

Chapter 2

MARTE vs. AADL for Discrete-Event and Discrete-Time Domains

Frédéric Mallet and Robert de Simone

Abstract Real-time embedded applications tend to combine periodic and aperiodic computations. Modeling standards must then support both discrete-time and discrete-event models of computation and communication whereas they historically pertain to two different communities: asynchronous and synchronous designers. In this article, two emerging standards of the domain (MARTE and AADL) are compared and their ability to tackle this issue is assessed. We plead for combining both standards and show how MARTE can be extended to integrate AADL features required for end-to-end flow latency analysis.

Keywords UML Marte · AADL · MoCC · Time requirement

2.1 Introduction

Embedded applications often combine aperiodic (or sporadic) and periodic computations. In the automotive industry, this has led to bus standards like FlexRay (<http://www.flexray.org>) or TT-CAN [10] that combine event-triggered messages (for aperiodic computations) with time-triggered messages (for periodic computations). In the avionic industry, an application generally mixes aperiodic events (e.g., interactions with the pilot, switching between air/ground modes ...) together with periodic events when updating the system (e.g., fuel quantity, update system data ...). On the one hand, time-triggered approaches enhance predictability by reducing latency jitters and provide higher dependability by making it easier to detect missed messages or illegal accesses to the bus. On the other hand, event-triggered systems are more flexible to support configuration changes without a complete redesign and adapt faster to asynchronous events. In electronic design automation (EDA) event-driven simulators (like those for VHDL or Verilog) provide a large flexibility and support the design of both synchronous and asynchronous architectures. Though, cycle-based simulators have better performances provided that architectures are mainly synchronous.

In EDA, Avionic and automotive industries, designers need models able to describe and compose these two communication models. The main point is that, even

F. Mallet (✉)

Aoste Project I3S-INRIA, INRIA Sophia Antipolis Méditerranée, Université de Nice
Sophia Antipolis, Sophia Antipolis Cedex, France
e-mail: Frederic.Mallet@sophia.inria.fr

in time-triggered sampled communications, the propagation of data in logically instantaneous communications introduces specific phenomena akin to event-based communication features. Indeed, the consumer must wait for data availability to start computing on it. This may introduce well-known problems of priority inversion when component blocks have to be executed atomically. All these phenomena deserve careful semantic treatment to be handled correctly, which is the true essence of this work.

Considering the large number of actors in the design of the very large systems (or even systems of systems) standard-based approaches are required to provide interoperability between models and to cover the whole design flow, from system requirements to code generation. These models must be precise enough to support various analyses at different refinement levels. We focus here on two particular standards, AADL (Architecture Analysis & Design Language) [12] standardized by the Society of Automotive Engineers and the UML (Unified Modeling Language) profile for MARTE (Modeling and Analysis of Real-Time and Embedded systems) [14], recently adopted by the Object Management Group (OMG). Both standards focus on modeling and analysis of embedded systems. Both offer constructs to model the application, the execution platform and to allocate the former onto the latter.

The expressiveness of MARTE and AADL is compared and their ability to combine periodic and aperiodic computations is assessed. Concerning MARTE, the discussion focuses on its Time Model [2], specifically devised to specify timed domains of computation and communication in a formal way. This is the continuation of some of our previous work [3, 9] to compare both formalisms. An AADL example [6], which comes with its implementation, is used for the comparison. In the selected example, several threads, periodic or not are connected through event, data or event-data ports. The combination of various parameters induces either asynchronous or sampled communications.

AADL two-layered model is compared to our three-layered UML-based approach. The latter gives more flexibility and avoids the mixing of different levels in the same model. Rather than opposing the two languages we have investigated how the two standards can be combined and gateways can be created. Indeed, a subset of MARTE can be combined with AADL to cover a larger scope than the one currently covered by AADL, thus benefiting to AADL users. Such a combination would also benefit to MARTE users because some of their models could then be analyzed by existing AADL tools.

Section 2.2 gives an overview of the MARTE time model. Section 2.3 introduces AADL and shows how MARTE is used to model the AADL constructs that address the two communication schemes under focus. Section 2.4 presents MARTE models for three configurations of an AADL-inspired example.

2.2 Marte Time Model

Time and time-related concepts of the UML profile for MARTE have been previously described [2]. This section recalls the main definitions.

2.2.1 Definitions

In MARTE, Time can be *physical* and considered as *continuous*, or *discretized*. 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 interest to tackle multiform time has been exhibited in synchronous languages [4]. MARTE Time subprofile, inspired from the theory of tags systems [8], provides a set of general mechanisms to define Models of Computations and Communications (MoCC). These modeling aspects should be hidden to end-users but not to model architects. This work intends to build a MoCC suitable for AADL.

The *time structure* is defined by a set of *clocks* and *relations* on these clocks. Here clocks are not a device used to measure the progress of physical time. It is rather a mathematical object lending itself to formal processing. Clocks referring to physical time are called *chronometric clocks*. A distinguished chronometric clock named *idealClk* is provided as part of the MARTE time library. This clock represents the “ideal” physical time used, for instance, in physics and mechanics laws. At the design level most of the clocks are *logical*. For instance, we consider the processor cycle or the bus cycle as being logical clocks. Making a distinction between chronometric and logical clocks is important. For chronometric clocks, a metric is associated with instant labels thus making relevant the distance between two instants. For logical clocks, the distance between two successive instants is irrelevant.

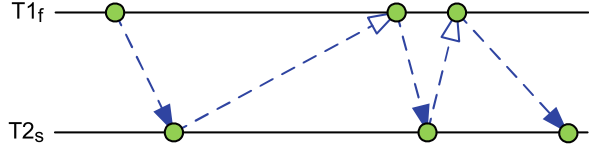
More precisely, a *Clock* is an ordered set of instants \mathcal{I} and a quasi-order relation $<$ on \mathcal{I} , named *strict precedence*. $<$ is a total, irreflexive, and transitive binary relation on \mathcal{I} . A *discrete-time clock* is a clock with a discrete set of instants \mathcal{I} . A *Time Structure* is a set of clocks \mathcal{C} with a binary, reflexive and transitive relation \preceq named *precedence*. *pred* is a partial order relation on the set of all instants of all clocks within a time structure. From \preceq we derive another instant relation named *Coincidence* ($\equiv \triangleq \preceq \cap \succ$).

Clocks are independent of each other unless some instant relations are imposed. To impose many—or infinitely many—instant relations at once, clock relations are used. A comprehensive description of clocks relations is available as a research report [1], we focus here on those required to represent event-triggered and time-triggered communications. Connections between UML model elements and MARTE clock relations are made through stereotype *ClockConstraint*, which extends meta-class *UML::Constraint*. The language to be used on these clock constraints is called *Clock Constraint Specification Language*, CCSL. It is defined as an annex of MARTE specification and its formal semantics is briefly introduced separately [11].

2.2.2 Event-Triggered Communications

Two clocks (t_s, t_f) are associated with each task t , the first one contains instants at which the task starts and the other the instants at which it finishes. A task cannot

Fig. 2.1 Clock relation *alternatesWith*



end before having started and every time a task starts it must finish, in one way or another (normal ending, preemption, interrupted). The clock relation *alternatesWith* can represent this causality relation between t_s and t_f . Equation (2.1) denotes that every i th instant of t_s strictly precedes every i th instant of t_f which in turns (weakly) precedes every $(i + 1)$ th instant of t_s . This relation is not symmetrical and does not assume the task t as being periodic.

$$t_s \text{ alternatesWith } t_f \quad (t_s \boxed{\sim} t_f) \quad (2.1)$$

$$t1_f \text{ alternatesWith } t2_s \quad (t1_f \boxed{\sim} t2_s) \quad (2.2)$$

Alternation is a very general relation and can also represent an event-triggered communication from a task $t1$ to another task $t2$, Eq. (2.2). Task $t1$ runs to completion and sends an event that triggers the execution of $t2$.

Figure 2.1 illustrates the alternation relation. Horizontal lines represent the clocks and their instants. Dashed arrows with a filled triangle as an arrowhead are strict precedence relations whereas arrows with a hollow triangle as an arrowhead are (weak) precedence relations. The precedence relations are directly induced by Eq. (2.2).

In that example, the termination of $t1$ asynchronously triggers the start of $t2$. Note that we only have partial orders i.e., no instant relation is induced between the start or end of $t2$ and the next start of $t1$.

2.2.3 Time-Triggered Communications

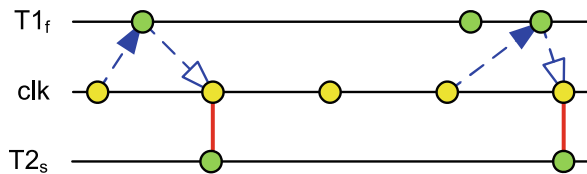
With time-triggered communications, the data is sampled from a buffer according to a triggering condition. Clock relation *sampledOn* is used to represent sampling and the triggering condition is given by the instants of clocks.

Following our previous example, we replace Eq. (2.2) by Eq. (2.3). clk is the sampling condition, i.e., the triggering clock.

$$t2_s \equiv t1_f \text{ sampledOn } clk \quad (2.3)$$

Figure 2.2 illustrates the use of clock relation *sampledOn*. It does not show the start of $t1$ since it is not relevant here. The start of task $t2$ is precisely given by sampling clock clk , however, some events may be missed if the sampling clock is not fast enough. Vertical lines denote coincidence relations.

Fig. 2.2 Clock relation sampledOn



2.2.4 Periodic Tasks and Physical Time

Logical clocks are infinite sets of instants but we do not assume any periodicity, i.e., the distance between successive instants is not known. The relation `discretizedBy` is used to discretize `idealClk`, a dense chronometric (related to physical time) perfect (with no jitter or any other flaw) clock. Equation (2.4) creates, as an example, a 100 Hz clock.

$$c_{100} \equiv \text{idealClk discretizedBy } 0.01 \quad (2.4)$$

Equation (2.4) states that the distance (duration) between two successive instants of clock c_{100} is 0.01 s. The unit second (s) is implied by the use of `idealClk`.

2.2.5 TimeSquare

TIMESQUARE is a software environment for modeling and analysis timed systems. It supports an implementation of the Time Model introduced in MARTE and its companion Clock Constraint Specification Language (CCSL). TIMESQUARE displays possible time evolutions—solutions to the clock constraint specification—as waveforms generated in the VCD format [7]. The VCD format has been chosen because it is an IEEE standard defined as part of the Verilog language and as such, is often used in EDA. TIMESQUARE is available at <http://www.inria.fr/sophia/aoste/>.

2.3 AADL

2.3.1 Modeling Elements

AADL supports the modeling of application software components (thread, subprogram, process), execution platform components (bus, memory, processor, device) and the *binding* of software onto execution platform. Each model element (software or execution platform) must be defined by a type and comes with at least one implementation.

Initially, there were plans to create a specific UML profile for AADL. However, the emerging profile for MARTE is now expected to be the basis for UML representation

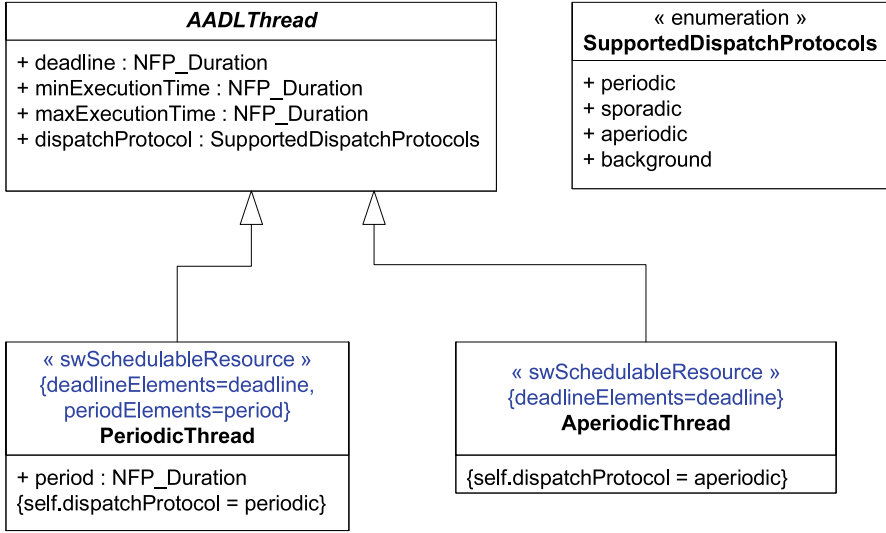


Fig. 2.3 MARTE model library for AADL threads

of AADL models [5]. The adopted MARTE specification provides guidelines in this direction. The main goal of this contribution is to further investigate how specific AADL concepts required for end-to-end flow latency analysis can be represented in MARTE. As such, this work is not (yet?) included in the official OMG specification.

2.3.2 AADL Application Software Components

Threads are executed within the context of a process, therefore the process implementations must specify the number of threads it executes and their interconnections. Type and implementation declarations also provide a set of properties that characterizes model elements. For threads, AADL standard properties include the dispatch protocol (periodic, aperiodic, sporadic, background), the period (if the dispatch protocol is periodic or sporadic), the deadline, the minimum and maximum execution times, along with many others.

We have created a UML library (see Fig. 2.3) to model AADL application software components [9]. Only elements of our library concerning the periodic and aperiodic threads are shown.

AADL threads are modeled using the stereotype `SwSchedulableResource` from the MARTE Software Resource Modeling sub-profile. Its meta-attribute `deadlineElements` and `periodElements` explicitly identify the actual properties used to represent the deadline and the period. Using a meta-attribute of type `Property` avoids a premature choice of the type of such properties. This makes it easier for the transformation tools to be language and domain independent. In our library, MARTE type `NFP_Duration` is used as an equivalent for AADL type `Time`.

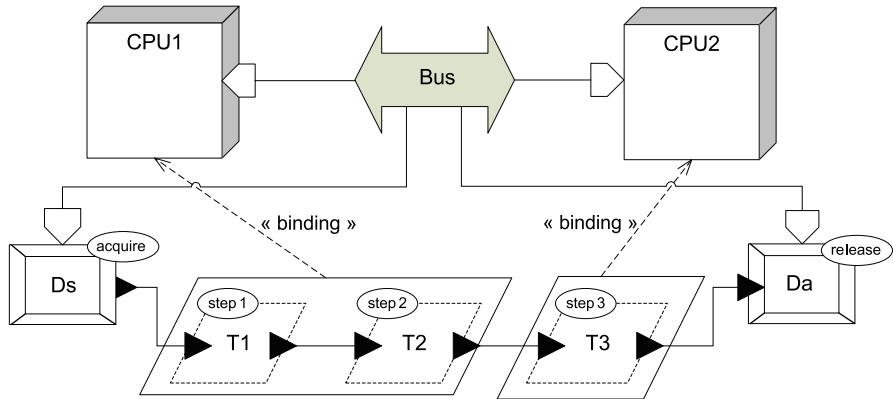


Fig. 2.4 The example in AADL

2.3.3 AADL Flows

AADL end-to-end flows explicitly identify a data-stream from sensors to the external environment (actuators). Figure 2.4 shows an example previously used [6] to discuss flow latency analysis with AADL models.

This flow starts from a sensor (Ds, an aperiodic device instance) and sinks in an actuator (Da, also aperiodic) through two process instances. The first process executes the first two threads and the last thread is executed by the second process. The two devices are part of the execution platform and communicate via a bus (db1) with two processors (cpu1 and cpu2), which host the three processes with several possible bindings. All processes are executed by either the same processor, or any other combination. One possible binding is illustrated by the dashed arrows. The component declarations and implementations are not shown. Several configurations deriving from this example are modeled with MARTE and discussed in Sect. 2.4.

2.3.4 AADL Ports

There are three kinds of ports: *data*, *event* and *event-data*. Data ports are for data transmissions without queueing. Connections between data ports are either *immediate* or *delayed*. Event ports are for queued communications. The queue size may induce transfer delays that must be taken into account when performing latency analysis. Event data ports are for message transmission with queueing, here again the queue size may induce transfer delays. In our example, all components have data ports represented as a solid triangle. We have omitted the ports of the processes since they are required to be of the same type than the connected port declared within the thread declaration and are therefore redundant.

UML components are linked together through ports and connectors. No queues are specifically associated with connectors. The queueing policy is better repre-

sented on a UML activity diagram, that models the algorithm. Activities are made of actions. The execution sequence is given by the control flow. Data communications between the actions are represented with object flows. In UML, by default, an object flow has a queue, the size of which can be parameterized with its property `upperBound`. So object flows can be used to represent both event and event-data AADL communication links. UML allows the specification of a customized selection policy to select which token is read among the ones stored in the object node. Unfortunately, the selection behavior is only allowed to select one single token making it impossible to represent the AADL dequeue protocol `AllItems`. This protocol dequeues all items from the port every time the port is read. Therefore, only the dequeue protocol `OneItem` is supported.

To model data ports, UML provides «datastore» object nodes. On these nodes, tokens are never consumed thus allowing for multiple readings of the same token. Using a data store node with an upper bound equal to one is a good way to represent AADL data port communications.

2.4 Three Different Configurations

This section illustrates the use of MARTE on three different configurations of the AADL example. First, we address a case where all threads are aperiodic. Then, we consider a mixed periodic/aperiodic case. We finish with a case where all threads are periodic and harmonic.

2.4.1 The Aperiodic Case

We rely on a model with three layers (see Fig. 2.5) where each layer denotes a different aspect of the system. The top-most layer represents the algorithm, i.e., different actions to be executed, and includes the data and control flow. The algorithm gives causal relations among the actions. All communications are through event-data ports with infinite queues, represented as object nodes. The two actions acquire and release model the behavior of the two devices.

The middle layer is a composite structure diagram that models AADL software components and represents the actual configuration under study. Here, all threads are aperiodic and therefore the classifier `AperiodicThread` defined in Fig. 2.3 is used. The bottom layer represents the execution platform.

This layer-oriented approach significantly differs from the AADL model where all the layers are combined. AADL models do not consider the pure applicative part but rather merge it either within the second or the third level (compare with Fig. 2.4). Layers can be changed independently of the others, which gives a great flexibility. If required, new layers can be added to model virtual machines or middleware, for instance.

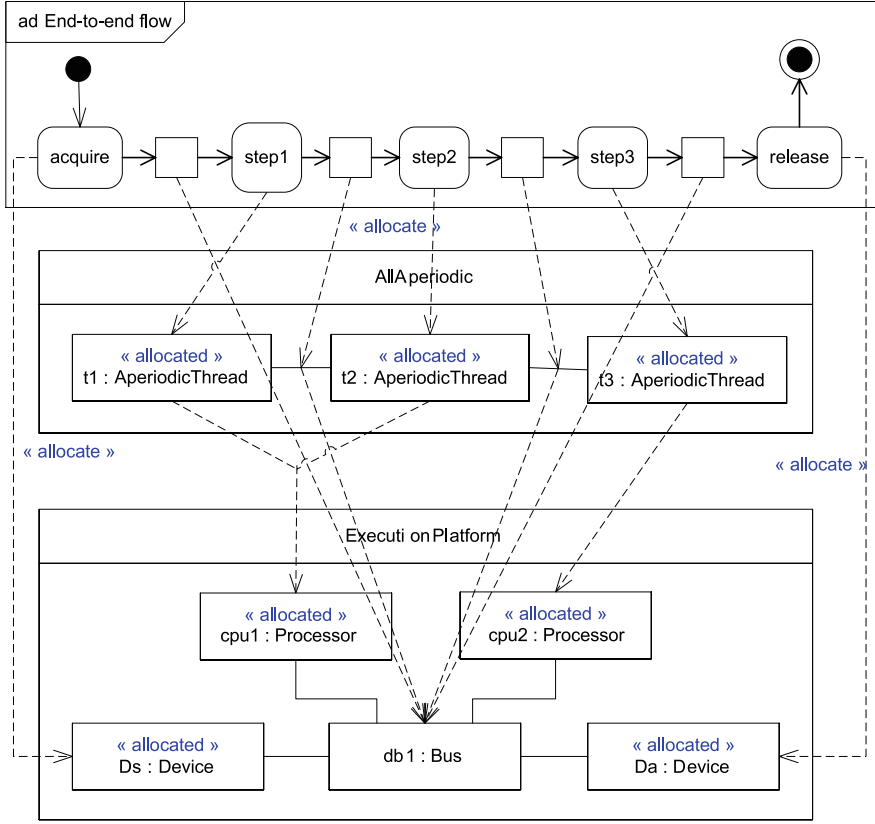


Fig. 2.5 MARTE model, fully aperiodic case

The AADL binding mechanism is equivalent to the MARTE allocation. Actions and object nodes are allocated (dashed arrows) to software components. All threads are aperiodic, therefore all communications are asynchronous and we only use clock relation `alternatesWith` (Eqs. (2.5)–(2.8)).

$$Ds \text{ alternatesWith } T1_s \quad (Ds \sqsim T1_s) \quad (2.5)$$

$$T1_f \text{ alternatesWith } T2_s \quad (T1_f \sqsim T2_s) \quad (2.6)$$

$$T2_f \text{ alternatesWith } T3_s \quad (T2_f \sqsim T3_s) \quad (2.7)$$

$$T3_f \text{ alternatesWith } Da \quad (T3_f \sqsim Da) \quad (2.8)$$

These clock relations are extracted using model-driven engineering techniques and fed into TIMESQUARE. TIMESQUARE checks the relation consistency and proposes, when possible, one execution conformant to the clock relations (see Fig. 2.6).

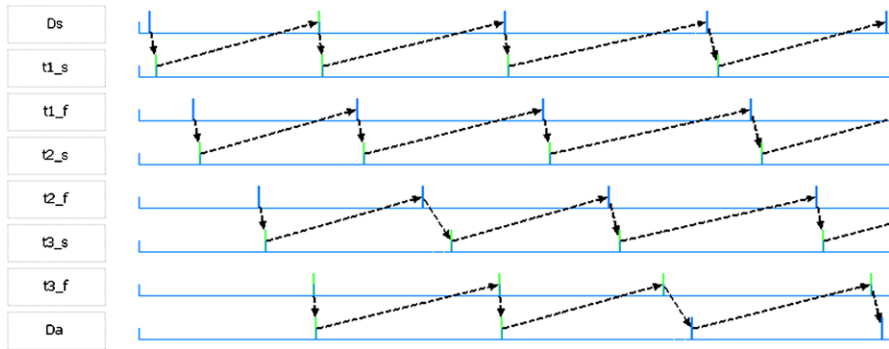


Fig. 2.6 VCD result, all aperiodic case

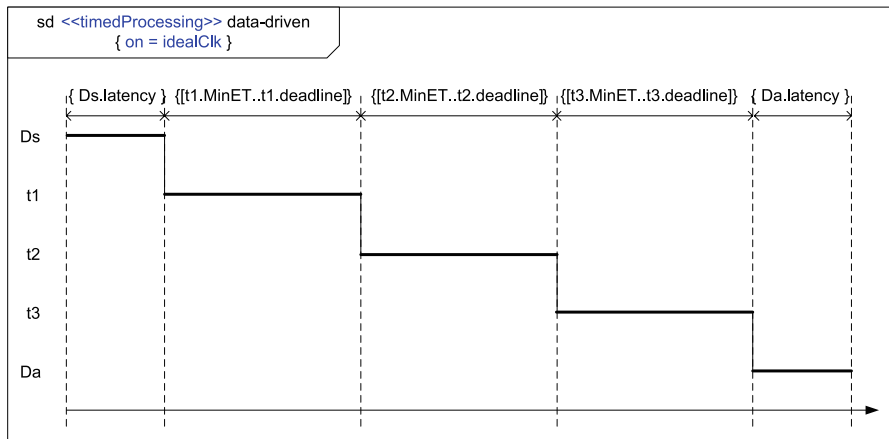


Fig. 2.7 Timing diagrams, all aperiodic case

With TIMESQUARE, the VCD output is annotated to embed clock relations so that instant relations are displayed. Dashed arrows denote precedences. The transformation could also target other analysis tools like, for instance, Cheddar [13], often used with AADL models.

After the analysis, it is important to bring back the results into the UML model. The model elements closest to VCD waveforms are the UML timing diagrams. By combining the two clocks relative to each task (e.g., $t1_s$ and $t1_f$) the whole information relative to the task itself (e.g., $t1$) is built. The result, illustrated in Fig. 2.7, represents a family of possible schedules for a given execution flow and a given pair application/execution platform.

Computation execution times (thick horizontal lines) equal the latency for devices and range between the MinimumExecutionTime (MinET) and the Deadline for threads.

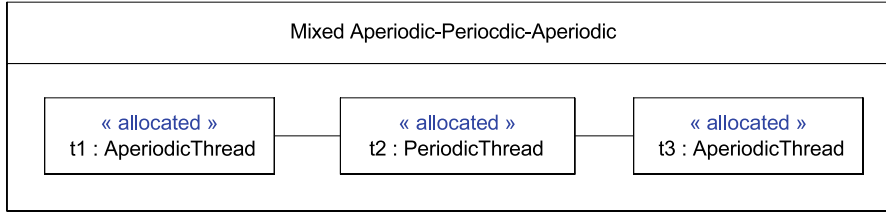


Fig. 2.8 MARTE model, mixed case

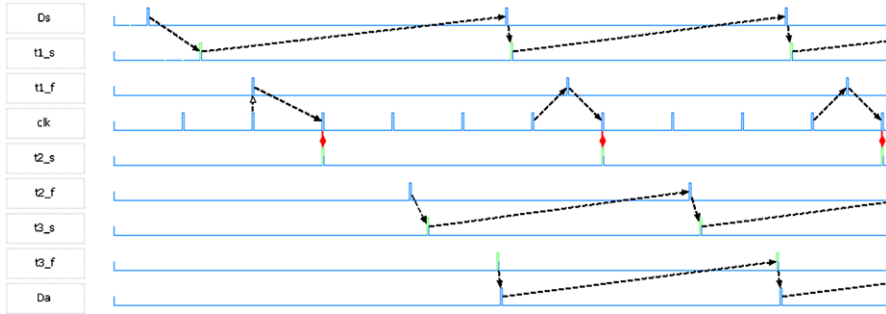


Fig. 2.9 VCD result, mixed case

2.4.2 The Mixed Event–Data Flow Case

We study here a second configuration that only differs by making periodic thread $t2$ (Fig. 2.8). Only the second layer of our model needs to be modified, whereas with AADL the whole model has been rebuilt.

The communication from $step1$ to $step2$ becomes a sampled one. In CCSL, Eq. (2.6) is replaced by Eqs. (2.9)–(2.10), where P is the sampling period of $t2$.

$$clk = IdealClk \text{ discretizedBy } P \quad (2.9)$$

$$t2_s = t1_f \text{ sampledOn } clk \quad (2.10)$$

The new simulation run processed by TIMESQUARE is shown in Fig. 2.9. Vertical plain lines with diamonds represent coincidence relations on instants.

We also get a different timing diagram (see Fig. 2.10). Oblique lines linking two computation lines represent communications and sampling delays. For sampled communications, this amounts to wait for the next tick of the receiver clock. The maximal sampling delay is when the communication waits for the full sampling period because the previous tick has just been missed. “Oblique” lines are not normative in UML timing diagrams but it is a convenient notation to represent intermediate communication states between two steady processing states (e.g., between $t1$ and $t2$).

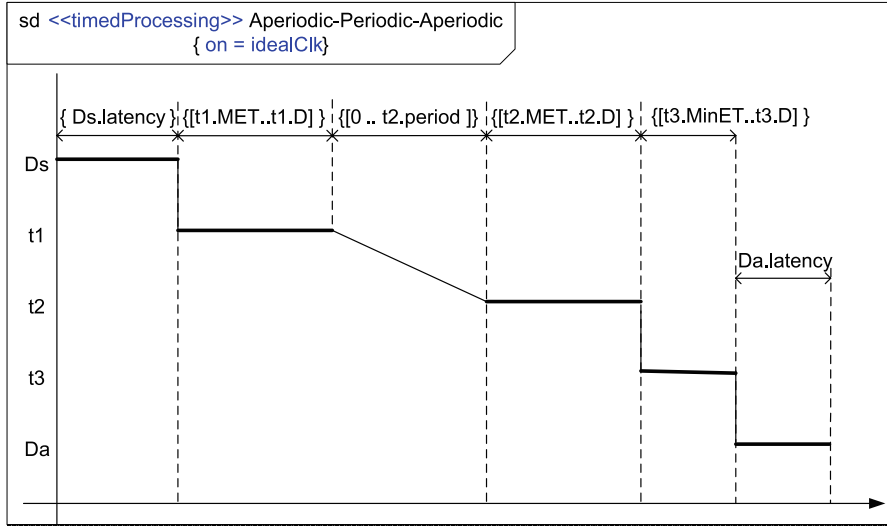


Fig. 2.10 Timing diagram, mixed case

Additionally, on this representation it is easy to process latencies for a given flow. On this configuration and assuming, as in [6], that the sampling delays are always maximal, we get the formulas given in the following equations.

$$\begin{aligned} Latency_{\text{worst-case}} = & Ds.latency + \sum_i (t_i.deadline) \\ & + t2.period + Da.latency \end{aligned} \quad (2.11)$$

$$\begin{aligned} Latency_{\text{best-case}} = & Ds.latency + \sum_i (t_i.MinET) \\ & + t2.period + Da.latency \end{aligned} \quad (2.12)$$

$$Latency_{\text{jitter}} = \sum_i (t_i.deadline - t_i.MinET) \quad (2.13)$$

The jitter (Eq. (2.13)) is identical to the fully asynchronous case, even though, due to the synchronization, the best-case and worst-case latencies are increased.

2.4.3 The Periodic Case

Finally, we address here the case where all threads are periodic but not fully synchronous. Thread t2 is twice as slow as threads t1 and t3, i.e., its period is twice larger. All threads being periodic, their deadline is assumed to be smaller than their period. Only the second layer needs to be replaced by a new configuration where all

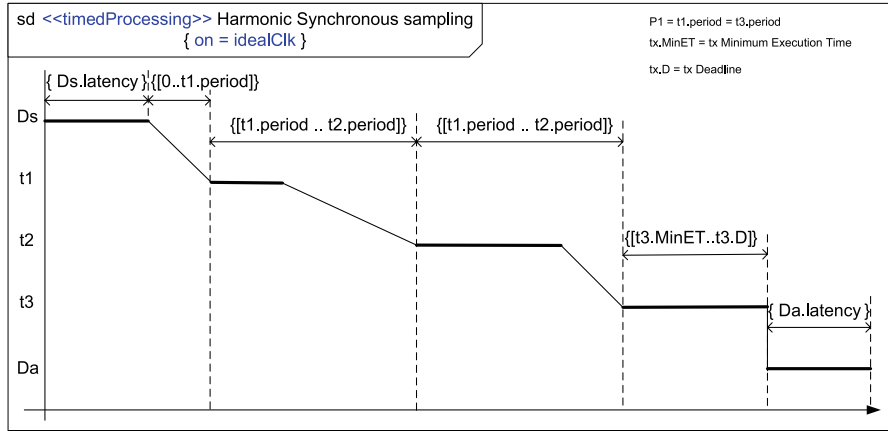


Fig. 2.11 Timing diagram, fully periodic case

threads are periodic. The timing diagram obtained with this configuration is shown in Fig. 2.11.

The first three communications in the flow (from acquire to step1, from step1 to step2, and from step2 to step3) are sampled communications. The last one (from step3 to release) is data-driven, since actuator Da is aperiodic. In this last configuration and as expected, becoming synchronous makes the system more predictable since the latency jitter is much smaller (Eq. (2.16)). However, both the best-case and worst-case are bigger than in the two previous cases.

$$\begin{aligned} Latency_{\text{worst-case}} &= Ds.latency + t1.period + t2.period \\ &\quad + t2.period + t3.deadline + Da.latency \end{aligned} \quad (2.14)$$

$$\begin{aligned} Latency_{\text{best-case}} &= Ds.latency + t1.period + t2.period \\ &\quad + t2.period + t3.MinET + Da.latency \end{aligned} \quad (2.15)$$

$$Latency_{\text{jitter}} = t3.deadline - t3.MinET \quad (2.16)$$

2.5 Conclusion

AADL offers lots of features very important to model and analyze computations and communications of embedded systems. However, combining all these features without a guideline (not part of the standard) can lead to models completely meaningless and impossible to analyze. We have shown how the MARTE Time model could be used to have the same expressiveness with less modeling concepts. More generally, MARTE and its time model could be used to model various timed models of computation and communication.

It is important to have specifications free, as much as possible, of implementation choices (platform independent models). To achieve this goal, we need model

elements of a higher level of abstraction than AADL threads. AADL two-level models assume that part of the application has already been allocated to a software execution platform made of threads. Our approach makes such an allocation explicit when required and allows alternative solutions. We propose to use for that purpose UML activities. Making a link to the software execution platform (runtime executive) is not a refinement but rather an allocation. The former implies models of the same nature, whereas the latter makes links between models of different natures.

Additionally, rather than building a specific graphical editor for AADL models it is more cost-effective to customize existing editors, give them the right semantics and perform model transformations towards analysis tools. For now, graphical customization supported by profiling tools is limited and big improvements are required. However, making AADL UML-friendly leads the path to interoperability with several other modeling standards like SysML [15], which is appropriate to model at the system level.

Glossary

AADL Architecture Analysis and Design Language, standardized by SAE

EDA Electronic Design Automation

MARTE Modeling and Analysis of Real-Time and Embedded systems

MET/MinET Minimum Execution Time

MoCC Model of Computation and Communication

NFP Non Functional Property

OMG The Object Management Group

SAE Society of Automotive Engineers

TT-CAN Time-Triggered Controller Area Network

UML The Unified Modeling Language, adopted by the OMG

VCD Value Change Dump

VHDL Very high speed integrated circuit Hardware Description Language

References

1. C. André and F. Mallet. Clock constraints in UML MARTE CCSL. Research Report 6540, INRIA, May 2008. <https://hal.inria.fr/inria-00280941>.
2. C. André, F. Mallet, and R. de Simone. Modeling time(s). In *MoDELS'07*, LNCS 4735, pages 559–573. Springer, Berlin, 2007.
3. C. André, F. Mallet, and R. de Simone. Modeling of AADL data-communications with UML Marte, In E. Villar, editor, *Embedded Systems Specification and Design Languages, Selected Contributions from FDL'07*, LNEE 10, pages 150–170. Springer, Berlin, 2008.
4. A. Benveniste, P. Caspi, S. A. Edwards, N. Halbwachs, P. Le Guernic, and R. de Simone. The synchronous languages 12 years later. *Proceedings of the IEEE*, 91(1):64–83, 2003.

5. M. Faugère, T. Bourbeau, R. de Simone, and S. Gérard. Marte: Also a UML profile for modeling AADL applications. In *ICECCS*, pages 359–364. IEEE Comput. Soc., Los Alamitos, 2007.
6. P.H. Feiler and J. Hansson. Flow latency analysis with the architecture analysis and design language. Technical Report CMU/SEI-2007-TN-010, CMU, June 2007.
7. IEEE Standards Association. *IEEE Standard for Verilog Hardware Description Language*. IEEE Std 1364TM-2005, Design Automation Standards Committee, 2005.
8. E.A. Lee and A.L. Sangiovanni-Vincentelli. A framework for comparing models of computation. *IEEE Transactions on CAD of Integrated Circuits and Systems*, 17(12):1217–1229, 1998.
9. S.-Y. Lee, F. Mallet, and R. de Simone. Dealing with AADL end-to-end flow latency with UML Marte. In *ICECCS*, pages 228–233. IEEE Comput. Soc., Los Alamitos, 2008.
10. G. Leen and D. Heffernan. TTCAN: a new time-triggered controller area network. *Microprocessors and Microsystems*, 26(2):77–94, 2002.
11. F. Mallet, C. André, and R. de Simone. CCSL: specifying clock constraints with UML Marte. *ISSE*, 4(3):309–314, 2008.
12. SAE. Architecture analysis and design language (AADL). AS5506/1, 2006. <http://www.sae.org>.
13. F. Singhoff and A. Plantec. AADL modeling and analysis of hierarchical schedulers. In A. Srivastava and L.C. Baird III, editors, *SIGAda*, pages 41–50. Assoc. Comput. Mach., New York, 2007.
14. The ProMARTE Consortium. UML profile for MARTE, beta 2. OMG document number: ptc/08-06-08, Object Management Group, 2008.
15. T. Weikiens. *Systems Engineering with SysML/UML: Modeling, Analysis, Design*. The MK/OMG Press, Burlington, 2008.

<http://www.springer.com/978-1-4020-9713-3>

Languages for Embedded Systems and their
Applications

Selected Contributions on Specification, Design, and
Verification from FDL'08

Radetzki, M. (Ed.)

2009, XIV, 324 p., Hardcover

ISBN: 978-1-4020-9713-3