Marte CCSL and East-ADL2 Timing Requirements

Frédéric Mallet, Marie-Agnès Peraldi-Frati, Charles André

INRIA

Marte CCSL and East-ADL2 Timing Requirements

Frédéric Mallet — Marie-Agnès Peraldi-Frati — Charles André

N° 6781

Thème COM

Rapport de recherche

Marte CCSL and East-ADL2 Timing Requirements

Frédéric Mallet* , Marie-Agnès Peraldi-Frati* , Charles André*

Abstract: In the automotive domain, several loosely-coupled Architecture Description Languages (ADLs) compete to provide a set of abstract modeling and analysis services on top of the implementation code. In an effort to make all these languages, and more importantly their underlying models, interoperable, we use the UML Profile for MARTE as a pivot to define the semantics of these models.

In this paper, we particularly focus on East-ADL2. We discuss the benefits of having an integrated, MARTE-centered, approach. We give a formal semantics of East-ADL2 timing requirements. Relying on this semantics, several kinds of analysis become possible. Requirements become executable and simulations are run. A constraint solver is used to detect logical inconsistencies. Our proposal is illustrated on an Anti-lock Braking System (ABS).

Key-words: MARTE, East-ADL2, Executable specifications, timing requirements

Marte pour les exigences temporelles de East-ADL2

Résumé : In the automotive domain, several loosely-coupled Architecture Description Languages (ADLs) compete to provide a set of abstract modeling and analysis services on top of the implementation code. In an effort to make all these languages, and more importantly their underlying models, interoperable, we use the UML Profile for MARTE as a pivot to define the semantics of these models.

In this paper, we particularly focus on East-ADL2. We discuss the benefits of having an integrated, MARTE-centered, approach. We give a formal semantics of East-ADL2 timing requirements. Relying on this semantics, several kinds of analysis become possible. Requirements become executable and simulations are run. A constraint solver is used to detect logical inconsistencies. Our proposal is illustrated on an Anti-lock Braking System (ABS).

Mots-clés : MARTE, East-ADL2, Spécifications exécutables, exigences temporelles

1 Introduction

Architecture Description Languages (ADLs) are more and more accepted as means to manage the engineering information related to automotive electronics and deal with the increasing complexity of the automotive software [8]. Several loosely-related ADLs are competing in that area (AADL [11], East-ADL [4], SysML [16, 9], AML [3]). Some propose connections with the emerging standard AutoSar (*http://www.autosar.org*), rely on it for the implementation and build on top some requirement and tracability facilities. Others provide analysis models and tools. However, the ever increasing cost of software in domains like automotive calls for a single framework that would bring together all tools and models and interpret them consistently. Building on the numerous existing, close to maturity, UML (Unified Modeling Language) tools seems like a good way to avoid building and maintaining separate domain-specific graphical editors when graphical aspects should be managed once and for all.

UML is becoming more and more popular for the design and modeling of software systems. Being a general-purpose language, it lacks key features to model software of the real-time and embedded (RTE) domain. The recently adopted UML Profile for Modeling and Analysis of Real-Time and Embedded systems (Marte) [15] aims at bringing in the missing features. The goal has never been for Marte to become the unified (universal) language for modeling RTE systems but rather to be a *pivot* that brings interoperability between the existing languages and formalisms of the RTE domain. The belief is that one single language or model of computation would never be able to cover all aspects for RTE systems whatever good it is for a given specific aspect (schedulability, dependability, requirement ... analyses).

In a continuous effort, we have been trying to show how to use Marte and its time model [1] to define formally the semantics of existing models of the RTE domain. Following some similar work on AADL [2], this paper focuses on East-ADL and explains how to express the semantics of East-ADL timing requirements in Marte. Based on this semantics, our constraint solver can detect inconsistencies and execute the timing requirements when there is no constraint violations. Giving a formal semantics to automotive-related models to make the specifications open for direct analysis resemble other work on AutoSar [10] and AADL [12]. However, instead of defining another specific time model, we reuse Marte time model in a consistent way so that different models and tools can be compared and can ultimately interoperate. Another direct benefit of using Marte, a UML Profile, is that any UML-compliant tool can be used to edit, store, version and maintain all required models of systems under study, from requirements and system specification to implementation models.

We start with a brief overview of East-ADL capabilities in Section 2. Then Section 3 introduces Marte time model. Follows our contribution in Section 4 where Marte is used to define the semantics of East-ADL timing requirements. Finally, Section 5 discusses the benefits of our proposal and possible modeling improvements. *Anti-lock Braking System* (ABS) is used as a running example.

2   Overview of East-ADL2

EAST-ADL (Electronic Architecture and Software Tools, Architecture Description Language) has been initially developed in the context of the East-EEA European project [14]. To integrate proposals from the emerging standard AUTOSAR and from other requirement formalisms like SYSML, a new release called EAST-ADL2 [4, 13] has been proposed by the ATESST project. We only consider this new release even though we abusively use the name EAST-ADL. This domain specific modeling language provides a UML-based notation for the development of complex automotive electronic embedded systems.

2.1   Structural modeling

The function modeling in EAST-ADL is performed at different abstraction levels. The FunctionalArchitecture provides the functional decomposition of an automotive system. Analysis and design levels contain model elements such as ADLFunctionType and ADLFunctionPrototype for modeling function parts of the system. Sensors and actuators are modeled with FunctionalDevices (analysis level) or with LocalDeviceManagers (design level).

These model elements are structural constructs. The ADLFunctionType and ADLFunctionPrototype give the hierarchical decomposition of the system. A trigger attached to the elementary ADLFunctionType components specifies the execution conditions (triggerCondition) and the period (Triggerperiod) of an elementary ADLFunctionType. ADLFlowPorts and ADLConnectorType interconnect the structural entities. An attribute period is attached to input ports.

2.2   Timing Requirements

EAST-ADL requirements extend the requirement concept of SYSML. Requirements express conditions that must be met by the system. They are usually defined to enrich the functional architecture of an automotive system with extra-functional characteristics such as variability, temporal behavior etc. In this paper, we focus on timing requirements. Figure 1 shows a partial view of EAST-ADL timing requirements meta-model, which exhibits three main concepts.

- A DelayRequirement constrains the delay "from" a set of entities "until" another set of entities. It specifies the temporal distance between the execution of the earliest "from" entity and the latest "until" entity.

- The RepetitionRate defines the inter-arrival time of data on a port or the triggering period of an elementary ADLfunction.

- The Input/outputSynchronization expresses a timing requirement on the input/output synchronization among the set of ports of an ADLFunction. It should be used to express the maximun temporal skew allowed between input or ouput events or data of an ADLfunction.
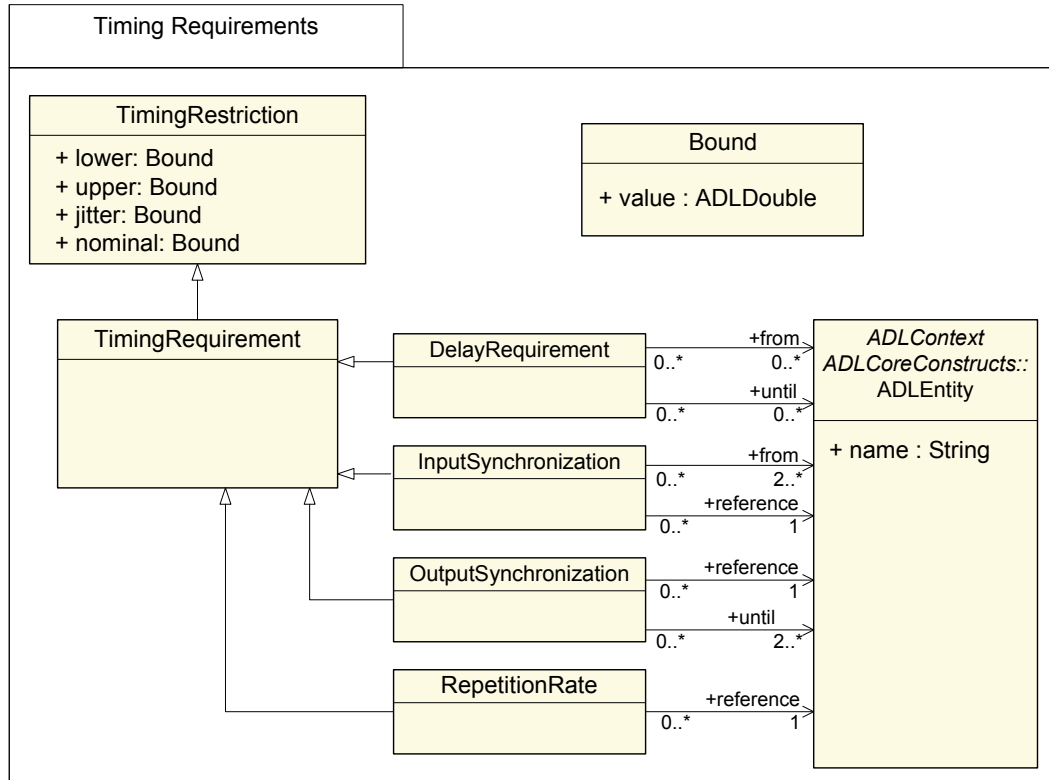
Figure 1: East-ADL timing requirements

Timing requirements specialize the meta-class TimingRestriction, which defines bounds on system timing attributes. The timing restriction can be specified as a *nominal* value, with or without a *jitter*, and can have *lower* and *upper* bounds. The jitter is the maximal positive or negative variation from the nominal value. A bound is a real value associated with an implicit time unit (ms, s, . . .).

2.3   Example

As an illustration, we use the example of an Anti-lock Braking System (ABS). This example and the associated timing requirements are taken from the ATESST report on EAST-ADL timing model [7]. The functional architecture model is shown in Figure 2. It is described in Papyrus (http://www.papyrusuml.org) with the UML profile for EAST-ADL. The ABS architecture consists of four sensors, four actuators and an indicator of the vehicle speed. The sensors measure the rotation speed of the vehicle wheels: i_{fl} (front left), i_{fr} (front
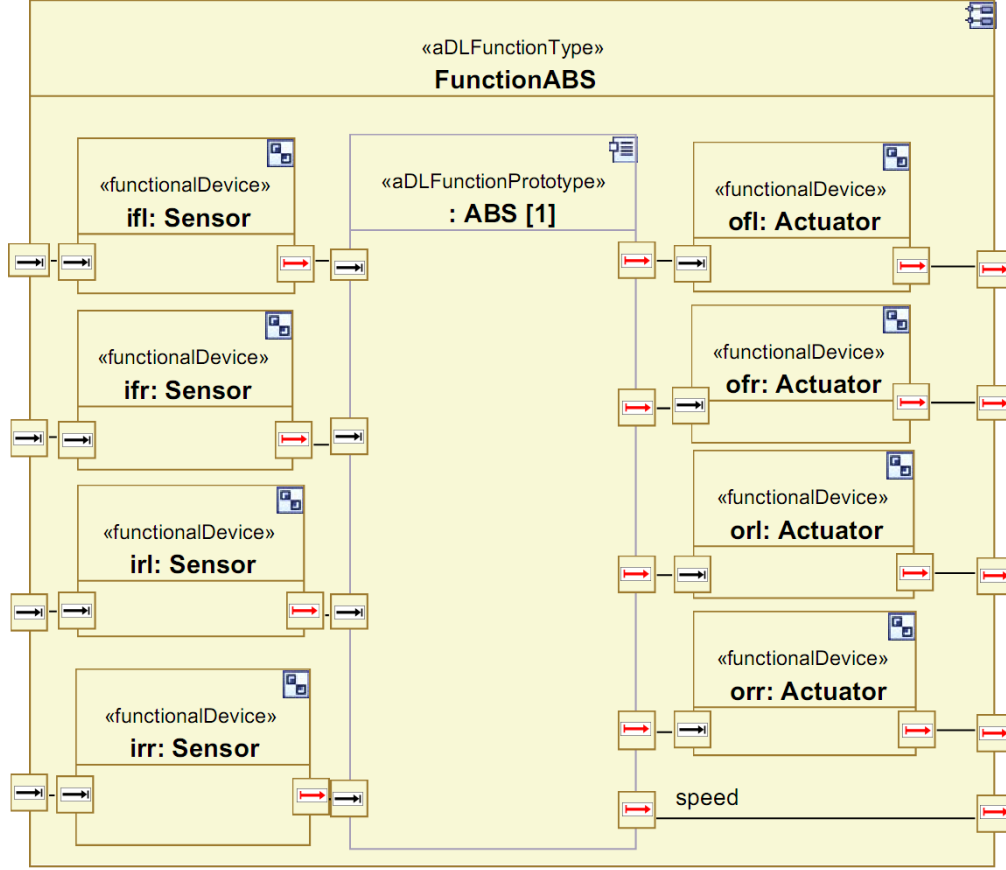
Figure 2: ABS system in East-ADL2

right), i_{rl} (rear left) and i_{rr} (rear right). The actuators indicate the brake pressure to be applied on the wheels (o_{fl}, o_{fr}, o_{rr} and o_{rl}). The FunctionalArchitecture is composed of FunctionalDevices for sensors and actuators and a ADLFunctionType for the functional part of the ABS. An ADLOutFlowPort provides the vehicle speed (speed).

The essential timing parameters for capturing the timing model of the ABS are represented in Figure 3. This figure is adapted from the ATESST report to remove ambiguities discussed in Section 5.

The execution of the ABS is triggered at the time R. L_s parameter measures the latency of sensor sampling. The values of the four sensors involved in the ABS computing must arrive on the input ADLFlowPorts within the delay J_{ii} (InputSynchronization). A similar OuputSynchronization delay J_{oo} is represented on the output interface side. The L_{io} represents the delay from the first event on the input set of the ABS until the last event

occurrence on the output set. The L_{ispeed} stands for the delay between the first event on the input set until the speed output occurrence. The sampling interval of the sensor is given by the H parameter. These parameters are modeled by timing requirements characterized by timing values or intervals with jitters. Values applied for the timing requirements are summarized in Table 1.
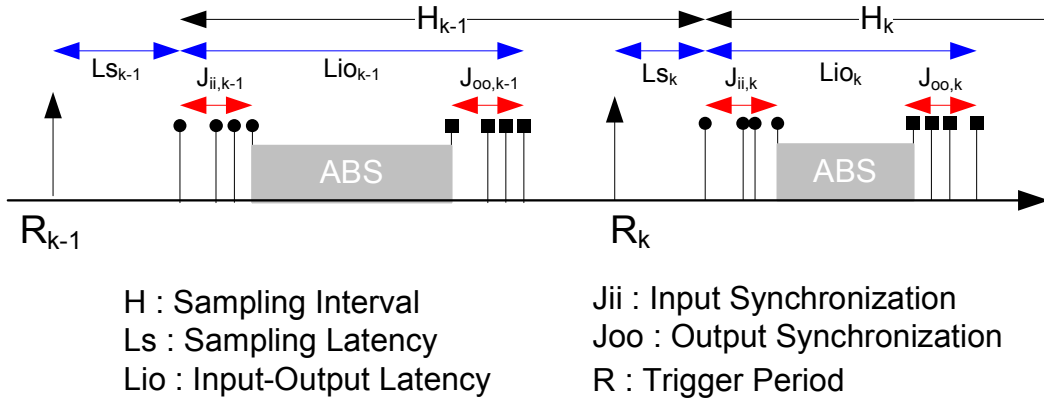

H : Sampling Interval   Jii : Input Synchronization
Ls : Sampling Latency   Joo : Output Synchronization
Lio : Input-Output Latency   R : Trigger Period

Figure 3: Timing model of the ABS

Parameter	nominal	upper	lower	jitter
R	5 ms	-	-	1 ms
L_s	-	3 ms	-	2 ms
J_{oo}	-	0.5 ms	-	-
J_{ii}	-	0.5 ms	-	-
L_{io}	-	5 ms	-	2 ms
L_{ispeed}	-	5 ms	-	3 ms

Table 1: Timing values

3 MARTE Time Model

3.1 Time Structure

MARTE defines a broadly expressive *Time Model* that provides for a generic timed interpretation of UML models. In MARTE, time can be explicitly associated with UML model elements like ValueSpecification, Constraint, Event, Behavior . . . Usage examples of these timed concepts

are available in a paper on MARTE applicability to complex real-time control systems [5]. In this presentation we focus on *clocks* and *clock constraints*.

MARTE time model deals with both *discrete* and *dense* time. A *clock* gives access to a *time structure*. A clock can be either *chronometric* or *logical*. The former implicitly refers to "physical time", whereas the latter does not. Logical clocks, which focus on the ordering of instants, *may* ignore the physical duration between instants, but this does not preclude quantitative information attached to (logical) clock instants. In MARTE, a discrete logical clock can be associated with any UML event. This clock ticks whenever the event occurs. Such clocks are useful in control applications. MARTE also allows *multiform* time modeling. This concept is inherited from synchronous languages: time observations (through clocks) can rely on different referentials. Thus, a timed model in MARTE relies on many clocks, which are generally interdependent. Interdependency results from relationships existing between instants i and j from different clocks: precedence ($i \prec j$), coincidence ($i \equiv j$) or exclusion ($i\#j$). The reader interested in the underlying mathematics may refer to a previously published paper [1]. Since instant relations are defined on pairs of instants, they are obviously not suitable to specify complex time structures (*i.e.*, complex clock dependencies). Instead, we rather use constraints on clocks (clock relations): a clock relation specifies many—usually infinitely many—instant relations. CCSL is a language to express these relations. It is briefly presented in the next subsection.

3.2   CCSL

CCSL (Clock Constraint Specification Language) is a non normative language annexed to MARTE specification. It is a declarative language that specifies *constraints* imposed on the clocks (activation conditions) of a model. These constraints can be classified into four categories: *synchronous*, *asynchronous*, *mixed*, and *non-functional*.

Synchronous clock constraints   rely on *coincidence*. *Subclocking* is such a constraint: each instant of the *subclock* must coincide with one instant of the *superclock*. Of course, the mapping must be order-preserving. To model the example (see Section 4) we need two other synchronous constraints: discretizedBy and isPeriodicOn. The former discretizes a dense clock. It is mainly used to derive a discrete chronometric clock from *IdealClk*. *IdealClk* is a dense chronometric clock, predefined in MARTE Time Library, and supposed to follow "physical time" faithfully. For instance:

$$\text{Clock } c_{10} = IdealClk \text{ discretizedBy } 0.0001 \qquad (1)$$

Eq. 1 specifies that c_{10} is a discrete chronometric clock whose period is 0.0001 s, where s is the time unit associated with *IdealClk*, therefore c_{10} is a 10 kHz clock.

$$\text{Clock } c_1 \text{ isPeriodicOn } c_{10} \text{ period } 10 \qquad (2)$$

Eqs. 2 reads that there is a tick of c_1 every 10th ticks of c_{10} (*i.e.*, c_1 is a 1 kHz clock). More precisely,

$$\forall k \in \mathbb{N}^\star, c_1[k] \equiv c_{10}[10(k-1)+1]$$

Asynchronous clock constraints   are based on *precedence*, which may appear in a *strict* ($\prec$) or a *non-strict* ($\preccurlyeq$) form.

The clock constraint "a isFasterThan b" (symbolically denoted by $a \boxed{\preccurlyeq} b$) specifies that a is (non-strictly) *faster than* b, that is for all natural number k, the kth instant of a precedes or is coincident with the kth instant of b ($\forall k \in \mathbb{N}, a[k] \preccurlyeq b[k]$). "$b$ isSlowerThan a" is equivalent to "a isFasterThan b".

Alternation is a form of mutual precedence. "a alternatesWith b" (symbolically denoted by $a \boxed{\sim} b$) states that $\forall k \in \mathbb{N}^{\star}, a[k] \prec b[k] \prec a[k+1]$.

Mixed clock constraints   combine coincidence and precedence. Given 2 clocks a and b, "$c = \mathsf{inf}(a, b)$" is the slowest clock among all clocks faster than both a and b
($\forall k \in \mathbb{N}^{\star}, c[k] \equiv$ if $a[k] \preccurlyeq b[k]$ then $a[k]$ else $b[k]$).
Similarly, "$d = \mathsf{sup}(a, b)$" is the fastest clock among all clocks slower than both a and b $\left(\forall k \in \mathbb{N}, d[k] \equiv$ if $a[k] \preccurlyeq b[k]$ then $b[k]$ else $a[k] \right)$. Most of the time, inf and sup clocks are neither a nor b. inf and sup are easily extended to sets of clocks.

Another mixed clock constraint enforces delayed coincidences. "$c = a$ delayedFor n on b" imposes c to tick synchronously with the nth tick of b following a tick of a. It is considered as a mixed constraint since a and b are not assumed to be synchronous.

Non Functional Property   constraints apply to chronometric clocks. While *IdealClk* is supposed to be perfect, an actual clock may have flaws. For instance, its period may not be strictly constant with respect to *IdealClk*. CCSL introduces special constraints to specify *stability*, *drift*, *offset*...of chronometric clocks. Examples of stability are given in Section 4. Here is an example of offset:

$$\text{Clock } c_2 = IdealClk \text{ discretizedBy } 0.0001 \tag{3}$$

$$c_{10}, c_2 \text{ haveOffset } 0.00002 \tag{4}$$

c_2 (Eq. 3) and c_{10} (Eq. 1) are both 10 kHz clocks, but they may be out of phase. Eq. 4 says that the offset between the two clocks is less than 0.00002 s = 20 us.

Stochastic parameters   are available in CCSL. Nondetermism introduced by such parameters may reflect a partial knowledge about the actual constraints. It may also be a deliberate choice for hiding unnecessary details. Several probability distributions are provided. The *uniform* distribution is often used to represent a tolerance interval on a duration. Examples are given in Section 4.

3.3   TimeSquare

TimeSquare is the software environment we propose to deal with Marte time model and CCSL. TimeSquare is an Eclipse plugin that has four main functionalities: 1) interactive

clock-related specifications, 2) clock constraint checking, 3) generation of a solution, 4) displaying and exploring waveforms.

TimeSquare has been designed to be used with UML tools applying Marte profile. In this profile, clocks and clock constraints can be associated with many and various model elements. A wizard is included in TimeSquare. It facilitates clock definitions, clock constraint specifications, model element browsing, and parameter setting.

The second functionality checks constraint sanity and is called when the above mentioned wizard is not used.

The third functionality relies on a *constraint solver* that yields a satisfying execution trace or issues an error message in case of inconsistency. The traces are given as waveforms written in VCD format. VCD (Value Change Dump) [6] is an IEEE standard textual format for dumpfiles used by EDA logic simulation tools. The solver intensively uses Binary Decision Diagrams (BDD) to manipulate boolean equations induced by CCSL clock constraints.

Waveforms can be displayed with any VCD viewer. TimeSquare has its own viewer enriched with interactive constraint highlighting and access facilities.

4   Modeling East-ADL requirements in CCSL

4.1   Clocks and events

The term clock used in CCSL may be misleading and deserves to be further discussed. Every event on which we want to attach time constraints can be associated with a clock. Event is taken here in the very broad sense (as in UML) to denote something that happens (the start of an action, the receipt of a message, ...). In that case, CCSL clocks represent the set of instants at which the concerned event occurs. CCSL clock constraints are relations amongst instants of the related clocks.

The first action to be undertaken is to select the events in the model on which clock relations should be applied. In this section, we give systematic rules to identify, for a given kind of timing requirement expressed in East-ADL, the events and the clock relations to apply.

Overall there are two kinds of relations supported by East-ADL timing requirements. Either, a timing requirement specifies a temporal relation between two successive occurrences of a single event (*i.e.,* two successive instants of the same clock), or between occurrences of different events. In the latter case, it is mostly between events that occur at the same rate and relations mostly concern the ith occurrence of one event and the ith occurrences of other events. Additionally, all requirements involve physical time but most also induce logical relations. A part of our contribution is to make explicit these logical relations by applying the adequate CCSL constraints. This section shows how to express in CCSL the example described in Section 2.

4.2 Repetition rate

A RepetitionRate concerns successive occurrences of the same event (data arriving to or departing from a port, triggering of a function). In all cases, it consists in giving a nominal duration between two successive occurrences/instants of the same event/clock. When the duration is specified in terms of number of occurrences of another event (for instance, number of clock cycles), CCSL relation isPeriodicOn must be used. When the duration is given in seconds (time unit s), then relation discretizedBy must be used. The two relations can also be combined to give both logical and physical constraints. Eq. 2 (Section 3.2) builds a discrete, 1 kHz chronometric clock c_1. Eq. 5 uses c_1 to specify a 5 ms repetition rate for function f. The latter defines $f.start$ as being a subclock of c_1 five times less frequent.

$$f.start \text{ isPeriodicOn } c_1 \text{ period } 5 \tag{5}$$

$$f.start \text{ hasStability } 1E{-}3 \tag{6}$$

$f.start$ is a clock associated with the beginning of function f. MARTE stereotype Timed-Processing enables the association of a clock with the start event of a given behavior.

If a jitter is associated with the period, CCSL relation hasStability should be used. Eq. 6 refines Eq. 5 and states that the jitter on the nominal 5 ms period is 1 ms. Eq. 5 combined with Eq. 6 are CCSL equivalent of EAST-ADL repetition rate and that is the way to implement the first row of Table 1 for requirement R.

It may also happen that the period assigned is not just a nominal value, but a time interval. In that case, a CCSL precedence-based relation like isFasterThan (denoted $\boxed{\preccurlyeq}$) can be used to specify the minimum and maximum bounds within which a clock must tick. Eqs. 7 state that the repetition rate of f is between 4 ms and 6 ms.

$$\text{Clock } f_{lower}, f_{upper}$$
$$f_{lower} \text{ isPeriodicOn } c_1 \text{ period } 4$$
$$f_{upper} \text{ isPeriodicOn } c_1 \text{ period } 6 \tag{7}$$
$$f_{lower} \boxed{\preccurlyeq} f.start \boxed{\preccurlyeq} f_{upper}$$

4.3 Delay requirements

A DelayRequirement constrains the delay between a set of inputs and a set of outputs. Input (resp. output) synchronizations are a specialization without outputs (resp. inputs). At each iteration, all inputs and outputs must occur. So for each iteration i, it is a matter of constraining the temporal distance between the ith occurrences of two events (occurring on inputs or outputs).

With general delay requirements, the delay applies between the earliest input and the latest output. With input synchronizations, it concerns the distance from the earliest input to the latest input. With output synchronizations, it is about the earliest and the latest outputs. Note that the earliest/latest input/output does not necessarily concern the same event at each iteration. CCSL relation inf builds a new clock so that its ith instant precedes

all i^{th} instants for a given set of clocks. CCSL relation sup builds a new clock so that its ith instant follows all i^{th} instants for a given set of clocks.

In our example, function abs has four inputs ($i_{fl}, i_{fr}, i_{rl}, i_{rr}$), one for each wheel. There also are four outputs to control the breaking systems ($o_{fl}, o_{fr}, o_{rl}, o_{rr}$), and one output for the vehicle speed (o_{speed}). Eqs. 8–11 specify the *sup* and *inf* for the inputs and outputs.

$$\text{Clock } i_{inf} = \inf(i_{fl}, i_{fr}, i_{rl}, i_{rr}); \tag{8}$$

$$\text{Clock } i_{sup} = \sup(i_{fl}, i_{fr}, i_{rl}, i_{rr}); \tag{9}$$

$$\text{Clock } o_{inf} = \inf(o_{fl}, o_{fr}, o_{rl}, o_{rr}); \tag{10}$$

$$\text{Clock } o_{sup} = \sup(o_{fl}, o_{fr}, o_{rl}, o_{rr}); \tag{11}$$

CCSL operator delayedFor builds from an initial clock a delayed clock for a given duration. Combining delayedFor with isFasterThan allows for specifying distances between two clocks. Here again, distance are *a priori* expressed in number of ticks of a reference clock (not necessarily a super clock). If the reference clock is chronometric, then the duration directly refers to physical time. When referring to IdealClk, duration are expressed in seconds. Eq. 12 denotes a delay requirement and states that the maximum end-to-end latency of the function abs is 3 ms. It characterizes the requirement L_{io} (see Table 1 in Section 2). Eq. 13 denotes an input synchronization of 0.5 ms (requirement J_{ii}) and Eq. 14 denotes an output synchronization (requirement J_{oo}). Note that these last two equations refer to clock c_{10} (a 10 khz discrete chronometric clock defined in Eq. 1, Section 3.2) instead of c_1 as in previous equations.

$$o_{sup} \quad \boxed{\preccurlyeq} \quad (i_{inf} \text{ delayedFor } 3 \text{ on } c_1) \tag{12}$$

$$i_{sup} \quad \boxed{\preccurlyeq} \quad (i_{inf} \text{ delayedFor } 5 \text{ on } c_{10}) \tag{13}$$

$$o_{sup} \quad \boxed{\preccurlyeq} \quad (o_{inf} \text{ delayedFor } 5 \text{ on } c_{10}) \tag{14}$$

When the latency is not a nominal value, similar mechanisms as the one explained in the previous subsection can be exploited to define lower or upper bounds, jitters.

4.4   Causal relationships

An ADLFunctionType like the one introduced in Figure 2 implies a data flow execution, even though the actual notation is similar to UML structural models. In this particular example, the flow goes from sensors to actuators and is repeated infinitely or at least until someone decides to shut the system down. This causal flow can be modeled in CCSL with the precedence-based relation alternatesWith ($\boxed{\sim}$). For instance, Eqs. 15-17 state that each release must be followed by one and only one capture of each input, one execution of the function *abs* and one output emission and all these actions must be performed before the next release (no pipeline is allowed here). Eq. 18 states that all inputs must occur before

any output is emitted and all outputs must be delivered before the next iteration.

$$R \boxed{\sim} i_{inf} \qquad\qquad R \boxed{\sim} i_{sup} \qquad\qquad (15)$$
$$R \boxed{\sim} abs \qquad\qquad\qquad\qquad\qquad\qquad (16)$$
$$R \boxed{\sim} o_{inf} \qquad\qquad R \boxed{\sim} o_{sup} \qquad\qquad (17)$$
$$i_{sup} \boxed{\sim} o_{inf} \qquad\qquad\qquad\qquad\qquad\qquad (18)$$

5 Discussion and results

5.1 ABS as a Time structure

The previous section has shown how to express the timing requirements of the ABS systems in a *declarative* way using CCSL. This specification can be used as a golden model to validate existing realizations of the system. With TIMESQUARE, we may go a step further towards *executable specifications*. Starting with timing requirements expressed in CCSL, we build a refined model of the timed behavior of the ABS system. Refining a constraint may consist in fixing some parameter values, in accordance with the initial specification. Then, TIMESQUARE simulates the refined model, possibly revealing inconsistency in the specification.

The Anti-Lock Braking system, the functional structure of which is described in Figure 2, can be modeled as a periodic *data flow* system. Since periods between successive occurrences of the same event can slightly vary (*jitter*), the system is not strictly periodic. In such models some processes produce data (or *tokens*) consumed by others whose executions are triggered by the receipt of their incoming data. MARTE clocks and clock constraints can easily capture such a behavior. We use *Timers* for modeling communication and processing. Timers rely on constraint delayedFor. For a communication, the delay is the communication duration, the sender (token producer) is the clock to be delayed (the trigger) and the receiver (token consumer) is the delayed clock. For a processing, the trigger represents the processing start, the delayed clock is the processing end, and the parameter the execution time. Often communication and processing are taken as a whole, and in this case the parameter stands for the *latency*. For instance, R triggers sensor sampling. In Eq. 19, R causes the availability of a sample (clock i) with a latency L_s. Note that L_s is a stochastic delay defined here by a uniform distribution Uniform(10..30). Since the delay is measured on clock c_{10} (10 kHz), Eq. 19 states that $1\ ms \le L_s \le 3\ ms$, in accordance with the specification given in Table 1.

$$i = R \text{ delayedFor } L_s \text{ on } c_{10} \qquad\qquad (19)$$

Clock i is not part of the ABS specification. We have introduced this auxiliary clock to respect faithfully Figure 3 borrowed from the ATESST example. Since there are 4 values to acquire, instead of one clock (i), we had to consider 4 clocks $(i_{fl}, i_{fr}, i_{rl}, i_{rr})$ each of them being delayed from i for a random duration Uniform(0..5) on c_{10} (*i.e.*, $0 \le delay \le 0.5\ ms$). This gives rise to the *input synchronization jitter* J_{ii} forced to be less than $0.5\ ms$ in Table 1.
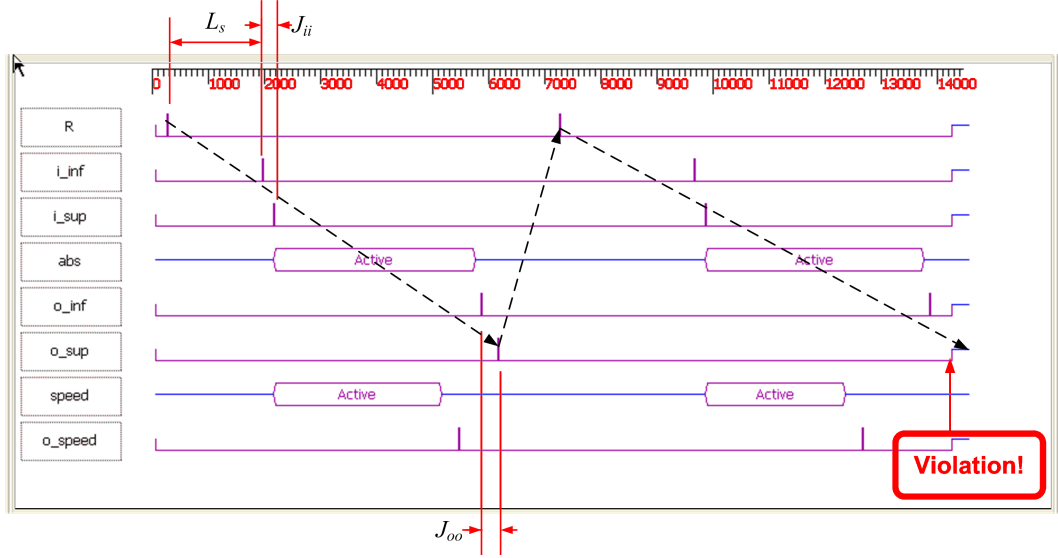
Figure 4: TimeSquare ABS simulation

As explained in Section 4.3, this jitter can be characterized by the two clocks i_{inf} (Eq. 8) and i_{sup} (Eq. 9). Nevertheless, such an implementation guarantees by construction the upper bound requirement for J_{ii}.

Besides system behavior modeling, CCSL can also be used to express properties. Eqs. 12–14 declare maximal response time. Eqs. 15–18 are other forms of (logical) deadline: an event a must occur once and only once before an event b. TimeSquare solver detects possible property violations.

5.2 Simulation

Figure 4 shows a screen copy of the ABS simulation, augmented with indications on timing parameters. Since chronometric clocks are used in the model, the top ruler indicates physical time and the unit is us (microsecond) here.

TimeSquare allows the user to hide traces (here all auxiliary clocks and timers). In the waveforms, clock instants are shown as pulses whereas a timer is depicted as a hexagon stretching over its active phase. TimeSquare can also represent clock constraints by special annotations: coincidence by a solid vertical line between coincident instants, precedence by black oblique dashed arrows (*e.g.,* the alternation between R and o_{sup} in Figure 4).

The current version of TimeSquare generates a sequence of steps that satisfies the set of constraints. An inconsistent specification can lead to a *deadlock*: after a finite sequence of steps, the simulation gets at a point where all the clocks are disabled (not allowed to tick). This is the case in Figure 4. Here, the cause of the deadlock is the violation of an
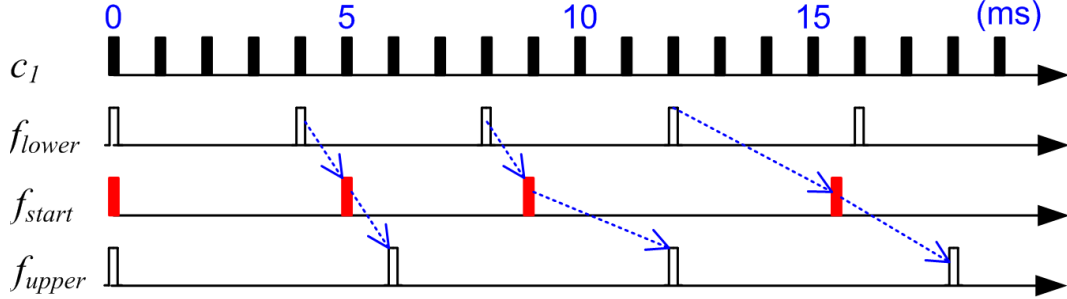
Figure 5: Specification lower–upper

alternation: R tries to occur while o_{sup} has not occurred yet. R being the (indirect) cause of all other events, the simulation cannot proceed on. This diagnostic is facilitated by the generation of an anomaly report file that contains the history upto the deadlock and the state of all clocks at the faulty step. Raising the period of R from 7 ms to 8 ms allows o_{sup} to occur in time and the simulation runs without abortion. Of course, this is not a proof of correctness. A simulation is conclusive only when it reveals a counter-example. Note that the initial specification (Table 1) allowed only 5 ms to R period, which was certainly insufficient and surely rejected by the simulator.

An inconsistent specification may also lead to a "*livelock*": a subset of clocks are mutually exclusive so that they cannot tick any more. This behavior is easily spotted on the simulation display by the persistent absence of pulses for these clocks. This is only a *presumption* of livelock, that should be confirmed or invalidated by inspection of the simulation trace.

5.3 Feedback on East-ADL specifications

The AUTOSAR community has screened the Time model proposed by the ATESST project and identified a number of delicate issues. Our approach meets these conclusions and can be considered as a proposal to solve some of the issues.

We have found it pretty hard to understand how attributes nominal, jitter, upper and lower combine themselves in EAST-ADL. Indeed, it seems that most of the time combining two among four should be enough to specify an interval. However, EAST-ADL specification gives example where the four attributes are used and where the upper bounds seems to be altered by the value of the jitter itself. There are also cases where a jitter (*e.g.,* a sampling latency jitter) is expressed as a timing requirement itself associated with the four attributes (including a jitter). In our model we can express both lower/upper and nominal/jitter. However, our upper value is an actual upper bound and when combined with a jitter, it only acts on the lower bound. In Section 4, Eqs. 7 defines a possible interval for the repetition rate and the expected result is illustrated by Figure 5. This specification can result in an unbounded jitter, which is surely not the expected behavior. Instead, we propose to fix the
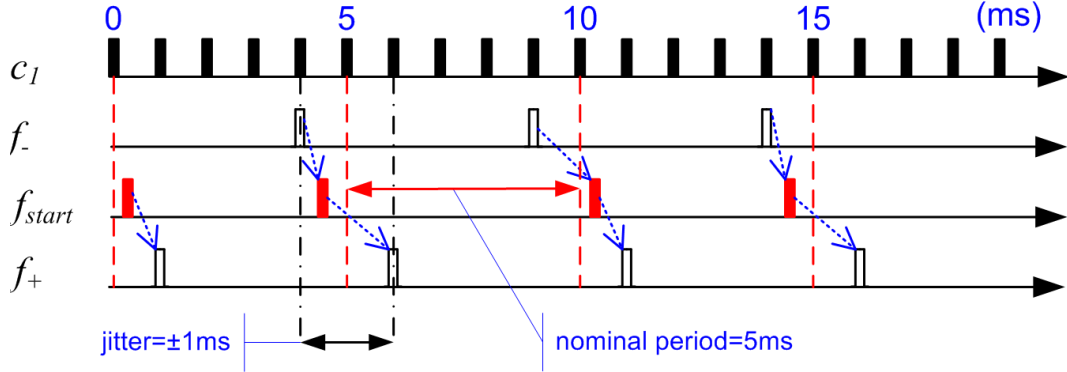
Figure 6: Specification nominal–jitter

nominal value and the jitter, leading to waveforms in Figure 6. The jitter is specified either with constraint `hasStability` as in Eq. 6 or with a constraint `isPeriodicOn` where the actual period is expressed with a probability distribution (see Eq. 20). This ability of CCSL to mix constant with distributed values is directly inspired from SYSML distributed properties.

$$f.start \text{ isPeriodicOn } c_1 \text{ period Uniform}(4..6) \qquad (20)$$

6 Conclusion

In this paper, we have expressed the semantics of EAST-ADL timing requirements using MARTE CCSL. Making the semantics formal allows for an automatic detection of logical inconsistencies. Then, requirements have been refined into an executable specification to run simulations and explore the dynamic properties of acceptable executions. We also give examples of ambiguities in EAST-ADL specification and explain choices made in our implementation.

Even though the simulation can detect some errors in the specification, it should be combined with exhaustive formal analyses. For instance, livelock detection can be automated with model-checking approaches, provided that the state-space is finite. TIMESQUARE already generates the state-space at each step. For now, our default simulation policy consists in randomly selecting a minimum acceptable firing sequence from the set of possible sequences. Other policies must be implemented and combined with validation of properties.

References

[1] C. André, F. Mallet, and R. de Simone. Modeling time(s). In G. Engels, B. Opdyke, D. C. Schmidt, and F. Weil, editors, *MoDELS*, volume 4735 of *Lecture Notes in Computer Science*,

pages 559–573. Springer, 2007.

[2] C. André, F. Mallet, and R. de Simone. *Modeling of AADL data-communications with UML Marte*, volume 10 of *LNEE*, chapter 11, pages 150–170. Springer, May 2008.

[3] P. Braun and M. Rappl. A model-based approach for automotive software development. In P. P. Hofmann and A. Schürr, editors, *OMER*, volume 5 of *LNI*, pages 100–105. GI, 2001.

[4] P. Cuenot, D. Chen, S. Gérard, H. Lönn, M.-O. Reiser, D. Servat, C.-J. Sjostedt, R. T. Kolagari, M. Torngren, and M. Weber. Managing complexity of automotive electronics using the East-ADL. In *ICECCS '07: Proceedings of the 12th IEEE International Conference on Engineering Complex Computer Systems (ICECCS 2007)*, pages 353–358, Washington, DC, USA, 2007. IEEE Computer Society.

[5] S. Demathieu, F. Thomas, C. André, S. Gérard, and F. Terrier. First experiments using the uml profile for marte. In *ISORC*, pages 50–57. IEEE Computer Society, 2008.

[6] IEEE Standards Association. *IEEE Standard for Verilog Hardware Description Language*. Design Automation Standards Committee, 2005. IEEE Std 1364TM-2005.

[7] R. Johansson, H. Lönn, and P. Frey. ATESST timing model. Technical report, ITEA, 2008. Deliverable D2.1.3.

[8] H. Lönn and U. Freund. *Automotive Architecture Description Language*, chapter 9. CRC Press, December 2008.

[9] OMG. *Systems Modeling Language (SysML) Specification 1.1.* Object Management Group, May 2008. OMG document number: ptc/08-05-17.

[10] K. Richter. Defining a timing model for AUTOSAR - status and challenges. In W. Maalej and B. Brügge, editors, *Software Engineering (Workshops)*, volume 122 of *LNI*, pages 93–97. GI, 2008.

[11] SAE. *Architecture Analysis and Design Language*. Society of Automotive Engineers, June 2006. Document AS5506/1.

[12] O. Sokolsky, I. Lee, and D. Clarke. Schedulability analysis of AADL models. In *IPDPS*. IEEE, 2006.

[13] The ATESST Consortium. East-ADL2 specification. Technical report, ITEA, March 2008. http://www.atesst.org, 2008-03-20.

[14] The East-EEA Project. Definition of language for automotive embedded electronic architecture approach. Technical report, ITEA, 2004. Deliverable D.3.6.

[15] The ProMARTE Consortium. *UML Profile for MARTE, beta 2.* Object Management Group, June 2008. OMG document number: ptc/08-06-08.

[16] T. Weilkiens. *Systems Engineering with SysML/UML: Modeling, Analysis, Design.* The MK/OMG Press, Burlington, MA, USA., 2008.

Contents