AA4MM coordination model and event-B specification

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

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Modèle de synchronization de l'approche AA4MM et spécification event-B

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

N° 7081

Mars 2010

Knowledge and Data Representation and Management

ISSN 0249-6399 ISRN INRIA/RR--7081--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
NANCY - GRAND EST

Modèle de synchronization de l'approche AA4MM et spécification event-B

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

Theme : Knowledge and Data Representation and Management
Networks, Systems and Services, Distributed Computing
Algorithmics, Programming, Software and Architecture
Équipes-Projets Maia et Madynes et Mosel

Rapport de recherche n° 7081 — Mars 2010 — 10 pages

Résumé : Agent and Artefact for Multiple Models coordination (AA4MM)* est un logiciel développé par Julien Siebert durant sa thèse. Le but de cet outil est de rendre la conception et l'implémentation de simulations de systèmes complexes modulaires et décentralisées. Le but principla est de pouvoir réutiliser des outils (modèles et simulateurs) existants et de les faire interagir afin de simuler des phénomènes à des niveaux d'abstraction différents. La contrainte principale est que les personnes impliquées dans la conception et l'implémentation de telles simulations ne sont pas obligatoirement des experts en théorie de la modélisation et de la simulation. En conséquance, les défis liés à la coordination (synchronisation, respect des contraintes de causalité) sont du ressort du logiciel AA4MM. Ce rapport présente la spécification du modèle de synchronisation. Celle ci est faite en Event-B et son but est de vérifier que le modèle de synchronisation ne conduit pas à des deadlocks (cas dans lesquels les simulateurs en interaction s'attendent les uns les autres indéfiniment).

Mots-clés : Interactions de modèles, co-evolution, spécification formelle, event-b, synchronisation

Ce travail a donné lieu à une collaboration entre Julien Siebert et Joris Rehm

* http://www.loria.fr/∼siebertj/aa4mm/aa4mm.html

Centre de recherche INRIA Nancy – Grand Est
LORIA, Technopôle de Nancy-Brabois, Campus scientifique,
615, rue du Jardin Botanique, BP 101, 54602 Villers-Lès-Nancy
Téléphone : +33 3 83 59 30 00 — Télécopie : +33 3 83 27 83 19

AA4MM synchronization model and event-B specification

Abstract: Agent and Artefact for Multiple Models coordination (AA4MM)[†] is a framework developed by Julien Siebert during its PhD. It is is intended to make the design and the implementation of complex systems simulation modular and decentralized. The 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 synchronization model proposed in [SCC10]. Its goal is to check that the system is never blocked (no deadlock): no simulator is waiting for another indefinitely.

Key-words: Models interactions, coevolution, formal specifiaction, event-b, synchronization

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

1 Introduction

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

1.1 Brief description of AA4MM

The Agent and Artefact for Multiple Models and simulators co-evolution (AA4MM) framework is a meta-model based upon the Agent and Artefact paradigm [RVO07]. It has been developed in order to couple heterogeneous models and simulators. Its implementation purpose is to build a society of interacting and co-evolving (existing) models and simulators. Since this society is composed by already existing modeling and simulation tools, some constraints and issues appear [SCC10]. The purpose of this report is not to present all of them. However, for the sake of understanding, some of the issues are presented hereafter.

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 (as in [BRC07]). The coordination process can either be done by a central scheduler or by a decentralized protocol. The originality of our model is its decentralized coordination. This raises the question to ensure the synchronization and to respect the causality constraint.

2 Coordination of multiple simulators

The goal of time management in distributed simulation is to ensure that local causality constraint is respected [Fuj01]. The next sections present the causality issues and the solution adopted by AA4MM approach.

2.1 Introduction

Let $S \subseteq \mathbb{N}$ be the finite set representing the indices of simulators. Let s_i ($i \in S$) be the simulator in charge of the model m_i (we assume that one simulator s_i is in charge of only one model m_i). Let m_i have X_i input ports and Y_i output ports.

2.2 Model execution process

Let d_i be the data produced by an execution of m_i. When a model m_i is executed, the simulator s_i reads input data d_j form other simulators ($j \in S$) and sets them into the input ports X_i. Then

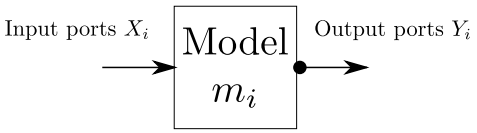


FIG. 1: Model representation, as defined in [ZPK00]

s_i executes the model m_i, increases its local simulation time from (current) t_i to (next) $t_i + \Delta t_i$ and finally proposes output data d_i to other simulators (d_i come from the output ports Y_i).

Remark 2.2.1 *Note that Δt_i is not fixed and depends on the execution policy (discrete events or step-by-step).*

The data d_i produced by the execution of m_i at simulation time t_i are timestamped with a time value t_i. A timestamped data is denoted $d_i^{t_i}$. However, since this notation is quite heavy, in this document, it is sometimes simplified in d_i for the sake of clarity.

2.3 Coordination issues

The goal of time management in distributed simulation is to ensure that all the models are executed in the correct order and that the causality constraint is respected [Fuj01]. It means that, if a simulator s_i depends on output data d_j produced by s_j, s_i cannot execute its model m_i until s_j has produced *valid* data (see definition 2.3.1).

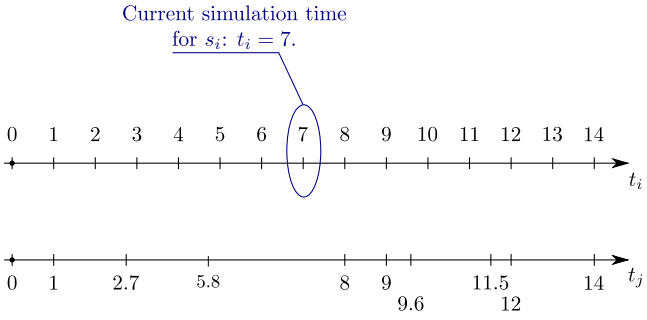
Definition 2.3.1 *Data are said to be valid (in the point of view of s_i at simulation time t_i) on two conditions. On the one hand, $d_j^{t_j}$ are produced by s_j at a simulation time $t_j < t_i$. On the other hand, the next data $d_j^{t_j^{next}}$ produced by s_j are produced at a simulation time value $t_j^{next} = (t_j + \Delta t_j) \geq t_i$.*

When all the input data for the model m_i are valid in the sense of s_i, the model execution is said to be *safe* and its execution will not provoke causality constraint violation (see definition 2.3.2, citation 2.3.1 and figure 2).

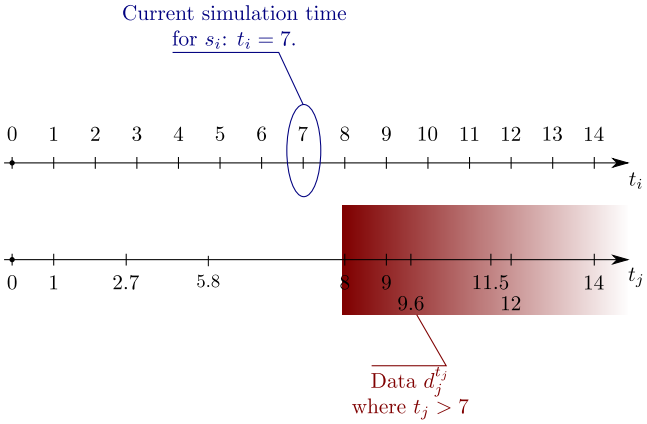
Definition 2.3.2 *For a simulator s_i ($i \in S$), a model execution associated with the current simulation time t_i is said to be safe to process if all the input data d_j ($j \in S$) received after this event execution are timestamped with a time value $\geq t_i$.*

Citation 2.3.1 *[...] if a process contains an unprocessed event E_{10} with a time stamp T_{10} (and no other with smaller time stamp)*[1]*, and that process can determine that it is impossible for it to later receive another event with time stamp smaller than T_{10}, then E_{10} is said to be* safe *because one can guarantee that processing the event will not later result in violation of local causality constraint.* (Section 3.1 in [Fuj01])

[1] It means s_i is going to executes m_i at simulation time $t_i = 10$

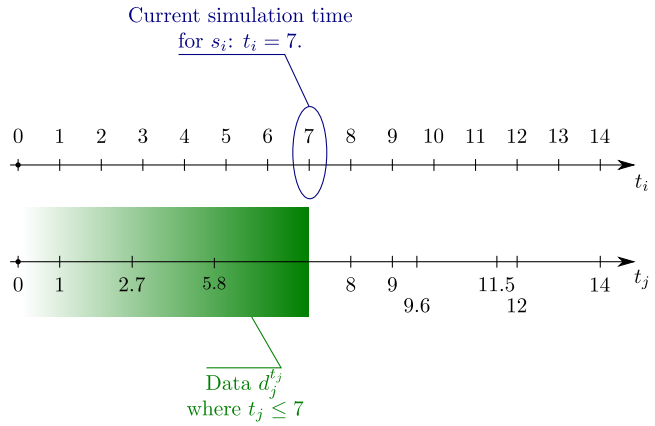


(a) At start : s_i at simulation time $t_i = 7$.

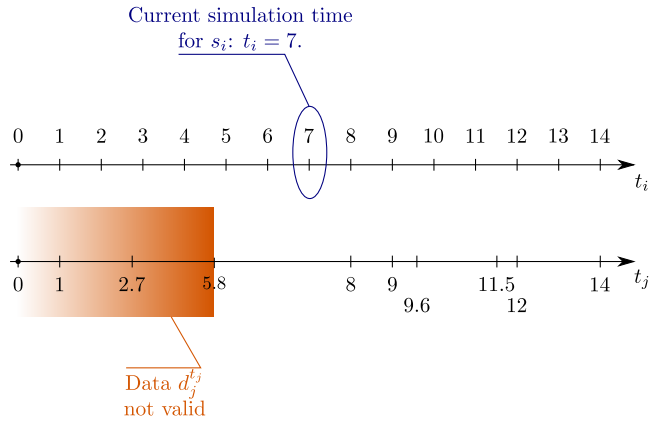


(b) Data that do not fulfil the first condition to be valid $t_i < t_j$ (see definition 2.3.1).

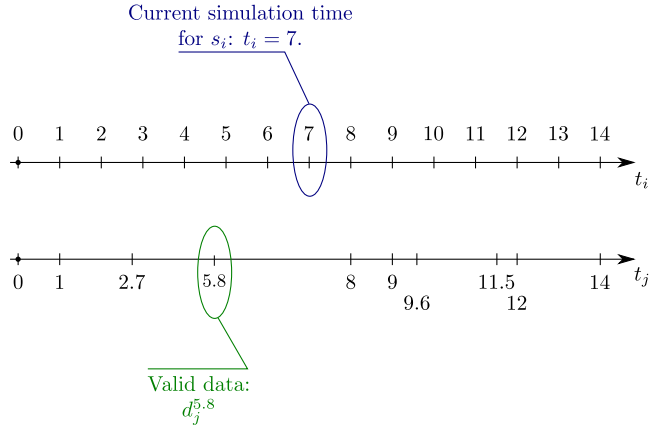
FIG. 2: Valid data example. Simulator s_i (upper time line) is waiting for *valid* data $d_j^{t_j}$ produced by the simulator s_j (lower time line)



(c) Data that fulfil the first condition to be valid $t_i < t_j$ (see definition 2.3.1).



(d) Data that fulfil the first condition to be valid $(t_i < t_j)$ but not the second $t_j^{next} = (t_j + \Delta t_j) \geq t_i$ (see defin



(e) Data $d_j^{5.8}$ is valid in the sense of s_i for the simulation time $t_i = 7$.

FIG. 2: Valid data example. Simulator s_i (upper time line) is waiting for *valid* data $d_j^{t_j}$ produced by the simulator s_j (lower time line)

2.4 Validity time interval

In the previous example (figure 2), the problem for the simulator s_i at simulation time $t_i = 7$, is to know that data $d_j^{5.8}$ is valid. In other sense, that no data are going to be produced between local simulation time values $t_j = 5.8$ and $t_j = 7$. This useful information is only known by the simulator s_j as it executes its model m_j.

The principle of our coordination model is to assign a validity interval (denoted by Γ_j, $\forall j \in S$) to the exchanged data d_j. This interval is proposed by the simulator s_j that produces the data d_j to be exchanged. Consequently, when a simulator s_i needs to read input data d_j ($j \in S$), it just has to check Γ_j to know if d_j is valid. Since s_i knows that whether the input data it needs are valid or not, s_i can wait for valid data or process the model execution (if all input data d_j are valid). A direct consequence is that the simulators stay synchronized (see section 2.7) and respect causality constraint. The coordination model is said to be correct (see citation 2.4.1).

Citation 2.4.1 *From the standpoint of synchronization, "correctness" only goes so far as to say that the parallel execution will produce identical results as a sequential execution of the same program. Also it should be pointed out that the adherence of this constraint is sufficient, but not always necessary, to guarranty that no causality errors occurs.* (Section 3.1 in [Fuj01])

2.4.1 Definition

The validity interval comes from the valid data definition (definition 2.3.1). It is defined, for every simulator s_i that executes a model m_i and increments its local simulation time t_i by a finite quantity of simulation time $\Delta_i \geq 0$, by the equation 1 :

$$\forall i \in S, \Gamma_i =]t_i; t_i + \Delta_i] \tag{1}$$

The validity interval Γ_i means that, when a piece of data is produced at simulation time t_i, then this piece of data is valid until the next time the model will be executed (i.e. at simulation time $t_i + \Delta_i$). In other words, the data $d_i^{t_i}$ is not modified for the simulation time interval $\Gamma_i =]t_i; t_i + \Delta_i]$.

2.5 Simulation algorithm

The following algorithm (see algorithm 1) represents one model execution *step* for s_i. One model execution *step* depends on the execution policy implemented by the simulator. That is, s_i executes m_i for one simulation step (if step-by-step simulation) or one simulation event (if discrete event simulation). The whole simulation process for s_i is a loop on this algorithm until the simulation ends.

Remark 2.5.1 *The process of waiting for valid input data (lines 2 to 8 in algorithm 1) can either be implemented in a sequential or a concurrent manner.*

Algorithm 1: One model execution step

MODEL EXECUTION STEP(I, s_i)

Description: s_i executes its own model m_i for one simulation step (if step-by-step simulation) or one simulation event (if discrete event simulation). Let $j \in S$ represent the indices of external simulators s_j interacting with s_i.

Input: A finite set of input data $d_j^{t_j}$ (associated with their validity interval Γ_j) called I

Output: A finite set of output data O

(1) $ct_i \leftarrow$ CurrentSimulationTime(s_i)
(2) **foreach** input data (d_j, Γ_j) in I
(3) **if** $ct_i \in \Gamma_j$
(4) set d_j to the right input port $x_i \in X_i$ of m_i
(5) **else**
(6) wait for a new input data (d_j, Γ_j)
(7) **end if**
(8) **end for**
(9) s_i executes m_i
(10) s_i increases its local simulation time
(11) $nt_i \leftarrow$ CurrentSimulationTime(s_i)
(12) $\Gamma_i \leftarrow]ct_i, nt_i]$
(13) **foreach** output data d_i in Y_i
(14) $O \leftarrow$ add$(O, (d_i, \Gamma_i))$
(15) **end for**
(16) **return** O

2.5.1 Remark on simultaneous simulation events

In discrete event simulation, in the case of simultaneous events, the simulation algorithm 1 can be either applied as it is (the simulator s_i executes events in a sequential manner) or the simulator s_i can execute all the simultaneous events during one *step*.

Indeed, it is worth noting that the lower boundary ct_i in Γ_i is excluded. It means that at current simulation time t_i, the simulator s_i can only read input data produced before that simulation time. It is useful to avoid incoherence problems with simultaneous simulation events. For example, consider one standalone simulator s_i. Its input data d_i correspond to its output data. Consider that it produces a set of output data made of three different pieces of data : $d_i^{t_i} = (\alpha, \beta, \gamma)$ (as described in figure 3).

1. s_i reads input data $d_i^{t_i} = (\alpha, \beta, \gamma)$
2. s_i executes m_i
3. s_i produces output data $d_i^{t_i+\Delta_i} = (\alpha_{new}, \beta_{new}, \gamma_{new})$ associated with $\Gamma_i =]ct_i, nt_i] =]t_i, t_i + \Delta_i]$

FIG. 3: Execution process

Assume that, each piece of data α, β, γ is produced by one simultaneous simulation event e_α, e_β and e_γ. Furthermore, each simulation event e_α, e_β and e_γ needs the three pieces of data values α, β, γ to be executed. In this case, the expected execution process is the following (see figure 4) :

1. at t_i, s_i reads $d_i^{t_i} = (\alpha, \beta, \gamma)$,
2. s_i executes e_α, e_β and e_γ,
3. s_i increases t_i by Δ_i,
4. s_i produces output data $d_i^{t_i+\Delta_i} = (\alpha_{new}, \beta_{new}, \gamma_{new})$

$$\left\{ \begin{array}{lcl} e_\alpha(\alpha,\beta,\gamma) & \rightarrow & \alpha_{new} \\ e_\beta(\alpha,\beta,\gamma) & \rightarrow & \beta_{new} \\ e_\gamma(\alpha,\beta,\gamma) & \rightarrow & \gamma_{new} \end{array} \right\}$$

FIG. 4: Expected execution process (concurrent) : each event depends on the original value α, β, γ

However, a problem can happen if, instead of executing all the simultaneous simulation events together in one simulation *step*, they are executed sequentially. That is (see figure 5) :

1. at t_i, s_i reads $d_i^{t_i} = (\alpha, \beta, \gamma)$,
   (a) s_i executes e_α,
   (b) s_i produces output data $d_i^{t_i} = (\alpha_{new}, \beta, \gamma)$
2. Then, s_i reads input data $d_i^{t_i} = (\alpha_{new}, \beta, \gamma)$,
   (a) s_i executes e_β,
   (b) s_i produces output data $d_i^{t_i} = (\alpha_{new}, \beta^*_{new}, \gamma)$
3. Finally, s_i reads input data $d_i^{t_i} = (\alpha_{new}, \beta^*_{new}, \gamma)$,
   (a) s_i executes e_γ,
   (b) s_i increases t_i by Δ_i
   (c) s_i produces output data $d_i^{t_i+\Delta_i} = (\alpha_{new}, \beta^*_{new}, \gamma^*_{new})$.

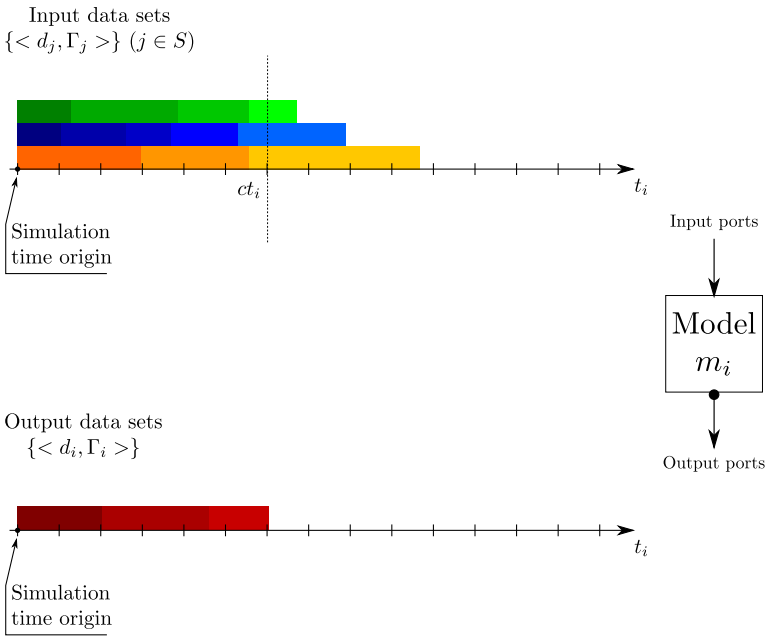
$$\left\{ \begin{array}{lcl} e_\alpha(\alpha,\beta,\gamma) & \rightarrow & \alpha_{new} \\ & & \\ & & \end{array} \right. \quad \left. \begin{array}{lcl} & & \\ e_\beta(\alpha_{new},\beta,\gamma) & \rightarrow & \beta^*_{new} \\ & & \end{array} \right. \quad \left. \begin{array}{lcl} & & \\ & & \\ e_\gamma(\alpha_{new},\beta^*_{new},\gamma) & \rightarrow & \gamma^*_{new} \end{array} \right\}$$

FIG. 5: Execution process to avoid (sequential) : each event does not depend on the original value α, β, γ but on the intermediate values

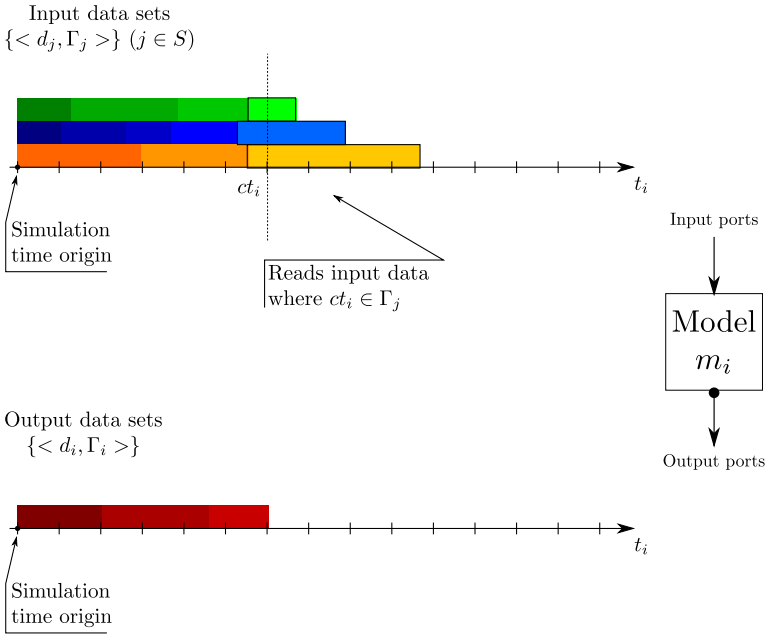
Whatever the execution policy chosen, the previously defined validity interval Γ_i avoids this kind of problem. Indeed, in the second case example, the data $d_i = (\alpha_{new}, \beta, \gamma)$ cannot be read by s_i because it is associated with a validity interval $\Gamma_i =]t_i, t_i] = \varnothing$. As a consequence, s_i always reads input data $d_i^{t_i} = (\alpha, \beta, \gamma)$ before executing the simultaneous events e_α, e_β and e_γ.

2.6 Example

The figure 6 describes the execution of the *model execution step* algorithm 1 for one model m_i. In this example, the exchanged data (d_i, Γ_i) are symbolized by colored rectangles (the green, blue, orange and red ones). The horizontal axis (denoted t_i) represents the simulation time axis of simulator s_i (graduations are only present for the sake of the visibility).

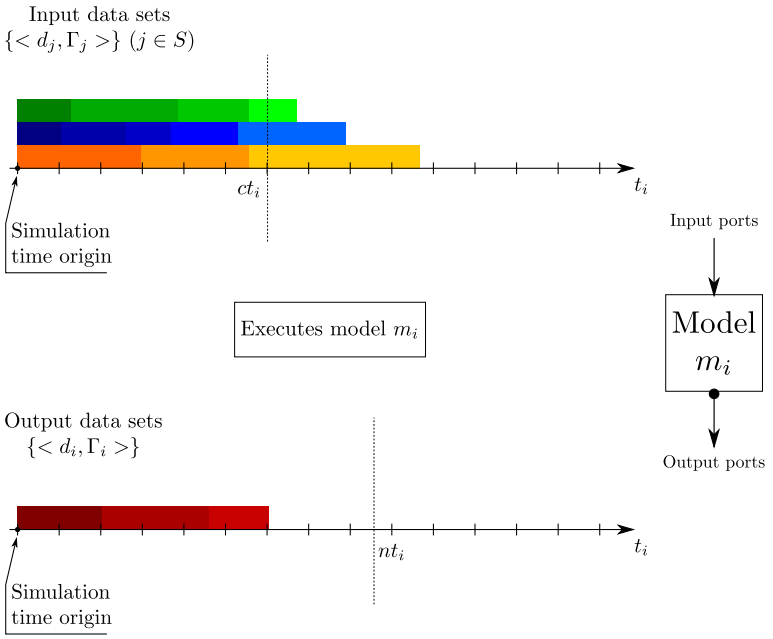


(a) At start.

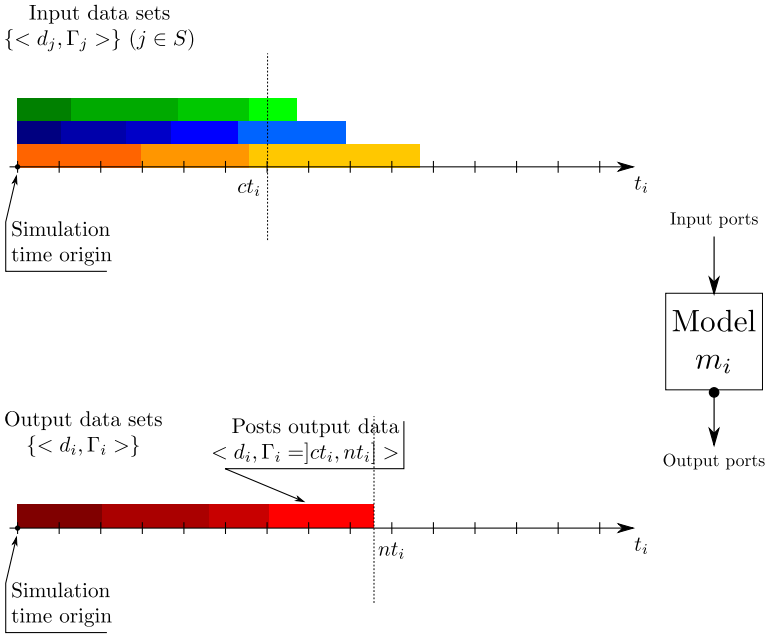


(b) Read input data.

FIG. 6: Algorithm 1 by the example : from the starting state (6a), input data are read (6b) (algorithm 1 lines 2 to 8). The model is then executed (6c) (algorithm 1 lines 9 to 11). Finally, output data are posted (6d) (algorithm 1 lines 13 to 15).



(c) s_i executes the model m_i.



(d) Post output data $O = \{(d_i, \Gamma_i)\}$.

FIG. 6: Algorithm 1 by the example : from the starting state 6a, input data are read 6b (algorithm 1 lines 2 to 8). Then, the model is executed 6c (algorithm 1 lines 9 to 11). Finally, output data are posted 6d (algorithm 1 lines 13 to 15).

2.7 Formalisation : no deadlock

We have developed a formal specification in event-B of this coordination model (see section 3). The goal of this formalisation is to ensure that synchronization always occurs between simulators. That is, the system is alive and deadlock free with n simulators ($n \in \mathbf{N}$). A sketch of the proof, using *reductio ad absurdum* is presented in the next section.

2.7.1 Notations

Let $S \subseteq \mathbb{N}$ be the finite set representing the indices of the simulators. Let s_i ($i \in S$) be the simulator in charge of the model m_i. Let $B \subseteq S$ be the finite set representing the indices of the blocked simulators. That is, simulators that cannot execute their own models because some input data are missing.

2.7.2 Hypothesis

Consider that the set of blocked simulators $B \neq \varnothing$. I.e. there exists at least one simulator that is blocked : simulator s_i ($i \in B$) is stopped at time $t_i = t_i^{stop}$ (see figure 7a). Our initial hypothesis is that s_i has executed m_i and produced output data (d_i, Γ_i) (the red rectangles in figure 7a) until it is stopped ($t_i = t_i^{stop}$).

Considering our coordination model (see algorithm 1 and figure 6), there exists a partial order relation between the time values of every blocked simulators (see an example in section 2.8.1). As a consequence, we propose the following lemma (used to derive the examples in sections 2.8.2 and 2.8.3) :

Lemma 2.7.1 *A simulator s_i is said to be blocked ($i \in B$) if and only if :*

1. *it depends of data d_j provided by a finite subset of simulators $j \in S^* \subseteq S$,*
2. *there exists at least one simulator s_k blocked : $\exists k \in (S^* \cap B) \neq \varnothing$,*
3. *$t_k^{stop} < t_i^{stop}$.*

2.7.3 No simulator can be blocked

In the finite set of blocked simulators $s_i (i \in B)$, the partial order relation between the time values t_i^{stop} implies that there exists one blocked simulator s_j with a minimum time value :

$$\forall i \in B, \exists j \in B \text{ such as } t_j^{stop} = \min_i(t_i^{stop})$$

The contradiction appears with the blocked simulator s_j described previously. Indeed, considering the lemma 2.7.1 if s_j is blocked, then there must exist a simulator s_k that is blocked and that possesses a time value $t_k^{stop} < t_j^{stop}$, which is not possible (t_j^{stop} being the minimum value). It means that, if s_j is blocked, then s_j is stop at time $t_j^{stop} = 0$. Either s_j is not running at all (maybe, it has not been initiated properly), or s_j is currently proccessing its first execution step and as a consequence, is not blocked.

We can reconsider this reasonning for all the blocked simulators and come to the same conclusion that in this coordination model, simulators cannot be blocked by waiting for each other (there is no deadlock). Next section give, as an example, a sketch of this proof.

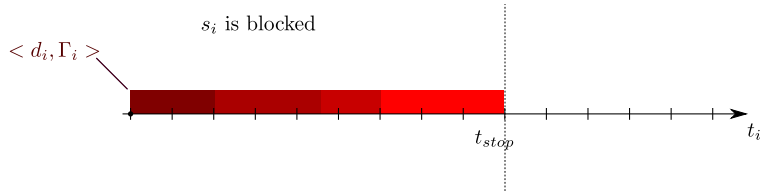
2.8 Sketch of proof

2.8.1 Simulator s_i is blocked

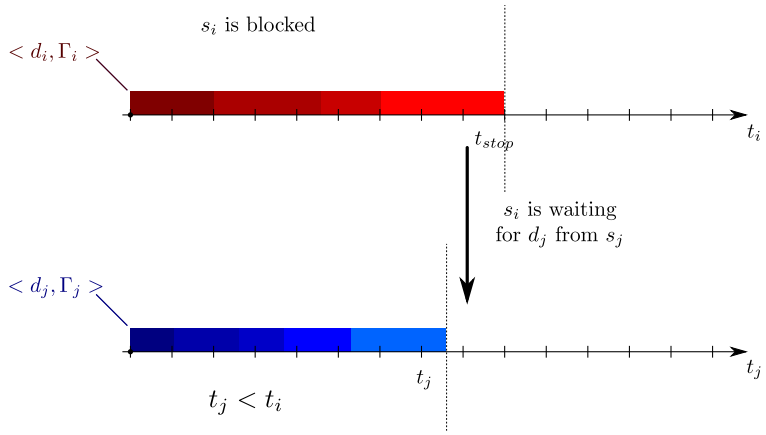
It is waiting for input data (d_j, Γ_j) from another simulator s_j (where $i \neq j$ and $i, j \in S$). This simulator cannot send data because it is also stopped but at a time $t_j < t_i$ (see figure 7b).

2.8.2 First case : simulator s_j is blocked but is waiting for s_i

s_j also waits for input data. Two cases appear. Either the simulator s_j is waiting for input data from s_i. This is impossible since our first assumption was that s_i is blocked at simulation time $t_i = t_i^{stop}$ and has produced all output data d_i until then. As a consequence, applying our coordination model means that s_j must be able to process the execution of m_j until $t_j = t_i^{stop}$ (see figure 7c).



(a) Starting assumption : s_i is blocked at simulation time $t_i = t_i^{stop}$.



(b) s_i waiting for intput data d_j from another simulator s_j.

FIG. 7: Sketch of proof

2.8.3 Second case : simulator s_j is blocked and is waiting for s_k

In the second case, the simulator s_j is waiting for input data (d_k, Γ_k) from another simulator s_k at time $t_k < t_j < t_i$ (where $i \neq j \neq k$ and $i, j, k \in S$. see figure 7d). It is worth noticing that this case is similar to the very state where s_i is blocked. So we can have the same reasoning to know why s_k is blocked, and by iteration on every simulator.

2.8.4 Conclusion

Since the number of simulator n is finite and since the number of exchanged data $(d_i, \Gamma_i)(\forall i \in S)$ (i.e. the number of blue, red, green and yellow rectangles in figure 7) is also finite, we can conclude that if s_i is blocked, it implies two distinct cases. On the one hand, one simulator is waiting for data d_i that has already been produced (like in our first case in section 2.8.2 and figure 7c). This leads to a contradiction and is not possible. On the other hand, every simulator is blocked. And as a consequence, the simulation process has never begun which is also a contradiction.

We can say that if the simulation process is properly instanciated (see definition 2.9.1), no deadlock could happen because of the coordination model[2].

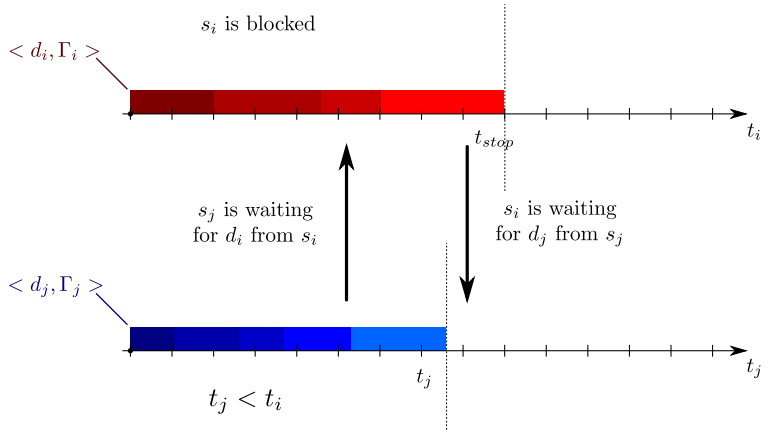
2.9 Intialisation process

In the previous section, it has been demonstrated that simulators cannot wait for each other indefinitively unless the initialization has not been set properly.

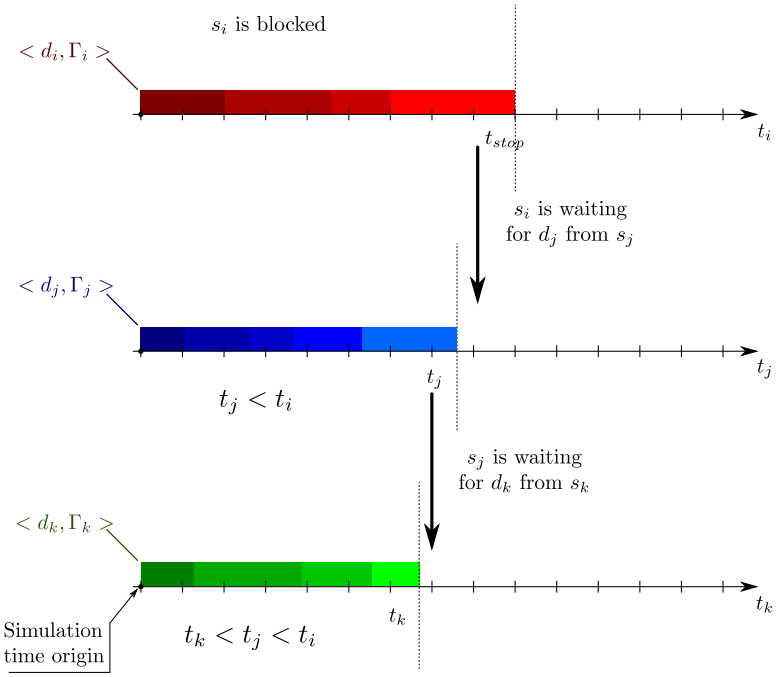
Definition 2.9.1 *$\forall i \in S$, at the origin of simulation time, every simulator s_i has been parametrized and possessed all the input data in order to execute its own model m_i once.*

The initialization process depends mainly on the models interactions network. For example, if all simulators begin at simulation time $t_i = 0$ $(\forall i \in S)$ one way to initialize the whole simulation process is that every simulator s_i has to post its initial data d_i^0 associated with the validity time interval

[2] Of course it can exist deadlocks caused by a concurent implementation but this is not the scope of this part.



(c) Simulator s_j cannot be blocked at simulation time $t_j < t_i$.



(d) Simulator s_j is waiting for another simulator s_k where $t_k < t_j^{stop}$.

FIG. 7: Sketch of proof

$\Gamma_i =]x, 0]$ (with $x < 0$). This condition allows every simulator to read its input data set and then, to process the simulation.

3 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.

3.1 Variables and constants used

Each simulator s_i holds a simulation time value : the current simulation time value $ct_i \in \mathbb{N}$. We define a constant np which is the number of interacting simulators : $np = \text{card}(S)$.

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

`t` Current Simulation Time

`r` is a process Running ?

INVARIANTS

`inv1` : $t \in 1 \mathinner{..} np \rightarrow \mathbb{N}$
Current simulation time

`inv2` : $r \in 1 \mathinner{..} np \rightarrow BOOL$

3.2 Initialisation

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

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

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` : $t := 1 \mathinner{..} np \times \{0\}$

`act2` : $r := 1 \mathinner{..} np \times \{FALSE\}$

end

3.3 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 which satisfy the guard condition enunciated in section 2 :

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

Since we only focus on deadlock, we only cares about the current simulation time values. 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 : $i \in 1 \mathinner{..} np$

grd2 : $r(i) = FALSE$

grd3 : $\forall j \cdot j \in 1 \mathinner{..} np \setminus \{i\} \Rightarrow t(i) \leq t(j)$

A process i can be proceed if every other processes have a current time value t(j) greater or equal than its own current time value t(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 t_i.

$$t_i \leftarrow t_i + \delta \text{ (act2)}$$

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

grd2 : $delta > 0$

grd3 : $r(i) = TRUE$

then

act1 : $r(i) := FALSE$

act2 : $t(i) := t(i) + delta$

end

3.4 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

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

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.

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

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.

thm1 (right part) : $\exists k, delta \cdot k \in 1 \mathinner{..} np \land r(k) = TRUE \land delta > 0$

4 Specification

An Event-B Specification of machine0
Generated Date: 23 Mar 2010 @ 03 :00 :05 PM

MACHINE machine0

SEES context0

VARIABLES

`t`

`r`

INVARIANTS

`inv1` : $t \in 1 \mathinner{..} np \rightarrow \mathbb{N}$
Current simulation time

`inv2` : $r \in 1 \mathinner{..} np \rightarrow BOOL$

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

EVENTS

Initialisation

begin

`act1` : $t := 1 \mathinner{..} np \times \{0\}$

`act3` : $r := 1 \mathinner{..} np \times \{FALSE\}$

end

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

any

i

where

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

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

`grd3` : $\forall j \cdot j \in 1 \mathinner{..} np \setminus \{i\} \Rightarrow t(i) \leq t(j)$
A process i can be proceed if every other processes have a current time value t(j) greater or equal than its own current time value t(i)

then

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

end

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

any

i

$delta$

where

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

`grd2` : $delta > 0$

`grd3` : $r(i) = TRUE$

then

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

`act2` : $t(i) := t(i) + delta$

end

END

Table des matières

Références

[BRC07] Stéphane 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.

[Fuj01] Richard M. Fujimoto. Parallel simulation : parallel and distributed simulation systems. In *WSC '01 : Proceedings of the 33nd conference on Winter simulation*, pages 147–157, Washington, DC, USA, 2001. IEEE Computer Society.

[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.

[RVO07] Alessandro Ricci, Mirko Viroli, and Andrea Omicini. Give agents their artifacts : the a&a approach for engineering working environments in mas. In *AAMAS '07 : Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, pages 1–3, New York, NY, USA, 2007. ACM.

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

[ZPK00] Bernard P. Zeigler, Herbert Praehofer, and Tag Gon Kim. *Theory of Modeling and Simulation*. Academic Press, January 2000.

Centre de recherche INRIA Nancy – Grand Est
LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399