Project ADVANCE

Grant Agreement 287563

"Advanced Design and Verification Environment for Cyber-physical System Engineering"

ADVANCE Deliverable D4.1

Specification of Multi-Simulation Framework

Public Document

December 19, 2011

http://www.advance-ict.eu

Contributors:

John Colley

Reviewers:

Jose Reis

Contents

Chapter 1

Preface

WP4 deliverable definition: Specification of multi-simulation framework.

This deliverable describes the various simulation tools and techniques that will have to be combined to provide an effective multi-simulation framework. The specification focuses on the structural mechanism needed to integrate a simulation tool into the framework, possibly as a plug-in, and the communication mechanisms that needs to be supported to ensure the efficient transfer of data between co-operating simulation tools. The requirements for the final multi-simulation framework are expressed and an architecture will be specified.

Chapter 2

Multi-Simulation Objectives

2.1 Motivation

Simulation is the dominant technology for the industrial verification of digital hardware and embedded systems. The use of the Verilog and VHDL modelling languages is widespread in simulation-based digital hardware verification and SystemVerilog and SystemC are used increasingly for the design and simulation of systems. The design and verification of cyber-physical systems, however, introduce new challenges which cannot be easily addressed by existing simulation frameworks. First, it is necessary in a cyber-physical system to model and simulate in the *continuous* as well as the digital domain. Second, the complexity of highly-concurrent systems cannot be addressed by simulation techniques alone. Refinement-based formal methods, such as Event-B [Abr10] help considerably to manage this complexity and to verify that the implemented system meets its specification.

It is not feasible to contemplate developing a single simulation language and verification environment that can meet all the requirements for cyber-physical development and verification. Legacy designs must be re-used and the specialised expertise of developers with existing tool chains leveraged. The primary objective of the ADVANCE multi-simulation framework is to address the needs for different design and verification tools, both discrete and continuous, test-based and formal to co-operate within a single development and verification framework.

2.2 Top-level Requirements

The top level requirements for the multi-simulation framework, as outlined in the ADVANCE proposal, are expanded and enumerated. Each requirement, *Rn* is accompanied by the motivation *Mn* behind that requirement.

R1 ADVANCE will implement a simulation framework, extending the existing RODIN platfom [Rod], within which independently-developed, heterogeneous components and sub-systems can be composed in a secure manner to enable *Cyber-Physical System* design and development.

M1 The great difficulty faced in assembling compatible models of the components and sub-systems which are at the appropriate level of abstraction to enable effective and efficient system verification presents a major barrier to the adoption of *Cyber-Physical System* engineering methods.

R2 In ADVANCE, the models of the components and sub-systems, developed using formal techniques, are imported directly into the simulation framework using two, complementary techniques.

R2.1 In the first, a simulation model is generated automatically from the formal model using techniques that build upon the code-generation methods developed in the FP7 DEPLOY project[Uni09].

M2.1 This technique will be used when the component model is mature and less-prone to frequent changes and will have the advantage of fast and efficient simulation.

R2.2 In the second, the model will be executed by its own host simulator and the ADVANCE multi-simulation framework will manage the communication of data between multiple simulation hosts, enabling simulation and verification of the whole system.

R2.2.1 More than one 3rd party simulator should be able to be employed during multi-simulation.

R2.2.2 The facility should support both discrete and continuous 3rd party simulators.

R2.2.3 If a 3rd party simulator is responsible for more than one system component, the connections between the components should still be managed by the multi-simulation master.

M2.2 This second technique will allow model development in other languages and environments is to be leveraged without the need to translate the models to the host ADVANCE format.

R3 The ProB formal model animator and model checker, also developed in the FP7 DEPLOY project, will be extended and its performance improved so that it too can be integrated within the ADVANCE multi-simulation framework.

M3 This will facilitate early system integration while the model is still being developed and allow interactive de-bugging. However, the execution performance of the embedded ProB model should still good enough to enable extensive regression tests to be run on the system so that modifications to the model can be verified in the system context.

R4 ADVANCE will extend and augment the test generation tool already proved in Rodin to facilitate constrained random testing.

M4 Hardware Verification Languages and tool environments such as Specman, VERA and SystemVerilog provide constraint-based random testing capabilities at a high level of abstraction and have been deployed successfully in the field of chip verification over the last decade. Providing the same capabilities for cyber-physical system verification within ADVANCE will leverage a proven verification technique in a new domain.

R5 The simulation model generation capability provided by ADVANCE will allow the option of collecting coverage measurements during simulation and test.

M5 Measuring the coverage achieved by testing is an essential component of any constrained random testing methodology. ProB already provides the facility to measure coverage of Nodes and operations of Event-B models. ADVANCE will bring these coverage measurement facilities to the wider simulation framework.

Chapter 3

Multi-Simulation Framework Top-level Specification

In order to meet the top-level requirements for the multi-simulation framework, it will be necessary to implement the fundamental, characteristic facilities of a typical discrete event simulator.

1. It will be necessary to implement the deterministic, simulation queuing mechanism that manages the timing and concurrency of simulation events.

2. It will be necessary to provide a Simulation Application Programming Interface (API) to this queuing mechanism with which simulation components can be developed and which the simulation model generation facility can use to generate efficient models.

3. The framework will provide a mechanism for 3rd party simulator executables to be initiated and for communication to be established with these executables

4. The framework will implement a closely-coupled interface with ProB within the Rodin Platform

5. The communication between the framework and the 3rd party simulators must be implemented and managed efficiently.

6. The framework must provide the capability for constrained-random test generation to be used on the system under test

7. The framework must provide the capability for coverage metrics to be collected and recorded during system simulation.

3.1 The Simulation Queuing Mechanism: the Two-list Simulation Algorithm

- Addresses Requirements: R1, R2.2, R3

The timing and synchronisation of communicating design components should be efficient and deterministic. This requirement can be met by implementing the well-established, two-list simulation algorithm which is employed in both VHDL and SystemC.

3.1.1 The Two-list Mechanism

Simulation operates in two, alternating *modes*, *update* mode and *evaluation* mode.

Consider a simple system with four *components*, A, B, C and D which communicate using *channels* C1 and C2, as shown in Figure 3.1. The channels are connected to the components using *output ports*, shown in black and *input ports*, shown in white.

Two lists, the *update* list and the *evaluation* list, a time-ordered queue, as shown in Figure3.2 are used to implement the algorithm.

In *update* mode, all value changes scheduled at current time to any of the channels are taken from the *update* list and executed. The *fanout* of each channel whose value has changed is examined and all components that have input ports connected to the channel are placed in the *evaluation* list. When all updates for current time have been completed, control switches to the *evaluation* list and all the components in this list are evaluated. These component evaluations result in future schedules which are placed in the *update* list. When all the component evaluations have been completed, control reverts to the *update* list and time is advanced to the next value change schedule in the list.

Note that the *order* in which the components are evaluated is immaterial since all communication between the components is managed by the update list and new channel input values are not visible until the next time that the components are evaluated.

Note also that the *minimum* delay for a future schedule is *one* unit. Zero delay is not supported.

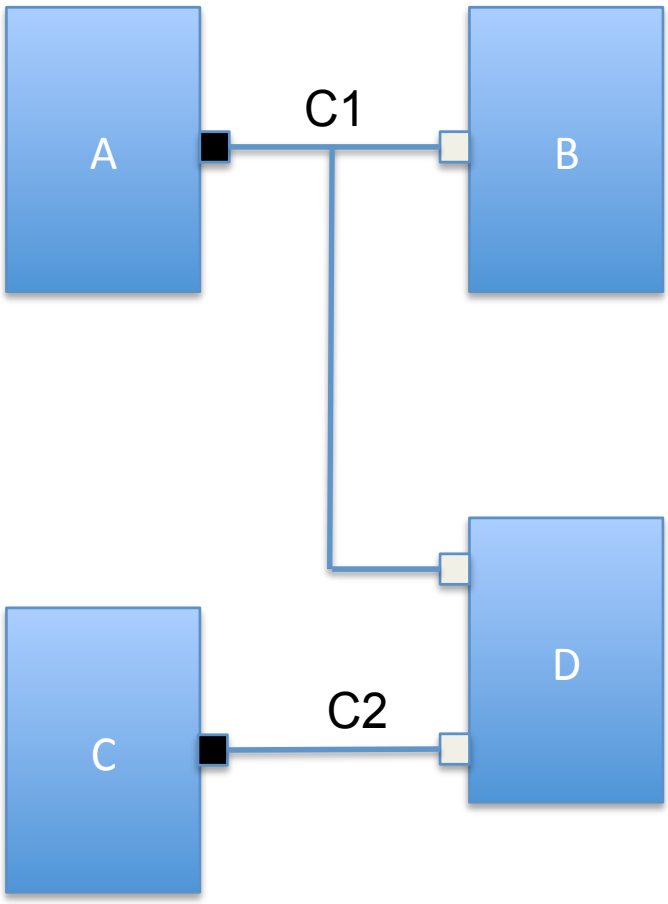

Figure 3.1: A Simple System

3.2 The Simulation API

- Addresses Requirements: R1, R2.1, R2.2, R3

Each component can only interact with other components running concurrently in the system using the Simulation API.

A component can write a value to a channel through its output port and assign an arbitrary delay to the write.

- *SetValue(OUT port, value, delay)*

This API call results in an entry being added to the update list at *current time* + *delay* to change the channel value to *value*.

Any component that has an input port connected to the channel that has changed will be evaluated when the delay on *SetValue* matures. The

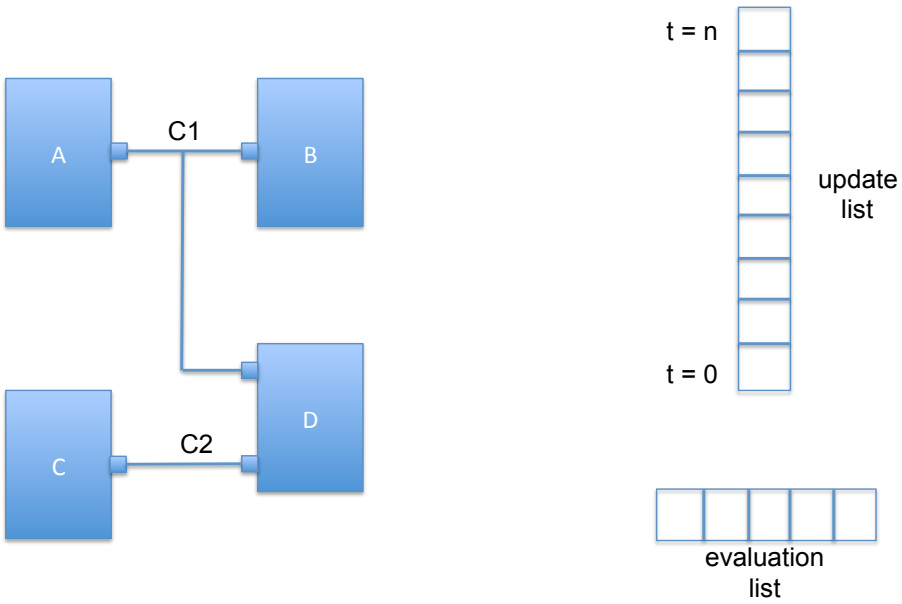

Figure 3.2: The two-list Algorithm

receiving component can then read the value of the channel connected to its input port.

- *GetValue(port)*

The component may also use this call to read the value of a channel connected to one of its output ports.

When a component is evaluated which has more than one input port, the component can determine which ports have changed.

- *HasChanged(port)*

Time cannot be advanced within a component execution, though a component can suspend its execution and ask to be evaluated at some time in the future.

- *ScheduleEval(delay)*

This API call results in an entry being added to the update list at *current time* + *delay* which will add the component to the evaluation list when the delay on *ScheduleEval* matures.

The developer of a component must ensure that evaluation of the component will suspend so that simulation can continue.

3.2.1 Simulating Systems without Discrete Delays: Unit Delay

In the case where the system is not concerned with actual delay, the notion of unit delay simulation will be implemented which has considerable performance advantages over general simulation with delay. There is no need for the update list to be implemented as a queue; a simple list will suffice as shown in Figure3.3.

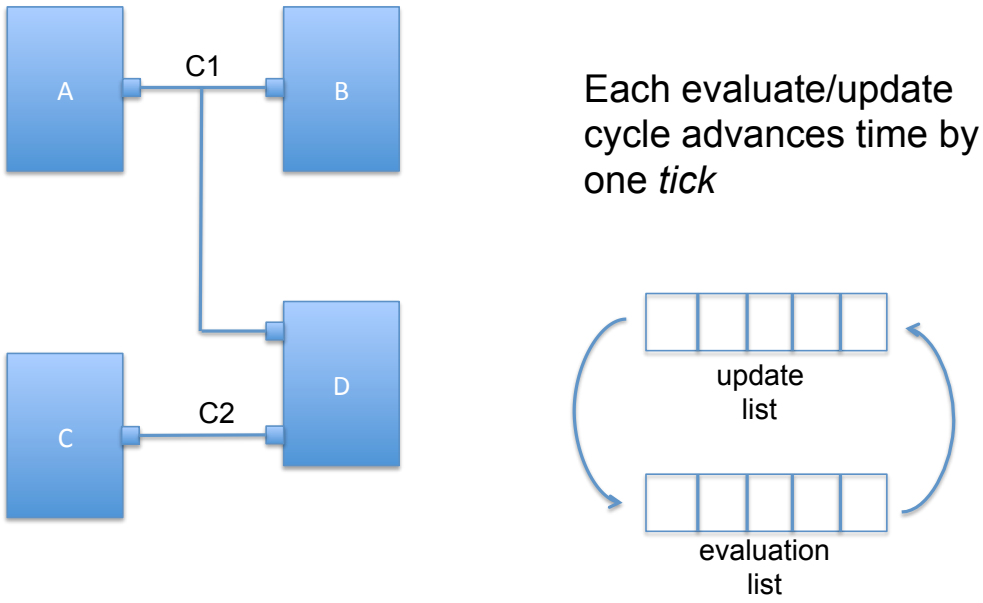

Figure 3.3: Unit Delay

3.3 Multi-Simulation with a 3rd Party Simulator

- Addresses Requirements: R1, R2.2

If one of the components of the system does not have a native implementation, as shown in Figure3.4, then the multi-simulation framework will initiate a *Master/Slave* multi-simulation with the 3rd party simulator in which that component is modelled.

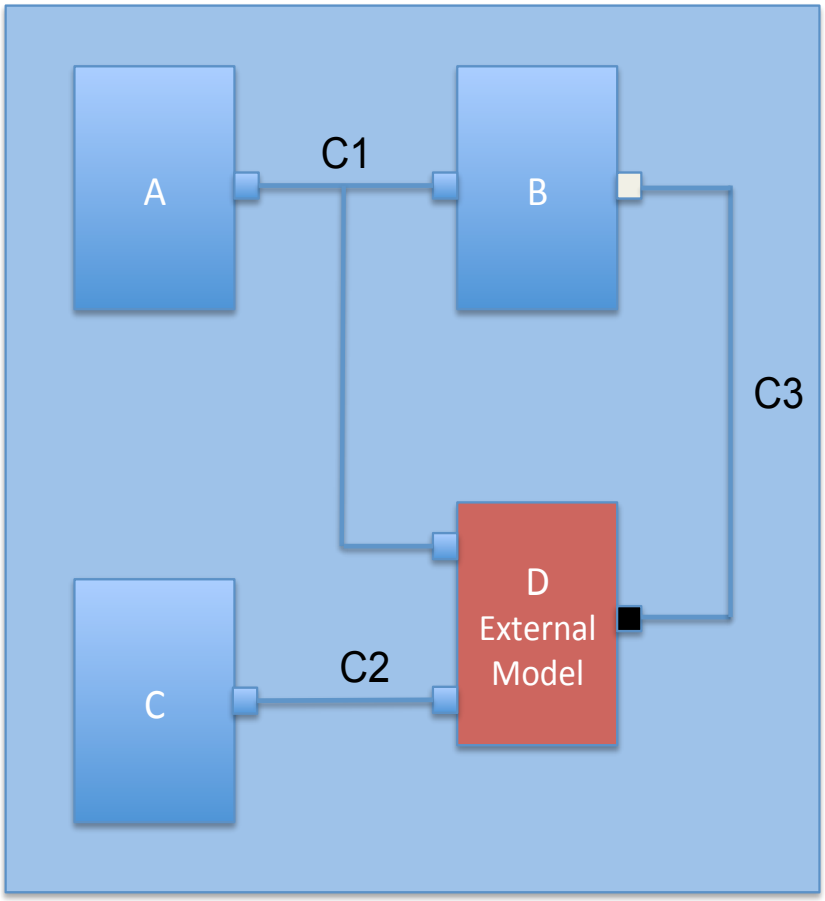

Figure 3.4: Co-Simulation

The multi-simulation framework, the *master* is responsible for the master event queue, the system topology (components and connectors), for initiating the slave 3rd party simulator and for establishing the inter-process communication. The component represented by the slave simulator is implemented

as a wrapper in the master which implements the simulator API and communicates directly with the slave simulator through its internal API using *sockets* or any like mechanism that the slave simulator supports.

3.4 Multi-simulation with ProB

- Addresses Requirements: R2.2, R3

Although the ProB formal animator and model checker will use the same, fundamental mechanism as any 3rd party simulator, the integration will be more tightly coupled and specialised within the Rodin environment to improve multi-simulation performance and to take advantage of its model checking and coverage collection capabilities.

The following three sections also describe multi-simulation facilities that will form part of the ProB integration.

3.5 Performance Optimisations with a Single 3rd Party Simulator

- Addresses Requirements: R1

When only one 3rd party simulator is used in multi-simulation as a Slave and the Slave provides an interface that allows the Master to see the Slave's future schedules, multi-simulation performance will be enhanced because it is not necessary to synchronise the simulators at each time step.

If the Slave simulator also has the ability to *backtrack*, the Master will allow the Slave to proceed freely with its simulation. If the Slave has moved in time beyond the next synchronisation point, the Master will instruct the Slave to backtrack to that synchronisation point before multi-simulation recommences.

3.6 Support for Constrained-Random Test Generation

- Addresses Requirements: R4

The multi-simulation framework will enable the development of comprehensive system test suites by providing an integrated testbench facility to support constrained-random test generation. The testbench represents the system environment and drives the system design during simulation through the *primary inputs* using the simulation API and records the results at the *primary outputs*, also using the API, as shown in Figure 3.5.

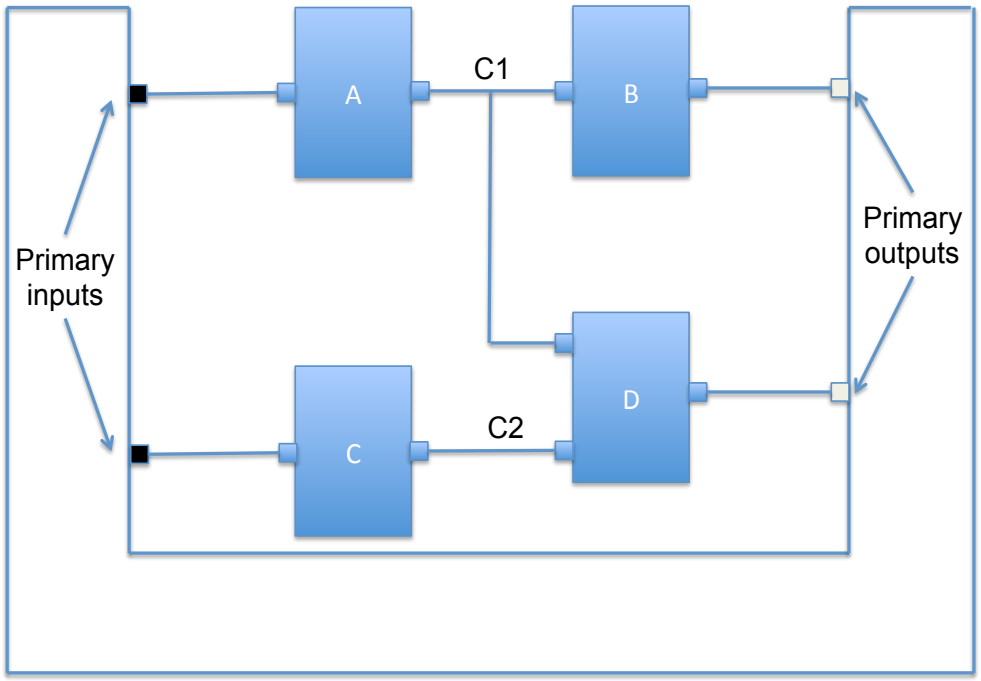

Figure 3.5: The Testbench

From the point of view of the multi-simulation architecture, the testbench is simply another component, connected to the system under test through input and output ports and using the multi-simulation API for communication and synchronisation.

The facility to produce a log file of simulation results will be provided for regression testing purposes.

When the testbench component is represented by ProB, the constrained-random test generation facilities supported by ProB will be used to verify the system under test.

3.7 Support for Coverage Metric Collection and Reporting

- Addresses Requirements: R5

When the testbench component is represented by ProB, the coverage metric facilities supported by ProB will be used to measure the coverage of the test suite developed for the system under test. ProB provides a measure of *transition coverage*.

Where a system component has been generated for simulation with the coverage option switched on, the transition coverage measurements for this component will be reported by the framework for each test.

Chapter 4

Proposed Multi-Simulation Architecture

ADVANCE will extend the RODIN platform to support the *multi-simulation core* as shown in Figure 4.1.

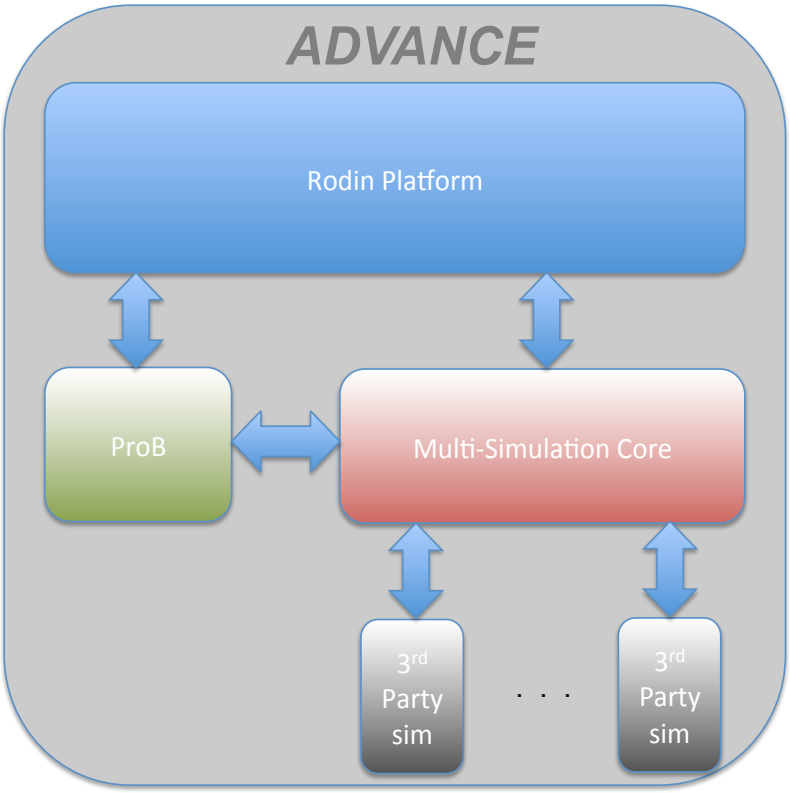

Figure 4.1: Proposed Multi-Simulation Architecture

The multi-simulation core will implement the specified simulation queues and API and will be responsible for establishing the synchronisation and communication mechanisms with the third-party simulators.

The Rodin platform user interface will be enhanced to allow the ADVANCE user to

- Describe the components and connections between the components for the cyber-physical system.
- Associate with each component the appropriate third party simulator.
- Describe and develop the simulation test bench.
- Specify the coverage metrics that will be collected during simulation.
- Collect a trace of the simulation results.

The multi-simulation architecture will be refined and further developed during the design phase.

Chapter 5

Summary

This deliverable presents the top-level requirements for the ADVANCE Multi-Simulation Framework and the motivation behind each of these requirements. It then presents a top-level specification for the multi-simulation core, the simulation queues and the API that the core should support and the mechanisms for communication and synchronisation with third-party simulators. It concludes with a description of the proposed multi-simulation architecture.

Bibliography

[Abr10] J.-R. Abrial. *The Event-B Book*. Cambridge University Press, Cambridge, UK, 2010.

[Rod] Rodin. The RODIN platform. Online at http://rodin-b-sharp.sourceforge.net/.

[Uni09] Newcastle University. DEPLOY - Industrial deployment of system engineering methods providing high dependability and productivity, May 2009. http://www.deploy-project.eu/.