AA4MM coordination model: event-B specification

Julien Siebert, Joris Rehm, Vincent Chevrier, Laurent Ciarletta, Dominique Mery

December 2, 2009

Abstract

We develop a framework called Agent and Artefact for Multiple Models coordination (AA4MM)[1]. It is is intended to make the design and the implementation of complex systems simulation modular and decentralized. Our main goal is to reuse existing models and simulators and to make them interact in order to simulate different levels of abstraction. The main constraint is that people involved into the design process do not have to care about anything else but modelling. Coordination challenges remain to the framework. This report presents the event-B specification of the coordination model proposed in [SCC10]. Its goal is to check that the system is never blocked (no deadlock): no simulator is waiting for another indefinitely.

Contents

[1] http://www.loria.fr/∼siebertj/aa4mm/aa4mm.html

1 Introduction

This document corresponds to the proof of properties of the AA4MM coordination model. It is not intended to describe or to argue this model. For these aspects see [SCC10].

1.1 Brief description of AA4MM

The Agent and Artefact for Multiple Models co-evolution (AA4MM) framework is a meta-model based upon the Agent and Artefact paradigm [ORV08] developed in order to couple heterogeneous models. Its implementation purpose is to build a society of interacting and co-evolving (existing) simulation models. This means that each simulation model possess its own representations of time and time advance, of the elements of the phenomena. Each model makes its representation states evolves and from time to time, exchanges information with another model.

1.2 Issues

1. **Data coherence between models** Scales or dimensions in which a piece of data is represented could be different from a model to another. For example, a position $pos_1 = < x, y, z >$ (with x, y and z expressed in $meters$) in a first model can be represented only in two dimensions in a second one: $pos_2 = < x', y' >$ (with x' and y' expressed in $kilometers$). A solution proposed in [BRC07] is to define operations (projection, discretisation, reduction) in order to achieve this coherence.

2. **Data compatibility between simulators** Each simulator could implement a single piece of data in its own way (integer, float...) or some simulators may not implement all aspects of a given model. These challenges are discussed in [RAF+04]. A solution is to add an entity (a program) between the simulators. Its role is to translate the data in order to respect the compatibility between simulation tools.

3. **Time coherence between models** Each model could have its own time representation. We need to assure, for example, that a time value $t_1 \in \mathbb{R}^+$ in a first model correspond to a time value $t_2 \in \mathbb{N}$ in a second one. A possible solution is to express an operation that makes the correspondence between both time values.

4. **Coordination** Each simulator process its own time execution (discrete events, time steps). We must beware of coordination between them. That is, if a simulator needs some data from another one, they must be synchronized in order to respect simulation time consistency.

We choose to use Agent and Artefacts paradigm in order to deal with coherence and compatibility issues. The coordination can be done with central and global scheduler that synchronizes all the simulators as in [BRC07] for example. The originality of our model is its decentralized coordination. This raises the question to ensure the synchronization. This document presents the proofs of synchronization.

1.3 Key elements of AA4MM to understand

1. Interaction between the different models is only exchange of data.

2. When a model M_i is executed, it produces data δ_i at time ct_i. These data δ_i will be updated the next time M_i is executed: nt_i. This value nt_i is the simulation time value of either the next simulation step or the next simulation event to execute. As a result, we can say that when M_i is executed at time ct_i, its simulation clock is increased from ct_i to nt_i and the output data produced δ_i are valid for the simulation time interval $\Gamma_i = [ct_i, nt_i[$.

3. Synchronization occurs when M_i has to read external data $< \delta_j; \Gamma_j >$. M_i can be executed if and only if it has received valid data for its current simulation time ct_i. Data $< \delta_j; \Gamma_j >$ are said to be valid if $ct_i \in \Gamma_j$. That is, δ_j are the last produced data by M_j and M_j will produced its next data δ_j at a time $ct_j > ct_i$. This is discussed in more details in section 2.

We formalize here this coordination model and prove that coordination occurs between models and that the system is alive and deadlock free with k models ($k \in \mathbf{N}$).

2 Coordination of multiple simulators

The goal of time management in distributed simulation is to ensure that simulation events (or steps) are executed in the correct order. So the coordination model purpose is to determine when a simulation event (or step) is *safe* to process.

Definition 2.0.1 *For a model M_i, a simulation event (or step) associated with the current simulation time ct_i is said to be safe to process if all the input events received after this event execution are timestamped with a time value $> ct_i$.*

Each model M_i holds a current simulation time value ct_i. The simulator S_i knows the simulation time value for the next event to be processed nt_i. When a model M_i is executed, it produces data δ_i at time ct_i. These data δ_i will not change until the next time M_i is executed (at time nt_i). As a result, we can say that δ_i are *valid* for the simulation time interval $\Gamma_i = [ct_i, nt_i[$.

A simulator can execute a model if the simulation event to process is safe (see definition 2.0.1). Then, the issue for the simulator is to know when an event is safe. It can be solved if the simulators exchange both the data and the corresponding validity interval: $< \delta_j; \Gamma_j >$. Indeed, a simulation event is safe to process if and only if:

$$\forall j \neq i : ct_i \in \Gamma_j$$

This way, the input data $< \delta_j; \Gamma_j >$ are safe for the simulator S_i and the latter can process the simulation event at time ct_i.

3 Sketch of livingness proof

3.1 Initial hypothesis

Assume, within this coordination model, that a simulator S_i is stopped at time ct_i. That is, S_i has been processed all the simulation events with a simulation time $< ct_i$. It also means that the simulation has been running until now and that initial parameters has been set properly.

3.2 Blocking case : S_i is waiting for S_j

S_i is waiting for input data $< \delta_j, \Gamma_j >$ from another simulator S_j $(i \neq j)$. This simulator cannot send data because it is also stopped but at a time $ct_j < ct_i$. S_j also wait for input data. Two cases appear.

3.2.1 First case : S_j is waiting for S_i

Either the simulator S_j is waiting for input data from S_i. In this case it means that S_i is waiting for itself. That contravenes our initial assumption in which S_i has processed all the simulation events associated with a simulation time $< ct_i$.

3.2.2 Second case ; S_j is waiting for S_k

Or the simulator S_j is waiting for input data $< \delta_k, \Gamma_k >$ from another simulator S_k at time $ct_k < ct_j < ct_i$ with $(i \neq j \neq k)$. In this case, we come back to the very first blocking case (section 3.2). Since the number of simulators and the number of simulation events are assumed to be finite, and since $\forall i : ct_i \geq 0$, we can show by recursion that the latter case means that initial conditions are not set correctly and then that the simulation cannot happen. This contravenes our initial assumption.

4 Coordination model specification

We develop a formal specification of the coordination model mentioned before. This specification has been written in Event-B. Its purpose is to check that, implementing the coordination model, no simulator stay blocked.

4.1 Variables and constants used

Each simulator S_i hold two simulation time values: the current simulation time value ct_i and the next simulation time value nt_i which is the simulation time value associated with the next simulation event (or step). Both ct_i and $nt_i \in \mathbb{N}$. We define a constant np which is the number of interacting simulators: $i \in \mathbb{N}$ and $i \leq np$.

In order to know if a simulator is running or waiting, we define a boolean r_i for each simulator S_i which takes the value TRUE if the simulator is running and FALSE if the simulator is waiting for some input data.

CONSTANTS

`np` The number of simulator (or processes)

VARIABLES

`ct` Current Simulation Time

`nt` Next Simulation Time

`r` is a process Running?

INVARIANTS

`inv1` : $ct \in 1 \mathinner{..} np \rightarrow \mathbb{N}$

`inv2` : $nt \in 1 \mathinner{..} np \rightarrow \mathbb{N}$

`inv3` : $r \in 1 \mathinner{..} np \rightarrow BOOL$

4.2 Initialisation

The initialisation process is the following: every simulator S_i begin with a current simulation time equal to zero.

$$\forall i \in \mathbb{N}, i \leq np : ct_i = 0$$

Then, the important thing is to remember that

$$\forall i \in \mathbb{N}, i \leq np : nt_i > ct_i$$

As a consequence, we can initiate every nt_i with a strictly positive value. In our case, we choose to initiate the nt_i to one. Note that it is not an obligation and that every strictly positive value could have been chosen.

$$\forall i \in \mathbb{N}, i \leq np : nt_i = 1$$

At the beginning of the simulation, every simulator is waiting for input data.

$$\forall i \in \mathbb{N}, i \leq np : r_i = \text{FALSE}$$

EVENTS

Initialisation

begin

`act1` : $ct := \{x \mapsto 0 | x \in 1 \mathinner{..} np\}$

`act2` : $nt := \{x \mapsto 1 | x \in 1 \mathinner{..} np\}$

`act3` : $r := \{x \mapsto FALSE | x \in 1 \mathinner{..} np\}$

end

4.3 Actions

The actions describe the sequence of event for each simulator S_i. It can be broke into two distinct actions. In the first one, called *begin_p*, the simulator S_i is waiting for input events from the other simulators S_j (where $i \neq j$) which satisfy the guard condition enunciated in section 2:

$$\forall j \neq i : ct_i \in \Gamma_j \text{ (see grd3)}$$

Once all the input events have been read, the simulator can execute the model for one simulation event (or step): $r_i \leftarrow$ TRUE (act1).

EVENTS

Event *begin_p* $\widehat{=}$

any

i

where

grd1 : $i \in 1 \,..\, np$
grd2 : $r(i) = FALSE$
grd3 : $\forall j \cdot j \in 1 \,..\, np \setminus \{i\} \Rightarrow nt(j) > ct(i)$

then

act1 : $r(i) := TRUE$

end

In the second action, called *end_p*, the simulator S_i has processed its model ($r_i = TRUE$ (grd2)) and updates its simulation time values ct_i.

$$ct_i \leftarrow nt_i \text{ (act2)}$$

The simulation time for the next event nt_i is also updated. Only S_i knows it. For the specification purposes, we increase nt_i by a finite and positive value δ (grd3).

$$nt_i \leftarrow nt_i + \delta \text{ (act3)}$$

Finally, S_i send the output data to the other simulators (through the artefacts see [SCC10]) and S_i now start to wait for new input events $r_i \leftarrow$ FALSE (see act1).

EVENTS

Event *end_p* $\widehat{=}$

any

i
delta

where

grd1 : $i \in 1 \,..\, np$
grd2 : $r(i) = TRUE$
grd3 : $delta > 0$

then

act1 : $r(i) := FALSE$
act2 : $ct(i) := nt(i)$
act3 : $nt(i) := nt(i) + delta$

end

4.4 Theorem

In event-B, a theorem is a predicate which has been demonstrated from the invariants. In our case, the theorem shows that either the action *begin_p* or the action *end_p* can always start. It means that no simulator stay blocked and that they synchronized themselves.

THEOREMS

$$\texttt{thm1}: (\exists i \cdot i \in 1 \mathinner{..} np \land r(i) = FALSE \land (\forall j \cdot j \in 1 \mathinner{..} np \setminus \{i\} \Rightarrow nt(j) > ct(i))) \lor (\exists i, delta \cdot i \in 1 \mathinner{..} np \land r(i) = TRUE \land delta > 0)$$

The first part of the theorem means that it always exists a simulator S_i that can go from the event *begin_p* from the event *end_p* because all the guards in event *begin_p* are valid.

THEOREMS

$$\texttt{thm1 (left part)}: \exists i \cdot i \in 1 \mathinner{..} np \land r(i) = FALSE \land (\forall j \cdot j \in 1 \mathinner{..} np \setminus \{i\} \Rightarrow nt(j) > ct(i))$$

The second part of the theorem (after the Or statement: $\lor$) means that there always exists a simulator S_i that can go from the event *end_p* to the event *begin_p* because all the in event *end_p* guards are valid.

5 Specification

An Event-B Specification of m0
Generated Date: 24 Mar 2009 @ 03:03:10 PM

MACHINE m0

SEES c0

CONSTANTS

`np` The number of simulator (or processes)

VARIABLES

`ct` Current Simulation Time

`nt` Next Simulation Time

`r` is a process Running?

INVARIANTS

`inv1` : $ct \in 1 \mathinner{..} np \rightarrow \mathbb{N}$

`inv2` : $nt \in 1 \mathinner{..} np \rightarrow \mathbb{N}$

`inv3` : $r \in 1 \mathinner{..} np \rightarrow BOOL$

THEOREMS

`thm1` : $(\exists i \cdot i \in 1 \mathinner{..} np \land r(i) = FALSE \land (\forall j \cdot j \in 1 \mathinner{..} np \setminus \{i\} \Rightarrow nt(j) > ct(i))) \lor$
$(\exists i, delta \cdot i \in 1 \mathinner{..} np \land r(i) = TRUE \land delta > 0)$

EVENTS

Initialisation

begin

`act1` : $ct := \{x \mapsto 0 | x \in 1 \mathinner{..} np\}$

`act2` : $nt := \{x \mapsto 1 | x \in 1 \mathinner{..} np\}$

`act3` : $r := \{x \mapsto FALSE | x \in 1 \mathinner{..} np\}$

end

Event $begin\_p \mathrel{\widehat{=}}$

any

i

where

`grd1` : $i \in 1 \mathinner{..} np$

`grd2` : $r(i) = FALSE$

`grd3` : $\forall j \cdot j \in 1 \mathinner{..} np \setminus \{i\} \Rightarrow nt(j) > ct(i)$

then

`act1` : $r(i) := TRUE$

end

Event $end\_p \mathrel{\widehat{=}}$

any

i

$delta$

where

grd1 : $i \in 1 \mathinner{..} np$

grd2 : $r(i) = TRUE$

grd3 : $delta > 0$

then

act1 : $r(i) := FALSE$

act2 : $ct(i) := nt(i)$

act3 : $nt(i) := nt(i) + delta$

end

END

References

[BRC07] Stphane Bonneaud, Pascal Redou, and Pierre Chevaillier. Pattern oriented agent-based multi-modeling of exploited ecosystems. In *6th EUROSIM congress on modelling and simulation*, september 9-13 2007.

[ORV08] Andrea Omicini, Alessandro Ricci, and Mirko Viroli. Artifacts in the a&a meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 17(3):432–456, 2008.

[RAF+04] George F. Riley, Mostafa H. Ammar, Richard M. Fujimoto, Alfred Park, Kalyan Perumalla, and Donghua Xu. A federated approach to distributed network simulation. *ACM Trans. Model. Comput. Simul.*, 14(2):116–148, 2004.

[SCC10] Julien Siebert, Laurent CiIarletta, and Vincent Chevrier. Agents & artefacts for multiple models coordination. In *25th Symposium On Applied Computing*, 2010.