Analysis Support for TADL2 Timing Constraints on EAST-ADL Models

Arda Goknil[1], Jagadish Suryadevara[2], Marie-Agnès Peraldi-Frati[1], Frédéric Mallet[1]

[1] AOSTE Project, UNS-I3S-INRIA, Sophia-Antipolis, France
[2] Mälardalen Real-Time Research Centre, Mälardalen University, Västerås, Sweden
arda.goknil@inria.fr, jagadish.suryadevara@mdh.se, map@unice.fr, frederic.mallet@unice.fr

Abstract. It is critical to analyze characteristics of real-time embedded systems such as timing behavior early in the development. In the automotive domain, EAST-ADL is a concrete example of the model-based approach for the architectural modeling of real-time systems. The Timing Augmented Description Language v.2 (TADL2) allows for the specification of timing constraints on top of EAST-ADL models. In this paper we propose a formal validation & verification methodology for timing behaviors given with TADL2. The formal semantics of the considered timing constraints is given as a mapping to the Clock Constraint Specification Language (CCSL), a formal language that implements the MARTE Time Model. Based on such a mapping the validation is carried out by the simulation of TADL2 specifications. The simulation allows for a rapid prototyping of TADL2 specifications. The verification is performed through a mapping to Timed Automata implemented by UPPAAL. The whole process is illustrated on a Brake-By-Wire application.

1 Introduction

Non-Functional properties and time are central concerns in real-time embedded systems. The increasing complexity of automotive systems requires the early identification of specification problems and the use of common/standard formalisms to cover all aspects of the systems. In the automotive domain, EAST-ADL [9] is a concrete example of the model-based approach for the architectural modeling of safety-critical embedded systems. EAST-ADL has been developed to provide a standard architecture description language aligned with Autosar [10]. The new release of EAST-ADL (v.2) has recently adopted the timing model proposed in the Timing Augmented Description Language (TADL) [19]. TADL allows for expressing and composing basic timing constraints such as repetition rates, end-to-end delays, or synchronization constraints.

The TIMMO-2-USE project [2] goes one step beyond TADL by recently introducing TADL2 [19]. The time model of TADL2 specializes the time model of the UML Profile for MARTE (Modeling and Analysis of Real-Time and Embedded systems) [18]. It elaborates on TADL and adds constructs borrowed from the MARTE companion language, the Clock Constraint Specification Language (CCSL) [8], a formal language dedicated to the specification of temporal and causality constraints. In particular, it adds new modelling capabilities such as the explicit notion of time base and the ability to use symbolic timing expressions in timing constraints.

In this paper, we propose a formal validation & verification methodology for timing behaviors given with TADL2. The validation is carried out by the simulation of TADL2 specifications, based on a mapping of TADL2 specifications to CCSL specifications. With such a mapping the simulation of TADL2 timing constraints becomes possible through TIMESQUARE [11], the framework dedicated to the analysis of CCSL specifications. This mapping gives a semantic reference interpretation for TADL2 constraints and the simulation with CCSL allows for a rapid prototyping of TADL2 specifications. Complementary to the simulation, the formal verification is performed by model checking, based on a mapping of TADL2 to Timed Automata for using the UPPAAL model-checker [17]. We use a Brake-By-Wire (BBW) application as a running example to illustrate and compare the results of those mappings.

The remainder of the paper is organized as follows. Section 2 introduces the BBW system used as a running example. Section 3 gives a brief overview of TADL2. Section 4 describes the mapping between TADL2 & CCSL and the analysis results provided by TIMESQUARE. In Section 5, we give the mapping between TADL2 and UPPAAL with the analysis results in the UPPAAL model-checker. Section 6 discusses the benefits of using both languages together. Section 7 discusses the related work.

2 Running Example: Brake-By-Wire Application

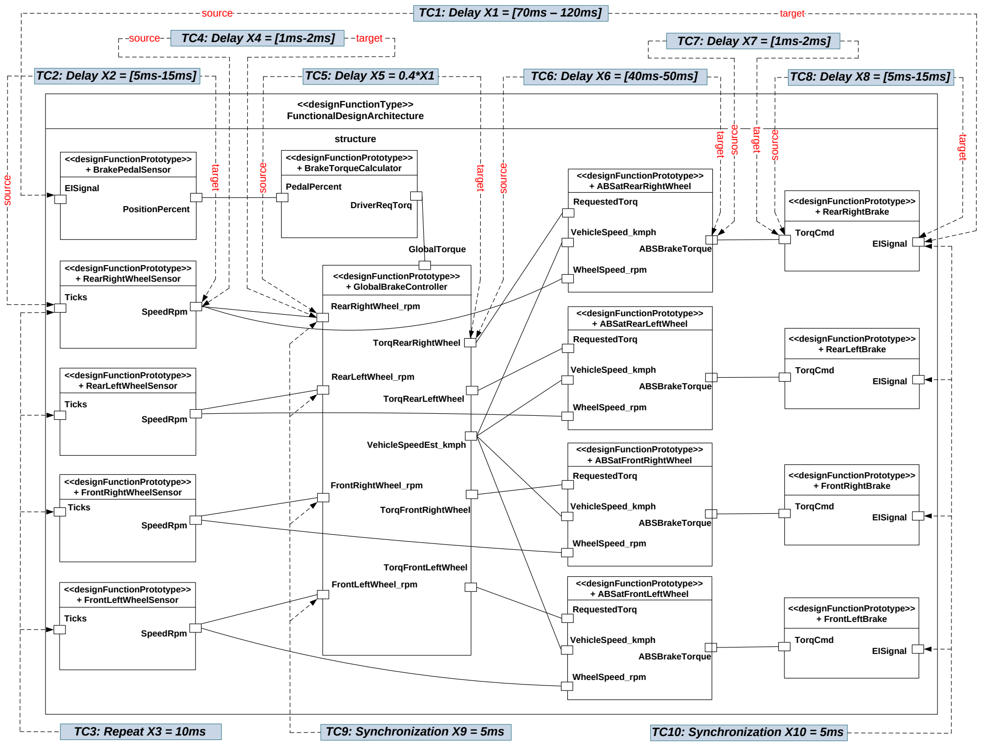
A distributed Brake-By-Wire (BBW) application with an anti-lock braking functionality illustrates our approach. The BBW application is one of the validator proposed by Volvo Technology in the TIMMO-2-USE project [2].

The structural decomposition of the braking functionality is shown in Figure 1. It gives the BBW functional design architecture in East-ADL: the parts denote subfunctions and the connectors represent data dependencies. The BBW is composed of two mains functions. First, the brake controller reads the wheel speed sensors and the brake pedal sensor. The brake controller computes the desired brake torque applied to the wheels. In addition to this basic brake controller functionality, a second function Anti-lock Braking System (ABS) adapts the brake force on each wheel if the speed of one wheel is significantly smaller than the estimated vehicle speed. The brake force is reduced on that wheel until it regains the speed that is comparable with the estimated vehicle speed. The braking functionality has the following components (Fig. 1):

- *BrakePedalSensor* (BPS) reads the pedal position percentage on port *EISignal*.
- *BrakeTorqueCalculator* (BTC) receives the pedal position percentage from BPS and computes the desired global torque.
- The wheel sensors—*RearRightWheelSensor* (RRWS), *RearLeftWheelSensor* (RLWS), *FrontRightWheelSensor* (FRWS) and *FrontLeftWheelSensor* (FLWS)—read the speed values for each wheel.
- *GlobalBrakeController* (GBC) receives the speed values measured by the wheel sensors and the global torque calculated by *BrakeTorqueCalculator*. It calculates the torque required for each wheel.

– The ABS components—*ABSatRearRightWheel* (ABSrrw), *ABSatRearLeftWheel* (ABSrlw), *ABSatFrontRightWheel* (ABSfrw) and *ABSatFrontLeftWheel* (ABSflw)—control the wheel braking to prevent locking the wheels.
– The brake actuators—*RearRightBrake* (RRB), *RearLeftBrake* (RLB), *FrontRightBrake* (FRB) and *FrontLeftBrake* (FLB)—apply the brake force on each wheel.

Sensors, actuators and the Electronic Control Units (ECUs) are distributed through a unique Controller Area Network (CAN).



TC1: Delay X1 = [70ms – 120ms]
TC2: Delay X2 = [5ms-15ms]
TC3: Repeat X3 = 10ms
TC4: Delay X4 = [1ms-2ms]
TC5: Delay X5 = 0.4*X1
TC6: Delay X6 = [40ms-50ms]
TC7: Delay X7 = [1ms-2ms]
TC8: Delay X8 = [5ms-15ms]
TC9: Synchronization X9 = 5ms
TC10: Synchronization X10 = 5ms
source
target
<<designFunctionType>> FunctionalDesignArchitecture
structure
<<designFunctionPrototype>> + BrakePedalSensor
ElSignal
PositionPercent
<<designFunctionPrototype>> + BrakeTorqueCalculator
PedalPercent
DriverReqTorq
GlobalTorque
<<designFunctionPrototype>> + RearRightWheelSensor
<<designFunctionPrototype>> + RearLeftWheelSensor
<<designFunctionPrototype>> + FrontRightWheelSensor
<<designFunctionPrototype>> + FrontLeftWheelSensor
Ticks
SpeedRpm
<<designFunctionPrototype>> + GlobalBrakeController
RearRightWheel_rpm
TorqRearRightWheel
RearLeftWheel_rpm
TorqRearLeftWheel
VehicleSpeedEst_kmph
FrontRightWheel_rpm
TorqFrontRightWheel
TorqFrontLeftWheel
FrontLeftWheel_rpm
<<designFunctionPrototype>> + ABSatRearRightWheel
<<designFunctionPrototype>> + ABSatRearLeftWheel
<<designFunctionPrototype>> + ABSatFrontRightWheel
<<designFunctionPrototype>> + ABSatFrontLeftWheel
RequestedTorq
VehicleSpeed_kmph
ABSBrakeTorque
WheelSpeed_rpm
<<designFunctionPrototype>> + RearRightBrake
<<designFunctionPrototype>> + RearLeftBrake
<<designFunctionPrototype>> + FrontRightBrake
<<designFunctionPrototype>> + FrontLeftBrake
TorqCmd
ElSignal

Fig. 1. Brake-By-Wire functional view augmented with TADL2 timing constraints.

This functional decomposition is augmented with thirty one TADL2 timing constraints. Figure 1 and Table 1 reflect the main types of timing constraints attached to the BBW architecture and provide an intuitive description of them. Some of these constraints are about a periodic sensor acquisition (see *TC3*). The distributed nature of the system generates some potential de-synchronizations. Therefore, some synchronization constraints that represent the temporal consistency of events (*TC9* and *TC10*) are introduced in the BBW timing specification. Delays on the ports are represented by delay constraints (*TC1*, *TC2*, *TC4*, *TC5*, *TC6*. *TC7* and *TC8*).

3 TADL2: Timing Augmented Description Language

In this section, we introduce TADL2 and give an informal semantics of the timing constraints. We also briefly describe the increment from TADL2 over TADL. The first

Table 1. The main timing constraints for the functional architecture of the BBW.

ID	Constraint Description
TC1	Four **delays** X1 are measured from the brake pedal stimulus (*ElSignal* on *BPS*) to the brake actuator responses (the *ElSignal* ports on *RRB*, *RLB*, *FRB* and *FLB*). The delays are bounded with a minimum value of **70** ms and a maximum value of **120** ms.
TC3	The acquisition of the wheel sensors (the *Ticks* ports on *RRWS*, *RLWS*, *FRWS*, *FLWS*) must be done **periodically** every X3=**10** ms.
TC5	Four **delays** X5 are measured from the wheel rpm signal (the *RRW_rpm*, *RLW_rpm*, *FRW_rpm* and *FLW_rpm* ports on *GBC*) to the brake torque calculation (the *TRRW*, *TRLW*, TFRW, *TFLW* ports on *GBC*). The delay constraint X5 applied on the global brake controller is 40 percent of the initial time budget **X1** given in **TC1**.
TC10	First and last wheel brake actuations (the *ElSignal* ports on *RRB*, *RLB*, *FRB*, *FLB*) must follow each other by no more than X10 =**5** ms.

improvement with TADL2 concerns *symbolic timing expressions* used to express durations such as *maximum/minimum delay* and *tolerance*. The second improvement is the ability to define explicit time bases by using modeling elements: *TimeBase*, *Dimension* and *Unit*. For a more detailed description of TADL2 please refer to [19].

3.1 TADL2 Timing Constraints

In this paper, we consider the following TADL2 timing constraints, sufficient to capture the constraints described in Figure 1 (See [9] for the whole set of constraints):

- `DelayConstraint` imposes duration bounds (minimum and maximum) between two events *source* and *target*.
- `SynchronizationConstraint` is a constraint on a set of events. All events must occur within a sliding window, specified by the *tolerance* attribute, i.e., maximum allowed skew between the events.
- `RepeatConstraint` imposes a period of the successive occurrences of a single event. *upper* and *lower* give the time interval between two subsequent occurrences.

The TADL2 timing constraints mostly constrain the identifiable state changes formulated as *Event*s. The causally related events are contained as a pair by *EventChain*s. Based on *Event*s and *EventChain*s, it is possible to represent data dependencies and critical execution paths as additional constraints for an EAST-ADL functional architecture model, and to apply timing constraints on these paths.

Timing attributes like *tolerance*, *upper* and *lower* are given as *Timing Expression*s. There are three types of timing expressions: *Value*, *Variable* and *Symbolic*. *Variable Timing Expressions* stand for free variables and constants. *Symbolic Timing Expression*s integrate basic arithmetic and relation operators associated with timing values.

3.2 TimeBase, Dimension and Unit in TADL2

TimeBase represents a discrete and totally ordered set of instants. An instant can be seen as an event occurrence called a tick. It may represent any repetitive event in the system. Events may refer to the classical time dimension or to some evolution of a hardware part (e.g., rotation of crankshaft, distance). The type of *TimeBase* is *Dimension* with a kind that represents the nature of *TimeBase*. *Time*, *Angle* and *Distance*, often used in automotive specifications, are proposed as a predefined dimension kind.

Dimension has a set of units to express durations measured on a given *TimeBase*. Each Unit is related to another Unit with *factor*, *offset* and *reference* to enable conversions. Only linear conversions are allowed. Because *Timebase* is a discrete set of instants, a discretization step is specified with *precisionFactor* and *precisionUnit*. Listing 1.1 gives examples of TADL2 declarations for Dimension and TimeBase. The *physicalTime* dimension has three units where 1 *second* is equal to 10^6 *micros* and 1 *ms* is equal to 10^3 *micros* (lines 2-4). *universal_time* is declared based on *physicalTime* (lines 7-8).

```
1  Dimension physicalTime {
      Units { micros{factor 1.0 offset 0.0},
3             ms{factor 1000.0 offset 0.0 reference micros},
              second{factor 1000000.0 offset 0.0 reference micros}  }
5     kind Time
   }
7  TimeBase universal_time { dimension physicalTime  precisionFactor 1
                             precisionUnit micros  }
```

Listing 1.1. Declaration of Dimension and TimeBase in TADL2

3.3 BBW Example in TADL2

Listing 1.2 gives some of the BBW timing constraints in TADL2. For the complete TADL2 specification of the BBW example, please refer to [1].

```
Event brakePedalSensorActivation {}  Event positionPercent {}
Event firstWheelBrakeActuation {}    Event firstWheel_rpm {}
Event firstWheelSensorAcquisition {} Event torqFirstWheel {}

EventChain ec1 {
   stimulus brakePedalSensorActivation     response firstWheelBrakeActuation
   eventChains ec1a, ec1b, ec1c, ec1d, ec1e, ec1f, ec1g
}
EventChain ec1a {stimulus brakePedalSensorActivation   response positionPercent}

var X1min ms on universal_time := 70.0
var X1max ms on universal_time := 120.0
DelayConstraint tc1a {
   source brakePedalSensorActivation     target firstWheelBrakeActuation
   lower = X1min       upper = X1max
}

var X3 ms on universal_time := 10.0
RepeatConstraint tc3a {
   event firstWheelSensorAcquisition    lower = X3    upper = X3    span = 1
}

var X5min ms on universal_time := (X1min*0.40)
var X5max ms on universal_time := (X1max*0.40)
DelayConstraint tc5a {
   source firstWheel_rpm    target torqFirstWheel    lower = X5min    upper = X5max
}

SynchronizationConstraint tc10 {
   events firstWheelBrakeActuation, secondWheelBrakeActuation,
          thirdWheelBrakeActuation, fourthWheelBrakeActuation
   tolerance = (5.0 ms on universal_time)  }
```

Listing 1.2. Some BBW Timing Constraints in TADL2

In Listing 1.2, we give only a part of events and event chains (lines 1-3). *ec1* gives the execution path between the activation of the brake pedal sensor and the actuation of the first wheel brake (lines 5-8). It contains other event chains ec1a, ... , ec1g (line 7) which give the intermediate executions. *ec1a* states that *positionPercent* is provided just after the activation of the brake pedal sensor (line 9). Each event is attached to a port in EAST-ADL. *brakePedalSensorActivation* and *firstWheelBrakeActuation* are attached to *EISignal* of *BPS* and *EISignal* of *RRB* in Figure 1 respectively.

We have variable declarations as *variable timing expression* (e.g., lines 11-12). All delay and repeat constraints in Figure 1 are replicated for the four wheels. *tc1a*, *tc3a* and *tc5a* (*TC1*, *TC3* and *TC5*) are only for the first wheel. The lower and upper bounds of *tc5a* (line 25) are computed by using symbolic timing expressions ("X1min*0.40" and "X1max*0.40" in lines 23-24). *tc3a* describes the occurrences of the first wheel sensor acquisition with a period (*lower* and *upper*). *tc10* is about the maximum tolerated time difference among the wheel brake actuations (*TC10*). Its *tolerance* attribute is equal to a value timing expression ("5 ms on universal_time" in line 32).

4 TADL2 to MARTE/CCSL: Simulation Approach

The TADL2 timing constraints are described informally in the previous section. However, to conduct validation and verification it is required to rely on a formal semantics. In this section, we use CCSL to capture the semantics of those constraints. We then rely on the CCSL operational semantics to execute the BBW example. This is the first part of our proposal is to make TADL2 specifications executable. CCSL was selected because it supports both kinds of constraints available in TADL2: causal ones (event chains) and temporal ones (delay, synchronization, repeat). After a brief introduction to CCSL, we give a mapping from TADL2 to CCSL. At the end, we illustrate our proposed validation framework for TADL2.

4.1 The Clock Constraint Specification Language (CCSL)

MARTE is the UML profile for Modeling and Analysis of Real-Time and Embedded systems [18,7]. It defines a broadly expressive formal *Time Model* [8] that provides a generic timed interpretation for UML models through the notion of *clock*. A clock c (not to be confused with the UPPAAL clocks) denotes particular UML events on which we want to impose a constraint. Clocks (events) are ordered sets of instants (event occurrences), $\mathcal{I}$. When the set is discrete, $c[i]$ denotes the i^{th} occurrence of event c. The Clock Constraint Specification Language (CCSL) was defined as a non-normative annex of MARTE as a language to build causal and timed constraints on clocks. CCSL considers two kinds of binary instant relations: *precedence* (denoted $\prec$) and *coincidence* (denoted $\equiv$). Given two instants i and $j \in \mathcal{I}$, $i \prec j$ denotes that the event occurrence i must be observed before j, whereas $i \equiv j$ denotes that i and j must be observed simultaneously. A labeling function $\lambda : \mathcal{I} \rightarrow T$ associates instants with a time tag.

Based on these two primitive relations on instants, CCSL derives relations on clocks. We only describe here the clock relations pertinent to our running example. Eq.1 gives an example of a *non-functional* clock relation where $mic\_universalTime$

is a logical clock, such that $\forall i \in \mathbb{N}^{\star}, \lambda(mic\_universalTime[i]) = i * 0.000001$, it models a discrete clock of period 1 *mic*, one microsecond since $IdealClock$ is defined relative to the unit $second$.

$$mic\_universalTime = IdealClock \texttt{ discretizedBy } 0.000001 \quad (1)$$

Eq.2 gives an example of *synchronous* clock relation that defines a new discrete logical clock ms such that $\forall i \in \mathbb{N}^{\star}, ms[i] \equiv micro\_universalTime[(i-1)*1000+1]$.

$$ms \texttt{ isPeriodicOn } mic\_universalTime \text{ period } 1000 \quad (2)$$

A basic asynchronous constraint is given by the clock relation `precedes`. "*a* precedes *b*" (symbolically denoted by $a \boxed{\prec} b$) specifies that for all natural number *k*, the k^{th} instant of *a* precedes the k^{th} instant of *b*: $\forall k \in \mathbb{N}^{\star}, a[k] \prec b[k]$.

Some clock constraints mix *precedence* and *coincidence* relations. "*a* causes *b*" or "*b* dependsOn *a*" (both denoted $a \boxed{\preccurlyeq} b$) specifies that for all natural number *k*, the k^{th} instant of *a* precedes or is coincident with the k^{th} instant of *b*: $\forall k \in \mathbb{N}^{\star}, a[k] \prec b[k] \vee a[k] \equiv b[k]$).

CCSL also provides expressions to build new clocks from existing ones. For instance, the CCSL expression *c* = `inf`(*a*, *b*) builds a new clock c such that c is the slowest clock that is faster than both a and b: ($\forall k \in \mathbb{N}^{\star}, c[k] \equiv a[k]$ `if` $a[k] \prec b[k]$, $c[k] \equiv b[k]$ `otherwise`). Similarly, "*d* = `sup`(*a*, *b*)" is the fastest clock slower than both a and *b*: $\forall k \in \mathbb{N}^{\star}, d[k] \equiv b[k]$ `if` $a[k] \prec b[k]$, $d[k] \equiv a[k]$ `otherwise`. Most of the time, *inf* and *sup* are neither *a* nor *b*. *inf* and *sup* are easily extended to sets of clocks.

Finally, the expression `delayedFor` builds a delayed clock."*c* = *a* `delayedFor` n on *b*" imposes *c* to tick synchronously with the n^{th} 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.

TimeSquare. TimeSquare [11] is a software environment (set of Eclipse plugins) dedicated to the analysis of MARTE time model and CCSL specifications. It has four main functionalities: 1) interactive clock-related specifications, 2) clock constraint checking, 3) generation of a solution and 4) displaying and exploring waveforms. The second functionality relies on a constraint solver that yields a satisfying execution trace for CCSL clocks. The traces are given as waveforms written in VCD (Value Change Dump) format [14]. The solver intensively uses Binary Decision Diagrams (BDD) to compose symbolically boolean equations induced by CCSL clock constraints.

4.2 Modelling TADL2 Constraints in CCSL

We first give the MARTE time model representation of basic TADL2 elements *TimeBase*, *Dimension* and *Unit* in CCSL. Then we express the semantics of *Event*, *EventChain* and some of the TADL2 constraints in CCSL.

4.2.1 TimeBase, Dimension and Unit. Each *Unit* of a *Dimension* in a *TimeBase* represents a set of ticks. Hence, we represent each *Unit* in a given *TimeBase* as a CCSL *clock*. The reference unit in a dimension is a special unit whose corresponding clock is derived by discretizing *IdealCLK*. Eq.1 defines a discrete chronometric clock *mic_universalTime* for the *micro* Unit of *physicalTime* in *universal_time* in Listing 1.1.

Clocks for other units in the *TimeBase* are defined as a subclock of the reference unit clock with a period. Eq.2 defines *ms* as a subclock of *mic_universalTime* with period 1000 (*factor* of the *ms* unit) for the *ms* unit of *universal_time*.

4.2.2 Timing Constraints. Each TADL2 *Event* on which we want to attach timing constraints is associated with a CCSL *Clock*. An event denotes something that occurs (e.g., the start of an action, the receipt of a message.). Therefore, a CCSL clock represents the set of instants at which the related event occurs.

An *EventChain* in TADL2 contains causally related events. It is mapped to the *causes* operator of CCSL. For instance, we have the following clock constraints for *ec1* and *ec1a* event chains: ($bpsa \boxed{\preccurlyeq} fwba$) and ($bpsa \boxed{\preccurlyeq} pp$) where the CCSL clocks *bpsa*, *fwba* and *pp* correspond to the TADL2 events *brakePedalSensorActivation*, *firstWheelBrakeActuation* and *positionPercent* respectively.

DelayConstraint. It specifies an *end-to-end* delay between the *source* and *target* events where the attributes *lower* and *upper* denote minimum and maximum values of the delay respectively. `tc1a` (Listing 1.2) specifies the permissible delay between the source event *brakePedalSensorActivation* and the target event *firstWheelBrakeActuation*. Eqs. 3-6 give the corresponding CCSL clocks and clock constraints for `tc1a`.

$$\begin{aligned}
&\textsf{Clock}\ bpsa,\ fwba && (3)\\
&\textsf{Clock}\ lower\ =\ bpsa\ \texttt{delayedFor}\ 70\ \texttt{on}\ ms && (4)\\
&\textsf{Clock}\ upper\ =\ bpsa\ \texttt{delayedFor}\ 120\ \texttt{on}\ ms && (5)\\
&\Big(lower\ \boxed{\preccurlyeq}\ fwba\Big) \wedge \Big(fwba\ \boxed{\preccurlyeq}\ upper\Big) && (6)
\end{aligned}$$

Eq.3 declares two CCSL clocks for the source and target events *brakePedalSensorActivation* (bpsa) and *firstWheelBrakeActuation* (fwba). The CCSL constraint *delayedFor* delays an initial clock (*bpsa*) for a given duration. Combining *delayedFor* and *causes* allows for specifying distances between two clocks. Eqs. 4-5 build two clocks *lower* and *upper* delayed for 70 and 120 ms respectively from the source event clock *bpsa*. Eq. 6 enforces the target event clock *fwba* to tick between the corresponding ticks of the clocks *lower* and *upper*.

SynchronizationConstraint. It specifies the bounds on the delay among event occurrences specified by the attribute *tolerance*. `tc10` (see Listing 1.2) specifies the output synchronization among the four brake actuators that must occur within the specified time duration. Eqs. 7-10 give the corresponding CCSL clocks and constraints for `tc10`.

$$\begin{aligned}
&\textsf{Clock}\ fwba,\ swba,\ twba,\ ftwba && (7)\\
&\textsf{Clock}\ fastest\ =\ inf(fwba,\ swba,\ twba,\ ftwba) && (8)\\
&\textsf{Clock}\ slowest\ =\ sup(fwba,\ swba,\ twba,\ ftwba) && (9)\\
&slowest\ \boxed{\preccurlyeq}\ (fastest\ \texttt{delayedFor}\ 5\ on\ ms) && (10)
\end{aligned}$$

The constraint concerns the distance from the earliest event to the latest event. It has four events, one for each wheel. Eq. 7 declares clocks for events *firstWheelBrakeActuation* (fwba), *secondWheelBrakeActuation* (swba), *thirdWheelBrakeActuation* (twba) and *fourthWheelBrakeActuation* (ftwba). Eqs. 8-9 we get the fastest/slowest clocks among all clocks slower/faster than *fwba*, *swba*, *twba* and *ftwba*. Eq. 10 states that $slowest$ must not tick later than 5 *ms* after the respective ticks of $fastest$.

RepeatConstraint. It specifies the periodic occurrence of an event. The duration of the period is specified by the attributes $lower$, $upper$ and $span$. This constraint defines the basic notion of repeated occurrences. If the $span$ attribute is 1 and the $lower$ and $upper$ attributes are equal, the accepted behaviors must be strictly periodic. If $lower$ is less than $upper$, the event occurrences may deviate from a strictly periodic one in an accumulating fashion. `tc3a` (Listing 1.2) specifies the strictly periodic nature of the sensor value acquisition, for one of the four wheels. Eqs.11-14 give the corresponding CCSL clocks and clock constraints for `tc3a`.

$$\textsf{Clock}\ fwsa \tag{11}$$

$$lower\ \texttt{isPeriodicOn}\ ms\ \textsf{period}\ 10 \tag{12}$$

$$upper\ \texttt{isPeriodicOn}\ ms\ \textsf{period}\ 10 \tag{13}$$

$$\Big(lower\ \boxed{\preccurlyeq}\ fwsa\Big) \wedge \Big(fwsa\ \boxed{\preccurlyeq}\ upper\Big) \tag{14}$$

Eq. 11 declares the CCSL clock $fwsa$ for the event *firstWheelSensorAcquisition*. Eqs. 12-13 build two clocks *lower* and *upper* of period 10 from *ms*. Eq. 14 enforces the event clock *fwsa* to tick between the corresponding ticks of the clocks *lower* and *upper*. Since *lower* and *upper* have the same period, *fwsa* ticks every 10 ticks of *ms*. We defined both *lower* and *upper* clocks to propose an exhaustive transformation in case lower and upper bounds differ.

4.3 Executing TADL2 specification with TimeSquare

Based on the mapping presented in Section 4.2, we obtain an executable CCSL specification from the Brake-By-Wire TADL2 description. A simulation trace produced by TimeSquare is partially shown in Figure 2. The focus here is on the constraint *TC10*. The dashed (blue) arrows are the *precedence* relations, whereas the vertical plain (red) connectors are the *coincidence* relations between two instants. The first entry shows the fastest ($fastest$) of the four wheel brake actuator events ($fwba$, $swba$, $twba$, $ftwba$). It is followed by the four events. The sixth entry is the slowest of the four events ($slowest$). Coincidence relations show that there is always one occurrence of each of the four actuator between an occurrence of $fastest$ and an occurrence $slowest$. Additionally, it also shows that $slowest$ always occur before the deadline, which is 5 ms after $fastest$. The deadline is shown as the last entry of the simulation.

In TimeSquare *runs* consist of multiple execution steps. At each step, the CCSL solver builds a boolean solution and computes a set of all the valid configurations. A configuration is a set of *enabled* clocks, i.e., clocks that are allowed to tick at the given step. If the CCSL specification is deterministic, there is only one valid configuration. If

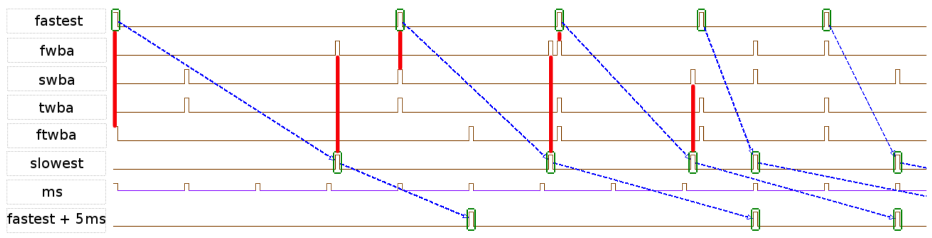


Fig. 2. CCSL Simulation focusing on the constraint TC10 of the BBW Example.

it is nondeterministic, for each step the simulator fires one of the valid configurations. This selection is based on a scheduling policy. When TimeSquare manages to produce a valid trace, this means there is a way to satisfy the constraints. If the system is not deterministic there may be other runs that do not satisfy all the constraints.This is why such a tool must be complement with exhaustive analyses when possible. However, it must be noted that in the general case it is not possible to conduct exhaustive analyses of CCSL specifications, whose state-space can be infinite. In such cases, TimeSquare provides an early support to validate and refine the specification. When focusing on TADL2 timing constraints (delay, repeat and synchronizations) the state-space is bounded and can be explored by model-checking. However, event chains with indeterminate delays may cause problems and need to be further refined. This is discussed in the next section.

On this example, TimeSquare has found one possible run that satisfies the TADL2 specification meaning that the specification is consistent and a solution exists. In the following section, UPPAAL shows that there also may be runs such that the synchronization constraint TC10 is violated (see eq. 20).

5 TADL2 to Timed Automata/UPPAAL: Verification Approach

In this section, we present a formal verification approach for TADL2 specifications. For this, we have chosen UPPAAL [17], a model-checking tool, and present a mapping for a subset of TADL2 into timed automata, the modeling language of UPPAAL.

5.1 UPPAAL model-checker: An overview

UPPAAL extends timed automata (TA), originally introduced by Alur and Dill [6], with a number of features, such as, global and local (bounded) integer variables, arithmetic operations, arrays, and a C-like programming language. The tool consists of three parts: a graphical editor for modeling timed automata, a simulator for trace generation, and a verifier for symbolic (exhaustive) verification of a system modeled as a network of timed automata. A subset of CTL (computation tree logic) is used as the input language for the verifier.

A timed automaton (TAn) is a tuple $< L, l_0, C, A, E, I >$, where L is a set of $locations$, $l_0 \in L$ is the initial location, C is the set of clocks, A is the set of actions, co-actions and the internal τ-action, $E \subseteq L \times A \times B(C) \times 2^C \times L$ is a set of edges between locations with an action, a guard, a set of clocks to be reset, and $I : L \rightarrow B(C)$

assigns clock *invariants* to locations. A location can be marked *urgent* (u) or *committed* (c) to indicate that the time is not allowed to progress in the specified location(s), the latter being stricter form indicating further that the next transition can only be taken from the corresponding locations. Synchronization between two automata is modeled by *channels* (e.g., **x**! and **x**?) with *rendezvous* or broadcast semantics.

Semantically, the *state* of a TAn represents the current location (several in case of a network of TA) and current evaluation of all the variables. An *enabled* edge (that is, when the guard becomes *true*) indicates a *transition* that may be taken in the current state. The semantics of a TAn defines transitions between locations as well as the time progress; an *enabled* edge at a current location may be taken (non-deterministically in case of many), when the invariant at the corresponding target location is preserved, otherwise no transition is taken and the time is allowed to progress as long as the invariant at the current location holds. For further details, we refer the reader to the UPPAAL tutorial [17].

5.2 Modeling TADL2 in UPPAAL

To begin with, we present the TA modeling of basic time (chronometric) aspects in TADL2, such as, *timebase*, *dimension* and *unit*. Next, we will show that the timing constraints in TADL2 can be modeled as TA.

5.2.1 TimeBase, Dimension and Unit. In the semantics of TA, time progresses symbolically, that is, through construction of so-called "region-graphs" [3]. Hence, we represent a time *dimension* and a given time *unit* in TADL2, corresponding to a given time base, as a single *step* of (chronometric) time progress in TA *clocks*. Concretely, a *timebase* can be modeled as a TAn using a *clock* variable which implicitly represents the associated *dimension* and the corresponding *unit*. As a timebase, the automaton is a *reference* clock for a TADL2 timing specification (or part of it, in case of multiple time bases in the specification).

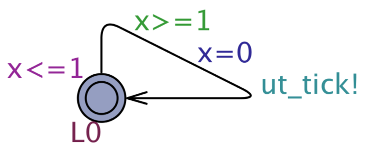


Fig. 3. `universal_time`: a timebase automaton.

In Figure 3, we present the TAn for the `universal_time` in BBW specification, a timebase as defined in Listing 1.1. The corresponding *dimension*, that is, the `physical_time` and the time unit `ms` (Eqs. 1 and 2) are implicitly represented by the *clock* variable 'x'. The duration of the time unit is represented by the invariant `x<=1` and the guard `x>=1` at the *location* L0; it represents a single step of the discrete time progress or tick' of the `universal_time`. The time progress can be observed by the successive ticks of `ut_tick` (modeled as a broadcast *channel*) during simulation.

[3] Makes reachability analysis *decidable* by transforming otherwise an infinite-state timed automaton into finite-state.

5.2.2 Timing Constraints. Timing constraints can be specified for $Event$ and $EventChain$ in TADL2. An event chain can be modeled as a TAn with synchronization channels representing the corresponding events. For instance, in Fig. 4(a), we present the TA modeling of the event-chain `ec1` (Listing 1.2). It consists of events $brakePedalSensorActivation$ ($bpsa$) as the source (stimulus) event and $firstWheelBrakeActuation$ ($fwba$) as the target (response) event, modeled as synchronization channels. The causality among the events is modeled by the receiving(?) and sending(!) signals of the corresponding channels respectively.

DelayConstraint. In Figure 4(b), we present the TA modeling of the delay constraint `tc1`, for the event chain `ec1`. The transition from L0 is taken when the source event $bpsa$ occurs. At $location$ L1, the permissible delay, specified in terms of the invariant and the guard (on outgoing edge) using clock variable x, is allowed before the target event $fwba$ occurrence. It can be observed that the TA modeling of the delay constraint `tc1` is a time constrained model of the corresponding event-chain automaton (Fig. 4(a)).

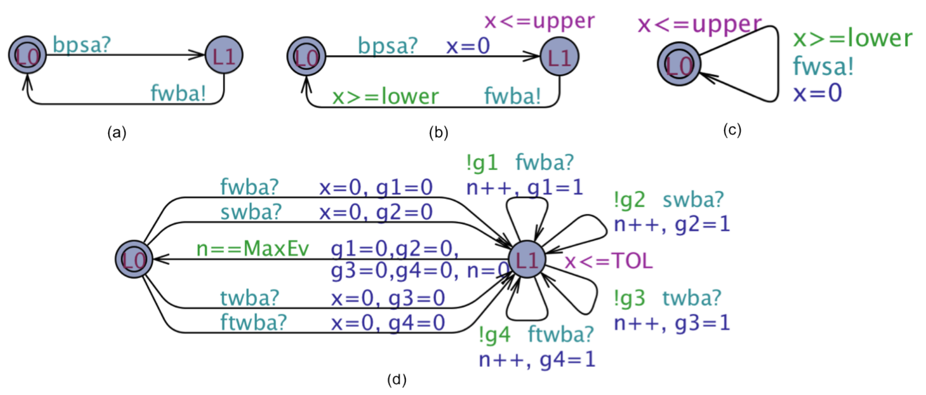


Fig. 4. TADL timing constraints as TA: (a) EventChain $ec1$ (b) DelayConstraint $tc1$ (c) RepeatConstriant $tc3a$ (d) SynchronizationConstraint $tc10$.

SynchronizationConstraint. In Figure 4(d), we present the TA modeling of the synchronization constraint `tc10` (Listing 1.2). It consists of two $locations$ L0 and L1; the edges between the locations contain synchronization channels (receiving signals) corresponding to the events of the specified event group, that is, $firstWheelBrakeActuation(fwba)$, $secondWheelBrakeActuation(swba)$, $thirdWheelBrakeActuation(twba)$ and $fourthWheelBrakeActuation(ftwba)$; corresponding transition is taken when anyone of these occur first. At location L1, the other three events need to occur, before the transition back to L0 is taken, within the specified $tolerance$ value denoted by the invariant `x <= TOL`.

RepeatConstraint. In Figure 4(c), we present the TA modeling of the constraint `tc3a` (Listing 1.2). When *span* is 1, and *upper* is equal to *lower*, the specified event is periodically generated. Thus the event $fwsa$ (*firstWheelSensorAcquistion*) periodically occurs with the specified period.

5.3 Verification Results

For a TADL2 specification, the corresponding network of TA models is an executable specification that can be simulated and verified. However, to support the verification, we further extend the composed model to enable verification. For example, for event chains with no associated delay constraint, we make the non-initial locations *urgent* to skip indeterminate delays. Further, we introduce a synchronization pattern (Fig. 5 (a) and (b)), for event chains with common response event. Also, we use an observer TAn, e.g. Fig. 5 (c), to verify delay constraints. To begin with, we can verify general properties, such as, well-formedness, deadlock-freeness, as discussed below:

- *well-formedness* : we define a TADL2 specification as *well-formed*, if every location in the composed TAn is *reachable*. For example, we can verify a location L in an automaton T is reachable by verifying that the property `E<> T.L` is satisfied (that is, there exists a path where the boolean predicate T.L *eventually* holds).
- *consistency*: a *deadlock* in the execution of TADL2 timed automata model, indicates an inconsistent specification, due to inconsistent timing constraints or even modeling errors. However, the diagnostic traces given by UPPAAL are useful to identify and resolve the deadlocks.

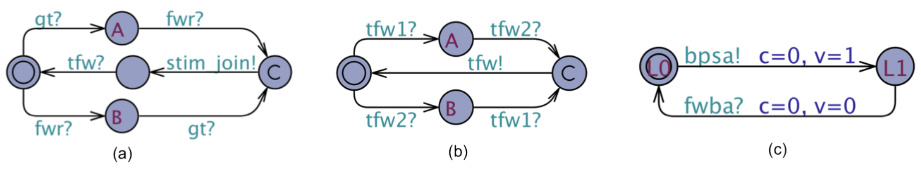


Fig. 5. (a) $join\_stimulus$ (b) $join\_response$ (c) Observer TA to verify '*tc1a*'

We have verified TADL2 timing constraints as follow. However, these constraints are verified in isolation whenever possible to keep the statespace minimal. While verifying the delay constraints, Eq. 15 shows there is no deadlock. And, Eq. 16 establishes that the event *'bpsa'* always `leads_to` ($\leadsto$) the corresponding response event *'fwba'*. Further, we can verify that the *DelayConstraints* are consistent. For BBW example, the event chain $ec1a$ is defined in terms of $ec1a$, ... , $ec1g$ (Listing 1.2). From this, we can verify the timing behavior of $ec1a$ given by $tc1a$ w.r.t. the combined timing/causality behavior of $ec1a$, ... , $ec1g$ (Fig. 1 and [1]). For this, we compose the specification model with an observer automaton for $tc1a$, by extending it with auxiliary variables `v` and `c` (Fig. 5 (c)). Eqs. 17 and 18 show the timing property $tc1a$ is *not* satisfied.

$$A[]\ \texttt{not deadlock} \qquad //satisfied \tag{15}$$
$$tc1a.L1\ \texttt{-->}\ tc1a.L0 \qquad //satisfied \tag{16}$$
$$A[]\ v\ \texttt{imply}\ c>=70 \qquad //satisfied \tag{17}$$
$$A[]\ v\ \texttt{imply}\ c<=120 \qquad //not\ satisfied \tag{18}$$
$$E<>\ v1\ \texttt{imply}\ c1<=5 \qquad //satisfied \tag{19}$$
$$A[]\ v1\ \texttt{imply}\ c1<=5 \qquad //not\ satisfied \tag{20}$$

For output synchronization constraint, *TC10*, we have extended the corresponding automata with auxiliary variables $c1$ and $v1$ (similar to Fig. 5(c)). Eq. 19, a *reachability* property, shows existence of a solution satisfying *TC10*, which confirms the TimeSquare simulation (Section 4.3). However, Eq. 20 shows that the constraint is not satisfied in all cases. This means that applying the constraints alone is not enough, these constraints should be complemented with a refined description of the actual behavior of the system.

6 Discussion of the Approach

This section discusses the benefits of combining two formal models CCSL and Timed Automata to offer a complete support for the analysis of TADL2 specifications. First it should be noted that one major extension of TADL2 over TADL is the addition of explicit references to time bases. Such time bases can be either logical or physical. This extension took a direct inspiration from the MARTE Time model. It recognizes the importance of logical clocks in high-level specifications where information about physical time is not always available. As a specification language, MARTE CCSL offers a full support to build both logical and physical clocks as well as capturing time expressions referring to such clocks. CCSL specifications can be analyzed by TimeSquare that was specifically designed for the purpose. TimeSquare provides several features. The first important one is a support for model simulation. Thus, the transformation from TADL2 to CCSL provides a support for making TADL2 specifications executable directly inside a UML environment (like Papyrus). Other features of TimeSquare offer support for exhaustive analyses of CCSL specifications. Those features mainly focus on logical aspects and offer little or no benefit for the exhaustive analysis of physical-based constraints such that those shown in this paper. Timed Automata, however, are a powerful formalism to handle physical time constraints through the UPPAAL clocks. By offering a transformation to Timed Automata, we then provide a support for the exhaustive analysis of physical-based constraints. This is why the two formalisms are used in a complementing way, CCSL to support model execution and analysis of logical time aspects, Timed Automata/UPPAAL for the exhaustive analysis of physical time constraints. However, having a coordinated analysis of mixed logical and physical constraints is out of the scope of this paper and is still an on-going research work.

Furthermore, exhaustive verifications can only be conducted with finite specifications. However, early specifications may remain incomplete and therefore may not be sufficiently refined to be bounded. For instance, event chains with indeterminate delays are typical unbounded specifications (the time may progress arbitrarily without any upper bound constraint). As discussed in the previous sections, we had to complete the event chains to conduct the analysis with UPPAAL.

7 Related Work

A number of approaches in the literature address modeling and analyzing timing constraints. Klein and Giese [16] present Timed Story Scenario Diagrams (TSSD), a visual notation for scenario specifications that takes structural system properties into account. In TSSD it is possible to specify *Time Constraints* that allow setting *lower* and *upper* bounds for delays. There is no mention of analysis support for TSSD. Alfonso et al. [5] present VTS, a visual language to define complex event-based constraints like freshness, bounded response, and event correlation. VTS does not support the notion of explicit time units coded as time bases. A mapping between VTS and timed automata is provided to model and analyze VTS scenarios. Aegedal [4] presents a general modelling language for Quality of Service (QoS). The language uses a time model where different clocks can be specified.

In the context of EAST-ADL, several approaches have been proposed for timed automata based modeling. In [20], Qureshi et al. presented timed automata templates for EAST-ADL timing constraints. These modeling templates capture various error scenarios and are based on EAST-ADL architectural models. In contrast, the automata templates presented in this paper specify event chains and associated causality and temporal behavior. While Kang et al. [15] have addressed the functional modeling for EAST-ADL models using UPPAAL, Enoiu et al. [13] have addressed a limited aspect of timing modeling for EAST-ADL models. However, none of the works mentioned above address the analysis of timing specifications including the explicit notion of timebase and symbolic timing expressions. Also they do not mention how to use different analysis approaches for timing constraints in a complementary fashion.

8 Conclusions

In this paper, we have presented both simulation and model-checking approaches for formal analysis of TADL2 specifications. We have mapped TADL2 specifications into CCSL specifications for simulations in TimeSquare and to timed automata for exhaustive verifications with UPPAAL model-checker. In addition to well-formedness and consistency checking, we have also verified the TADL2 timing constraints. The main limitation of the verification approach is the statespace explosion problem with model-checking. However, this may be addressed by using compositional techniques based on event chains in TADL2 specifications.

We have used a real industrial example proposed by Volvo Technology in the TIMMO-2-USE project [2] to show the capability of our approach for handling timing behavior of industrial systems. However, a natural question arises about the scalability and the efficacy of the proposed analysis approach on larger case studies. As a future work we plan to conduct experimental analysis on larger case studies. We also plan to consider a detailed comparison of analysis features of TimeSquare and UPPAAL for TADL2 specifications. Further, the mappings can be extended to multiple timebases and timebase relationships in TADL2, for specification and verification of timing constraints for distributed embedded systems, i.e., systems with multiple ECUs where each ECU has its own timebase. The mappings provide a basis for automated model transformations from TADL2 specifications to CCSL and UPPAAL. The automated model transformation from TADL2 to CCSL [3] is already implemented with QVTo [12]. Currently, we are working on the model transformation from TADL2 to UPPAAL.

References

1. BBW Spec in TADL2, http://www-sop.inria.fr/members/Arda.Goknil/bbw/.
2. ITEA TIMMO-2-USE Project, http://timmo-2-use.org/.
3. TADL2-CCSL QVTo Transformation, http://www-sop.inria.fr/members/Arda.Goknil/bbw/.
4. J. Aegedal. Quality of service support in development of distributed systems. *PhD Thesis*, 2001.
5. A. Alfonso, Víctor A. Braberman, Nicolas Kicillof, and Alfredo Olivero. Visual timed event scenarios. In *ICSE'04*, pages 168–177, 2004.
6. R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
7. C. André. Syntax and semantics of the Clock Constraint Specification Language (CCSL). Research Report 6925, INRIA, May 2009.
8. C. André, F. Mallet, and R. de Simone. Modeling time(s). In *Models'07*, volume 4735 of *LNCS*, pages 559–573. Springer, 2007.
9. ATESST (Advancing Traffic Efficiency through Software Technology). East-ADL2 specification, http://www.atesst.org, 2008-03-20.
10. Autosar Consortium. AUTOSAR specification, release 4.0, 2009, http://www.autosar.org/.
11. J. Deantoni and F. Mallet. Timesquare: Treat your models with logical time. In *TOOLS Europe 2012*. Springer LNCS, vol 7304, May 2012.
12. R. Dvorak. Model transformation with operational qvt. *EclipseCon'08*, 2008.
13. E. P. Enoiu, R. Marinescu, C. C. Seceleanu, and P. Pettersson. Vital: A verification tool for east-adl models using uppaal port. In *ICECCS'12*, pages 328–337, 2012.
14. IEEE Standards Association. *IEEE Standard for Verilog Hardware Description Language*. Design Automation Standards Committee, 2005. IEEE Std 1364TM-2005.
15. E. Y. Kang, P. Y. Schobbens, and P. Pettersson. Verifying functional behaviors of automotive products in east-adl2 using uppaal-port. In *SAFECOMP'11*, pages 243–256, 2011.
16. F. Klein and H. Giese. Joint structural and temporal property specification using timed story scenario diagrams. In *FASE'07*, pages 185–199, 2007.
17. K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a Nutshell. *Int. Journal on Software Tools for Technology Transfer*, 1(1–2):134–152, October 1997.
18. OMG. *UML Profile for MARTE, v1.0*. Object Management Group, November 2009. formal/2009-11-02.
19. M. A. Peraldi-Frati, A. Goknil, J. DeAntoni, and J. Nordlander. A timing model for specifying multi clock automotive systems: The timing augmented description language v2. In *ICECCS'12*, pages 230–239, 2012.
20. T. N. Qureshi, D. J. Chen, and M. Törngren. A timed automata-based method to analyze east-adl timing constraint specifications. In *ECMFA'12*, pages 303–318, 2012.