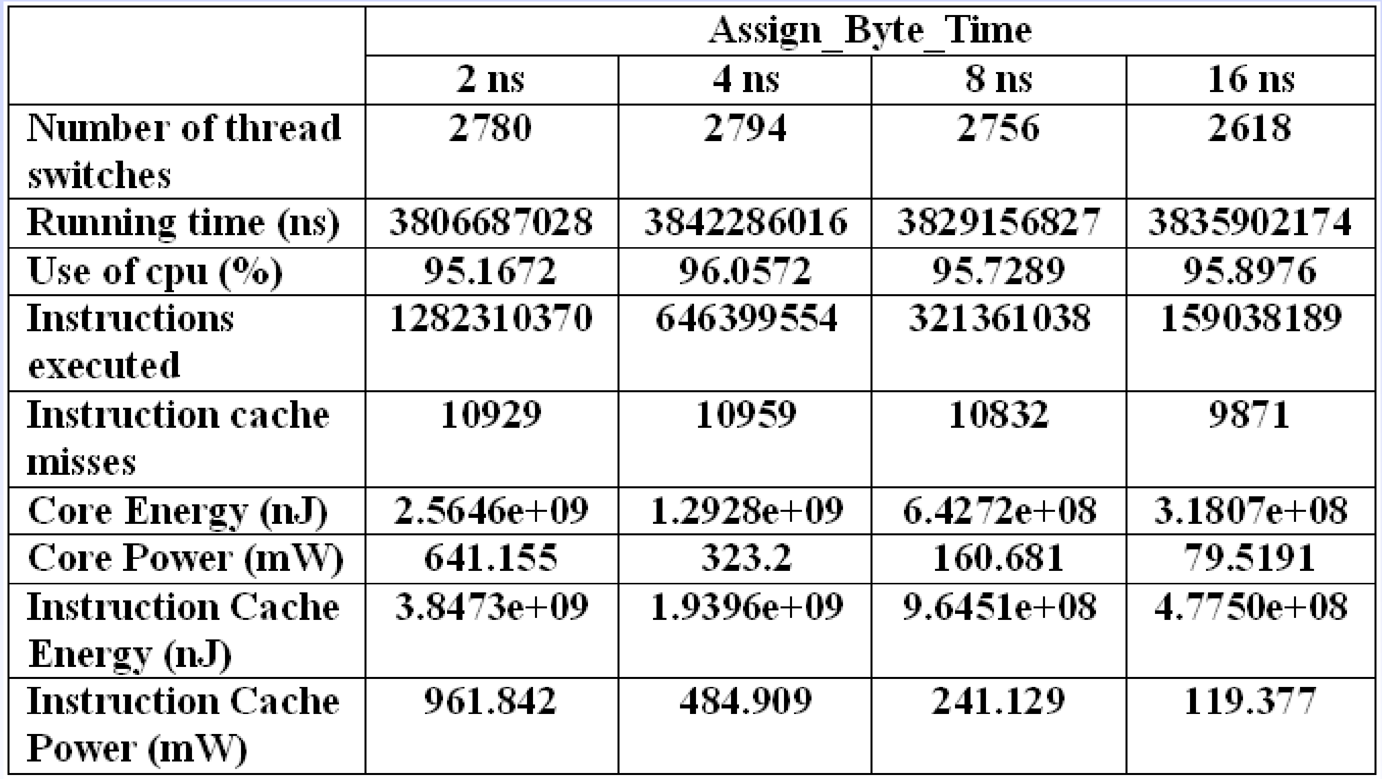
Case Study

- The files produced by AADS are compiled with SCoPE to simulate the model. The results obtained in the simulation are used to refine the model of the cruise control as needed.
- The value of the period of the threads has been refined to permit the correct interaction among threads. After trying different values of periods, 20 ms was found to be the best for all threads' interaction. If one thread had a different period from the other, it had to wait for the first one to send/get data.
- The size of the message queues from the thread *Refspd* was refined from a prior value (10) to a value (200) that avoids missing messages in the reception. The other message queues did not need to be refined.
- Some SW subcomponents such as subprograms, composite data, etc. have been added to the AADL model to obtain the desired system performance.

Case Study

- Properties have been defined and their values refined to achieve this purpose. Minimum and maximum times of some properties (e. g. *Compute_Execution_Time)* were adjusted depending on the results obtained.
- Connections among threads can be varied (and indeed they were varied) from the different types to achieve the desired interaction.
- The properties of the HW subcomponents were changed (e. g. as *Assign_Byte_Time* of the processors was increased the instructions executed, core and instruction energies decreased) to ascertain the different behaviors of the system. Thus, the most suitable HW subcomponent can be chosen for the system according to the initial constraints.

Case Study



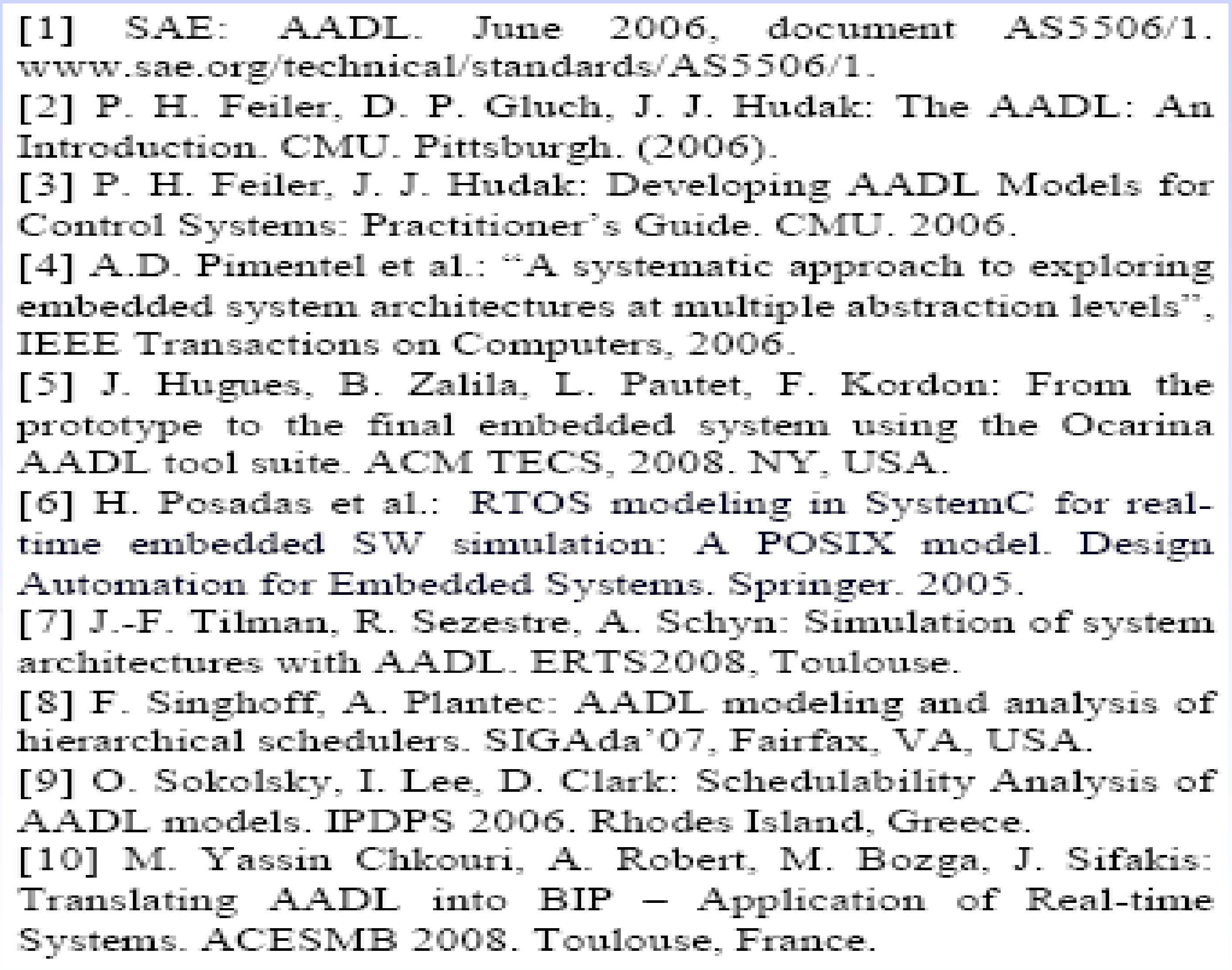
	Assign_Byte_Time			
	2 ns	4 ns	8 ns	16 ns
Number of thread switches	2780	2794	2756	2618
Running time (ns)	3806687028	3842286016	3829156827	3835902174
Use of cpu (%)	95.1672	96.0572	95.7289	95.8976
Instructions executed	1282310370	646399554	321361038	159038189
Instruction cache misses	10929	10959	10832	9871
Core Energy (nJ)	2.5646e+09	1.2928e+09	6.4272e+08	3.1807e+08
Core Power (mW)	641.155	323.2	160.681	79.5191
Instruction Cache Energy (nJ)	3.8473e+09	1.9396e+09	9.6451e+08	4.7750e+08
Instruction Cache Power (mW)	961.842	484.909	241.129	119.377

E. g. *Assign_Byte_Time* variation of the processors.

Conclusions

- The paper presents 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 behavior specification annex and in V2.0 of the AADL standard.

References



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

[2] P. H. Feiler, D. P. Gluch, J. J. Hudak: The AADL: An Introduction. CMU. Pittsburgh. (2006).

[3] P. H. Feiler, J. J. Hudak: Developing AADL Models for Control Systems: Practitioner's Guide. CMU. 2006.

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

[5] J. Hugues, B. Zalila, L. Pautet, F. Kordon: From the prototype to the final embedded system using the Ocarina AADL tool suite. ACM TECS, 2008. NY, USA.

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

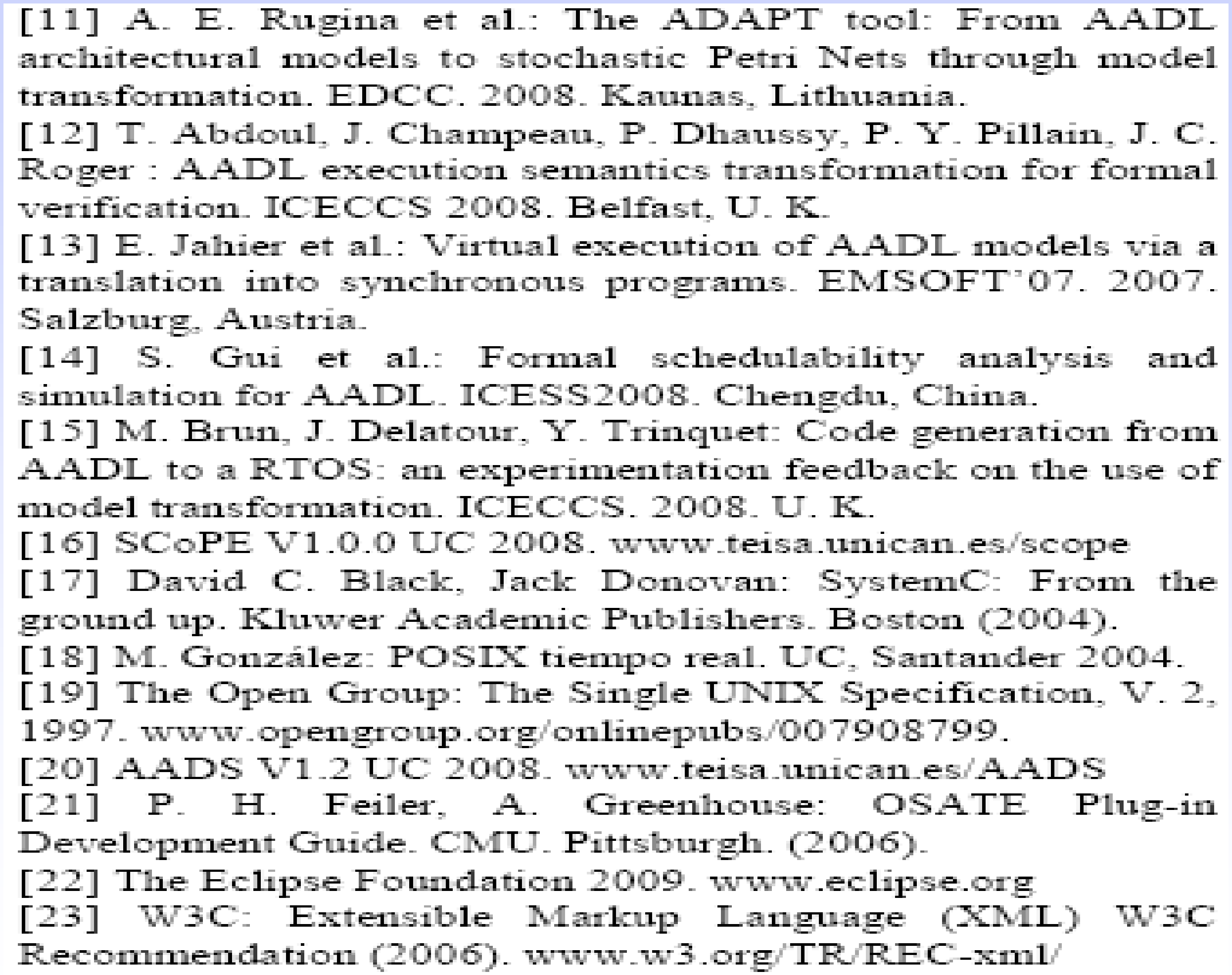
[7] J.-F. Tilman, R. Sezestre, A. Schyn: Simulation of system architectures with AADL. ERTS2008, Toulouse.

[8] F. Singhoff, A. Plantec: AADL modeling and analysis of hierarchical schedulers. SIGAda'07, Fairfax, VA, USA.

[9] O. Sokolsky, I. Lee, D. Clark: Schedulability Analysis of AADL models. IPDPS 2006. Rhodes Island, Greece.

[10] M. Yassin Chkouri, A. Robert, M. Bozga, J. Sifakis: Translating AADL into BIP – Application of Real-time Systems. ACESMB 2008. Toulouse, France.

References



[11] A. E. Rugina et al.: The ADAPT tool: From AADL architectural models to stochastic Petri Nets through model transformation. EDCC. 2008. Kaunas, Lithuania.
[12] T. Abdoul, J. Champeau, P. Dhaussy, P. Y. Pillain, J. C. Roger : AADL execution semantics transformation for formal verification. ICECCS 2008. Belfast, U. K.
[13] E. Jahier et al.: Virtual execution of AADL models via a translation into synchronous programs. EMSOFT'07. 2007. Salzburg, Austria.
[14] S. Gui et al.: Formal schedulability analysis and simulation for AADL. ICESS2008. Chengdu, China.
[15] M. Brun, J. Delatour, Y. Trinquet: Code generation from AADL to a RTOS: an experimentation feedback on the use of model transformation. ICECCS. 2008. U. K.
[16] SCoPE V1.0.0 UC 2008. www.teisa.unican.es/scope
[17] David C. Black, Jack Donovan: SystemC: From the ground up. Kluwer Academic Publishers. Boston (2004).
[18] M. González: POSIX tiempo real. UC, Santander 2004.
[19] The Open Group: The Single UNIX Specification, V. 2, 1997. www.opengroup.org/onlinepubs/007908799.
[20] AADS V1.2 UC 2008. www.teisa.unican.es/AADS
[21] P. H. Feiler, A. Greenhouse: OSATE Plug-in Development Guide. CMU. Pittsburgh. (2006).
[22] The Eclipse Foundation 2009. www.eclipse.org
[23] W3C: Extensible Markup Language (XML) W3C Recommendation (2006). www.w3.org/TR/REC-xml/

AADL Simulation and Performance Analysis in SystemC

Thanks for your attention.