Co-simulation of Event-B and Ptolemy II Models via FMI

Jean-Charles Chaudemar[1], Vitaly Savicks, Michael Butler, John Colley[2]

[1] Institut Supérieur de l'Aéronautique et de l'Espace, France
[2] University of Southampton, United Kingdom

1 Introduction

In the framework of model-based design formal modelling, verification and simulation of safety-critical systems are supported by several methods and tools. Interfacing these tools often becomes challenging for heterogeneous systems. The FMI standard enables integration of different simulation tools through artefacts called Functional Mockup Units (FMU) [1]. The FMI standard is mainly based on the concept of scalability of the simulation as it deals with heterogeneous cyber-physical systems. The combination of discrete behaviour and continuous-time environment is a common case study in hybrid simulation. Moreover, another aspect of the FMI is to enhance the capability of the tools. Thus, a collaborative simulation between the Rodin [2] and Ptolemy [3] is leveraged by both platforms. While Event-B is enhanced by new models of computation of Ptolemy, Ptolemy leverages the expressivity and properties validation (theorem/invariant proofs) implemented by Event-B. The main rationale of the co-simulation between Event-B and Ptolemy relies on the intention of dissimilarity and complementarity of the modelling viewpoints. Event-B provides formal modelling by specifying conditions, actions and properties that manage discrete event behaviour, whereas Ptolemy gives a structural viewpoint in terms of actors, components or functions with relation to concerned behaviour. Thus, the association of Ptolemy and Event-B puts together structural and formal aspects.

This paper focuses on the collaborative simulation of models supported by both Ptolemy II and Event-B. The ongoing work consists of the design of a diagrammatic co-simulation surface and its application to a controller case study.

2 Event-B

Event-B is a refinement-based formal method for modelling and analysis of complex safety-critical systems [4]. A system is modelled in Event-B as a collection of state variables and events (guarded actions) that act on variables. System properties are specified as invariants on state variables and are formally verified by deductive proof. The key mechanism of the method is the refinement, i.e. incremental development from an abstract to a concrete model that splits the complex task of formal verification into manageable proofs, allowing to detect the errors as soon as they are introduced along the modelling process. The extensible Rodin platform aids this process with automatic provers and additional modelling features from third-party plug-ins. Models can also be validated by the ProB model checker and animator tool [5].

In general an Event-B model consists of a static *context*, which contains *constants* and *axioms*, and a dynamic *machine* that comprises *variables* (sets, binary relations, functions, etc.), *invariants* (constraints on variables that must hold) and *events* that model the behaviour. An event may contain *parameters*, *guards* (event-enabling conditions) and *actions* that modify state variables. Events are atomic (all actions happen instantaneously) and their choice is non-deterministic, i.e. when multiple events are enabled a single event is selected non-deterministically.

There may exist relationships in Event-B. A machine can *see* a context in order to use its constants. On refinement: 1) a context can *extend* another context by introducing new constants; 2) a machine can *refine* another machine by introducing more details either in the systems state (data refinement) or in the behaviour (horizontal refinement).

3 Ptolemy II

Ptolemy II is an open source framework for modelling and simulation of cyber-physical systems. This platform enables the hierarchical composition of components for the design of embedded systems. A system component is described by a set of interfaced actors in interaction. The design of a system is actor-oriented in Ptolemy [6].

Actors are executable Ptolemy components which have syntax and semantics. An actor reacts to stimuli (token) at its input port and produces stimuli at its output port. Different relations could be applied to actors like composition (hierarchy), or high-order relation (an actor which enables a reduction and a simplification of a model).

Directors are special components which give meaning to a model. Various directors are implemented in Ptolemy so as to provide different models of computation and determine how actors communicate. The execution of actors is composed of three main phases:

- Setup phase divided in two steps: preinitialise and initialise. The preinitialise step is performed once at the very beginning of the execution. This step relates to a static analysis of the actor's actions and communication. It is just followed by the initialise step. The latter sets initial values to data and resets local state.
- Iterate phase or sequence of iterations divided into three steps: prefire, fire, postfire. The prefire step enables to meet preconditions related to the execution termination. The fire step reads the input data, computes the actor's behaviour and produces output data. Then the postfire step updates the state. Iteration is resumed possibly several times to reach a steady state.
- Wrapup phase: this phase ensures that the execution is properly terminated.

The execution of an actor is enabled through the implementation of a model of computation (MoC) associated with this actor. The MoC describes how the actor interaction is related to other actors. Indeed the MoC defines the communication semantics.

4 Co-simulation Approach

The FMI standard relies on the development of artefacts called Functional Mockup Units (FMU). An FMU is a zipped file, which contains an xml document describing the variables and capabilities of the model, and a dynamic-link library (dll) implementing the behaviour performed by the tool that simulates it. We are using the FMI standard and the structure of the FMU to bundle Ptolemy models for co-simulation framework developed in the Rodin platform [7].

Our approach to perform a co-simulation between Rodin and Ptolemy consists of several steps. First, we develop an Event-B model of controller in Rodin and a Ptolemy model of the environment behaviour we want to co-simulate. The latter model consists of a composite actor and sub-actors coordinated by the Synchronous Data Flow (SDF) director. The model takes an input signal, generated by the model of controller, and produces a dependant feedback output signal. The controller model is a standard Event-B machine extended with modal semantics using the state machines plug-in for Rodin [8].

As a second step we generate C code from the Ptolemy model via automatic code generator module and transform it into a .dll using FMU SDK[3]. DLL and corresponding model description XML are bundled into an FMU file.

Next we import the FMU into the Rodin co-simulation environment together with Event-B model and compose two models into a component graph. The component composition is achieved using the Component diagram plug-in for Rodin – a front-end of the co-simulation framework, developed as part of the ADVANCE project[4].

Finally we implement the FMI master algorithm in Groovy scripting language on top of ProB animator for Event-B and integrate it with the composition graph model. This algorithm initialises and coordinates the simulation process between two components, as well as provides the simulation output. The Event-B component (controller) is coordinated by the master via ProB API, while the Ptolemy component (environment) is coordinated using FMI.

5 AOA Sensor Processing

5.1 Case Study

Traditionally in aeronautics, safety-critical systems are based on a rigorous dissimilarity applied to the architectures to avoid common points of failure. One of

[3] FMU SDK source code is available at `https://github.com/cxbrooks/fmusdk`

[4] ADVANCE: EU Project IP-287563, `http://www.advance-ict.eu`

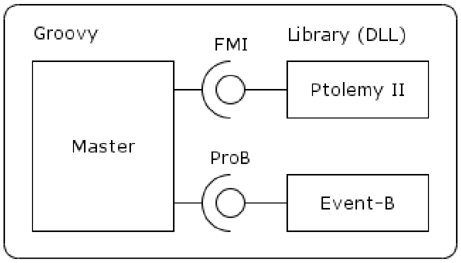


Fig. 1. Co-simulation process in Rodin

the widely used technics is the COM/MON architecture which consists of two separated command and monitoring units [9]. All input and output data are monitored permanently and only the command unit sends the command data to actuators under the control of the monitoring unit. This dual behaviour is of interest in the application of the co-simulation between Ptolemy and Event-B. The environment and the command data generation are devoted to Ptolemy model whereas Event-B model is in charge of the monitoring unit and controls the output data to be sent.

Likewise our case study extracted from a J. Rushby's presentation [10] relates to the utilisation of three Angle of Attack (AOA) sensors in the Flight Control System (FCS). As safety is the first objective that is required for aircraft avionics systems, these three sensors are combined in an algorithm which provides the most relevant value of measurement. The algorithm is synthetically based on the computation of the median of the three measures so as to detect erroneous values and erroneous devices, along with on the computation of the average value between two of the three sensors.

From this illustration, our study consists in modelling the strategy of combination of the sensors by distinguishing an AOA sensors algorithm modelled in Ptolemy and the same algorithm modelled in Event-B.

5.2 Ptolemy Model

The Ptolemy model of AOA sensors consists of a SDF MoC as a first abstraction in order to only focus on the specification of the algorithm part (figure 2). Indeed the time aspect is modelled by series of iterations based on the measurement sampling frequency of 20Hz. Thus to represent a delay of 1.2 seconds, 24 iterations are performed. The first *CompositeActor* mainly describes the processing for the detection of erroneous measures: the median of the three measures is computed, then the difference between each measure and the median is compared to a threshold; the result of this comparison indicates if the measure is erroneous or not. This processing is followed by the *FSMActor* for each sensor. The *FSMActor* enables to identify a erroneous sensor after 1 second of erroneous measures sent by this sensor.

Similarly a second *CompositeActor2* follows the previous and deals with the strategy about the final AOA measure (figure 3). Its output is either the current mean of sensor 1 and sensor 2 or the previous mean held during 1.2 seconds. It is provided by the *Case* actor associated with the *FSMActor*. The *CompositeActor2* behaviour is also modelled in Event-B.

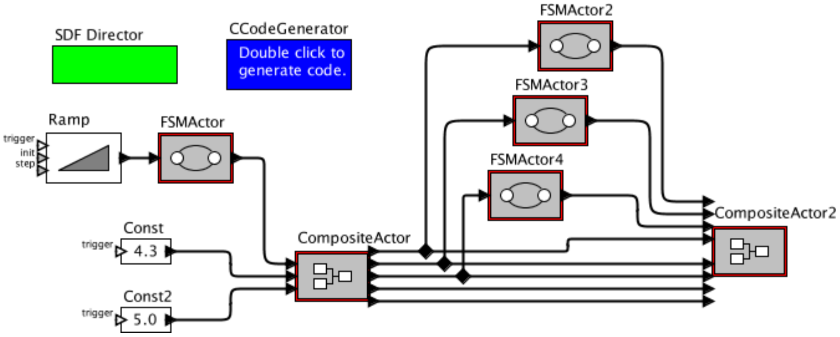


Fig. 2. Ptolemy model of AOA processing

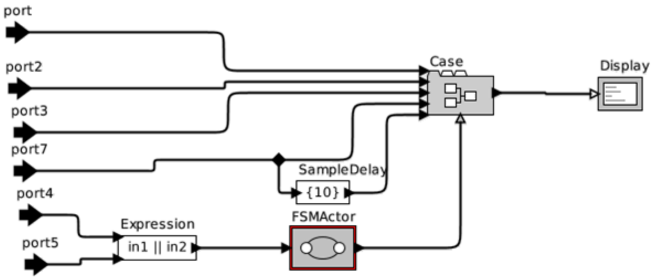


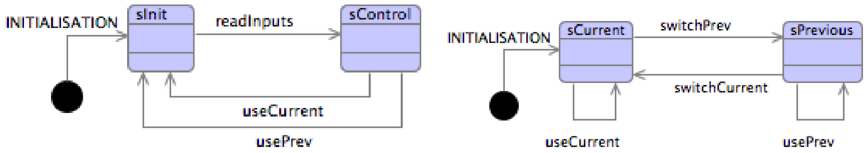
Fig. 3. Modelling of output strategy in CompositeActor2

5.3 Event-B Controller

The controller actor of the Ptolemy model has been duplicated, which is a standard practice in aeronautics [9], by an Event-B model. The model has been implemented using a state machine plug-in for Rodin [8], which allowed the control flow, i.e. the ordering of events, to be specified. Two state machines have been defined in the controller:

– *controlSM* (Figure 4(a)), which defines the control flow from an interface (to master) perspective, with a `readInputs` event for reading sensor inputs and two output events `useCurrent` and `usePrev` that model current and previous mean output, respectively.

- *modeSM* (Figure 4(b)), which defines the modal semantics of the controller and models a delay in the output. The controller operates in either `sCurrent` or `sPrevious` mode, switching to the latter when a threshold is exceeded, and to the former after 1.2 seconds since the threshold crossing.



(a) Control (b) Mode

Fig. 4. State machines of the Event-B controller

The delay of 1.2 seconds is modelled by a variable `modeCounter` that is set on a `switchPrev` execution and is decremented by `usePrev` event until it reaches 0, at which point a `switchCurrent` event becomes enabled. The Event-B code for switch events is given below. Note that `sPrevious` and `sCurrent` are modelled as boolean variables (translated from the state machines automatically).

```
event $switchPrev$
  where $sCurrent = TRUE$
   $medianS1 > THRESHOLD \vee medianS2 > THRESHOLD$
  then $sCurrent := FALSE$
   $sPrevious := TRUE$
   $modeCounter := DELAY$
  end

event $switchCurrent$
  where $sPrevious = TRUE$
   $modeCounter = 0$
  then $sPrevious := FALSE$
   $sCurrent := TRUE$
  end
```

5.4 Simulation Master

Our simulation master, displayed below, is written in the Groovy language to facilitate the scripting capabilities of the ProB animator tool, which is used in the Rodin for animating (simulating) Event-B models. The design of the master is based on a proposed algorithm demonstrated in the FMI for Co-simulation specification [11].

```
import de.prob.cosimulation.FMU
```

```
ctrl = api.eventb_load("./eventBControl.bum") as Trace
ctrl = ctrl.anyEvent(); // setup constants
ctrl = ctrl.anyEvent(); // initialize machine

fmu = new FMU("./ptolemySensors.fmu")
fmu.initialize(0.0, 30.0)

time = 0.0
decimals = 2

while(time < tStop) {
   // read sensor outputs
   mds1 = (fmu.getDouble("v1") * decimals) as int // |median - s1|
   mds2 = (fmu.getDouble("v2") * decimals) as int // |median - s2|
   mn = (fmu.getDouble("v3") * decimals) as int   // mean

   // write control input
   ctrl = ctrl.readInputs("mds1="+mds1+"&mds2="+mds2+"&mn="+mn)

   // simulate a step
   while (!ctrl.current.edge.name in ["useCurrent","usePrev"])
      ctrl = ctrl.anyEvent()
   time = fmu.doStep(time, 1);
}
```

The key aspects of the master script are the passing of the sensor values, read from the Ptolemy FMU, to the Event-B controller via `readInputs` event parameters, and the semantics of a simulation step of the Event-B component – a non-deterministic execution of events until one of the step-finishing events (`useCurrent` or `usePrev`) is executed. The step of an FMU component is performed by a standard FMI call `doStep`. Another important aspect is the treatment of the Real-typed signals by Event-B, which currently does not support the Real type [12]. Real values from the FMU are converted to Event-B integers by agreed decimal point shift.

5.5 Co-Simulation Results

A successful simulation has been carried out for 30 seconds with a step size 1. The results are shown in Figure 5.

The specification of the system defines the operational frequency of 20Hz, which we have mapped to 20 steps. Consequently, the delay of the mean value for 1.2 seconds was modelled by 24 steps, visible as a `meanOut` signal on the plot. Notice that the actual delay is 25 steps due to a delay of the control signal for 1 step, caused by the discrete data synchronisation and simplistic algorithm of the master, i.e. a mean signal received by the controller at t=N is processed

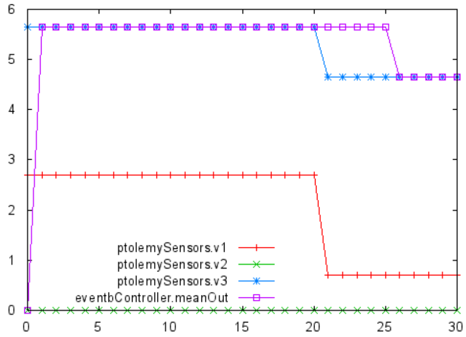


Fig. 5. Co-simulation output (simulation time = 30s, step size = 1s)

and output at t=N+1. This behaviour can be improved by adapting a predictive control logic and a master with a rollback and step-size prediction [13].

6 Conclusions and Outlook

In this work we have tried to demonstrate the proof of concept of a co-simulation between a well-known formalism, such as Event-B, and a heterogeneous simulation tool Ptolemy II. A mixture of domain specific modelling and simulation tools, which is based on the rigorous analysis approaches for verifying safety and reliability constraints, is arguably an essential technology in the face of the development of cyber-physical systems [14–16]. Our simulation results have showed that an integration is possible, even more advantageous, as it is based on the tool-independent FMI standard, which in theory allows a Ptolemy FMU to be used in any other FMI-compliant simulator. The results have also revealed the existing problems and limitations of both tools, namely limited support of directors and actors by the Ptolemy's C code generator and a strong requirement for Real type support in the Event-B language. We hope that these issues will be addressed as the tools develop.

This ongoing work highlights the ability of the FMI co-simulation between Ptolemy and Event-B, although both tools do not provide a support for FMI standard yet. The main difficulties have been to generate the FMU file from Ptolemy model manually since its code generation component is working partially. Therefore, several Ptolemy models have been carried out to ease the code generation.

Within the framework of the FMI co-simulation the outlook is twofold: to take into account continuous time and to implement the verification of combined system properties.

References

1. Blochwitz, T., Otter, M., Arnold, M., Bausch, C., Clauß, C., Elmqvist, H., Junghanns, A., Mauss, J., Monteiro, M., Neidhold, T., et al.: The functional mockup

interface for tool independent exchange of simulation models. In: Modelica'2011 Conference, March. (2011) 20–22

2. Abrial, J.R., Butler, M., Hallerstede, S., Hoang, T.S., Mehta, F., Voisin, L.: Rodin: an open toolset for modelling and reasoning in Event-B. International journal on software tools for technology transfer **12**(6) (2010) 447–466
3. Brooks, C., Lee, E.A., Liu, X., Zhao, Y., Zheng, H., Bhattacharyya, S.S., Cheong, E., Goel, M., Kienhuis, B., Liu, J., et al.: Ptolemy II-heterogeneous concurrent modeling and design in Java. (2005)
4. Abrial, J.R.: Modeling in Event-B: system and software engineering. Cambridge University Press (2010)
5. Leuschel, M., Butler, M.: ProB: an automated analysis toolset for the B method. International Journal on Software Tools for Technology Transfer **10**(2) (2008) 185–203
6. Lee, E.A.: Heterogeneous actor modeling. In: Proceedings of the ninth ACM international conference on Embedded software. EMSOFT '11, New York, NY, USA, ACM (2011) 3–12
7. Savicks, V., Bendisposto, J., Butler, M., Colley, J.: Co-simulation of Event-B and Continuous Models in Rodin. In Butler, M., Hallerstede, S., Waldén, M., eds.: Proceedings of the 4th Rodin User and Developer Workshop. Volume 18 of TUCS Lecture Notes., TUCS (2013)
8. Savicks, V., Snook, C., Butler, M.: Event-B Wiki: Event-B Statemachines. `http://wiki.event-b.org/index.php/Event-B\_Statemachines` (2011)
9. Sghairi, M., Aubert, J.J., Brot, P., De Bonneval, A., Crouzet, Y., Laarouchi, Y.: Distributed and Reconfigurable Architecture for Flight Control System. In: Digital Avionics Systems Conference, 2009. DASC '09. IEEE/AIAA 28th. (2009) 6.B.2–1–6.B.2–10
10. Rushby, J.: Safety, Fault-tolerance, Verification, and Certification for Embedded Systems. `http://chess.eecs.berkeley.edu/eecs149/lectures/Rushby-SFVC.pdf` (2009)
11. MODELISAR: Functional Mock-up Interface for Co-Simulation, Version 1.0. `https://svn.modelica.org/fmi/branches/public/specifications/FMI_for_CoSimulation_v1.0.pdf` (October 2010)
12. Abrial, J.R., Su, W., Zhu, H.: Formalizing hybrid systems with Event-B. In: Abstract State Machines, Alloy, B, VDM, and Z. Springer (2012) 178–193
13. Broman, D., Brooks, C., Greenberg, L., Lee, E.A., Masin, M., Tripakis, S., Wetter, M.: Determinate composition of FMUs for co-simulation. In: Proceedings of the Eleventh ACM International Conference on Embedded Software, IEEE Press (2013) 2
14. Lee, E.A.: Cyber physical systems: Design challenges. In: International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC). (May 2008) Invited Paper.
15. Rajkumar, R., Lee, I., Sha, L., Stankovic, J.: Cyber-physical systems: the next computing revolution. In: Proceedings of the 47th Design Automation Conference, ACM (2010) 731–736
16. Marwedel, P.: Embedded and cyber-physical systems in a nutshell. (2010)