Modeling of Immediate vs. Delayed Data Communications: from AADL to UML MARTE

Charles André, Frédéric Mallet, Robert De Simone

Modeling of immediate vs. delayed data communications: from AADL to UML MARTE

Charles André, Frédéric Mallet, Robert de Simone
I3S, Université de Nice-Sophia Antipolis, CNRS, F-06903 Sophia Antipolis
Aoste Project, I3S/INRIA

E-mail: {candre,fmallet,rs}@sophia.inria.fr

Abstract

The forthcoming OMG UML Profile for Modeling and Analysis of Real-Time Embedded systems (MARTE) *aims, amongst other things, at providing a referential Time Model subprofile where semantic issues can be explicitly and formally described. As a full-size exercise we deal here with the modeling of* immediate *and* delayed *data communications in AADL. It actually reflects an important issue in RT/E model semantics: a propagation of immediate communications may result in a combinatorial loop, with ill-defined behavior; introduction of delays may introduce races, which have to be controlled. We describe here the abilities of MARTE in this respect.*

I. Introduction

The modeling phase in Real-Time Embedded design is increasingly required to allow various types of timing analysis prior to final code production and testing. AADL [1] and MARTE [2] are two such modeling formalisms, in part similar in their objectives. They both allow independent descriptions of the functional applications and the execution platforms, and the possible allocation of the former onto the latter. They also allow the description of both the structural organization of systems, and to some extent of their dynamic behaviors.

Our belief here is that AADL relies on a number of assumptions that make the definition of dynamic behaviors visibly simple, but largely implicit and informal (with the risk of ambiguity or misdesign, which various analysis tools then try to spot and identify). Conversely, MARTE explicit Time model with powerful *logical time* constraints allows to specify precisely and thoroughly the scheduling aspects of application elements (which induces a relevant definition apparatus). Multiform logical time supported by MARTE, is inspired from the theory of tag systems [3].

AADL applications comprise threads, often of periodic nature (with distinct periods), connected through event or data parts. Data communications can be *immediate* or *delayed*. As can be seen here, the same model provides structural information (the thread connections) together with a crude abstraction of behaviors usually needed for schedulability analysis (the relative speeds of threads). Delayed communications are needed in particular to break down cycle propagation of data. They implicitly impose a partial order on how various threads (and their containing processes) can be executed/simulated in a simultaneous step. The issues of priority inversion involved here are dealt with in [4]. AADL thread modeling thus requires the conjunct of two MARTE models (one behavioral and one structural), with the relevant logical clocks defining the relative ordering of dispatch events for the threads according to the desired semantics. The operational semantics is now explicit, and the various protocols (immediate/delayed) can be constructed in a formal way. This is the topic of the current paper. The hope is that such construction can then allow by analytic techniques to prevent non-determinism and pathological priority inversions to occur, in a way that is predicted and garanteed rather than monitored by non-exhaustive model simulations.

II. Background

A. Time in MARTE

The metamodel for time and time-related concepts is described in the "Time modeling" chapter of the UML profile for MARTE, soon available at the OMG site. The time chapter is briefly described in another paper [5].

In MARTE, Time can be *physical*, and considered as *continuous* or *discretized*, but it can also be *logical*, and related to user-defined clocks. Time may even be *multiform*, allowing different times to progress in a non-uniform fashion, and possibly independently to any (direct) reference to physical time. The *time structure* is defined by a set of *clocks* and *relations* on these clocks. Here clock is not a device used to measure the progress of physical time. It is a mathematical object lending itself to formal processing instead. A clock that refers

to physical time is called a *chronometric* clock. A distinguished chronometric clock called idealClk is provided in the MARTE time library. This clock represents the "ideal" physical time, used, for instance, in physical and mechanics laws. At the design level most of the clocks are *logical* ones.

The mathematical model for a clock is a 5-tuple $(\mathcal{I}, \preccurlyeq, \mathcal{D}, \lambda, u)$ where $\mathcal{I}$ is a set of instants, $\preccurlyeq$ is an order relation on $\mathcal{I}$, $\mathcal{D}$ is a set of labels, $\lambda : \mathcal{I} \rightarrow \mathcal{D}$ is a labeling function, u is a symbol, standing for a *unit*. For a chronometric clock, the unit can be the SI time unit s (second) or one of its derived units (ms, us...). The usual unit for logical clocks is tick, but clockCycle, executionStep ...may be chosen as well. Since instants of a clock are fully ordered, $(\mathcal{I}, \prec)$ is an ordered set.

Clock are *a priori* independent. They become dependent when their instants are linked by *instant relations* imposing either *coincidence* between instants (coincidence relation $\equiv$) or *precedence* (precedence relation $\preccurlyeq$). *Clock relations* are a convenient way to impose many—often infinitely many—instant relations. Examples of clock relations are given in Section III-B.

A *Time Structure* is a 4-tuple $(\mathcal{C}, \mathcal{R}, \mathcal{D}, \lambda)$ where $\mathcal{C}$ is a set of clocks, $\mathcal{R}$ is a relation on $\bigcup_{a,b \in \mathcal{C}, a \neq b} (\mathcal{I}_a \times \mathcal{I}_b)$, $\mathcal{D}$ is a set of labels, $\lambda : \mathcal{I}_\mathcal{C} \rightarrow \mathcal{D}$ is a labeling function. $\mathcal{I}_\mathcal{C}$ is the set of the instants of a time structure. $\mathcal{I}_\mathcal{C}$ is not simply the union of the sets of instants of all the clocks; it is the quotient of this set by the coincidence relation induced by the time structure relations represented by $\mathcal{R}$. A time structure specifies a poset $(\mathcal{I}_\mathcal{C}, \preccurlyeq_\mathcal{C})$.

During a design we introduce several (logical) clocks that are progressively constrained. This causes strengthenings of the ordering relation of the application time structure.

B. AADL inter-thread communications

As a demonstration of the expressiveness of MARTE, we take as an example the inter-thread data communication semantics of AADL.

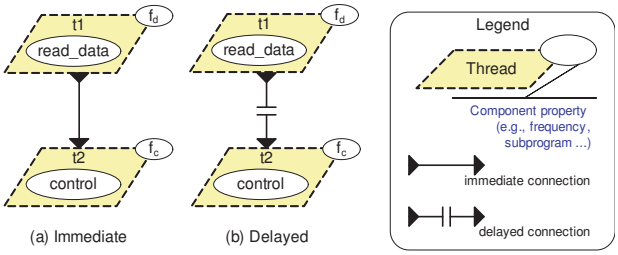

Fig. 1. AADL inter-thread data communication.

In AADL, the communications can be *immediate* (Fig. 1a) or *delayed* (Fig. 1b). The threads are concurrent schedulable units of sequential executions. Several properties can be assigned to threads, the one of concern here is the *dispatch protocol*. We actually consider only periodic threads, associated with a period and a deadline, specified as chronometric time expressions (*e.g.*, period=50ms or frequency=20Hz). By default, when the deadline is not specified it equals the period.

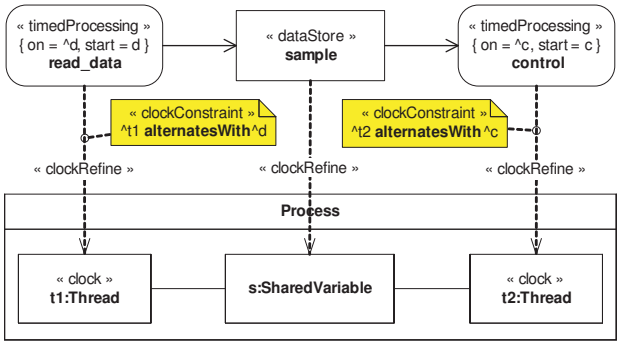

Fig. 2. Application/Execution platform in MARTE.

III. The explicit modeling of AADL communication aspects

A. Application and clock refinement

A first difference with AADL is that MARTE differentiates the algorithm, which can be represented as an activity diagram (Fig. 2, upper part), from the underlying structure, which is modeled here as a composite structure diagram (Fig. 2, lower part), and that implies a logical scheduling. Each part has its own causality constraints. MARTE refinement mechanism, and its associated clock constraints, allows for expliciting relations amongst the clocks of both parts. In MARTE, activation conditions of all application model elements are represented by clocks identified with the appropriate stereotypes, for instance TimedProcessing. As a starting point, we consider the clocks of each element as independent, then the context (dependencies and refinement) constrains these clocks. At last, a timing analysis tool may resolve the constraints to determine a (family of) possible schedules. We strive to avoid overspecification and keep the model as generic as possible, adding only required constraints. From the algorithmic point of view, the actions read_data and control are CallBehaviorAction that execute a given behavior repetitively according to their activation condition (clocks $\widehat{d}$ and $\widehat{c}$ respectively).

B. Introducing clock constraints

From the structural point of view, the threads t1 and t2 are also associated with clocks ($\widehat{t1}$ and $\widehat{t2}$ respectively). These clocks, purely logical, represent the dispatches of the threads. In AADL, the period of a thread is expressed as a chronometric time expression and therefore, at some point, we need to establish relations between these clocks and chronometric clocks. This aspect is addressed in section III-E, but we need to set up some causality relations first.

Deciding that a given behavior (*i.e.*, read_data) is executed by a periodic thread (*i.e.*, t1) implies that each

thread dispatch (modeled by clock $\widehat{t1}$) causes and therefore precedes a new execution of subprogram read_data, and that this execution must complete before the deadline (the next dispatch by default). In MARTE, we differentiate atomic behaviors, for which the execution time is considered negligible as compared to the period, from non-atomic ones. If we consider the behaviors as atomic, the association of a behavior with a thread is simply expressed with the constraint given by Eq. 1. Note that this constraint is not symmetrical since t1 may cause d, but not the converse.

$$\widehat{t1} \texttt{ alternatesWith } \widehat{d} \tag{1}$$

If the execution time is not negligible, each action can be represented by two events, the start (*e.g.*, ds for d, cs for c) and the finish (*e.g.*, df for d, cf for c), and a duration. In that latter case, we need three constraints to express that the behavior read_data is repetitively executed on thread t1 (Eqs. 2–4).

$$\widehat{t1} \texttt{ alternatesWith } \widehat{ds} \tag{2}$$

$$\widehat{t1} \texttt{ alternatesWith } \widehat{df} \tag{3}$$

$$\widehat{ds} \texttt{ isFasterThan } \widehat{df} \tag{4}$$

The first two constraints express that the behavior starts and finishes between two consecutive dispatches of thread t1. The last constraint, which reads clock $\widehat{ds}$ is faster than clock $\widehat{df}$, specifies that the action read_data starts before it finishes; it is sufficient to impose that it finishes within the same cycle of execution.

The next constraint comes from the communication itself. We use a UML data store to mean that the action read_data can overwrite the existing value (in the object node) without generating a new token and this very same value can be read several times by the action control (non depleting read). In UML, there must be at least one writing before any reading (Eq. 5).

$$\widehat{d}[1] \texttt{ precedes } \widehat{c}[1] \tag{5}$$

Let $\widehat{wr}$ be the (logical) clock for *significant writings* in the data store. There could be several consecutive writings in the datastore before one reading. In that case, only the last one is considered significant. Let $\widehat{rd}$ be the corresponding (logical) clock for *significant readings* from the data store. When the same value is read several times, only the first reading is considered to be significant. Furthermore, AADL assumes that communicating threads must have common dispatches. A simple way to achieve that is if all threads start their execution at the same time (they are in phase). The AADL standard considers three cases: *synchronous* threads with the same period, *oversampling* (the period of control is evenly divided by the period of read_data), *undersampling* (the period of read_data is evenly divided by the period of control). Let $q1$ and $q2$ be natural numbers such that $f_d/f_c = q1/q2$. They represent the relative periods of read_data and control. Section III-F discusses how to compute $q1$ and $q2$ in the general case. When the threads are synchronous (Eq. 6), $q1 = q2 = 1$. When oversampling (Eq. 7), $q1 = 1$ and $q2 > 1$. When undersampling (Eq. 8), $q1 > 1$ and $q2 = 1$. $\max(q1, q2)$ is called the hyper-period. In Eq. 7 (resp. Eq. 8), the binary word [6] following the keyword filteredBy expresses that each instant of $\widehat{t1}$ (resp. $\widehat{t2}$) is synchronous with every $q2^{\text{th}}$ (resp. $q1^{\text{th}}$) instant of $\widehat{t2}$ (resp. $\widehat{t1}$).

$$\widehat{t1} \boxed{\equiv} \widehat{t2} \tag{6}$$

$$\widehat{t1} \boxed{\equiv} \widehat{t2} \texttt{ filteredBy } (1.0^{q2-1}) \tag{7}$$

$$\widehat{t2} \boxed{\equiv} \widehat{t1} \texttt{ filteredBy } (1.0^{q1-1}) \tag{8}$$

Selecting the significant writings and readings consists in choosing one every $q1^{\text{th}}$ instant of $\widehat{d}$ (Eq. 9) and one every $q2^{\text{th}}$ instant of $\widehat{c}$ (Eq. 10).

Additionally, Eq. 11 states that each significant writing must precede its related significant reading.

$$\widehat{wr} \texttt{ isPeriodicOn } \widehat{d} \texttt{ period } q1 \tag{9}$$

$$\widehat{rd} \texttt{ isPeriodicOn } \widehat{c} \texttt{ period } q2 \tag{10}$$

$$\widehat{wr} \texttt{ alternatesWith } \widehat{rd} \tag{11}$$

We restrict our comparison to the three cases considered by the AADL standard. However, in subsection III-F we elaborate on the general case.

We have defined all general constraints. In particular, note that contrary to Eqs. 7–8, Eqs. 9–10 do not specify which instant is chosen as a significant writing or reading. The actual instant depends on the semantics of the communication. The following two subsections study the three different cases (synchronous, oversampling, undersampling) with both an immediate and a delayed communication, each subsection gives stronger constraints compatible with Eqs. 9–11.

C. Immediate communication

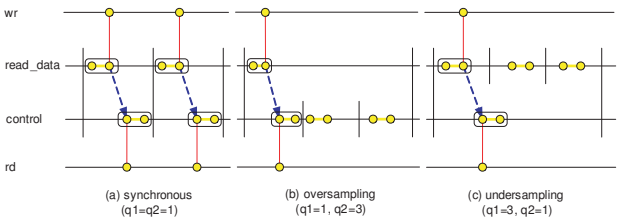

Fig. 3. Immediate communications.

An immediate communication means that the result of the sending thread (here read_data) is immediately available to the receiving thread (here control). When threads are synchronous (Fig. 3a), this is denoted by " $\widehat{wr} \boxed{\equiv} \widehat{d}$ " and " $\widehat{rd} \boxed{\equiv} \widehat{c}$ ", or more precisely by " $\widehat{wr} \boxed{\equiv} \widehat{df}$ "and " $\widehat{rd} \boxed{\equiv} \widehat{cs}$ ". In case of oversampling

(Fig. 3b), the result of the action read_data must be written in the object node early enough so that the *first* (for each $q2$-long hyper-cycle) execution of the action control can use it. This is denoted by " $\widehat{wr} \boxed{\equiv} \widehat{d}$ " and " $\widehat{rd} \boxed{\equiv} \widehat{c}$ filteredBy $\left(1.0^{q2-1}\right)$ ". The latter constraint is stronger than Eq. 10: it implies it. In case of undersampling (Fig. 3c), AADL specifies that the execution of the *first* (for each $q1$-long hyper-cycle) execution of the action read_data must complete before the execution of the action control. This is stated by " $\widehat{rd} \boxed{\equiv} \widehat{c}$ " and " $\widehat{wr} \boxed{\equiv} \widehat{d}$ filteredBy $\left(1.0^{q1-1}\right)$ ".

D. Delayed communication

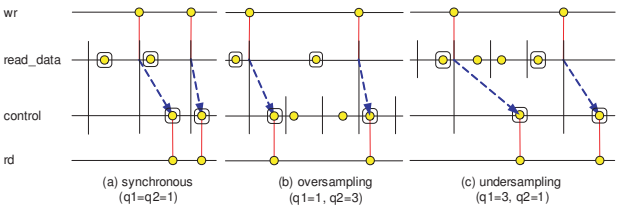

Fig. 4. Delayed communications.

A delayed communication means the result of the sending thread is made available only at its *next* dispatch while the receiving thread only reads *after* its own dispatch and *ultimately* when the data is required. The dispatches of the sending and the receiving threads are not necessarily all synchronous, even if there must be a synchronization at some point. When the thread are synchronous (Fig. 4a), the constraint is denoted by Eqs. 12–13. Note that $\delta 4$ offers the possibility to delay the actual execution of read_data. The thread t1 can either be idle or be executing another action before starting to execute read_data. Eq. 12 states that $(\exists \delta 4 \in \mathbb{N})\,(\forall k \in \mathbb{N}^\star)\left(\widehat{wr}[k] \equiv \widehat{t1}[\delta 4 + k]\right)$.

$$(\exists \delta 4 \in \mathbb{N})\left(\widehat{wr} \boxed{\equiv} \widehat{t1} \text{ filteredBy } 0^{\delta 4}\,(1)\right) \quad (12)$$

$$\widehat{rd} \boxed{\equiv} \widehat{c} \quad (13)$$

For oversampling (Fig. 4b), the result is available for the *first* execution of the action control of the *next* $q2$-long hyper-cycle. This leaves lots of freedom to schedule the action read_data anywhere within the current hyper-cycle. We keep the relation Eq. 12 while Eq. 13 is replaced by Eq. 14.

$$\widehat{rd} \boxed{\equiv} \widehat{c} \text{ filteredBy } \left(1.0^{q2-1}\right) \quad (14)$$

For undersampling (Fig. 4c), the result of the *last* execution (for each $q1$-long hyper-cycle) of the action read_data is available for the action control at the *next* hyper-cycle. This is denoted by combining Eq. 15 with Eq. 13.

$$(\exists \delta 4 \in \mathbb{N})\left(\widehat{wr} \boxed{\equiv} \widehat{t1} \text{ filteredBy } 0^{\delta 4}\left(1.0^{q1-1}\right)\right) \quad (15)$$

Note that the relations are not fully symmetrical. This is due to the AADL semantics that changes the rule depending on the kind of communication.

Up to here, we have only defined logical constraints. In some cases, these constraints are strong enough to get a total order, and thus a possible schedule, on all instants belonging to the defined clocks. For instance, in the delayed synchronous case, whenever the first execution of read_data occurs, the first significant writing occurs at the very next dispatch. However, in some other cases, we need additional stronger constraints to get a schedule. These constraints reflects additional choices that are mainly implicit in the AADL semantics. Depending on these choices we get different deterministic schedules. These cases are studied in the next section.

E. Getting a schedule

Figure 3 shows that for immediate communications, the constraints given define a total order between instants of $\widehat{d}$ and $\widehat{c}$ in both the synchronous and the oversampling cases. Combining our constraints we get the same result analytically. One question remains, it is whether or not both executions (read_data and control) can be performed within the period of thread t2. If not, there is no possible schedule, otherwise, the schedule is given by Figure 5, assuming both threads are executed on the same process.

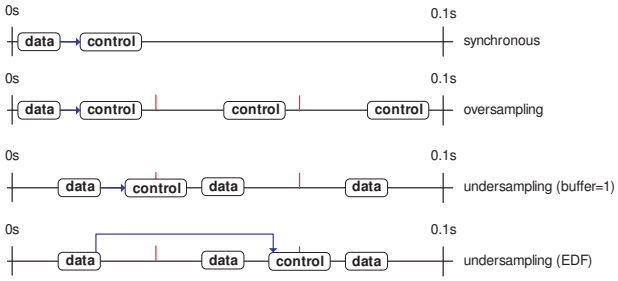

Fig. 5. Schedules with immediate communications.

For delayed communications, additional constraints are required to get a deterministic schedule. Several criteria can be considered, like for instance, the size of the buffer used for the communication, or applying a well-know scheduling policy, like Earliest Deadline First (EDF).

An apparent easy way to force a total order is to project the logical clocks onto chronometric clocks. Logical clocks only give an order amongst instants (sometimes partial), while chronometric clocks give an absolute position in time. The use of chronometric clocks is implied in AADL because of the units used to describe either the frequency (Hz) or the period (s). In MARTE, we create models of chronometric clocks by discretizing idealClk (Sec. II-A). For instance, we create three chronometric clocks c_{100}, c_{10} and c_{30} of respective frequency 100Hz, 10Hz and 30Hz (Eqs. 16–18). Note that these are relations, whence the definition of the 30Hz-clock from c_{10}.

Now, we replace the three equations (Eqs. 6–8) by the three following constraints. $\widehat{t1} \boxed{\equiv} \widehat{t2} \boxed{\equiv} c_{10}$ (synchronous), $\widehat{t1} \boxed{\equiv} c_{10}$ and $\widehat{t2} \boxed{\equiv} c_{30}$ (oversampling), $\widehat{t1} \boxed{\equiv} c_{30}$ and $\widehat{t2} \boxed{\equiv} c_{10}$ (undersampling). The only additional information we have here is the distance (expressed in seconds) between two consecutive dispatches. This information is useful for comparing the duration of executions with the period of the threads, however it does not change in any way the causality relations expressed.

$$c_{100} \boxed{\equiv} \text{idealClk } \texttt{discretizedBy } 0.01 \quad (16)$$

$$c_{10} \boxed{\equiv} c_{100} \texttt{ filteredBy } (1.0^{9}) \quad (17)$$

$$c_{10} \boxed{\equiv} c_{30} \texttt{ filteredBy } (1.0^{2}) \quad (18)$$

For the immediate undersampling, we can infer from the specified constraints that, for each hyper-cycle, the first execution of read_data must complete before the execution of control. However, we cannot decide when to execute control relatively to other executions of read_data. We need another criterion. For instance, we choose to minimize the actual size of the buffer used for the communication. To get this buffer as small as possible (size=1), we have to schedule control before the second execution of read_data. Were we to schedule according to an EDF policy we would get another schedule, see Fig. 5.

For a delayed communication, we just have partial orders and we need additionnal criteria. For synchronous threads, the use of an EDF policy is of no help. However, reducing the size of the communication buffer gives a schedule (top-most part of Fig. 6). For oversampling, both criteria are compatible and we get the second schedule on Fig. 6. For undersampling, we get two different schedules depending on whether we apply an EDF policy or we attempt to reduce the buffer size.

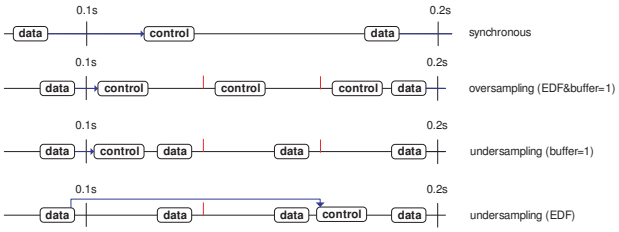

Fig. 6. Schedules with delayed communications.

F. Generalization

We can generalize the constraints to get only two sets of constraints, one for the immediate communication and one for the delayed communication.

In this section we do not restrict to the three special cases addressed in the AADL standard. This generalization does not assume that the frequencies of the threads are natural numbers, it just assumes that they are rational numbers. It also assumes that in the notation of our binary words $Y.x^0 = Y$, for any binary word Y and any bit x.

Let $f_d = n_r/d_r$ and $f_c = n_c/d_c$, $f_d/f_c = (n_r * d_c) / (n_c * d_r)$ with $n_r, n_c, d_r, d_c \in \mathbb{N}^\star$. Let $r_1 = n_r * d_c$ and $r_2 = n_c * d_r$. We choose $q1$ and $q2$ such as $q1 = r_1/\gcd(r_1, r_2)$ and $q2 = r_2/\gcd(r_1, r_2)$. Note, that we still have $f_d/f_c = q1/q2$ and that the constraints given by Eq. 15 and Eq. 14 are general. However, Eqs.6–8 are replaced by a single one, Eq. 19.

$$\widehat{t1} \texttt{ filteredBy } \left(1.0^{q1-1}\right) \boxed{\equiv} \widehat{t2} \texttt{ filteredBy } \left(1.0^{q2-1}\right) \quad (19)$$

Again, these constraints are purely logical. In the general case, these constraints are not strong enough to identify deterministically the significant writings and readings. If we take for instance, the case where $q1 = 2$ and $q2 = 5$ (Fig. 7). If we apply the AADL semantics, we can only say that, within an hyper-cycle (of period $\text{lcm}(q1, q2)$), the first execution of read_data produces the sample for the first control, but we cannot know what sample is used by other executions of control. In particular, there is no relation between $t1[2 * n + 1]$ and $t2[5 * n + 2]$.

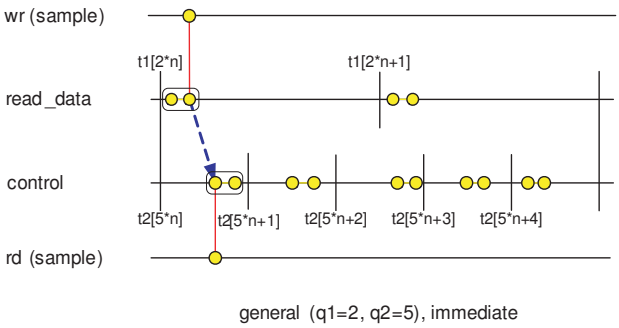

Fig. 7. General case with immediate communications and purely logical clocks.

To get a deterministic behavior, we need to give more constraints. For instance we can project our clock to chronometric clocks and we model as an example the case where $f_d = 10\text{Hz}$ and $f_c = 25\text{Hz}$. We proceed by using the clock c_{100} defined in Eq. 16 and we add two new constraints given below.

$$\widehat{t1} \boxed{\equiv} c_{10} \quad (20)$$

$$\widehat{t2} \boxed{\equiv} c_{100} \texttt{ filteredBy } (1.0^{3}) \quad (21)$$

With such constraints, we get a total order (Fig. 8) and then there are two possible cases. The first case appears when $duration(read\_data) + duration(control) \geq 0.02s$. Then, we exactly get the result presented in Fig. 8, where, within an hyper-cycle, the third execution of control uses the sample computed by the first execution of read_data and the fourth execution of control uses the sample computed by the second execution of read_data. In the second case, if $duration(read\_data) +$

$duration(control) < 0.02s$, the third execution of control should use the sample computed by the second execution of read_data. However, note that such systems that very much depend on the exact duration of tasks are not very robust.

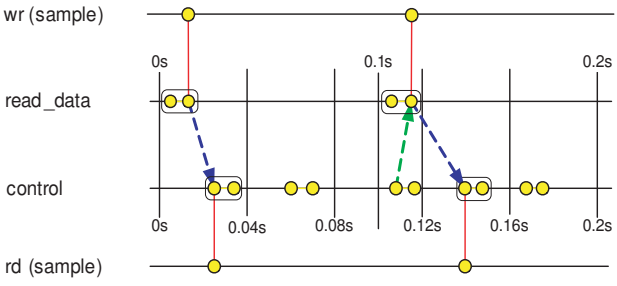

Fig. 8. General case with immediate communications and chronometric clocks.

If we now take a look at the situation with a delayed communication, there are several possible interpretations of a generalized AADL semantics. The simplest interpretation is that the data is made available (written in the object node) at the first dispatch (of the sending thread) following the execution of the behavior that has produced it (read_data). And the data is read at the first dispatch of the receiving thread following the writing (see Fig. 9).

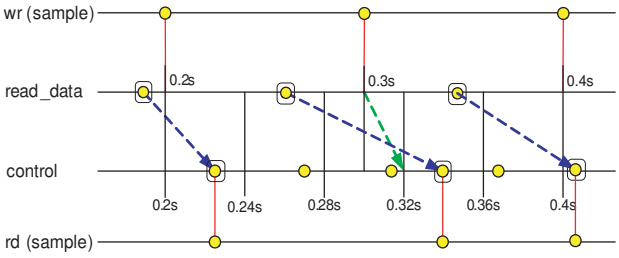

Fig. 9. General case with delayed communications (first interpretation).

A second interpretation could be that the data is read at the first dispatch of the receiving thread following the actual production of the data (not waiting for the following dispatch of the sending thread). This interpretation leads to make the second significant reading synchronous the third instant of control (for each hypercycle) instead of the fourth as in Figure 9. These cases are studied in detail in [5]. Note these two interpretations can all be valid and deterministic. It is just a matter of making explicit the semantics. The first interpretation is very simple to implement and the second one requires to be able to control very tighly the communication times.

A UML object node has two interesting attributes: it has an upper bound, possibly unlimited, and it can order events, by default according to a FIFO policy. Thus, there is no reason to assume that the threads are in phase, the sending thread writes (and possibly overwrites) tokens in the object node, while the receiving thread reads them when required. Our definition of the significant writings and readings helps defining when the token is the same—the content must be ovewritten—and when the token is different, which implies that a new token must be created. Actually, the occurrence of $\widehat{wr}$ should create a new token.

IV. Conclusion

We have briefly introduced the Time model of MARTE and we have illustrated its use on an example taken from AADL. We think that our clock constraint language could be used to make formal the semantics of UML-like graphical representations that is often partially implicit. In this language, we borrowed some notations on binary words from the N-synchronous approach but in our case we do not limit ourself to synchronous relations. We have implemented a constraint parser that has been made available with the XMI of the Time subprofile on the OMG website. This parser can be used to parse constraints extracted from UML models. Some analytic tools should reduce the constraints or compute new ones and put them back in the models. For now, all these formal computations are manual but we intend to transform our constraints into languages amenable to clock computations (time automata or synchronous languages like Signal or Esterel). Ultimately, our constraint language could be used to drive a UML simulator, in a constructive way, according to the model time semantics rather than an untimed event-driven semantics.

References

[1] S. Standards, *SAE Architecture Analysis and Design Language (AADL)*, June 2006, document number: AS5506/1. [Online]. Available: http://www.sae.org/technical/standards/AS5506/1

[2] OMG, *UML profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE), Request for proposals*, February 2005, oMG document number: realtime/2005-02-06.

[3] E. A. Lee and A. L. Sangiovanni-Vincentelli, "A framework for comparing models of computation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, no. 12, pp. 1217–1229, December 1998.

[4] P. H. Feiler, D. P. Gluch, J. J. Hudak, and B. A. Lewis, "Embedded System Architecture Analysis using SAE AADL," Carnegie Mellon University, Tech. Rep. CMU/SEI-2004-TN-005, June 2004. [Online]. Available: //www.sei.cmu.edu/pub/documents/04.reports/pdf/04tn005.pdf

[5] C. André, F. Mallet, and R. de Simone, "Modeling time(s)," in *MoDELS 2007*, October 2007.

[6] A. Cohen, M. Duranton, C. Eisenbeis, C. Pagetti, F. Plateau, and M. Pouzet, "N-synchronous kahn networks," in *POPL 2006*, January 2006.