Tool Support for the Analysis of TADL2 Timing Constraints using TimeSquare

Arda Goknil, Julien DeAntoni, Marie-Agnès Peraldi-Frati, Frédéric Mallet
AOSTE Research Team
UNS-I3S-INRIA
Sophia-Antipolis, France
{*arda.goknil, julien.deantoni, marie-agnes.peraldi_frati, frederic.mallet*}*@inria.fr*

***Abstract*—Modeling and analysis of non-functional properties are central concerns in distributed real-time embedded systems. In automotive domain, EAST-ADL is one of the main architectural modeling approaches for real-time embedded systems. In our previous work we introduced the Timing Augmented Description Language V2 (TADL2), which is the new release of the time model for EAST-ADL. It provides new modeling capabilities such as explicit notion of timebase and symbolic timing expressions. In this paper we propose an approach to simulate and analyze TADL2 timing constraints. The formal semantics of TADL2 is given by an exogenous model transformation in QVTo to the Clock Constraint Specification Language (CCSL), a formal language that implements the MARTE Time Model. With this transformation, the analysis of TADL2 constraints become possible through TIMESQUARE framework dedicated to the analysis of CCSL specifications. The approach is illustrated on the Brake-By-Wire example.**

***Keywords*-timing constraints; analysis; EAST-ADL**

I. INTRODUCTION

Model Driven Development (MDD) is more and more applied in automotive domain. EAST-ADL (Electronic Architecture and Software Tools, Architecture Description Language) [1] is a concrete example for architectural modeling of safety-critical embedded systems. It is specified through a metamodel and implemented as a UML2 profile [1]. EAST-ADL mainly focuses on functional design specifications.

The new release of EAST-ADL (v2) has recently adopted the time model proposed in the Timing Augmented Description Language (TADL) [2] [3]. TADL allows for expressing and composing basic timing constraints such as repetition rates, end-to-end delays, and synchronization constraints on top of EAST-ADL models. The TIMMO-2-USE project [4] goes one step beyond TADL by recently introducing TADL2 [2]. The time model of TADL2 specializes the time model of the UML Profile for MARTE (Modeling and Analysis of Real-Time and Embedded systems) [5]. TADL2 augments TADL with new constructs borrowed from MARTE companion language, the Clock Constraint Specification Language (CCSL) [6]. CCSL is a formal language dedicated to the specification of temporal and causal constraints. In particular, TADL2 provides new modelling capabilities such as explicit notion of time base and symbolic timing expressions.

This paper proposes an automatic model transformation approach to simulate and analyze TADL2 timing constraints conjointly with an EAST-ADL architecture (see Figure 1). The formal semantics of TADL2 is given by an exogenous model transformation to CCSL. We use the Event Constraint Language (ECL) [7] to explicitly denote the semantics of both EAST-ADL and TADL2 metamodels. ECL is a lightweight extension of the Object Constraint Language (OCL) [8] with notions borrowed from CCSL. From the ECL specification given for the EAST-ADL and TADL2 metamodels, it is possible to generate a QVTo transformation [9] that takes EAST-ADL/TADL2 models as input and produces a CCSL model. The resulting CCSL model can then be used to simulate and analyze TADL2 constraints with TIMESQUARE [10].

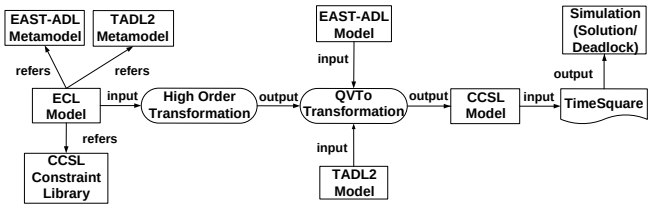

Figure 1. Overview of the Approach.

More than yet another transformation between two languages, we propose an approach to reason and refine EAST-ADL/TADL2 specifications. This is very important since syntactically well-formed TADL2 models can be underspecified or inconsistent. Inconsistency may be caused by two (or more) contradictory constraints. In such cases, TIMESQUARE simulation will highlight *disabled constraints* and *dead clocks*. Under-specification results in many possible solutions to satisfy the constraints. TIMESQUARE then attempts to find one valid solution amongst the set of valid solutions. When several simulation runs result in different solutions, the system is underspecified. Finding either satisfying solutions or falsifying ones helps refine the constraint system until the specification matches the expectations.

The paper is organized as follows. Section II introduces the Brake-By-Wire (BBW) system as a running example. Section III briefly overviews EAST-ADL and TADL2. Section IV gives the semantics of EAST-ADL and TADL2. Section V

gives some simulation results on the BBW example. Section VI discusses the related work.

II. Running Example

A distributed Brake-By-Wire (BBW) application with an anti-lock braking functionality is used to illustrate our approach. The BBW example is one of the validator proposed by Volvo Technology in the TIMMO-2-USE project [4].

The running example is a real-time distributed system that is in charge of computing the torque that must be applied on the brakes. The torque is a function of the brake pedal position and the speed of the wheels. The goal is to avoid blocking wheels. This is achieved by releasing momentarily the pressure on the brake(s) even though the driver is pressing the brake pedal. This example is interesting for a couple of reasons. First, sensors, actuators and controls are usually deployed on different Electronic Control Units (ECUs) due to the car physical architecture; second, its functional architecture supports possible synchronizations (time triggered and/or event triggered). Third, its real-time nature is auspicious for the specification of multiform and symbolic timing constraints.

The BBW architecture is given in EAST-ADL (see Figure 2) and then augmented with TADL2 timing constraints. These two languages are described in section III.

III. Modeling Approach for Timing Constraints

This section describes the core of our approach. It first presents EAST-ADL and focuses on the behavioral semantics hidden behind the stereotypes. The EAST-ADL semantics is important and needs to be made explicit because it enforces synchronization and causality constraints that impact the timing analysis. After introducing EAST-ADL, we show how an EAST-ADL model can be augmented with a TADL2 specification to obtain a global model for timing constraints.

A. EAST-ADL *Functional Model*

EAST-ADL is an architecture description language specified through a metamodel and implemented as a UML profile [1]. It has been initially developed in the context of the EAST-EEA European project [11]. Its modeling process is structured into different abstraction levels: *vehicle level*, *analysis level*, *design level* and *implementation level*. The features of the vehicle electronic system are modeled at the *Vehicle Level*. Abstract functional definitions of these features are modeled at the *Analysis Level*. The details of the functional definition are given at the *Design Level*. The *Implementation Level* describes reusable code and software & system configurations for hardware deployment.

At both analysis and design levels, the system is described by a *FunctionalArchitecture* (called *FunctionalDesignArchitecture* in Figure 2). The functional architecture is composed of a number of interconnected *FunctionPrototype*s where each prototype instantiates a *FunctionType*. In the Brake-by-Wire example we have five function prototypes that manage the acquisition of data from the environment (four prototypes for wheel speed sensors called *XXWheelSpeedSensor* and one prototype for the brake pedal sensor called *BrakePedalSensor*). The *StartBrakingDetector* function prototype represents the act of changing pedal position which is detected by *BrakePedalSensor*. There are also four function prototypes that manage the actuation of the physical brake for each wheel (called *XXBrake*). Other function prototypes in Figure 2 are used to control the execution between the acquisition and actuation. The most important one is *GlobalBrakeControler*. Others are interfacing the brake pedal sensor and the brake actuators with the global brake controller. To provide the communication among function prototypes, the function types and consequently their function prototype instances have ports that can be either *FlowPort* or *ClientServerPort*. The connections between ports are provided by *FunctionConnector*s. Connected flow ports represent shared data from an output port to an input port.

One specificity of EAST-ADL is *functionTrigger*. It specifies when a function should be executed. It is a flexible mechanism allowing a function to be activated in a time-driven mode as well as an event-driven mode. In both event and time driven modes, *triggerCondition* specifies when to trigger the function. This condition is a key point to perform timing analysis in an EAST-ADL model because changing when a function is executed has an impact on the system behavior and consequently on the timing analysis.

Unfortunately no language is given to express *trigger conditions*. We propose to denote the semantics of EAST-ADL and TADL2 in CCSL (see Section IV). More precisely in CCSL we propose to denote formally (i) the causal and synchronization semantics of EAST-ADL reflected by the definition of the design and/or analysis functional architecture, (ii) the timed and causal semantics of TADL2 to augment the functional architecture with the temporal constraints imposed by real-time distributed systems. The result is providing an operational framework (TimeSquare) which is able to conduct simulations and formal analysis of EAST-ADL/TADL2 models. Additionally, we provide simple but sufficient ways to specify *trigger conditions* for:

- `Event-triggered functions`. The condition is expressed by a set of ports. It means that at least one data must be received on each port after the last function execution to trigger the function again. This is a very simple way to express the trigger condition and one can use more complex expressions (for instance by directly using a CCSL expression). For all cases, it must be possible to denote the semantics of the triggering condition in CCSL.
- `Time triggered functions`: Because TADL2 is dedicated to specifying timing expressions, we propose to use it to specify time trigger conditions.

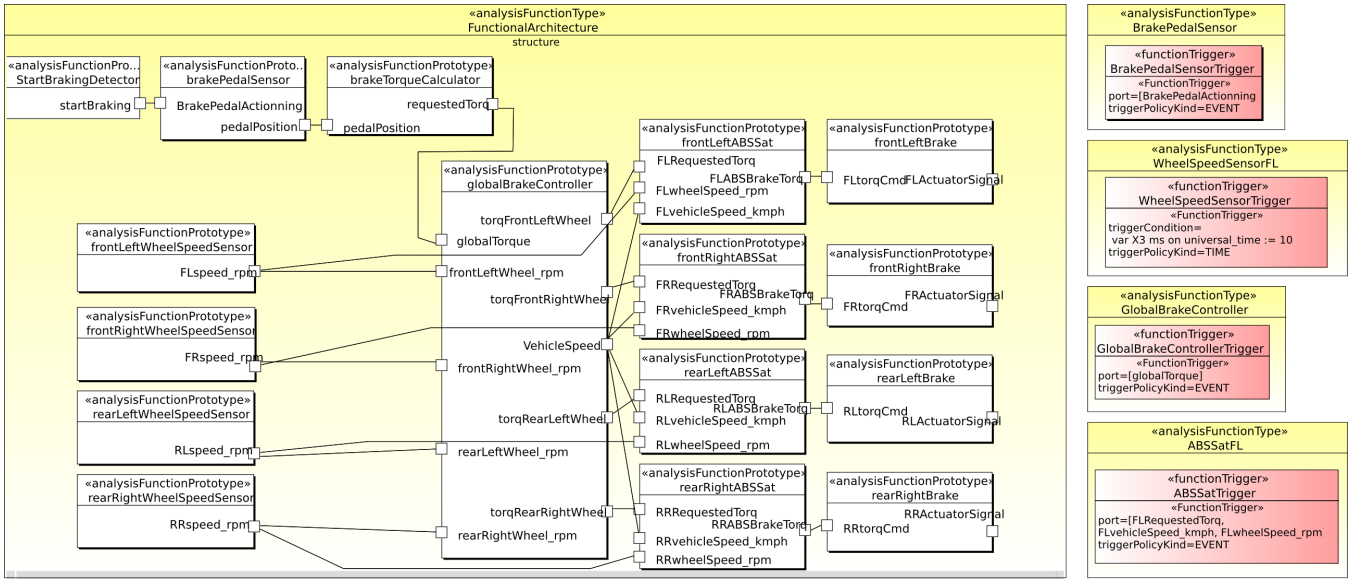

Figure 2. The Brake-By-Wire Example Functional Architecture in EAST-ADL

In the following subsection, we detail TADL2 constructs.

B. TADL2: *Timing Augmented Description Language*

In this section, we introduce TADL2 which supports modeling timing constraints with *Symbolic Timing Expressions* and *Time Bases*. *Symbolic timing expressions* are used to express durations such as *maximum/minimum delay* and *tolerance* with variables. TADL2 provides the ability to define relationships between *TimeBases* along with *Dimension* and *Unit*. For a more detailed TADL2 description please refer to [2].

1) TADL2 *Timing Constraints:* The TADL2 timing constraints constrain the identifiable state changes of the system formulated as *Events*. Usual events are sending/receipt of information through ports. The timing constraints allows events to occur only when some conditions are met. In this paper we consider the following TADL2 timing constraints which are sufficient to represent the timing model of BBW example with symbolic timing expressions and time bases.

- `SynchronizationConstraint` is a constraint on a set of events, which constrains the time duration between the n^{th} occurrence of all the events in the set (*i.e.,*maximum allowed time between the arrival of the event occurrences). All event occurrences must occur within a sliding window specified by the *tolerance* attribute and measured on an explicit timebase.
- `PeriodicConstraint` describes the behavior of an event with a strict periodic occurrence pattern given by the *period* attribute.
- `ReactionConstraint` defines how long after the occurrence of a *stimulus* a corresponding *response* must occur. The causal relation between event occurrences is taken into account.

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

Additionally, the causally-related events are contained as a pair by *EventChain*. Based on *Events* and *EventChains*, it is possible to represent data dependencies and critical execution paths as additional constraints of an EAST-ADL functional architecture model, and to apply timing constraints on these paths.

Some timing constraints are attached to the BBW architecture in Figure 2. One of these constraints is about periodic sensor acquisition. Another constraint is the synchronization constraint that represents the temporal consistency of events. Delays on the ports are represented by reaction constraints. Listing 1 gives some of the event declarations in TADL2 for the EAST-ADL model in Figure 2. For the complete TADL2 specification of the BBW example, please refer to [12].

```
1 Event firstWheelTorqCmd{FunctionalArchitecture::
      frontLeftBrake::FLTorqCmd}
  Event globalTorqueRead {FunctionalArchitecture::
      globalBrakeController::GlobalTorque}
3 Event startBraking {FunctionalArchitecture::
      StartBrakingDetector::startBraking}
  Event firstWheelBrakeActuation {FunctionalArchitecture::
      frontLeftBrake::FLActuatorSignal}
5 Event brakePedalSensorActivation {FunctionalArchitecture::
      brakePedalSensor::BrakePedalActionning}
  Event torqFirstWheel {FunctionalArchitecture::
      globalBrakeController::torqFrontLeftWheel}
7 Event firstWheelSensorAcquisition {FunctionalArchitecture
      ::frontLeftWheelSpeedSensor::WheelSpeedSensorTrigger}
```

Listing 1. Event Declarations in TADL2

For instance, the *brakePedalSensorActivation* and *firstWheelSensorAcquisition* events are attached to the *BrakePedalActionning* port of the *BrakePedalSensor* function prototype and the *WheelSpeedSensorTrigger* time function trigger of the *FrontLeftWheelSpeedSensor* function prototype in Figure 2 respectively.

Listing 2 gives some of the timing constraints in TADL2 for the BBW functional architecture.

```
1  var reactionTimeMin ms on universal_time := 0.0
2  var reactionTimeMax ms on universal_time := 330.0
3  var X3 ms on universal_time := 10.0
4
5  ReactionConstraint tc1a {
6    source startBraking
7    target firstWheelBrakeActuation
8    lower = reactionTimeMin     upper = reactionTimeMax
9    scope pedalPositionWrite , pedalPositionRead ,
10     GlobalTorqueWrite , globalTorqueRead , torqFirstWheel ,
11     requestedTorqFL , FLABSRead, firstWheelTorqCmd
12 }
13
14 PeriodicConstraint tc3a {
15   event firstWheelSensorAcquisition     period = X3
16 }
17
18 ReactionConstraint tc5a {
19   source globalTorqueRead    target torqFirstWheel
20   lower = (reactionTimeMin*0.275)
21   upper = (reactionTimeMax*0.275)
22 }
23
24 ReactionConstraint tc8a {
25  source firstWheelTorqCmd  target firstWheelBrakeActuation
26   lower = 0     upper = (10 ms on ecu1)
27 }
28
29 SynchronizationConstraint tc10 {
30   events firstWheelBrakeActuation ,
31    secondWheelBrakeActuation , thirdWheelBrakeActuation ,
32    fourthWheelBrakeActuation
33    tolerance = (5.0 ms on universal_time)  }
```

Listing 2. Some BBW Timing Constraints in TADL2

All reaction and periodic constraints in Figure 2 are replicated for the four wheels. Listing 2 only shows the constraints for the first wheel (*tc1a*, *tc3a*, *tc5a* and *tc8a*). *tc1a* gives the end-to-end delay between the brake pedal activation and the brake actuation on the front left wheel. The *scope* attribute gives the sequence of events involved in the reaction from pressing the pedal to the brake activation (lines 9-11). *tc3a* specifies that the first wheel sensor is sampled periodically at period $X3$ (line 14-16). *tc5a* specifies the minimum and maximum delays between the receipt of the information of the first wheel speed and the computation of the torque by the *GlobalBrakeController* function. The lower and upper bounds of *tc5a* are computed by using symbolic timing expressions ("reactionTimeMin*0.275" and "reactionTimeMax*0.275" in lines 20-21). *reactionTimeMin* and *reactionTimeMax* are declared as variable timing expressions (lines 1-2). Since *tc5a* is between the input and output ports of the same function prototype (*GlobalBrakeController*), it does not need its *scope* to be specified. *tc10* is about the maximum tolerated time difference among the wheel brake actuations. Its attribute *tolerance* is equal to a value timing expression ("5 ms on universal_time" in line 33). In Listing 2, all constraints except *tc8a* specify timing values based on universal time. The wheel sensors and brake controllers have their own Electronic Computing Units (ECUs). Therefore, they could use their own time bases as a time reference [2].

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

A *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 3 shows TADL2 declarations for *Dimension* and *TimeBase*.

```
1  Dimension physicalTime {
2    Units {
3      micros{factor 1.0 offset 0.0},
4      ms{factor 1000.0 offset 0.0 reference micros},
5      second{factor 1000000.0 offset 0.0 reference micros}
6    }
7    kind Time
8  }
9  TimeBase universal_time {
10      dimension physicalTime  precisionFactor 1.0
11      precisionUnit micros
12 }
```

Listing 3. Declaration of Dimension and TimeBase in TADL2

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 3 - 5). *universal_time* represents an ideal measurement of time on Earth.

The hardware platform of a real-time system consists of sensors/actuators and ECUs. Each ECU in the system has its own temporal reference (*TimeBase*) which is not necessarily (well) synchronized with others. Communications among the ECUs are mainly asynchronous despite the apparition of Time Triggered buses. The drifts between the ECU time bases can be captured with *TimeBaseRelation* (see Listing 4).

```
1  TimeBase ecu1 {
2     dimension physicalTime  precisionFactor 0.1
3     precisionUnit micros
4  }
5  TimeBaseRelation tbr {
6   (1 ms on ecu1) = (1100 micros on universal_time)
7  }
```

Listing 4. TimeBaseRelation Declaration in TADL2

ecu1 is an ECU timebase. *tbr* states that 1 *ms* on *ecu1* is equal to 1100 *micros* on *universal_time* (lines 5-7).

For the modeling of timing constraints we have a TADL2 editor [2]. The textual concrete syntax for TADL2 is tooled by using Xtext [13] which is a framework for the development of programming and domain specific languages. The next section explains how we formally define the semantics of EAST-ADL and TADL2.

IV. Semantics denotation

In order to define the formal semantics of TADL2 and EAST-ADL, we need a formal language which covers synchronous, asynchronous and polychronous aspects. As a consequence, we decided to use CCSL (Clock Constraint Specification Language). CCSL has been first introduced in the annex of the MARTE profile and after several improvements it became a formal language whose associated tool (TimeSquare) is dedicated to the analysis of CCSL specifications. CCSL has been used to give the semantics of models without changing the technological space. It also provides a convenient feedback to the modeler like model animation and timing diagrams linked to models. Please refer to [14] for a detailed description of the language with its semantics and to [10] for an overview of the tool[1].

A. CCSL and ECL

1) CCSL: In CCSL, the main concept is *Clock* which represents a (possibly infinite) totally ordered set of *instants* noted $\mathcal{I}$. Clocks can be seen as events and their instants as event occurrences. They can be logical or physical, dense or discrete. In the remainder of this paper we only consider discrete clocks, rather logical or physical. A dense clock can be used to represent a physical magnitude like physical time. A discrete clock is derived from the dense clock with discretization.

For a discrete clock $c1$, we denote by $c1[k]$ the k^{th} instant of $c1$ where $k \in \mathbb{N}^+$. CCSL also provides *clock constraints* which refer to at least two clocks in order to constrain the respective evolution of their instants.

The CCSL formal semantics can be exploited to process a correct execution, if any. If no correct execution can be processed, TimeSquare indicates the presence of a deadlock. Foundational CCSL constraints are defined in a kernel library. CCSL allows building new libraries for user-defined constraints by composing existing relations in the kernel library or in other user-defined libraries to support constraints for any specific domain.

CCSL is a means of specifying constraints for the evolution of clocks. Constraints can be either a relation or an expression. A relation can be synchronous or asynchronous. CCSL also provides a set of expressions whose goal is to define a new clock based on other clocks. In this paper we do not give all details of the CCSL semantics but we informally describe the necessary material to intuit the language as well as the relations and expressions used in this paper.

[1]available here: `http://timesquare.inria.fr`

Clock relations are based on a reflexive and transitive instant relation named *causality* (or *non strick precedence*) and noted $\preccurlyeq$. From $\preccurlyeq$ four new instant relations are derived: *Coincidence* ($\equiv$), *Strict precedence* ($\prec$), *Independence* ($\|$), and *Exclusion* ($\#$).

A clock relation is the generalization of an instant relation on all instants of a clock. For instance, $(a \boxed{\preccurlyeq} b)$ denotes that a is non-strictly faster than b, that is for all positive natural numbers k, the k^{th} instant of a precedes or is coincident with the k^{th} instant of b. The *strick precedence* clock relation (denoted $\boxed{\prec}$) between two clocks a and b is asynchronous and specifies that for all positive natural numbers k, the k^{th} instant of a occurs before the k^{th} instant of b: $\forall k \in \mathbb{N}^+, a[k] \prec b[k]$. Another example is the *coincidence* relation (denoted $\boxed{=}$) between two clocks a and b that imposes a stronger synchronous dependency: the k^{th} instant of a must be coincident with the k^{th} instant of b: $\forall k \in \mathbb{N}^+, a[k] \equiv b[k]$. The same mechanism applies for all relations. Informally, the *exclusion* relation (denoted $\boxed{\#}$) between two clocks a and b specifies that no instants of the clock a coincide with the instants of the clock b. The *independence* relation between two clocks a and b states that there is no relation between the instants of these two clocks. It is possible to build new relations based on existing relations. For instance, the *alternatesWith* relation (denoted $\boxed{\sim}$) between the two clocks a and b is built upon the *strict precedence* relation and specifies that the instants of the clock b are the interleaving instants of the clock a ($\forall k \in \mathbb{N}^+, a[k] \prec b[k] \prec a[k+1]$).

Expressions are directly defined on clocks. For instance, the *isPeriodicOn* expression takes three parameters: (i) a clock specifying the super clock, (ii) a positive natural number specifying the *period* and (iii) a positive natural number specifying the *offset*. It results in a clock which is a subclock of the super clock and whose instants are always separated by the *period* instants of the super clock. Moreover, the first instant of the resulting clock coincides with the *offset*th instant of the super clock. Another expression used in this paper is *delayedFor*. It takes three parameters: (i) a reference clock which represents the clock we want to delay, (ii) a counter clock which represents the clock on which the delay is counted and (iii) a positive natural number which specifies the number of ticks of the counter clock the reference clock is delayed for.

A CCSL specification is the conjunction of a set of constraints. Since synchronous and asynchronous relations are used conjointly, the execution of a CCSL specification is a partially ordered set of coincidence-equivalence classes of instants [6].

2) ECL: ECL (Event Constraint Language) [7] is a lightweight extension of OCL (Object Constraint Language) [8] with the explicit notion of events (clocks) and constraints (clock relations and expressions) borrowed from CCSL. From an ECL specification expressed for a metamodel *MM1*, it is

possible to automatically generate a QVTo transformation which transforms any model that conforms to *MM1* to CCSL specifications. CCSL is able to provide the semantics of a specific model. In the following we use ECL to denote the semantics of both EAST-ADL & TADL2 metamodels and consequently every model that conforms to these metamodels.

ECL exploits the *def* construct of OCL to declare *Events* at the metamodel level. To constraint these event(s), ECL extends the OCL invariants with the ability to add CCSL like constraints on these events. Consequently, in a specific context (for a specific metaclass) some constraints can be added at the metamodel level. Please note that the notion of *Event* in ECL is not the same with the notion of *Event* in TADL2. ECL is quite intuitive for the readers familiar to OCL. The details of the language can be found in [7]. The semantics of EAST-ADL and TADL2 are given with ECL in Sections IV-B and IV-C.

B. EAST-ADL *semantics*

In this section we present how the informal presentation in Section III-A can be equipped with a formal semantics given in ECL. The first step is to define the ECL events which are relevant for the EAST-ADL semantics definition. For instance, the *start* and *stop* events of a function prototype must be made explicit. The trigger of a function prototype is also important. Finally, the writing of data on an 'out' flow port and the reading of a value on an 'in' port should also be made explicit. These are the minimum relevant events which are needed to specify the behavioral semantics of EAST-ADL (see Listing 5[2]).

```
context FunctionTrigger
    def activate : Event(self) = GenericEvent

context FlowPort
    def if ((self.direction) = PortDirection::OUT):
      write : Event(self) = produceEvent
    def if ((self.direction) = PortDirection::IN):
      read : Event(self) = ConsumeEvent

context AnalysisFunctionPrototype
    def start : Event(self) = StartEvent
    def stop : Event(self) = StopEvent
```

Listing 5. Behavioral Invariants for Some EAST-ADL Elements

Once the relevant events are defined, we have to constrain their respective evolution. At this point some choices have to be made since the informal semantics of EAST-ADL has some ambiguities. For instance, nothing is said about the re-entrance of function prototypes in the EAST-ADL specification. Here we provide only some of the constraints that fix such ambiguities. In Listing 6, we explicitly define the function prototypes as being non re-entrant (*i.e.,* start and stop alternates).

[2]Please note that to avoid the stereotype navigation, we provide the ECL specification for the EAST-ADL domain model. The version based on the profile is available here: `http://timesquare.inria.fr/ecl/EASTMoC/`

```
context AnalysisFunctionPrototype
  inv nonReEntrance:
      Relation Alternates(self.start, self.stop)
```

Listing 6. Behavioral Invariants for Non Re-entrance of EAST-ADL Function Prototype

In the *nonReEntrance* invariant we directly use the *alternatesWith* clock relation of CCSL to state that the next start of the function prototype cannot occur before the execution of the function prototype ends ($\forall k \in \mathbb{N}^+, start[k] \prec stop[k] \prec start[k+1]$).

Another example concerns the function triggers with the *event* triggering policy (see the *BrakePedalSensorTrigger*, *GlobalBrakeControllerTrigger* and *ABSSatTrigger* function triggers in Figure 2). In Listing 7, we define the *event* triggering policy as being activated once at least one data has been provided on each input port referenced by the *port* collection of the function trigger.

```
context FunctionTrigger
  inv EventTriggerRule:
     (self.triggerPolicyKind = TriggerPolicyKind::EVENT)
          implies
  let doIt : Event = Expression Sup(self.port.read) in
  Relation  Coincides(doIt, self.activate)
```

Listing 7. Behavioral Invariants for the Event Trigger Policy based on Input Port Reading

The *EventTriggerRule* invariant states that if the triggering policy kind is "Event" (line 3), then the slowest input port of the function prototype coincides with the activation of the function prototype ($\forall k \in \mathbb{N}^+, dolt[k] \equiv activate[k]$). Please note that *Sup* is a CCSL clock expression creating a clock which coincides with the slowest clock of its input clocks ("d = sup(a, b)" is the fastest clock slower than both a and b: $\forall k \in \mathbb{N}^+, d[k] \equiv b[k]$ if $a[k] \prec b[k]$, $d[k] \equiv a[k]$ otherwise).

All other invariants for EAST-ADL are provided in the web page (http://timesquare.inria.fr/ecl/EASTMoC/). A QVTo transformation is automatically generated from the whole set of invariants so that any EAST-ADL model can be transformed to its corresponding CCSL model. It is then possible to simulate and analyze the EAST-ADL models. In addition, the EAST-ADL semantics is mandatory to enable the analysis of the TADL2 timing constraints on top of EAST-ADL models.

C. TADL2 *semantics*

We denote the TADL2 semantics with ECL based on the TADL2- CCSL mappings given in Table I.

These mappings are quite direct since TADL2 has been inspired by CCSL. However, some parts of the TADL2 semantics have to be clarified. For instance, an event chain specifies a set of causality for related events if the event chain is not used in the context of a reaction constraint. If it is used in a reaction constraint, the event chain means that in the *scope* started by the reaction constraint stimulus, a sampling chain must be observed to consider the reaction occurrence that results in the response.

Table I
MAPPING SUMMARY FROM TADL2 TO CCSL.

TADL2	CCSL	Remark
Dimension	CCSL Dense Clock and *Discretize* expression	Each *Dimension* represents a physical reference and is equivalent to a dense clock in CCSL. To ease the mapping of time-bases, a Dimension is also mapped to a *Discretize* expression that discretizes the dense clock according to its default unit factor
Event	CCSL Discrete Clock	Each TADL2 *Event* is associated with a CCSL *Discrete Clock*. The CCSL *Clock* represents the set of instants at which the concerned TADL2 *Event* occurs.
TimeBase	*Periodic* expression	The precision factor and precision unit are used to compute the period according to the dimension referred by the timebase.
Timing Expression	*Periodic* expression	The *Timebase*, *Unit* and *value* given in a *TimingExpression* are used to compute the *period* and *offset* of a CCSL *Periodic* expression.
TimeBase Relation	*Coincides* relation	A *coincides* relation between the clocks for the timebases to which the timebase relation refers is created (*i.e.,* between the periodic expressions in CCSL).
EventChain	Precedence Relation	if an *EventChain* in TADL2 is not used in the context of a *ReactionConstraint*, it contains causally related events. It is mapped to a set of *causes* relations in CCSL.
Timing Constraints	Relations declared in the CCSL Library	Each TADL2 timing constraint (*Reaction*, *Synchronization*, *Periodic*, etc.) is mapped to a set of CCSL clock relations and expressions declared either in the CCSL built-in library or in any dedicated CCSL library for TADL2.

We obtain a set of ECL invariants. In Listing 8, an event chain invariant is guarded by the fact that the event chain is not used in the context of a reaction constraint (lines 4 to 7). For each pair in the event chain there is a non strict precedence relation (line 8).

```
1 context EventChain
2  inv eventChainCausality:
3   --if not used in a reaction constraint
4   (not self.oclContainer().oclAsType(Timing).constraints
5   ->select(te | te.oclIsKindOf(ReactionConstraint))
6   .oclAsType(ReactionConstraint)->exists(
7   rc | rc.scope = self)) implies
8     (Relation Causes(self.segment.tadlEvent))
```

Listing 8. One of the Behavioral Invariants for the TADL2 Event Chain Semantics

Another mapping in Table I is the *Dimension* mapping. A *Dimension* represents a physical magnitude used as a reference in the system description (*e.g., Time*, *Angle*, *Distance*). Such physical references are represented in CCSL by a dense clock with a base unit. The only thing CCSL can do on a dense clock is the discretization. To do so, we chose to use the first declared unit of a dimension as a base unit. In addition, because CCSL constraints deal with discrete clocks, we discretize the dense clock to provide a discrete reference for the dimension. In the following we show the CCSL code generated by the QVTo transformation for the *physical_time* dimension with the *universal_time* timebase given in Listing 3.

$$DenseClockType\ physicalTime \quad (1)$$
$$(baseunit\ =\ micros); \quad (2)$$
$$Clock\ dense\_universalTime : physicalTime; \quad (3)$$
$$universal\_time = \quad (4)$$
$$dense\_universalTime\ \texttt{discretizedBy}\ 1.0; \quad (5)$$

Eqs.1-3 define a dense clock (*dense_universalTime*) of the physical time dimension. In Eqs. 4-5 the discrete clock *universal_time* is created for the *micro* reference Unit of the *universal_time* TimeBase.

Each TADL2 timing constraint (*Reaction*, *Synchronization*, *Periodic*, etc.) is mapped to a set of CCSL clock relations and expressions declared either in the CCSL built-in library or in any dedicated CCSL library for TADL2. In Listing 9 we give the ECL invariant for the TADL2 synchronization constraint.

```
1 context SynchronizationConstraint
2  inv synchronisation:
3   let fastest:Event=Expression Inf(events.tadlEvent) in
4   let slowest:Event=Expression Sup(events.tadlEvent) in
5   let tolerance : Integer = self.tolerance.value *
6         self.tolerance.unit.factor
7   let delayedFastest:Event=Expression DelayFor(fastest,
8         self.tolerance.timeBase.ref,
9         tolerance) in
10   Relation Precedes(slowest, delayedFastest)
```

Listing 9. Part of the Behavioral Invariants for the TADL2 Synchronization Constraint Semantics (navigation simplified)

First, we get the fastest/slowest events among all input events for the synchronization constraint (lines 3-4). Please note that *Sup* is a CCSL clock expression creating a clock which coincides with the slowest clock of its input clocks and *Inf* is a CCSL clock expression creating a clock which

coincides with the fastest clock of its input clocks. In the synchronization constraint the *tolerance* attribute is specified by a timing expression with a *value*, a *unit* and a *timebase*. *tolerance* is then computed according to the value (line 5) and the factor of the used unit with regard to the reference timebase unit (line 6). *tolerance* specifies the delay of the fastest event counted on the timebase (line 7-9). Finally, the slowest event should not occur after this delay (line 10). In the following we give the CCSL code generated by the QVTo transformation for the *tc10* synchronization constraint in Listing 2.

$$
\begin{aligned}
fastest\_tc10 \;=\; & \mathsf{Inf}(firstWheelBrakeActuation,\\
& secondWheelBrakeActuation,\\
& thirdWheelBrakeActuation,\\
& fourthWheelBrakeActuation)\\
slowest\_tc10 \;=\; & \mathsf{Sup}(firstWheelBrakeActuation,\\
& secondWheelBrakeActuation,\\
& thirdWheelBrakeActuation,\\
& fourthWheelBrakeActuation)\\
& \mathsf{Integer}\ tolerance\_tc10 = 5000\\
slowest\_tc10 \;& \boxed{\prec}\; (fastest\_tc10\ \mathsf{delayedFor}\\
& tolerance\_tc10\ \mathsf{on}\ universal\_time)
\end{aligned}
$$

The QVTo transformation is able to create a CCSL specification from any TADL2 model and EAST-ADL model. The architecture and timing constraints can then be analyzed with TIMESQUARE. It is important to have the formal semantics of both languages since a minor change in the structural part (mainly in the triggering condition) may lead to the violation of timing constraints (see Section V).

V. EXECUTING TADL2 SPECIFICATIONS WITH TIMESQUARE

In this section we present the simulation results of the whole BBW specification (*i.e.,* EAST-ADL + TADL2). The simulations have been done under two different configurations of the system. The first configuration (*Time Triggered BBW*) is fully time triggered and the second one (*Time/Event Triggered BBW*) mixes time and event triggerings. For both configurations we use a unique clock (the universal time) but it is of course possible to use various timebases as supported by TADL2.

A. Time Triggered BBW

The first configuration considers the BBW example as totally time triggered. It means that all function prototypes are triggered periodically and their periods are specified in the TADL2 specification. In addition, each function prototype has a fixed execution time specified by some TADL2 reaction constraints (see Table II for the period and execution time values). Finally, a global end-to-end reaction constraint (the *tc1a* reaction constraint in Listing 2) is put from a brake action on the brake pedal (the *startBraking* port in the *StartBrakingDetection* function prototype) to the corresponding brake actuations (the *FLActuatorSignal* ports in the *XXBrake* function prototypes). In the requirements of the BBW example, the interval of the global end-to-end reaction constraint is set to [0 ; 330] ms.

Table II
TIMING CONFIGURATION OF THE BBW EXAMPLE.

Function Prototype	Period	Execution Time
Wheel Speed Sensors (XXWheelSpeedSensor)	20ms	10ms
BrakePedalSensor	20ms	10ms
BrakeTorqueCalculator	50ms	40ms
GlobalBrakeController	90ms	80ms
ABS Function Prototypes (XXABSSat)	40ms	30ms
Brakes (XXBrake)	40ms	10ms

One of the benefits of our approach is to simulate the system with timing constraints based on the specified EAST-ADL and TADL2 semantics in TIMESQUARE that provides timing diagrams and model animation. Timing diagrams and model animation help to understand causal and temporal relationships of the system under simulation. During the simulation with the values in Table II a deadlock occurred and the simulation has been stopped by TIMESQUARE. This is represented in Figure 3 which shows the timing diagram of the event chain in the *scope* of the global reaction constraint *tc1a* obtained by the simulation. The deadlock is due to a lack of synchronization between the brake torque calculator and the global brake controller as highlighted in Figure 3.

Since it is a synchronization problem, we decided to change the activation condition of the *GlobalBrakeController* function prototype from time triggered to event triggered (change of the triggering policy in EAST-ADL).

B. Time/Event Triggered BBW

For the second configuration, we changed only the triggering policy kind and the triggering condition of the *GlobalBrakeController* function prototype. We decided that *GlobalBrakeController* is triggered every two updates of its *globalTorque* input port. Note that we do not want *GlobalBrakeController* to be triggered every time its *globalTorque* port is updated since the execution time of *GlobalBrakeController* is bigger then the minimum time between two updates of the *globalTorque* port. If we did so, the non reentrance semantics chosen for EAST-ADL would lead to a deadlock again. In this configuration, since the period of the *BrakeTorqueCalculator* function prototype is 50 ms, the *globalTorque* port is updated every 50 ms. Therefore, *GlobalBrakeController* is now triggered every 100 ms instead of the previous period of 90 ms (see Table II). However,

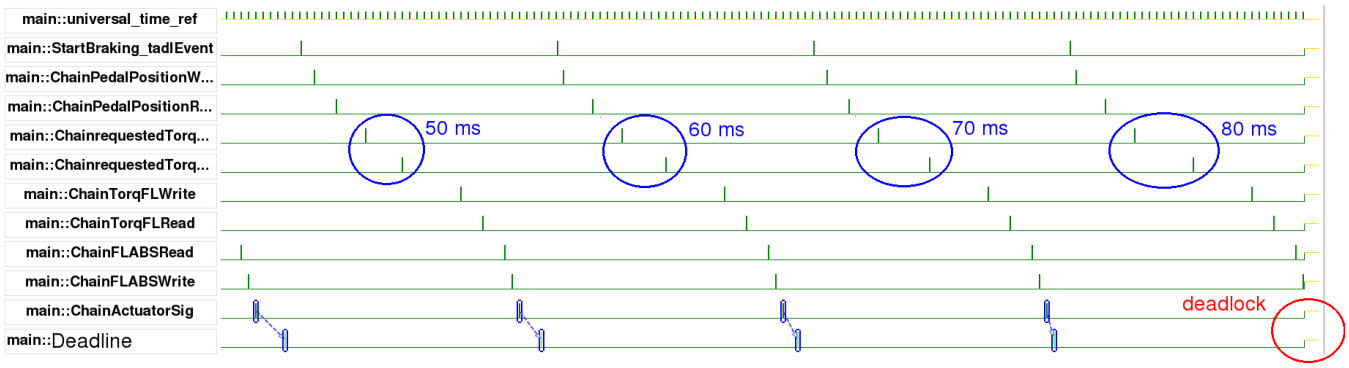

Figure 3. A simulation of the Brake-By-Wire Time Triggered Configuration

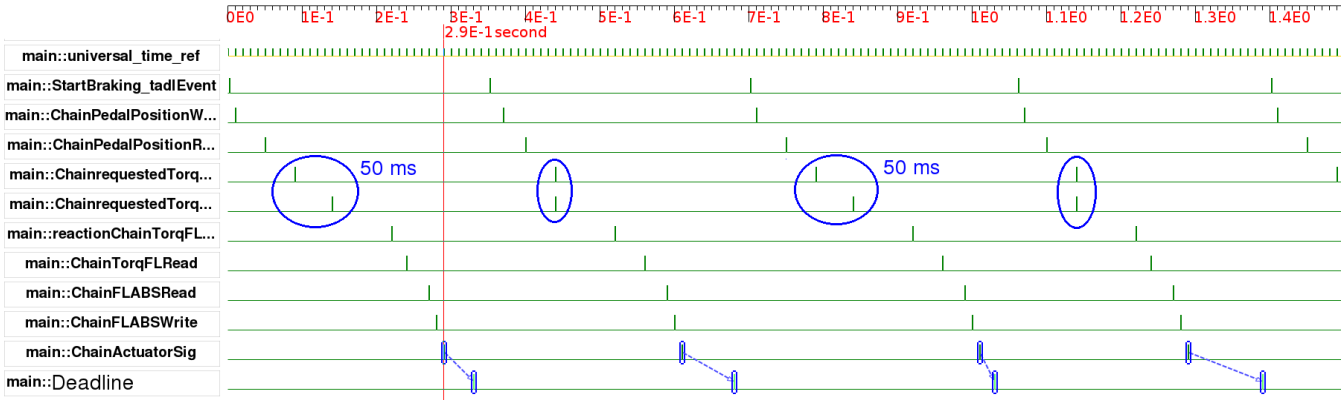

Figure 4. A simulation of the Brake-By-Wire Time/Event Triggered Configuration

the event trigger activation of the *GlobalBrakeController* function prototype helps to keep the synchronization of the consignment between the *BrakeTorqueCalculator* and *GlobalBrakeController* function prototypes as highlighted in Figure 4. *GlobalBrakeController* is triggered every two updates of the *globalTorque* port. Therefore, the maximum time between the *globalTorque* port update (considered in the event chain scope of the reaction constraint) and the *GlobalBrakeController* activation is then the period of the *BrakeTorqueCalculator* triggering. With this configuration, the deadlock is not observed anymore. The global end-to-end reaction constraint is shows by the blue arrows in Figure 4.

We believe this approach gives confidence on the correctness of the system and features like model animation and timing diagram help to understand the synchronizations in the system. Beyond this, the approach provides the semantics of TADL2 and EAST-ADL with ECL. The semantics helps clarify ambiguities in TADL2 and EAST-ADL. The ambiguities are not resolved in an opaque way by the tool implementation. Moreover, the approach outlines the possibility to denote the behavioral semantics of two different languages in order to provide a global heterogeneous simulation. We showed that it is possible to detect some problems in timing constraints via simulation in TIMESQUARE. In order to fully validate the system an exhaustive simulation is needed. The formal semantics of EAST-ADL and TADL2 in ECL can be used as a reference for transformations to other analysis tools as described in [15].

VI. 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. [17] 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 [18] presents a general modelling language for Quality of Service (QoS). The presented modelling language is based on enterprise architecture modeling. It uses a time model where different clocks can be defined. These clocks are related to the chosen standard clock with skew, drift and offset. They are

indirectly related to each other via the chosen standard clock. Zschaler [19] presents QML/CS, a specification language used to model non-functional properties of component-based systems including response time.

In the context of EAST-ADL, several approaches based on timed automata have been proposed. In [20], Qureshi et al. presented timed automata templates for EAST-ADL timing constraints. These modeling templates capture various error scenarios based on EAST-ADL architectural models. While Kang et al. [21] have addressed the functional modeling for EAST-ADL models using Uppaal, Enoiu et al. [22] have addressed a limited aspect of time modeling for EAST-ADL models. However, none of the above works address the analysis of timing constraints with the explicit notion of time base and symbolic timing expressions.

VII. Conclusion

In this paper we presented an approach for the conjoint simulation and analysis of EAST-ADL and TADL2 specifications. We denoted the semantics of both languages by using ECL. It allows for an automatic transformation to CCSL for the simulation in TimeSquare.

We used a real industrial example proposed by Volvo Technology in the TIMMO-2-USE project [4] to show the capability of our approach with the tool support for handling timing behavior of industrial systems. The example highlighted the need for a precise semantics of both EAST-ADL and TADL2 for an accurate timing analysis. It also shows the benefits of an early simulation to understand the synchronizations that exist directly at the model level. 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 study the semantic impact of the interaction among languages.

References

[1] ATESST (Advancing Traffic Efficiency through Software Technology). (2008, March) East-ADL2 specification. [Online]. Available: http://www.atesst.org, 2008-03-20

[2] M.-A. Peraldi-Frati, A. Goknil, J. Deantoni, and J. Nordlander, "A timing language for specifying multi clock automotive systems: The timing augmented description language v2," in *ICECCS'12*, July 2012.

[3] M.-A. Peraldi-Frati, A. Goknil, M. Adedjouma, and P. Y. Gueguen, "Modeling a bsg-e automotive system with the timing augmented description language," in *ISoLA (2)*, 2012, pp. 111–125.

[4] ITEA TIMMO-2-USE Project. (2012, July). [Online]. Available: http://timmo-2-use.org/

[5] OMG, *UML Profile for MARTE, v1.0*, Object Management Group, November 2009, formal/2009-11-02.

[6] C. André, F. Mallet, and R. de Simone, "Modeling time(s)," in *MoDELS'07*, 2007, pp. 559–573.

[7] J. Deantoni and F. Mallet, "ECL: the Event Constraint Language, an Extension of OCL with Events," INRIA, Research Report RR-8031, Jul. 2012. [Online]. Available: http://hal.inria.fr/hal-00721169

[8] OMG, Object Constraint Language. [Online]. Available: http://www.omg.org/spec/OCL/2.0/

[9] R. Dvorak, "Model transformation with operational qvt," in *EclipseCon'08*, 2008.

[10] J. DeAntoni and F. Mallet, "Timesquare: Treat your models with logical time," in *TOOLS (50)*, 2012, pp. 34–41.

[11] The East-EEA Project. (2004) Definition of language for automotive embedded electronic approach, technical report, itea. [Online]. Available: http://timmo-2-use.org/

[12] (http://www-sop.inria.fr/members/Arda.Goknil/BBW2-tadl2.pdf) BBW Spec in TADL2.

[13] XText. [Online]. Available: http://www.eclipse.org/Xtext/

[14] C. André, "Syntax and Semantics of the Clock Constraint Specification Language (CCSL)," INRIA, Research Report RR-6925, 2009. [Online]. Available: http://hal.inria.fr/inria-00384077

[15] L. Yin, F. Mallet, and J. Liu, "Verification of marte/ccsl time requirements in promela/spin," in *ICECCS'11*, 2011, pp. 65–74.

[16] F. Klein and H. Giese, "Joint structural and temporal property specification using timed story scenario diagrams," in *FASE'07*, 2007, pp. 185–199.

[17] A. Alfonso, V. A. Braberman, N. Kicillof, and A. Olivero, "Visual timed event scenarios," in *ICSE'04*, 2004, pp. 168–177.

[18] J. Aegedal, "Quality of service support in development of distributed systems," *PhD Thesis*, 2001.

[19] S. Zschaler, "Formal specification of non-functional properties of component-based software systems," *Software and Systems Modeling*, vol. 9, no. 2, pp. 161–201, 2010.

[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*, 2012, pp. 303–318.

[21] E.-Y. Kang, P.-Y. Schobbens, and P. Pettersson, "Verifying functional behaviors of automotive products in east-adl2 using uppaal-port," in *SAFECOMP'11*, 2011, pp. 243–256.

[22] E. P. Enoiu, R. Marinescu, C. Seceleanu, and P. Pettersson, "Vital : A verification tool for east-adl models using uppaal port," in *ICECCS'12*, July 2012.