Railroad Crossing Heterogeneous Model

Matias Vara Larsen[1], Arda Goknil[2]

[1] University of Nice Sophia Antipolis,CNRS, I3S
route des Lucioles, BP 93 06902 Sophia Antipolis Cedex, France
<varalars@i3s.unice.fr>
[2] INRIA Sophia Antipolis
route des Lucioles, BP 93 06902 Sophia Antipolis Cedex, France
<arda.goknil@inria.fr>

Abstract. Systems are getting more and more complex and usually involve many stakeholders. Stakeholders are concerned by different aspects of the system, potentially supported by multiple Domain Specific Modeling Languages (DSMLs). The DSMLs are usually different not only in their syntax but also in their behavioral semantics. In order to provide simulation and/or verification of the overall system, it is mandatory to compose the DSMLs behavioral semantics. The composition of the DSMLs behavioral semantics results in the coordination of different models that conform to the DSMLs. This paper presents the coordination of the models representing a railroad crossing management system. The system is composed of two models, conforming to two different DSMLs. The paper explains the behavioral semantics of these DSMLs and presents a simple coordination of the models used in the example.

1 Introduction

Complex heterogeneous systems often rely on several Domain Specific Languages (DSMLs). A DSML efficiently specifies the domain concepts as well as their behavioral semantics within a particular domain. Several models following different behavioral semantics are combined by heterogeneous systems. It is mandatory to compose the DSMLs behavioral semantics in order to provide simulation and/or verification of the overall system. The composition of the DSMLs behavioral semantics results in the coordination of different models that conform to the DSMLs.

We present the coordination of the models representing a Railroad Crossing Management System (RCMS). The system is composed of two models, conforming to two different DSMLs. The description of each DSML is split into the language units Abstract Syntax (AS), Domain Specific Actions (DSA), Model of Computation(MOC) and Domain Specific Events (DSE). The AS specifies the concepts of the language and their relationships. An instance of the AS is a model. The DSA represent both the execution state and the execution functions of a DSML. An instance of the DSA represents the state of a specific model during the execution and the functions to manipulate such a state. The MOC represents the concurrency aspects in the language. An instance of a MOC is defined for a specific model, conforming to the DSML. It is the part of the concurrency model that specifies the possible partial orderings between the events instantiated with regards to the model. The DSE specify the coordination between the events from the MOC and the execution function calls from the DSA. In this paper we use the DSE as a behavioral interface to coordinate different models through constraints between domain specific events. Only the concurrency part (MoC) of DSMLs is detailed. It is described in ECL (Event Constraint Language) [1]. ECL is used to specify, at the language level, the constraints used in a concurrency model for any model. Then, for any model conforming to the DSML, the constraints given in ECL are used to automatically create an executable concurrency model (an instance of MOC) in CCSL [2]. This model is used by TimeSquare [3] to provide a simulation that gives the partial order between the domain specific events in the model.

2 Example: the Railroad Crossing Management System

The Railroad Crossing Management System (RCMS) regulates the traffic that crosses the rails to avoid possible collisions between the train and cars. The overall system is composed of two subsystems: *the Barrier Detection Controller* (BDC) and *the Barrier Motor Controller* (BMC). The BDC detects the location of the train and commands the BMC to control the barrier. The BMC opens and closes the barrier with the correct handling of the motor. When the train's approach is detected, the BDC commands the BMC to close the barrier. The BMC starts the motor of the barrier and reads the position of the barrier constantly. It stops the motor and informs the BDC when the barrier is completely down. When the BDC detects that the train is far away from the road crossing, it commands the BMC to reopen the barrier. The BMC starts the motor and reads the position of the barrier constantly. It stops the motor and informs the BDC when the barrier is completely up.

The subsystems are given in two different models, conforming to two different DSMLs. The opening and closing of the barrier have to be synchronized with the detection of the train location. The synchronization need requires the coordination of the concurrency parts (MoC) of the models for the BDC and the BMC subsystems.

2.1 The Barrier Detection Controller

The BDC is given in a Timed Final State Machine (TFSM) model (see Figure 1). The TFSM model waits for the *start* event after its initialization and then goes to the *Up* state. In this state it waits for the reception of an event called *TrainComing*. When the *TrainComing* event is received, it sends the *doClose* event and goes to the *Closing* state. The *doClose* event makes a request to close the barrier. In the *Closing* state, it waits for the reception of the *Closed* event. When the *Closed* event is received, it goes to the *Down* state. The *Closed*

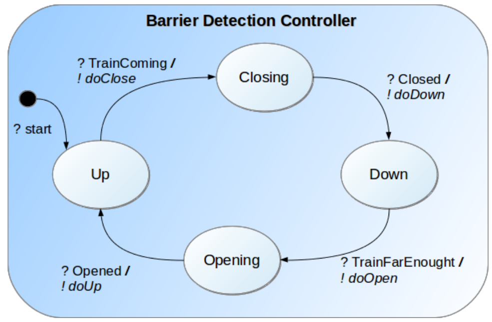

Fig. 1. Barrier Detection Controller TFSM

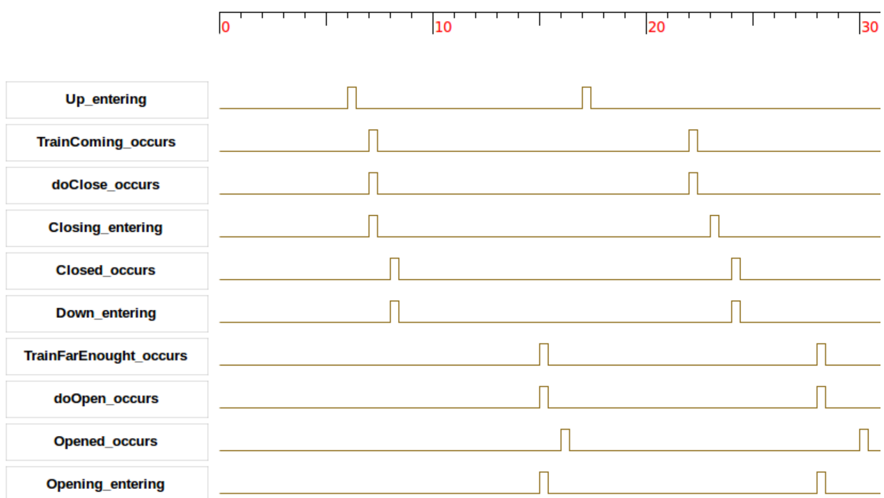

Fig. 2. Timing output of the simulation with TFSM semantics

event informs that the barrier is completely down. It stays in this state until the reception of the *TrainfarEnough* event. When this event occurs, the *doOpen* event is sent and the *Opening* state is reached. The *doOpen* event is a request to open the barrier. In this state it waits for the reception of the *Opened* event and then, it returns to the *Up* state.

The instance MOC of the TFSM model is specified in the CCSL language. The events *TrainComing*, *TrainFarEnough*, *Closed* and *Opened* are not constrained by the CCSL specification. In order to simulate them, we create a scenario. The scenario specifies the occurrence of those events. In the case of *TrainComing* and *TrainFarEnough*, we specify the occurrence periodically and alternatively in time. Then, the scenario specifies that *Closed* precedes *TrainComing* and *Opened* precedes *TrainFarEnough*. These constraints are added to the generated CCSL specification manually. We execute the concurrency model (the CCSL specification) of the TFSM model in TimeSquare. The execution of the concurrency model produces a timing diagram representing the occurrences of the events (see Figure 2). The occurrence of *doClose* coincides with the occurrence of *TrainComing*. These synchronizations are defined in the concurrency model and they are part of the behavioral semantics. The *Closed* event fires the transition and the *Down* state is reached (*Down_entering* in Figure 2).

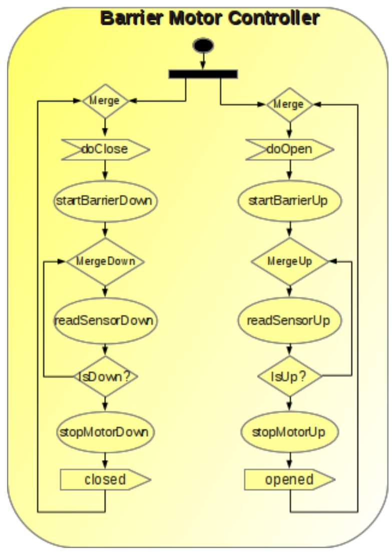

Fig. 3. Barrier Motor Controller Activity Diagram

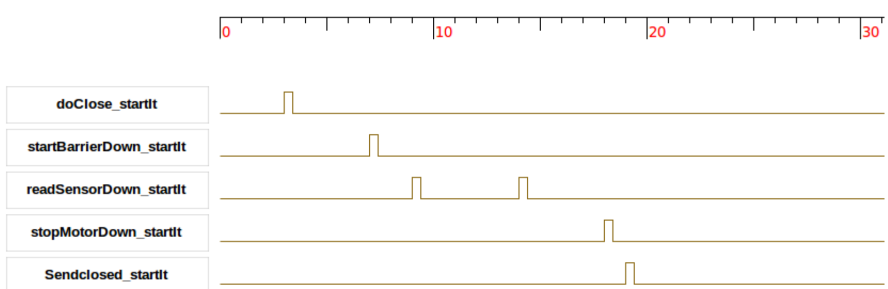

Fig. 4. Timing output of the simulation with fUML semantics

2.2 The Barrier Motor Controller

The BMC is represented by an activity diagram following the fUML semantics (see Figure 3). When the activity is initialized, it goes through a fork node. After the fork node, the activity is split into two concurrent control flows. One branch does the opening of the barrier and the other one does the closing. When the *doClose* event is received, the activity invokes the *startBarrierDown* action and goes to the *readSensorDown* action (through a merge node). Then, it goes to a decision node to check if the barrier is completely down. If it is not down, it goes to the *readSensorDown* action and checks the barrier again. If the barrier is down, it invokes the *stopMotorDown* action, sends the signal *closed* and goes back to the event reception node. The workflow of the opening branch is similar. When the *doOpen* event is received, the activity invokes the *startBarrierUp* action and goes to the *readSensorUp* action. The barrier is checked for being completely Up. If the barrier is completely up, the motor is stopped and the *opened* event is sent.

After the instance MoC of the activity diagram is given in CCSL and the scenario is correctly defined, the concurrency model is executed by TimeSquare. In Figure 4 we see the execution of one of the branches which corresponds to the closing of the barrier.

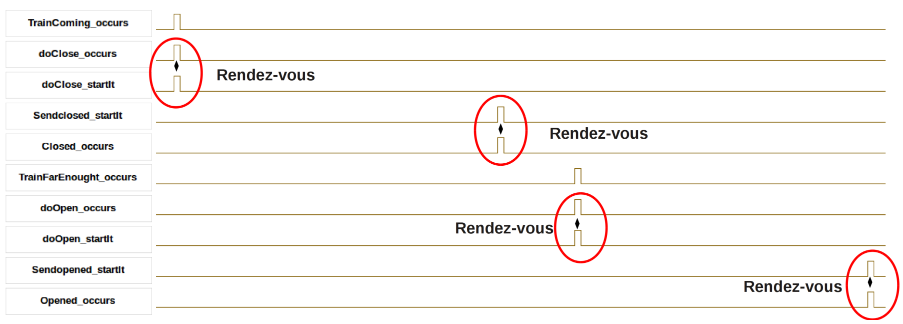

Fig. 5. Timing output of the simulation of TFSM and fUML composition

2.3 Composition of the Models

We are able to model and simulate each subsystem in isolation. Now the models for subsystems have to be composed to simulate and verify the overall system. The composition is expressed through constraints between the following pairs of events: [doClosed_startIt, doClose_occurs], [doOpen_occurs, doOpen_startIt], [Closed_occurs, Sendclose_startIt] and [Opened_occurs, Sendopen_startIt]. The instance DSEs of each model is used to compose the behavioral semantics of the models. The constraints between the instance DSEs provides the composition. In this example we decided to specify a strong synchronization between the pairs of the events (a rendez-vous semantics). For instance, the *doClosed_occurs* event coincides with the *doClose_startIt* event. The same strong synchronization is specified between the rest of the event pairs.

We manually created a CCSL specification which includes the TFSM, the fUML and the composition constraints. For the events *TrainComing_occurs* and *TrainFarEnough_occurs*, we keep the scenario created before. The CCSL specification of the two composed models was executed in TimeSquare. Figure 5 shows the timing diagram of the composed models. Although we only have the *coincides* constraint in this example, other types of constraints like *causality* can be specified for the composition of the models.

The composition constraints also constrain the possible behaviors of the individual models. This is the case for the *doOpen_startIt* and *doClose_startIt* events in the activity diagram. The fUML behavioral semantics does not constrain these events and their concurrent occurrences are allowed. After the composition, these events are synchronized with the *doOpen_occurs* and *doClose_occurs* events in the TFSM model. The TFSM behavioral semantics does not allow the concurrent occurrence of the *doOpen_occurs* and *Close_occurs* events. Consequently, the TFSM constrains the *doClose_startIt* and *doOpen_startIt* events and the concurrent occurrences of them are no longer possible. Figure 5 shows the alternation for the *doClose_startIt* and *doOpen_startIt* events. The resulting behavior of the overall model is then a subset of the union of both possibles behaviors.

3 Conclusion

In this paper we presented the model of a Rail Crossing Management System. The system is composed of two subsystems modeled in different DSMLs: the Barrier Detection Controller (BDC) given in TFSM and the Barrier Motor Controller (BMC) given as an activity diagram in fUML. We presented a simple approach to compose these models with their behavioral semantics. A DSML is defined as a tuple composed of the Abstract Syntax (AS), the Domain Specific Actions (DSA), Model of Computation (MOC) and Domain Specific Events (DSE). The DSE of each DSML has been used as a crude behavioral interface for the composition. The composition constrains the behavior of each MoC so that the final behavior is a subset of the union of both possible behaviors.

Currently, the composition is done manually at the model level and we are investigating the reification of the composition with operators specified at the language level.

References

1. Deantoni, J., Mallet, F.: ECL: the Event Constraint Language, an Extension of OCL with Events. Research report RR-8031, INRIA (July 2012)
2. Mallet, F., DeAntoni, J., Andr, C., de Simone, R.: The Clock Constraint Specification Language for building timed causality models. Innovations in Systems and Software Engineering 6 (2010) 99106
3. Deantoni, J., Mallet, F.: TimeSquare: Treat your Models with Logical Time. In: TOOLS. Volume 7304 of LNCS., Springer (May 2012) 3441