ITEA 2

INFORMATION TECHNOLOGY FOR EUROPEAN ADVANCEMENT

ITEA 2 - 09033

TIMMO-2-USE

Timing Model – Tools, algorithms, languages, methodology, USE cases

Report type	Deliverable D11
Report name	Language syntax, semantics, metamodel V2
Report status	Public
Version number	Version 1.2
Date of preparation	2012-08-30

TIMMO-2-USE Partners

AbsInt Angewandte Informatik GmbH

Arcticus Systems AB

Chalmers University of Technology

Continental Automotive GmbH

Delphi France SAS

dSpace GmbH

INCHRON GmbH

Institute National de Recherche en Informatique et Automatique INRIA

Mälardalen University

Rapita Systems Ltd, UK

RealTime-at-Work

Robert Bosch GmbH

Symtavision GmbH

Technische Universität Braunschweig

University of Paderborn

Volvo Technology AB

Project Coordinator

Dr. Daniel Karlsson

Volvo Group Trucks Technology

Advanced Technology & Research

Dept 6260, M2.7

SE-405 08 Göteborg

Sweden

Tel.: +46 31 322 9949

Email: Daniel.B.Karlsson@volvo.com

Authors

Hans Blom, Volvo Technology AB

Dr Lei Feng, Volvo Technology AB

Dr Henrik Lönn, Volvo Technology AB

Dr Johan Nordlander, Chalmers University of Technology

Stefan Kuntz, Continental Automotive GmbH

Dr Björn Lisper, Mälardalen University

Dr Sophie Quinton, Technische Universität Braunschweig

Dr Matthias Hanke, Technische Universität Braunschweig

Dr Marie-Agnès Peraldi-Frati, Institute National de Recherche en Informatique et Automatique INRIA

Dr Arda Goknil, Institute National de Recherche en Informatique et Automatique INRIA

Dr Julien Deantoni , Institute National de Recherche en Informatique et Automatique INRIA

Gilles Bertrand Defo, University of Paderborn

Kay Klobedanz, University of Paderborn

Mesut Özhan, INCHRON GmbH

Olha Honcharova, AbsInt Angewandte Informatik GmbH

Document History

Version	Date	Description
1.2	2012-08-30	Public version. Corrected typo in chapter 3.6.2.
1.1	2012-08-20	Typos corrected in chapter 3.6.7 and 12.1.1.
1.0	2012-07-10	Result from TIMMO-2-USE WP2

Table of contents

1 Executive Summary

Work package 2 (WP2) of the TIMMO-2-USE project defines a language called TADL2 (Timing Augmented Description Language Version 2) for advanced handling of timing information. It is based on the existing TADL, developed in the TIMMO project.

Project requirements and use cases from WP1 serve as the definition of the needed updates. Additional features are documented as the result of interaction with other project work packages.

Harmonization with EAST-ADL Timing from the MAENAD project and the AUTOSAR 4.0 Timing Extensions has been performed. This gave an updated and integrated syntax and semantic definition of TADL2. These results may lead to improvement suggestions for these related standards.

Semantics of mode dependency to be used in timing constraints have been defined.

Syntax for symbolic time expressions will allow for advanced expressions in constraints; this also includes the use of multiple time bases.

In close cooperation with WP3 (Tools and Algorithms), the need for introduction of probabilistic timing information is included for TADL2.

2 Introduction

The main goal of WP2 in TIMMO-2-USE is to provide a modeling framework for timing information in the early steps of the design process of embedded real-time systems in the automotive industry. Following this idea, it is expected that test effort, development time, and cost will be reduced. A language is developed to formalize the information related to timing; this language is the Timing Augmented Description Language Version 2, TADL2.

The scope of TADL2 is the definition of timing constraints and the description on how these relate to events exposed by a system. The system is described by models using EAST-ADL and AUTOSAR concepts. This concept of relating timing constraints with events that can be observed in a system described by EAST-ADL and AUTOSAR is shown in Figure 1.

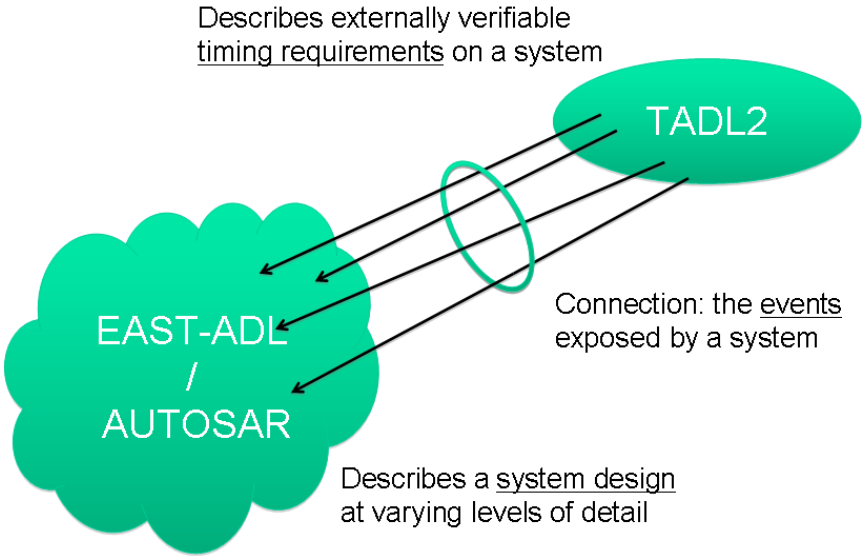

Figure 1. Events connect TADL2 to the system design modeled with EAST-ADL and AUTOSAR.

2.1 Parts of this Deliverable

The deliverable D11 consists of two parts:

- This document with introduction and background, description of language syntax and semantics. And new concepts in TADL2 are described. In this document Appendix A gives the justification for symbolic time extensions and Appendix B presents the TADL2 metamodel.
- Metamodel, which is supplied separately as an XMI export of the metamodel in Enterprise Architect. This metamodel should be imported in an Enterprise Architect model of EAST-ADL and AUTOSAR 4, see [3] and [8]. The elements in the metamodel are documented, which facilitates the generation of documentation from the metamodel

2.2 Motivation and Overview of TADL2

TADL was first defined in the TIMMO project [1]. These results were integrated in EAST-ADL by the ATESST2 project [2]. The metamodel was contained in the package Timing [3]. In this process there was some renaming of the events connected to the EAST-ADL function concepts. This was due to the renaming of ADLFunctionType to FunctionType in EAST-ADL. Hence the “ADL” part of the name was also removed from the events.

The set of different events defined for particular flow ports in EAST-ADL were replaced by one concept EventFunctionFlowPort. One single concept EventFunctionClientServerPort replaced two separate concepts for client and server ports.

These events remain in use by TADL2, however it has been realized that the semantics of the specific events belong to the model describing the structure of the solution or proposed solution. This semantics is therefore not defined by TADL2. Rather, only the concepts of abstract events and event chains, and the definition of constraints belong to TADL2, see Figure 2. To avoid modification of EAST-ADL and AUTOSAR, the references are consistently defined from TADL2 to EAST-ADL and AUTOSAR respectively.

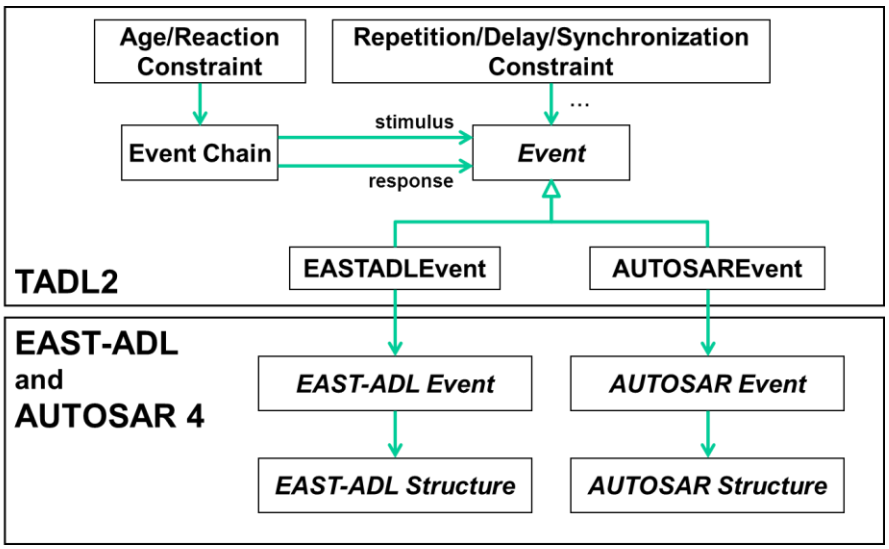

Figure 2. Different events are defined to refer to structural elements of EAST-ADL and AUTOSAR.

Since the TIMMO project ended, there has been an extension added in AUTOSAR that deals with timing [4]. AUTOSAR belong in the Implementation abstraction layer as defined by EAST-ADL. However, TADL2 still spans all the abstraction levels as defined in EAST-ADL. This is to provide a common semantics for all levels. See Figure 3.

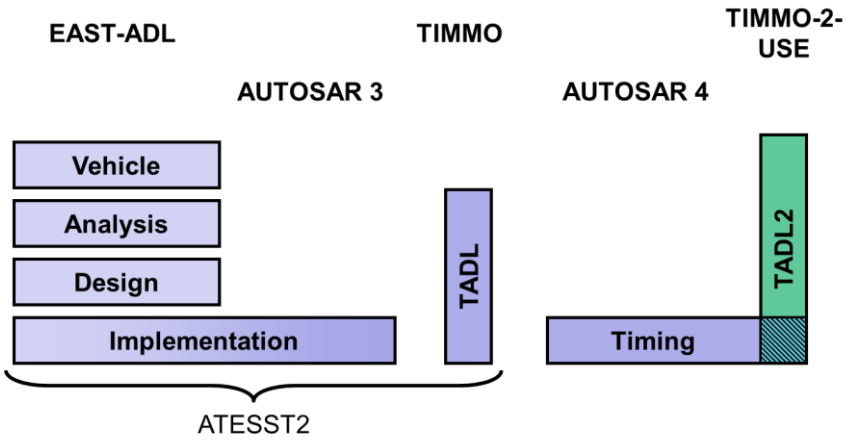

Figure 3. This figure shows how TADL and TADL2 spans the abstraction levels of EAST-ADL where AUTOSAR is available in the implementation level. AUTOSAR 4 includes a timing extension.

TADL2 as developed by TIMMO-2-USE differs from TADL from TIMMO in a number of ways:

- The semantics of TADL is updated w.r.t. timing constraints and mode dependency.
- TADL2 adds new concepts for multiple time bases, symbolic time expressions and probabilistic timing.

Details on the differences between the timing constraints in TADL2 and EAST-ADL and AUTOSAR Timing Extensions are described in a later chapter.

3 TADL2 Specification of Syntax and Semantics

This chapter defines the syntax and semantics for TADL2. By the semantics of a constraint we mean the unambiguous interpretation of the conditions that must hold for the constraint to be satisfied.

As mentioned in the introduction, the natural border of responsibilities between the TADL2 constraint language and the models it constrains goes through the definition of events. This chapter therefore begins with a section on the semantics of events, and the assumptions TADL2 makes on event occurrences in running systems or simulated system models. An introduction to the notations used then follows, as well as descriptions of the classes used to express events and arithmetic expressions, before the topic moves on to the different constraints and their semantic interpretations.

3.1 Events and occurrences

An event denotes a distinct form of state change in a running system, taking place at distinct points in time called *occurrence* of the event. That is, a running system can be observed by identifying certain forms of state changes to watch for, and for each such observation point, noting the times when changes occur. This notion of observation also applies to a hypothetical predicted run of a system or a system model — from a timing perspective, the only information that needs to be in the output of such a prediction is a sequence of times for each observation point, indicating the times that each event is predicted to occur.

In system models, events appear syntactically as names indicating the state changes of interest. Semantically, an event name is a variable standing for some statically unknown *set of occurrences*. Note that this connection is purely conceptual; occurrences never exist concretely in any system model as they are a purely semantic notion representing the state changes that can be observed when a system is executed, or simulated, or perhaps only mathematically predicted.

In the TADL2 semantics, an occurrence is basically just a timestamp — a real value of the SI unit seconds. However, a few constraints will also make use of an additional piece of information for each occurrence: a *color* annotation set by the producer of an event, which a constraint may utilize to identify related occurrences (those that are causally connected, for example). Colors are drawn from some abstract, possibly infinite type whose only restriction is that it must support an equality test on its values. This reveals that semantically an occurrence is really a (timestamp, color) pair, although in most contexts, the color component may be ignored and the occurrence simply identified with its timestamp.

3.2 Notations

Syntactic and semantic objects like events, constraints and time, will be referenced by simple variable names in this chapter; for example,

a constraint *c*, an occurrence *x,* or a set of occurrences *Y*. To denote attributes of such an object an object-oriented notation will be used, where for example *c.jitter* means the attribute *jitter* of constraint *c*.

If *ev* is an event name, its dynamic occurrences are referenced in constraint definitions by reference to a variable *x* that is a member of *ev*. Variable *x* thus stands for a particular occurrence, which in all arithmetic contexts simply means its timestamp value. When the color annotation of an occurrence *x* is intended, it is explicitly written as *x.color*.

The following standard logical connectives and set operators on will be used in the formal semantic definitions:

$c_1 \wedge c_2$	constraints c_1 and c_2 *both* true
$c_1 \Rightarrow c_2$	implication (c_1 is false, or c_1 and c_2 are both true)
$c_1 \Leftrightarrow c_2$	equivalence (c_1 and c_2 have the same truth value)
$\forall x : c$	c is true for all possible values of x
$\exists x : c$	c is true for at least one value of x
$\|Y\|$	the number of elements in set Y
$X \cap Y$	the elements that are in both X and Y
$X \cup Y$	the elements that are in either X or Y
$X \setminus Y$	the elements in X that are not elements of Y
$\mathbf{C}(X)$	complement (all elements that are not in X)
$X \subseteq Y$	all elements in X are also in Y
$x \in Y$	x is an element of Y
$\forall x \in Y : c$	for each x in Y, c is true
$\exists x \in Y : c$	there is an x in Y such that c is true

Because occurrences are ordered via their timestamps, a set of occurrences might just as well be thought of as a *sequence* of occurrences. This gives rise to a *sub-sequence* relation between such sets, as well as the notion of *indexing*:

$X \leq Y$	X is a sub-sequence of Y
$x = Y(i)$	x is element number i in Y (counting from 0)

Just as events are semantically understood as sets of occurrences, *time intervals* can also be seen as occurrence sets, with the important characteristic that any non-empty interval will contain infinitely many dense occurrences. The following interval operators will play a role in coming definitions:

$[x..Y]$	the set of all occurrences from x to the nearest greater element of Y
$[X..Y]$	the set of all occurrences between any x in X to its nearest greater element in Y (may result in alternating intervals and gaps)
$[X]$	the set of all occurrences between the smallest and greatest occurrences in X
$\lambda(X)$	the total length of all continuous intervals in X, such that if X contains all occurrences between times 3 and 5, as well as between times 6 and 8, $\lambda(X)$ is 4.

It should be noted since both events and intervals correspond to sets of occurrences, expressions like $X \cap Y$ or $X \subseteq Y$ are perfectly valid, even if X is an event and Y stands for an interval.

A full definition of the interval and sequence operators above is contained in a separate technical paper [5]. For a thorough

explanation on the logic and set notations used, the reader is referred to any undergraduate text in mathematical logic, for example [6].

3.3 Event classes

TADL2 events include all events expressible in AUTOSAR, all events expressible in EAST-ADL, as well as events of the external physical world not part of a particular system model. This is all captured by a common abstract superclass *Event*, which is specialized in three different ways: *AUTOSAREvent*, *EASTADLEvent*, and *ExternalEvent*. Any event can furthermore be part of an *EventChain*, which is a class referencing two *Event* instances.

3.3.1 Event

Description

The *Event* class stands for all the forms of identifiable state changes that are possible to constrain with respect to timing using TADL2.

Attributes

(none)

3.3.2 AUTOSAREvent

Description

An *AUTOSAREvent* instance refers to an event of the form defined by AUTOSAR.

Generalization

Event

Attributes

ref : *AUTOSAR*::*TimingDescription*::*TimingDescriptionEvent*

3.3.3 EASTADLEvent

Description

An *EASTADLEvent* instance refers to an event of the form defined by EAST-ADL.

Generalization

Event

Attributes

ref : *EAST-ADL*::*Timing*::*Event*

3.3.4 ExternalEvent

Description

An *ExternalEvent* instance stands for some particular form of state change.

Generalization

Event

Attributes

description : *String*

Note

It is implied that the attribute description uniquely identifies the intended form of state change. It is also assumed that a description string is sufficiently informative to determine an unambiguous set of occurrences for each observation.

This event is named *ExternalEvent* as it does not refer to an element in the model. Events referring to structural elements in the model, as e.g. ports, are defined and available in EAST-ADL and AUTOSAR.

3.3.5 EventChain

Description

An *EventChain* is a container for a pair of events that must be causally related.

Attributes

stimulus : *Event*
response : *Event*

Semantic implications

A system behavior is consistent with respect to an event chain *ec* if and only if

for each occurrence *x* in *ec.stimulus*,
for each occurrence *y* in *ec.response*,
if *x.color* = *y.color* then *x* < *y*

Logic formulation

$\forall x \in stimulus : \forall x \in response : x.color = y.color \Rightarrow x < y$

3.4 ArithmeticExpression

Description

An arithmetic expression, denoted *aexp*, is a term built from literals, arithmetic variables and arithmetic operators. It stands for a value in the real number system extended with positive and negative infinity.

Grammar

aexp ::= *LIT*
| *VAR*
| **-** $aexp_1$

| $aexp_1$ **+** $aexp_2$
| $aexp_1$ **-** $aexp_2$
| $aexp_1$ ***** $aexp_2$
| $aexp_1$ **/** $aexp_2$
| ***(***$aexp_1$***)***
| **infinity**

The grammar notation used here and in subsequent sections is the standard BNF form, with keywords in boldface and non-terminal symbols written using lower-case names.

LIT is a terminal symbol standing for a non-negative real-valued literal.

VAR is a terminal symbol standing for an alphanumeric identifier.

For more information on formal grammars and the BNF notation, see for example reference[7].

Variable assignments

The meaning of an arithmetic expression is defined relative to some assignment of values to all referenced arithmetic variables. Such a variable assignment is part of the constrained system behavior; that is, every system behavior will define a particular variable assignment and thus ascribe a particular meaning to the arithmetic expressions it references in its constraints.

Semantics

Given a particular variable assignment, the meaning of an arithmetic expression *aexp* in that assignment is a value in the real number system extended with positive and negative infinity. Depending on the form of *aexp*, this value is defined as follows:

- If *aexp* is of the form *LIT*, its meaning is the value denoted by *LIT*.
- If *aexp* is of the form *VAR*, its meaning is the value of *VAR* in the given variable assignment.
- If *aexp* is of the form **–***aexp*$_1$, its meaning is $–r_1$, where r_1 is the meaning of $aexp_1$ in the given variable assignment.
- If *aexp* is of the form $aexp_1$ **+** $aexp_2$, its meaning is $r_1 + r_2$, where r_1 is the meaning of $aexp_1$, and r_2 is the meaning of $aexp_2$, in the given variable assignment.
- If *aexp* is of the form $aexp_1$ **-** $aexp_2$, its meaning is $r_1 - r_2$, where r_1 is the meaning of $aexp_1$, and r_2 is the meaning of $aexp_2$, in the given variable assignment.
- If *aexp* is of the form $aexp_1$ ***** $aexp_2$, its meaning is $r_1 * r_2$, where r_1 is the meaning of $aexp_1$, and r_2 is the meaning of $aexp_2$, in the given variable assignment.
- If *aexp* is of the form $aexp_1$ **/** $aexp_2$, its meaning is r_1 / r_2, where r_1 is the meaning of $aexp_1$, and r_2 is meaning of $aexp_2$, in the given variable assignment.
- If *aexp* is of the form **(** $aexp_1$ **)**, its meaning is the meaning of $aexp_1$ in the given variable assignment.

- If *aexp* is of the form **infinity**, its meaning is the infinite value ∞.

3.5 TimingExpression

Description

A *TimingExpression* is identical to an arithmetic expression for what this chapter is concerned. However, in chapter 5, the *TimingExpression* grammar is extended to allow for expressions that use alternative time bases and a variety of units. The semantics of this extension is provided as a reduction of any *TimingExpression* to a form equivalent to a simple arithmetic expression, so the constraint definitions of the current chapter remain valid in spite of the expression forms added in chapter 5.

Whenever a *TimingExpression* attribute is referenced in the subsequent semantic definitions, it is the meaning of the attribute – i.e., an extended real value, given the variable assignment defined by a particular system behavior – that is implied.

3.6 TADL2 Constraints

TADL2 offers a palette of means to constrain the time occurrences of events. These can roughly be grouped into restrictions on the recurring *delays* between a pair of events, restrictions on the *repetitions* of a single event, and restrictions on the *synchronicity* of a set of events. All constraints provided by TADL2 are defined in this section.

Each constraint is documented using the following headings:

- *Description*.
- *Attributes* of the constraint, referencing *Events* as described in the previous sections, and the *TimingExpressions* that will be defined in detail in chapter 5. For the purposes of this chapter, TimingExpressions can be understood as a synonym for the *ArithmeticExpressions* introduced in chapter 3.4. Default attribute values, which apply in a right-to-left manner whenever a constraint argument list is too short to match all defined attributes, are given when applicable.
- *Semantics*, that defines which event occurrence patterns that satisfy the constraint and which do not.
- *Logic equivalence*, which expresses the constraint semantics using an equivalent first-order logic formula.
- *Notes*, which gives some motivation and intuitions to the constraint definition.
- There is also an *illustrating figure* showing a trace of event occurrences fulfilling the constraint.

3.6.1 DelayConstraint

Description

A DelayConstraint imposes limits between the occurrences of an event called *source* and an event called *target*.

Attributes

source : *Event*
target : *Event*
lower : *TimingExpression* = 0
upper : *TimingExpression* = **infinity**

Semantics

A system behavior satisfies a DelayConstraint *c* if and only if

for each occurrence *x* of *c.source*,
there is an occurrence *y* of *c.target* such that
$c.lower \leq y - x \leq c.upper$

Logic equivalence

DelayConstraint (*source*, *target*, *lower*, *upper*)
$\Leftrightarrow$
$\forall x \in source : \exists y \in target : lower \leq y - x \leq upper$

Note

This notion of delay is entirely based on the distance between *source* and *target* occurrences; whether a matching *target* occurrence is actually *caused* by the corresponding *source* occurrence is of no importance. This means that one-to-many and many-to-one *source-target* patterns are allowed, and so are stray *target* occurrences that are not within the prescribed distance of any *source* occurrence.

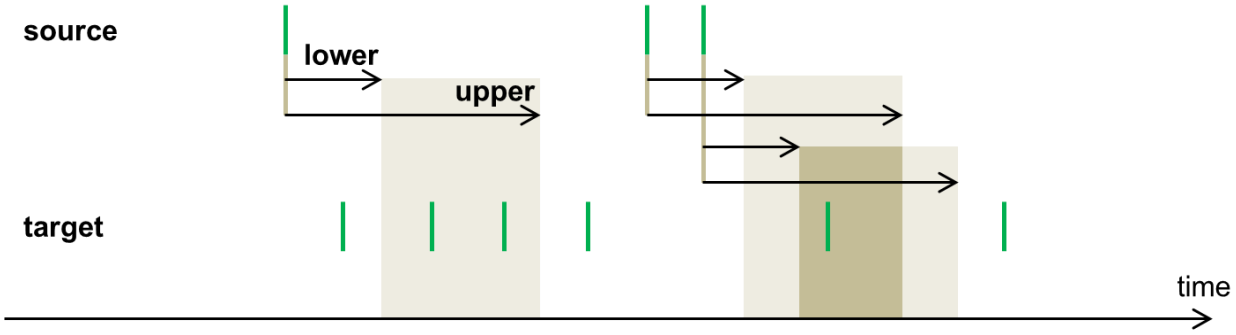

Figure 4. A set of event occurrences satisfying a DelayConstraint. Note the stray target occurrences outside the bounds set by the constraint.

3.6.2 StrongDelayConstraint

Description

A StrongDelayConstraint imposes limits between each indexed occurrence of an event called *source* and the identically indexed occurrence of an event called *target*.

Attributes

source : *Event*
target : *Event*
lower : *TimingExpression* = 0
upper : *TimingExpression* = **infinity**

Semantics

A system behavior satisfies a StrongDelayConstraint *c* if and only if

c.source and *c.target* have the same number of occurrences, and for each index *i*,
if there is an *i*:th occurrence of *c.source* at time *x*
there is also an *i*:th occurrence of *c.target* at time *y*
such that
$c.lower \leq y - x \leq c.upper$

Logic equivalence

StrongDelayConstraint (*source*, *target*, *lower*, *upper*)
$\Leftrightarrow$
$|source| = |target| \wedge$
$\forall i : \forall x : x = source(i) \Rightarrow \exists y : y = target(i) \wedge lower \leq y - x \leq upper$

Note

The strong delay notion requires *source* and *target* occurrences to appear in lock-step. Only one-to-one *source-target* patterns are allowed, and no stray *target* occurrences are accepted.

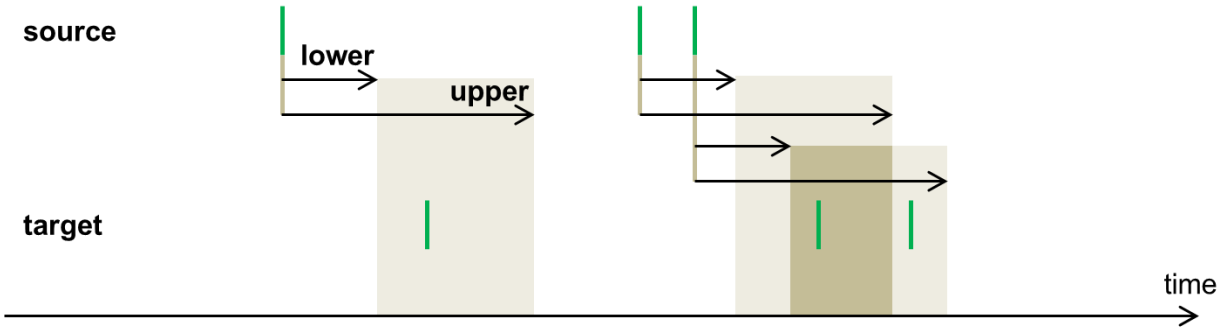

Figure 5. A set of event occurrences satisfying a StrongDelayConstraint. Note that no stray target occurrences are allowed outside the bounds set by the constraint.

3.6.3 RepeatConstraint

Description

A RepeatConstraint describes the repeated distribution of the occurrences of a single event.

Attributes

event : *Event*
lower : *TimingExpression* = 0
upper : *TimingExpression* = **infinity**
span : *int* = 1

Semantics

A system behavior satisfies a RepeatConstraint c if and only if

for each subsequence *X* of *c.event,*
if *X* contains *span* + 1 occurrences then
e is the distance between the outermost occurrences in *X*
and
c.lower ≤ *e* ≤ *c.upper*

Logic equivalence

RepeatConstraint (*event*, *lower*, *upper*, *span*)
⇔
$\forall X \leq event : |X| = span+1 \Rightarrow lower \leq \lambda([X]) \leq upper$

Note

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 *span* is still 1 but *lower* is strictly less than *upper*, the pattern may deviate from a periodic one in an accumulating fashion, making the window within which occurrence number *N* may appear as wide as *N*(*upper-lower*) time units. A *span* attribute greater than 1 similarly constrains every sequence of span+1 occurrences, but places no restriction on the distances within shorter sequences.

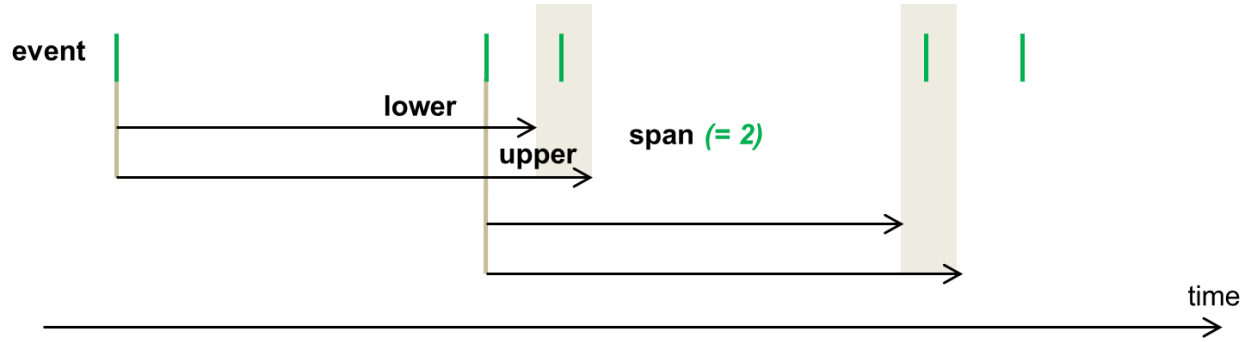

Figure 6. A set of event occurrences satisfying a RepeatConstraint with a span of 2.

3.6.4 RepetitionConstraint

Description

A RepetitionConstraint describes the distribution of the occurrences of a single event, including the allowance for jitter.

Attributes

event : *Event*
lower : *TimingExpression* = 0
upper : *TimingExpression* = **infinity**
span : *int* = 1
jitter : *TimingExpression* = 0

Semantics

A system behavior satisfies a RepetitionConstraint *c* if and only if

the same system behavior concurrently satisfies

RepeatConstraint { *event* = *X*,
lower = *c.lower*,
upper = *c.upper*,
span = *c.span* }

and

StrongDelayConstraint { *source* = *X*,
target = *c.event*,
lower = 0,
upper = *c.jitter* }

Logic equivalence

RepetitionConstraint (*event*, *lower*, *upper*, *span*, *jitter*)
⇔
∃*X* : RepeatConstraint (*X*, *lower*, *upper, span*)
∧ StrongDelayConstraint (*X*, *event*, *0*, *jitter*)

Note

The RepetitionConstraint extends the basic notion of repeated occurrences by allowing local deviations from the ideal repetitive pattern described by a RepeatConstraint. Its *jitter*, *lower* and *upper* attributes all contribute to the width of the window in which occurrence number *N* is accepted, according to the formula *N*(*upper-lower*) + *jitter*. That is, with *lower* = *upper*, the uncertainty of where occurrence *N* may be found does not grow with an increasing *N*, unlike the case when *lower* differs from *upper* by a similar amount and *jitter* is 0. By adjusting all three attributes, a desired balance between accumulating and non-accumulating uncertainties can be obtained.

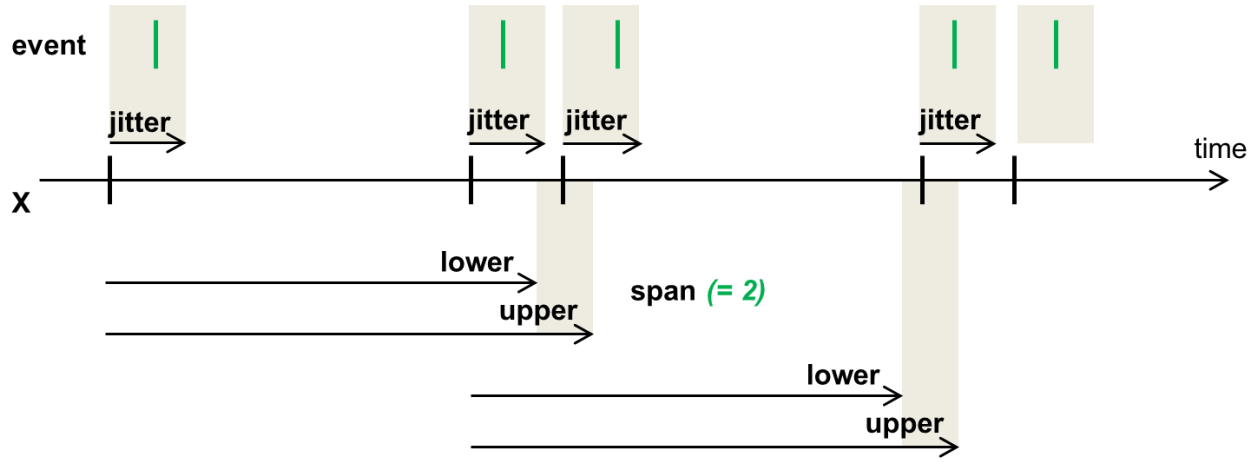

Figure 7. A set of event occurrences satisfying a RepetitionConstraint with a span of 2.

3.6.5 SynchronizationConstraint

Description

A SynchronizationConstraint describes how tightly the occurrences of a group of events follow each other.

Attributes

event : *Event* [2..*]
tolerance : *TimingExpression* = **infinity**

Semantics

A system behavior satisfies a SynchronizationConstraint *c* if and only if

there is a set of times *X* such that for each *c.event* index *i*, the same system behavior concurrently satisfies

DelayConstraint { *source* = *X*,
target = $c.event_i$,
lower = 0,
upper = *c.tolerance* }

and

DelayConstraint { *source* = $c.event_i$,
target = *X*,
lower = *-c.tolerance*,
upper = 0}

Logic equivalence

SynchronizationConstraint ($event_1$, …, $event_n$, *tolerance*)
⇔
∃*X* : ∀*i* : DelayConstraint (*X*, $event_i$, 0, *tolerance*)
∧ DelayConstraint ($event_i$, *X*, *-tolerance*, 0)

Note

This form of synchronization only takes the width and completeness of each occurrence cluster into account; it does not care whether some events occur multiple times within a cluster or whether some clusters overlap and share occurrences. In particular, event occurrences are not partitioned into clusters according to their role or what has caused them. Stray occurrences of single events are not allowed, though, since these would just count as incomplete clusters according to this constraint.

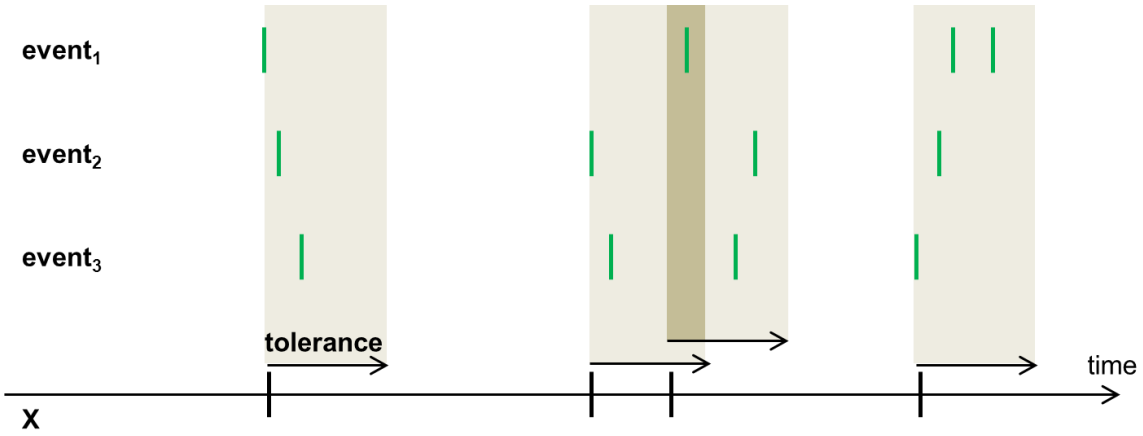

Figure 8. A set of event occurrences of three events satisfying a SynchronizationConstraint.

3.6.6 StrongSynchronizationConstraint

Description

A StrongSynchronizationConstraint describes how tightly the occurrences of a group of events follow each other.

Attributes

event : *Event* [2..*]
tolerance : *TimingExpression* = **infinity**

Semantics

A system behavior satisfies a StrongSynchronizationConstraint *c* if and only if

there is a set of times *X* such that for each *c.event* index *i*, the same system behavior satisfies
StrongDelayConstraint { *source* = *X*,
target = $c.event_i$,
lower = 0,
upper = *c.tolerance* }

Logic equivalence

StrongSynchronizationConstraint ($event_1$, …, $event_n$, *tolerance*)
⇔
∃*X* : ∀*i* : StrongDelayConstraint (*X*, $event_i$, 0, *tolerance*)

Note

Strong synchronization differs from the ordinary form (section 3.6.5) by grouping event occurrences into synchronization clusters strictly according to their index. This means that multiple occurrences of a single event cannot belong to a single cluster, and clusters may not share occurrences. Strong synchronization tightens the requirements compared to ordinary synchronization in much the same way as strong delay (section 3.6.2) refines the ordinary delay constraint (section 3.6.1).

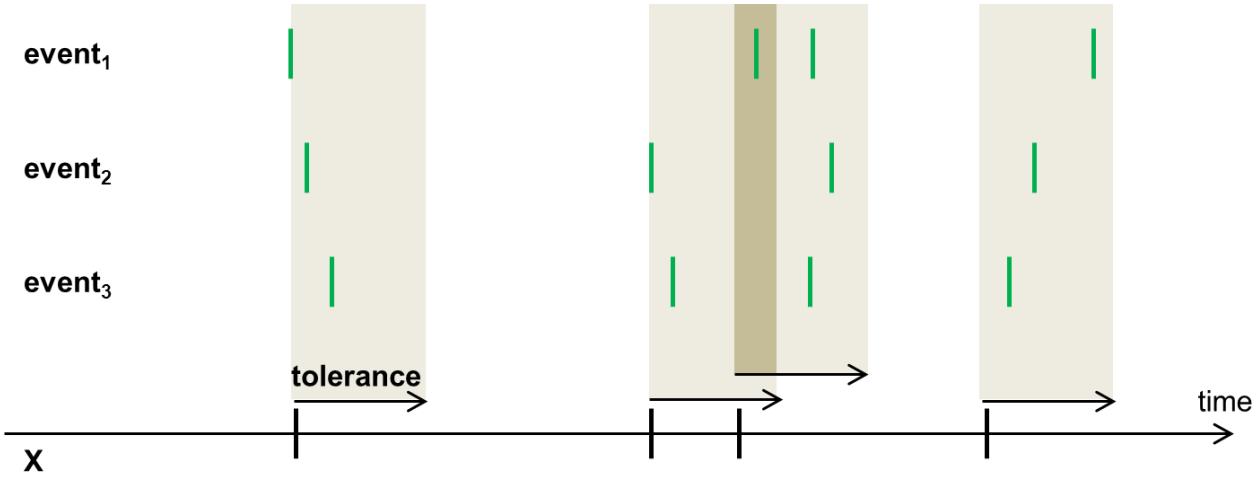

Figure 9. A set of event occurrences of three events satisfying a StrongSynchronizationConstraint.

3.6.7 ExecutionTimeConstraint

Description

An ExecutionTimeConstraint limit the time between the starting and stopping of an executable entity (function), not counting the intervals when the execution of such an executable entity (function) has been interrupted.

Attributes

start : *Event*
stop : *Event*
preempt : *Event*
resume : *Event*
lower : *TimingExpression*
upper : *TimingExpression*

Semantics

A system behavior satisfies an ExecutionTimeConstraint *c* if and only if

for each occurrence *x* of event *c.start*,
E is the set of times between *x* and the next *c.stop* occurrence, excluding the times between any *c.preempt* occurrence and its next *c.resume* occurrence,
and
c.lower ≤ length of all continuous intervals in *E* ≤ *c.upper*

Logic equivalence

ExecutionTimeConstraint (
start, *stop*, *preempt*, *resume*, *lower*, *upper*)
⇔
$\forall x \in start : lower \leq \lambda([x..stop] \setminus [preempt..resume]) \leq upper$

Note

The execution time of a task (function, runnable) is defined as the time from each activation of the task to the point of the corresponding task termination, not counting the intervals where the task has been preempted. TADL2 assumes that all points of interest in this respect are available as events. The length operator λ(*X*) plays a crucial role in this definition, as it computes the sum of the length of all continuous intervals in set *X*. A discontinuous series of intervals will typically result if the times belonging to any preemption interval (that is, elements of [*preempt*..*resume*]) are removed from the interval describing a task activation (i.e., [*x*..*stop*]).

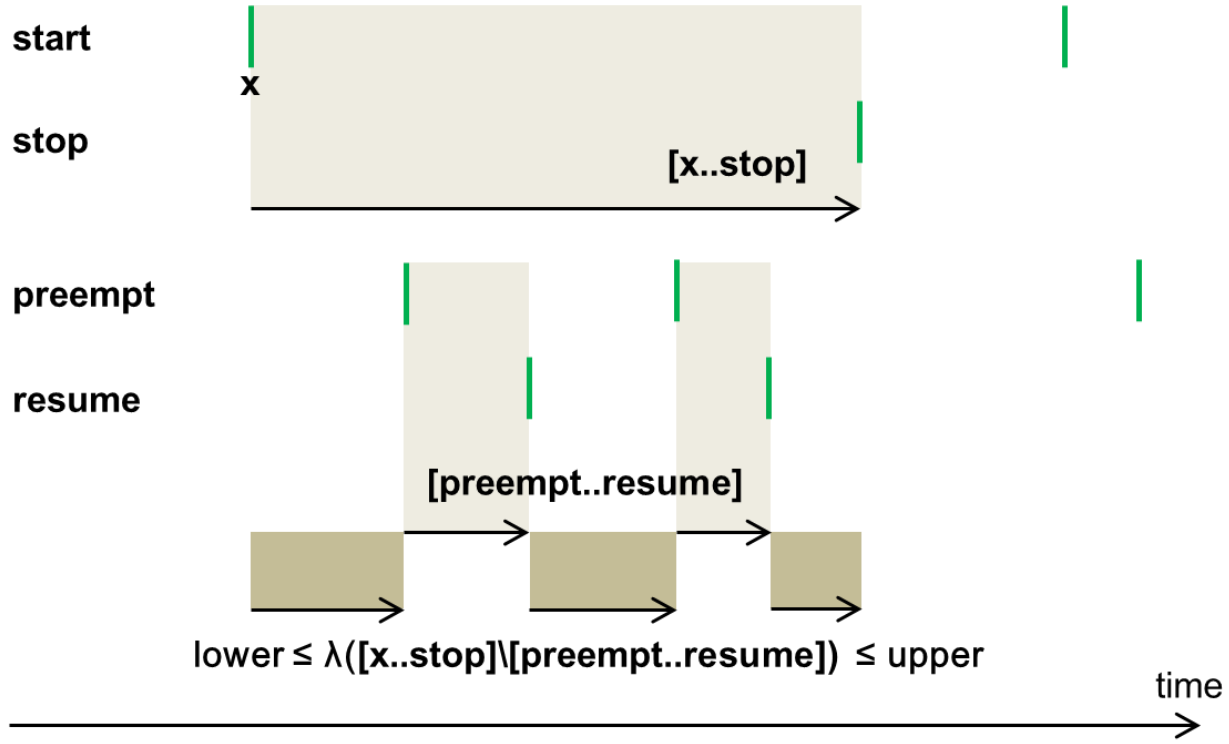

Figure 10. A set of event occurrences satisfying an ExecutionTimeConstraint.

3.6.8 OrderConstraint

Description

An OrderConstraint imposes an order between the occurrences of an event called *source* and an event called *target*.

Attributes

source : *Event*
target : *Event*

Semantics

A system behavior satisfies an OrderConstraint c if and only if

c.source and *c.target* have the same number of occurrences, and for each index *i*,
if there is an *i*:th occurrence of *c.source* at time *x,* there is also an *i*:th occurrence of *c.target* at time *y* such that
$x < y$

Logic equivalence

OrderConstraint (*source*, *target*)
$\Leftrightarrow$
$|source| = |target| \wedge$
$\forall i : \exists x : x = source(i) \Rightarrow \exists y : y = target(i) \wedge x < y$

Note

The OrderConstraint is a minor variant of an application of StrongDelayConstraint with *lower* set to 0 and *upper* to **infinity**;

the difference being that the OrderConstraint does not allow matching target and source occurrences to coincide.

Figure 11. A set of event occurrences satisfying an OrderConstraint.

3.6.9 ComparisonConstraint

Description

A ComparisonConstraint states that a certain ordering relation must exist between two timing expressions.

Attributes

leftOperand : *TimingExpression*
rightOperand : *TimingExpression*
operator : *ComparisonOperator*

Semantics

A system behavior satisfies a ComparisonConstraint *c* if and only if

c.leftOperand and *c.rightOperand* are related according to the ordering relation given by *c.operator*.

Logic equivalences

ComparisonConstraint (*leftOperand*, *rightOperand*, LessThanOrEqual)
⇔
leftOperand ≤ *rightOperand*

ComparisonConstraint (*leftOperand*, *rightOperand*, LessThan)
⇔
leftOperand < *rightOperand*

ComparisonConstraint (*leftOperand*, *rightOperand*, GreaterThanOrEqual)
⇔
leftOperand ≥ *rightOperand*

ComparisonConstraint (*leftOperand*, *rightOperand*, GreaterThan)
⇔
leftOperand > *rightOperand*

ComparisonConstraint (*leftOperand*, *rightOperand*, Equal)
⇔
leftOperand = *rightOperand*

Note

This constraint is special in that it does not reference any events. Its main purpose is to express relations between arithmetic variables used in other constraint; for example, stating that the sum of the variables denoting segment delays in a time-budgeting scenario must be less than the maximum end-to-end deadline allowed.

3.6.10 SporadicConstraint

Description

A SporadicConstraint describes an event that occurs sporadically.

Attributes

event : *Event*
lower : *TimingExpression* = 0
upper : *TimingExpression* = **infinity**
jitter : *TimingExpression* = 0
minimum : *TimingExpression* = 0

Semantics

A system behavior satisfies a SporadicConstraint *c* if and only if

the same system behavior concurrently satisfies
RepetitionConstraint { *event* = *c.event*,
lower = *c.lower*,
upper = *c.upper*,
jitter = *c.jitter* }
and
RepeatConstraint { *event* = *c.event*,
lower = *c.minimum* }

Logic equivalence

SporadicConstraint (*event*, *lower*, *upper*, *jitter*, *minimum*)
⇔
RepetitionConstraint (*event*, *lower*, *upper*, *jitter*)
∧ RepeatConstraint (*event*, *minimum*)

Note

The SporadicConstraint is just an application of the RepetitionConstraint with a default *span* attribute of 1, combined with an additional requirement that the effective minimum distance between any two occurrences must be at least the value given by *minimum* (even if *lower-jitter* would suggest a smaller value).

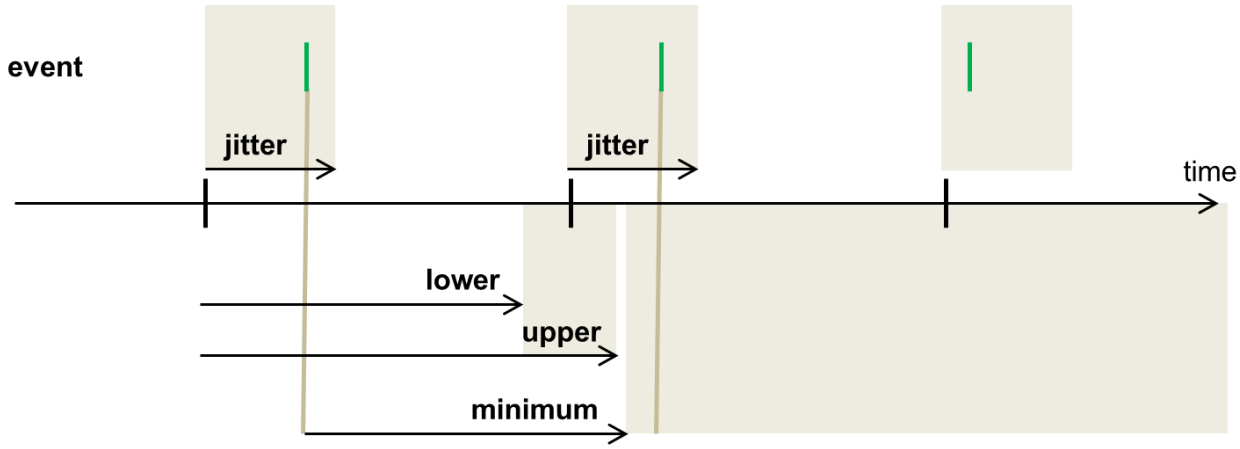

Figure 12. A set of event occurrences satisfying a SporadicConstraint.

3.6.11 PeriodicConstraint

Description

A PeriodicConstraint describes an event that occurs periodically.

Attributes

event : *Event*
period : *TimingExpression*
jitter : *TimingExpression* = 0
minimum : *TimingExpression* = 0

Semantics

A system behavior satisfies a PeriodicConstraint *c* if and only if the same system behavior satisfies

SporadicConstraint { *event* = *c.event*,
lower = *c.period*,
upper = *c.period*,
jitter = *c.jitter*,
minimum = *c.minimum* }

Logic equivalence

PeriodicConstraint (*event*, *period*, *jitter*, *minimum*)
⇔
SporadicConstraint (*event*, *period*, *period*, *jitter*, *minimum*)

Note

The PeriodicConstraint is a mere synonym for an application of the SporadicConstraint with equal values for the *lower* and *upper* attributes (i.e., not accumulating uncertainties). See sections 3.6.3 and 3.6.4 for further discussions on the use of attributes *lower*, *upper* and *jitter*.

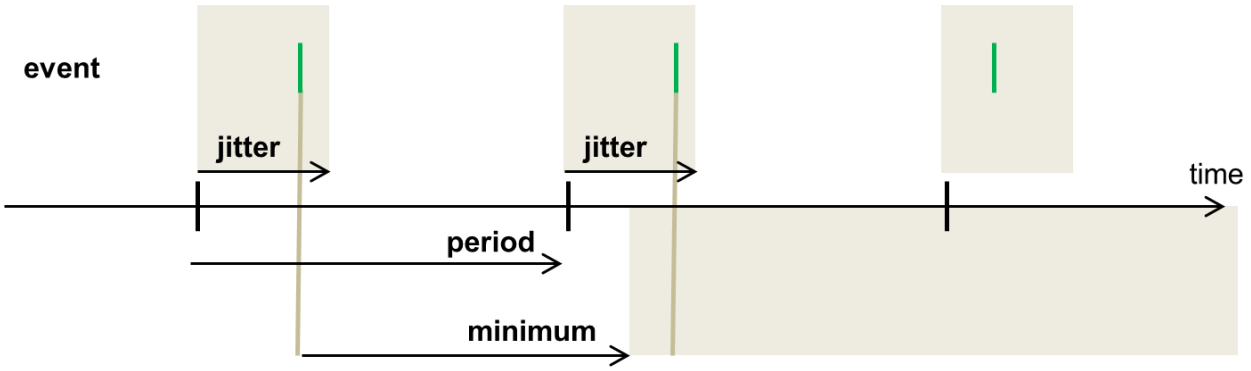

Figure 13. A set of event occurrences satisfying a PeriodicConstraint.

3.6.12 PatternConstraint

Description

A PatternConstraint describes an event that exhibits a known pattern relative to some periodic points in time.

Attributes

event : *Event*
period : *TimingExpression*
offset : *TimingExpression* [1..*]
jitter : *TimingExpression* = 0
minimum : *TimingExpression* = 0

Semantics

A system behavior satisfies a PatternConstraint *c* if and only if

there is a set of times *X* such that the same system behavior concurrently satisfies

PeriodicConstraint { *event* = *X*,
period = *c.period* }

and for each *c.offset* index *i*,

DelayConstraint { *source* = *X*,
target = *c.event*,
lower = $c.offset_i$,
upper = $c.offset_i$ + *c.jitter* }

and

RepeatConstraint { *event* = *c.event*,
lower = *c.minimum* }

Logic equivalence

PatternConstraint
(*event*, *period*, $offset_1$, …, $offset_n$, *jitter*, *minimum*)
⇔
∃*X* : PeriodicConstraint (*X*, *period*)
∧ (∀*i* : DelayConstraint (*X*, *event*, $offset_i$, $offset_i$+*jitter*))
∧ RepeatConstraint(*event*, *minimum*)

Note

A PatternConstraint requires the constrained event occurrences to appear at a predetermined series of offsets from a sequence of

reference points in time that are strictly periodic. The exact placement of these reference points is irrelevant; if one placement exists that is periodic and allows the event occurrences to be reached at the desired offsets, the constraint is satisfied.

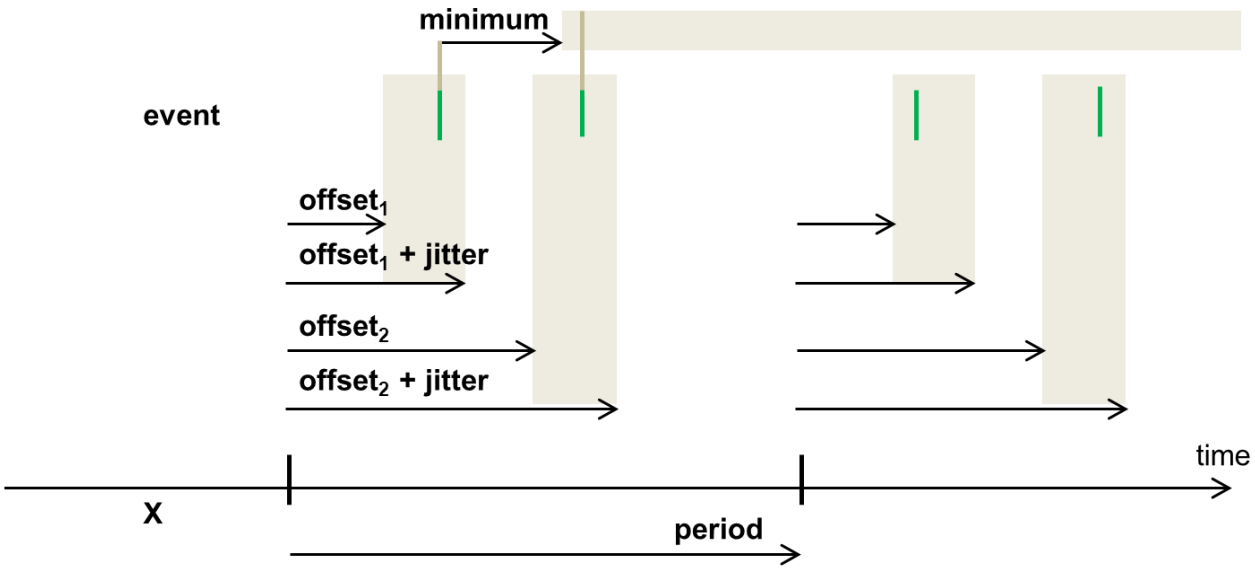

Figure 14. A set of event occurrences satisfying a PatternConstraint.

3.6.13 ArbitraryConstraint

Description

An ArbitraryConstraint describes an event that occurs irregularly.

Attributes

event : *Event*
minimum : *TimingExpression* [1..*]
maximum : *TimingExpression* [1..*]

Syntactic constraints

The number of elements in *minimum* and *maximum* must be equal.

Semantics

A system behavior satisfies an ArbitraryConstraint *c* if and only if

for each *c.minimum* index *i*, the same system behavior satisfies
RepeatConstraint { *event* = *c.event*,
lower = $c.minimum_i$,
upper = $c.maximum_i$,
span = *i* }

Logic equivalence

ArbitraryConstraint (*event*, $minimum_1$, …, $minimum_n$,
$maximum_1$, …, $maximum_n$)
⇔
∀*i* : RepeatConstraint (*event*, $minimum_i$, $maximum_i$, *i*)

Note

An ArbitraryConstraint is equivalent to a combination of Repeat constraints, each one constraining sequences of *i*+1 occurrences (that is, *i* repetition spans), with *i* ranging from 1 to some given *n*.

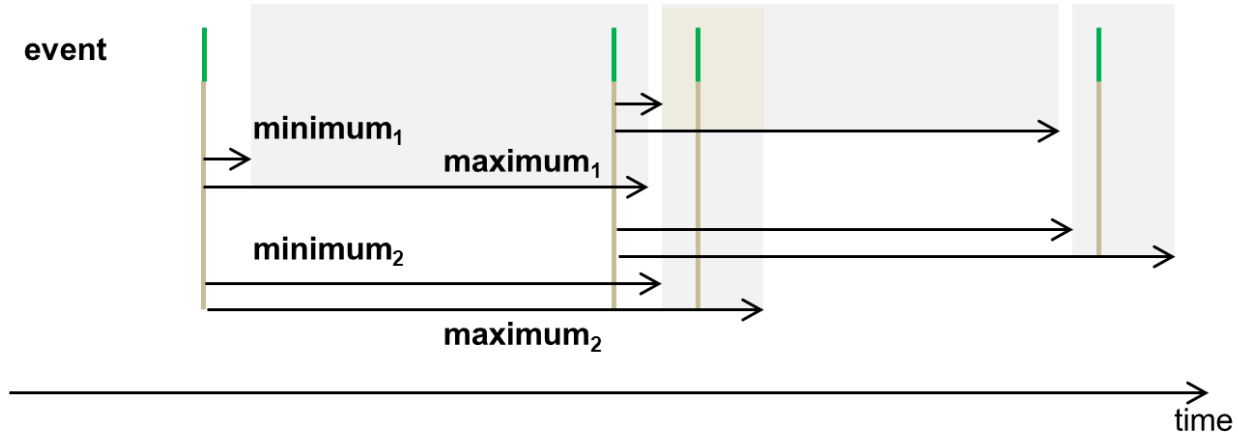

Figure 15. A set of event occurrences satisfying an ArbitraryConstraint.

3.6.14 BurstConstraint

Description

A BurstConstraint describes an event that occurs in semi-regular bursts.

Attributes

event	: *Event*
length	: *TimingExpression*
maxOccurrences	: *int*
minimum	: *TimingExpression* = 0

Semantics

A system behavior satisfies a BurstConstraint *c* if and only if

the same system behavior concurrently satisfies
RepeatConstraint { *event* = *c.event*,
lower = *c.length*,
upper = **infinity**,
span = *c.maxOccurrences* }
and
RepeatConstraint { *event* = *c.event*,
lower = *c.minimum* }

Logic equivalence

BurstConstraint (*event*, *length*, *maxOccurrences*, *minimum*)
⇔
RepeatConstraint (*event*, *length*, **infinity**, 0, *maxOccurrences*)
∧ RepeatConstraint (*event*, *minimum*)

Note

A BurstConstraint expresses the maximum number of event occurrences that may appear in any interval of a given *length*, which is equivalent to constraining the same number of repeat spans (which count one extra occurrence at the end) to have a minimum width of *length*.

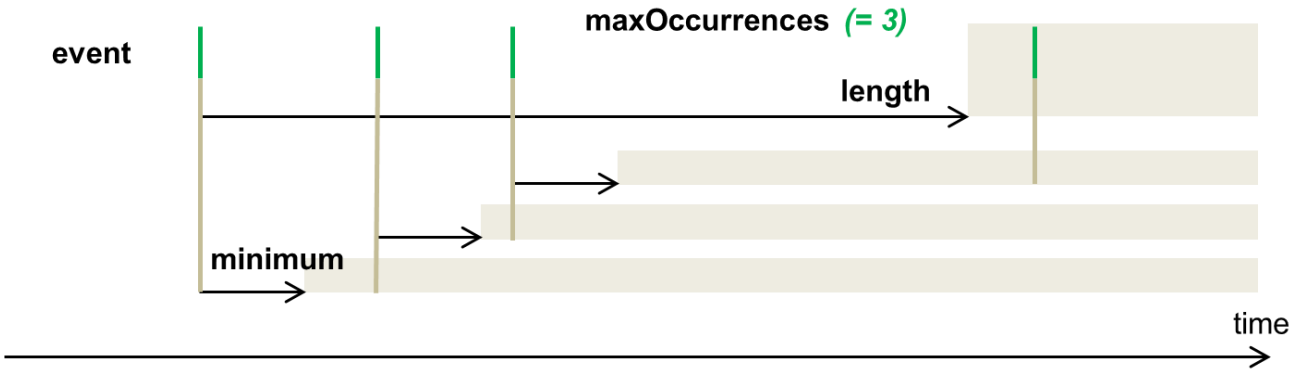

Figure 16. A set of event occurrences satisfying a BurstConstraint with maxOccurrences = 3.

3.6.15 ReactionConstraint

Description

A ReactionConstraint defines how long after the occurrence of a stimulus a corresponding response must occur.

Attributes

scope : *EventChain*
minimum : *TimingExpression* = 0
maximum : *TimingExpression* = **infinity**

Semantics

A system behavior satisfies a ReactionConstraint *c* if and only if

for each occurrence *x* in *c.scope.stimulus*,
there is an occurrence *y* in *c.scope.response* such that
y.color = *x.color*
and
y is minimal in *c.scope.response* with that color
and
$c.minimum \le y - x \le c.maximum$

Logic equivalence

ReactionConstraint (*scope*, *minimum*, *maximum*)
$\Leftrightarrow$
$\forall x \in scope.stimulus : \exists y \in scope.response :$
$x.color = y.color$
$\wedge (\forall y' \in scope.response : y'.color = y.color \Rightarrow y \le y')$
$\wedge\ minimum \le y - x \le maximum$

Note

This constraint provides an alternative to the ordinary DelayConstraint (section 3.6.1) for situations where the causal relation between event occurrences must be taken into account. It differs from the DelayConstraint in that it applies to an event chain, and only looks at the *response* occurrences that have the same color as each particular *stimulus* occurrence. It is the earliest of these *response* occurrences that is required to lie within the prescribed time bounds. If the roles of *stimulus* and *response* are swapped, and the time bounds negated, an AgeConstraint is obtained (see section 3.6.16).

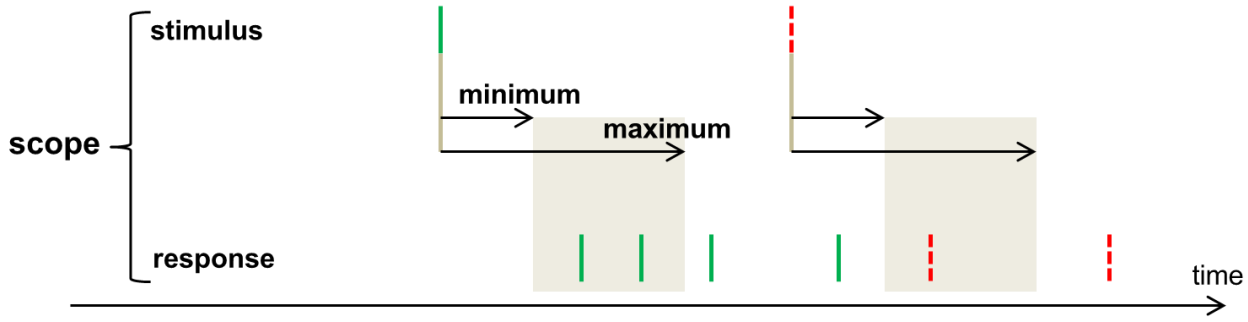

Figure 17. A set of event occurrences satisfying a ReactionConstraint. Causally connected event occurrences are shown by the same line style. Two sets are shown, one by a solid line and one by a dashed line.

3.6.16 AgeConstraint

Description

An AgeConstraint defines how long before each response a corresponding stimulus must have occurred.

Attributes

scope : *EventChain*
minimum : *TimingExpression* = 0
maximum : *TimingExpression* = **infinity**

Semantics

A system behavior satisfies an AgeConstraint *c* if and only if

for each occurrence *y* in *c.scope.response*,
there is an occurrence x in *c.scope.stimulus* such that
x.color = y.color
and
x is maximal in *c.scope.stimulus* with that color
and
$c.minimum \leq y - x \leq c.maximum$

Logic equivalence

AgeConstraint (*scope*, *minimum*, *maximum*)
$\Leftrightarrow$
$\forall y \in scope.response : \exists x \in scope.stimulus :$
$x.color = y.color$
$\wedge\ (\forall x' \in scope.stimulus : x'.color = x.color \Rightarrow x' \leq x)$
$\wedge\ minimum \leq y - x \leq maximum$

Note

This constraint provides an alternative to the ordinary DelayConstraint (section 3.6.1) for situations where the causal relation between event occurrences must be taken into account. It differs from the DelayConstraint in that it applies to an event chain, and only looks at the *stimulus* occurrences that have the same color as each particular *response* occurrence. It is the latest of these *stimulus* occurrences that is required to lie within the prescribed time bounds. If the roles of *stimulus* and *response* are swapped, and the time bounds negated, a ReactionConstraint is obtained (see section 3.6.15).

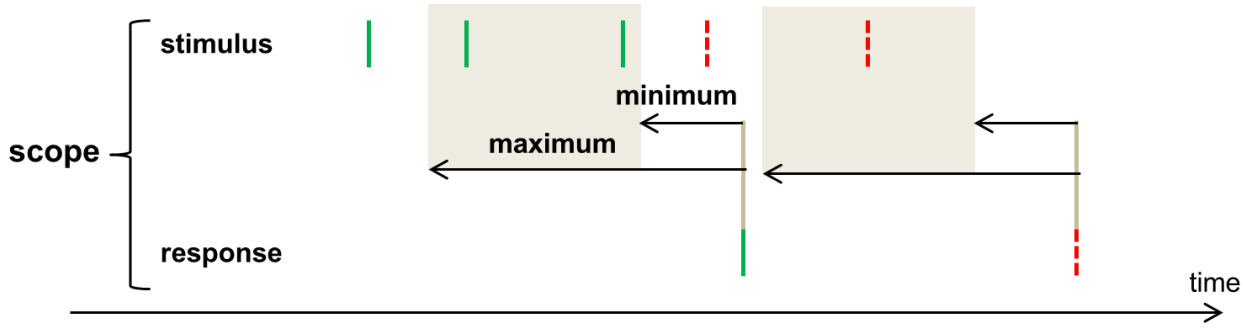

Figure 18. A set of event occurrences satisfying an AgeConstraint. Causally connected event occurrences are shown by the same line style. Two sets are shown, one by a solid line and one by a dashed line.

3.6.17 OutputSynchronizationConstraint

Description

An OutputSynchronizationConstraint defines how far apart the responses that belong to a certain stimulus may occur.

Attributes

scope : *EventChain*[2..*]
tolerance : *TimingExpression* = **infinity**

Syntactic constraints

All scopes must reference one common stimulus event.

Semantics

A system behavior satisfies an OutputSynchronizationConstraint *c* if and only if

for each occurrence *x* in $c.scope_1.stimulus$,
  there is a time *t* such that for each *c.scope* index *i*,
    there is an occurrence *y* in $c.scope_i.response$ such that
      $y.color = x.color$
    and
      y is minimal in $c.scope_i.response$ with that color
    and
      $0 \le y - t \le c.tolerance$

Logic equivalence

OutputSynchronizationConstraint ($scope_1$, …, $scope_n$, *tolerance*)
⇔
$\forall x \in scope_1.stimulus : \exists t : \forall i : \exists y \in scope_i.response :$
$x.color = y.color$
$\wedge (\forall y' \in scope_i.response : y'.color = y.color \Rightarrow y \leq y')$
$\wedge\ 0 \leq y - t \leq tolerance$

Note

This constraint provides an alternative to the ordinary SynchronizationConstraint (section 3.6.5) for situations where the causal relation between event occurrences must be taken into account. It differs from the SynchronizationConstraint in that it applies to a set of event chains, and only looks at the *response* occurrences that have the same color as each particular *stimulus* occurrence. It is the earliest of these *response* occurrences for each chain that are required to lie no more than *tolerance* time units apart. If the roles of *stimuli* and *responses* are swapped, an InputSynchronizationConstraint is obtained (see section 3.6.18).

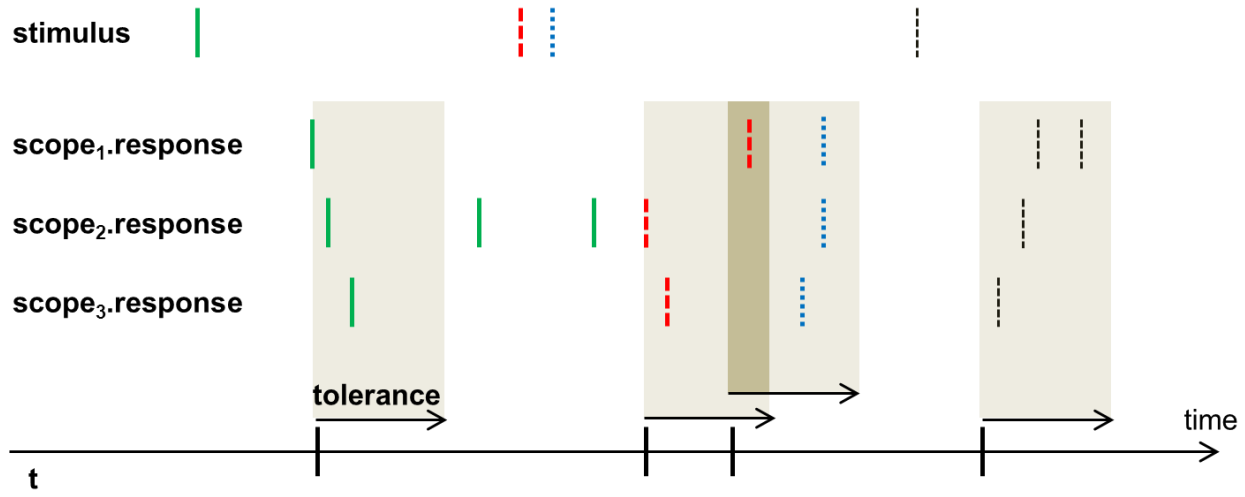

Figure 19. A set of event occurrences satisfying an OutputSynchronizationConstraint. Response event occurrences that are causally connected to a stimulus event occurrence are shown by the same line style as the stimulus event occurrence. Four sets are shown with different line styles.

3.6.18 InputSynchronizationConstraint

Description

An InputSynchronizationConstraint defines how far apart the responses that belong to a certain stimulus may occur.

Attributes

scope : *EventChain*[2..*]
tolerance : *TimingExpression* = **infinity**

Syntactic constraints

All scopes must reference one common response event.

Semantics

A system behavior satisfies an InputSynchronizationConstraint *c* if and only if

for each occurrence *y* in $c.scope_1.response$,
there is a time *t* such that for each *c.scope* index *i*,
there is an occurrence *x* in $c.scope_i.stimulus$ such that
$y.color = x.color$
and
x is maximal in $c.scope_i.stimulus$ with that color
and
$0 \le x - t \le c.tolerance$

Logic equivalence

InputSynchronizationConstraint ($scope_1, \ldots, scope_n, tolerance$)
$\Leftrightarrow$
$\forall y \in scope_1.response : \exists t : \forall i : \exists x \in scope_i.stimulus :$
$x.color = y.color$
$\wedge\ (\forall x' \in scope_i.stimulus : x'.color = x.color \Rightarrow x' \le x)$
$\wedge\ 0 \le x - t \le tolerance$

Note

This constraint provides an alternative to the ordinary SynchronizationConstraint (section 3.6.5) for situations where the causal relation between event occurrences must be taken into account. It differs from the SynchronizationConstraint in that it applies to a set of event chains, and only looks at the *stimulus* occurrences that have the same color as each particular *response* occurrence. It is the latest of these *stimulus* occurrences for each chain that are required to lie no more than *tolerance* time units apart. If the roles of *stimuli* and *responses* are swapped, an OutputSynchronizationConstraint is obtained (see section 3.6.17).

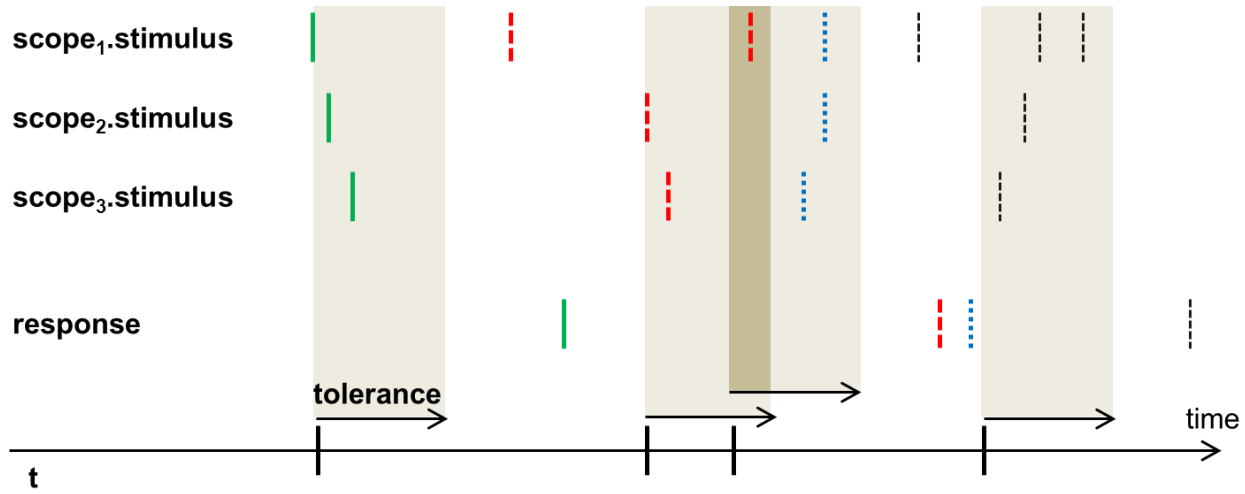

Figure 20. A set of event occurrences satisfying an InputSynchronizationConstraint. Stimuli event occurrences that are causally connected to a specific response event occurrence are shown by the same line style as the response event occurrence. Four sets are shown with different line styles.

4 Mode Dependency

Modes capture certain states in a system where it is supposed to operate in a certain way. For instance, if the hazard warning signal is turned on, then the warning lights would be flashed synchronously with a certain periodicity and when the signal is turned off then the system should return to its usual mode, where the lights do not flash. Obviously, timing constraints should also be possible to associate with modes since different modes of operation have different timing requirements on different events.

TADL2 therefore allows a timing constraint *c* to be dependent on a mode by setting *c.mode* to a mode identifier *M (or, in the logic format, by writing c % M*). When a mode is turned on then all its dependent timing constraints become active and they remain so until the mode is turned off and they- the constraints are again deactivated.

TADL2 does not provide any means to define modes: it assumes that, for each mode *M*, there is a special event turning *M* on (semantically denoted $M\uparrow$) and another event turning *M* off (written $M\downarrow$). These events provide the interface of TADL2 for modes, according to Figure 21. A time interval between an event occurrence turning on *M*, and the subsequent event occurrence turning it off, is a *mode window* for *M*; the timing constraints that depend on *M* are active exactly in these windows.

TADL2 does not require the modes that are mutually exclusive. However, a timing constraint can only be dependent on one mode. If a constraint is to be active in several modes, then a mode corresponding to the union of these modes has to be defined outside TADL2, such that it's on and off events delimit the desired combined mode windows.

Semantics

Assume *c* is a mode-dependent constraint that is an instance of class *Constr*, has *Event* parameters $event_1,\ldots,event_n$ and additional parameters $par_1,\ldots,par_m$, and that *c.mode* = *M*.

Then, a system behavior satisfies *c* if and only if

for each occurrence *x* of event $M\uparrow$,

Y is the interval from *x* to the next $M\downarrow$ occurrence

and

$X_1,\ldots,X_n$ are freely chosen sets of occurrences outside Y

and

$$Constr\ \{\ event_1 = (c.event_1 \cap\ Y) \cup\ X_1,$$
$$\ldots$$
$$event_n = (c.event_n \cap\ Y) \cup\ X_n,$$
$$par_1 = c.par_1,$$
$$\ldots$$
$$par_m = c.par_m\ \}$$

Logic Equivalence

$$Constr\ (\ event_1,\ \ldots,\ event_n,\ par_1,\ \ldots,\ par_m\ )\ \%\ M$$
$$\Leftrightarrow$$

$\forall x \in M\uparrow : \exists Y = [x..M\downarrow] : \exists X_1,\ldots,X_n \subseteq \mathbf{C}(Y) :$

Constr ((*event*$_1$ ∩ *Y*) ∪ *X*$_1$, …, (*event*$_n$ ∩ *Y*) ∪ *X*$_n$, *par*$_1$, …, *par*$_m$)

Note

The mode-restricted version of the constraint *C* holds exactly when-for each mode window, an instance of *C* holds where each constrained event is first restricted to the mode interval, then extended in some arbitrary way outside the mode interval to make the constraint hold. The occurrences extending an event do not have to occur, they are only "imaginary" occurrences showing that the part of an event that intersect the mode window could have been part of an event fulfilling the mode-unrestricted constraint. Note that we do not care about the actual occurrences of such an event outside each mode window.

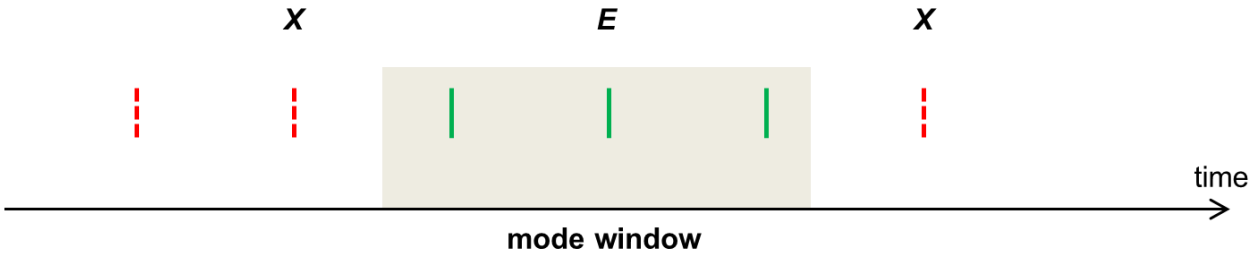

Figure 21. The mode window with three occurrences of event *E*.

Figure 21 illustrates the semantics of mode-dependent constraints. For simplicity, consider a mode that is active only in a single mode window. The three occurrences of the event *E* that occur within the mode window all occur periodically, with a constant distance in time between each consecutive occurrence. Thus, they can be extended with "hypothetical" occurrences belonging to some event *X*, outside the mode window, in such a way that the combined event is periodic also outside the mode window. This shows that *E* satisfies a mode-dependent periodic constraint. For modes with several mode windows, the event *E* must be possible to extend in this way for each mode window separately.

5 Multiple Time Bases and Symbolic Timing Expressions

This work on symbolic time expression is twofold: firstly, it concerns the concepts of TADL2 to manage in a same design, time bases of multiple types (universal time – i.e. chronometric time, angular time, etc.). The second aspect concerns with the extension of constant time expressions and the possibility to define time as an algebraic expression that is able to manipulate symbolic identifiers. So, a *value* expression in a TADL2 time constraint may refer to an expression made of a suitable set of arithmetic operators mixing symbolic identifiers and referring to different time bases.

A typical use for this feature is to capture unknown configuration parameters; another one is to relate constraints in different time-bases to each other. Inherent to this work is also the study of the allowable ranges for symbolic values that are dictated by a set of constraints, using both logic reasoning and systematic simulations of event patterns.

5.1 Requirements and Examples for Symbolic Timing Expressions

Depending on the abstraction level in a design, timing requirements could refer to different time bases (continuous, discrete or logical).

Some examples of typical requirements which refer to different time bases are given below:

Time Measured on Multiple Time Bases

- The language should be capable of expressing constraints with different time bases. It is supposed that time bases are explicit model elements and the language can refer to these elements.
- The spark ignition correction should be within the interval [-15 ° CRK ; +15 ° CRK] from the reference Top Dead Center (TDC)
- A phase in a 4-stroke engine lasts after 90°CAM
- The knock control shall be computed each 720 ° CRK/number of cylinders
- The offset of the time bases relative to a global time base needs to be specified. For example, if t=0 on one ECU actually represents t=2 on a global time scale due to start up delays.

Relation between Time Bases

- The expected drift in time bases defined per ECU. For instance, if 1 ms elapsed time on one ECU represents 1.2 ms on another.
- Time bases can be related to each other by either constant values or dynamic relations (example dependency between °crk and RPM).

Parameterized Timing Expression and Intervals

- Expressing time budgeting: “the budget for this part is the whole minus the time for the other part”.

- Expressing time budgets: An overall response time is shared with 30% for the sensor segment, 50% for the controller and 20% for the actuator. Resp_Total=250 ms, parts are 0.3* Resp_Total, 0.5* Resp_Total and 0.3* Resp_Total

- Expressing harmonic periods: A period of a sensor is oversampled whereas a period of an actuator is undersampled for a 50 Hz process: T=20ms, T_{sensor}=0.5*T, $T_{controller}$=T, $T_{actuator}$=2*T

- During design, it could be useful to be able to place constraints such as period < delay/2 in order to "automatically" fulfill a delay constraint.

- Knock control on the cycle n shall be performed before the engine cycle n+2

- Reaction time is unknown but shall be minimized and smaller than 50ms

- Reaction time is unknown but shall be maximized (leaving max margins for implementation while meeting overall requirements)

- The acquisition of the knock signal shall be performed in less than X ms;

- Like "upper = 1.1*nominal", i.e. in this example the language needs to allow for references to the nominal value of a constraint. There are two ways of doing this, 1) define nominal as 0.1 seconds and define a model parameter "nominalTime" for this, use the parameter in upper. 2) Refer directly to the nominal value from the upper attribute.

- Express "x seconds if n, y seconds if m", where n and m may be expressions containing references to model attributes or defined parameters.

- Acquisition duration = MIN {sampleNb*10 ms, 30 on crkClk} i.e. the duration time of a function should not exceed the minimum between a constant expression value and a value linked to the dynamic of the system.

5.2 Multiple Time Bases

This section presents new extensions in TADL2 to manage, time bases of multiple types (universal time, angular time, etc.). A second aspect concerns the extension of constant time expressions with the possibility to define time as an algebraic expression that can manipulate symbolic identifiers and of a suitable set of arithmetic operators mixing symbolic identifiers and referring to different time bases.

5.2.1 Dimension

The type of *TimeBase* is *Dimension*. *Dimension* defines the set of units that can be used to express duration measured on a given TimeBase. The Dimension can be seen as the type of a TimeBase. Each *Unit* relates to another unit to enable conversions. The *factor*, *offset* and the *reference* attributes in *Unit* are used for such

conversions. Only linear conversions between units of the same dimension are allowed. As a unit conversion example, the unit *second = 1000 * millisecond* so *factor = 1000 and offset = 0*. Because a *TimeBase* is a discrete set of instants, a discretization step is specified with the *precisionFactor* attribute which relies on a *precisionUnit*.

Description

Dimension is an identifier introduced in a dimension declaration.

Grammar

DimensionDecl	::=	**Dimension** *DI* **{** *UnitDecl*+ **}**
UnitDecl	::=	*UN* **:** *LIT*

Here *DI* and *UN* are terminal symbols standing for alphanumeric identifiers, and *LIT* is a real-valued literal.

Syntactic constraints

Every dimension declaration must introduce a unique *DI* identifier.

The *UN* identifiers within a single dimension declaration must be locally unique.

A dimension declaration with the name *universal* must exist.

Terminology

If **Dimension** *DI* { UN_1 : LIT_1, …, UN_n : LIT_n } is a dimension declaration, we say that the *factor* of UN_i in *DI* is LIT_i.

5.2.2 TimeBase

TimeBase has been introduced to cope with the need of modeling various temporal referential used in an automotive distributed systems design (clocks from different ECUs, motor position, etc.).

TADL2 timing expressions may contain an explicit *TimeBase* which 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 a system. Events may refer even to “classical” time dimension or to some evolution of a mechanical part like the rotation of crankshaft, distance, etc. Figure 22 presents the modeling elements and their relationships.

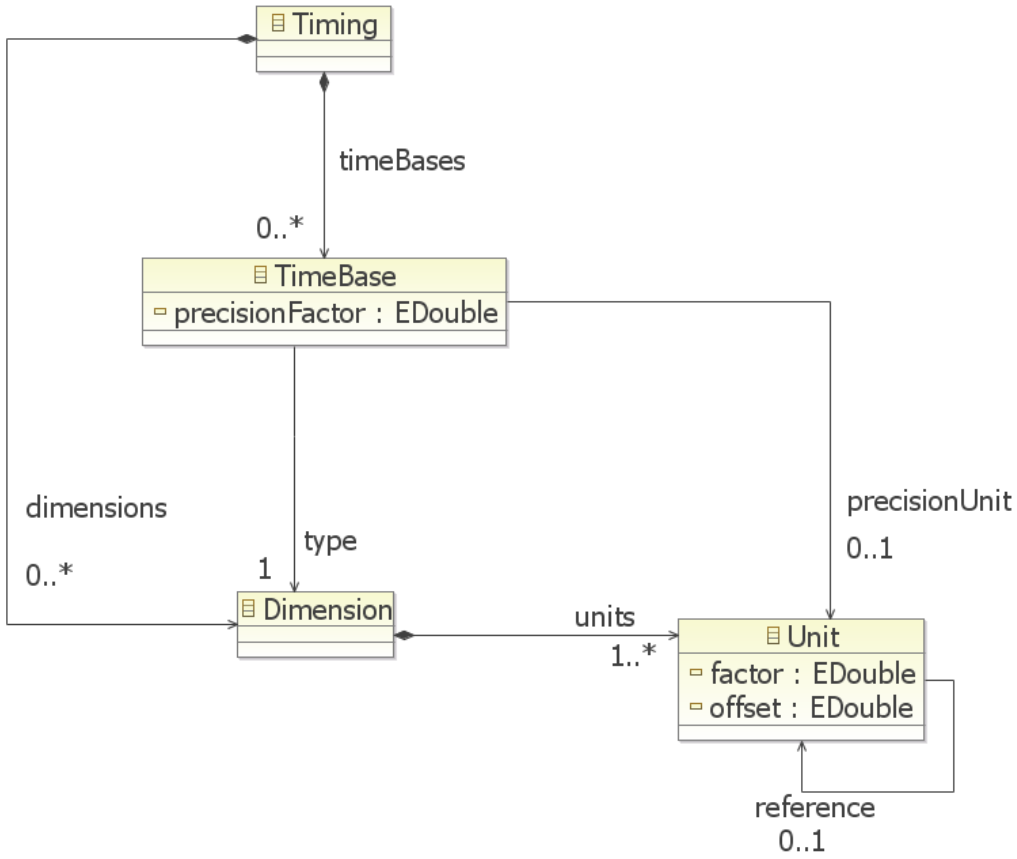

Figure 22: Extension of the metamodel with explicit time bases.

Textual Syntax for TimeBase

TimeBase is an identifier introduced in a *TimeBase declaration*.

Grammar

TimeBaseDecl ::= **TimeBase** TB_1 **:** *DI* **{** *LIT UN* **on** TB_2 **=** *texp* **}**

Here TB_1 and TB_2 are terminal symbols standing for alphanumeric identifiers and *texp* is an expression which evaluates to a time value (see section 5.3), and *LIT* is a real-valued literal.

Syntactic constraints

For a time base declaration of the form above, it must hold that:

1. TB_1 is a globally unique identifier ≠ *Universal*.
2. TB_1 is identical to TB_2.
3. *DI* has a matching dimension declaration.
4. *texp* does not directly or indirectly (via some other time base declaration) refer to time base TB_1.

Terminology

For a time base declaration of the form above, we say the dimension of TB_1 is *DI*. The dimension of the predefined time base *Universal* is *universal*.

Semantics

Given a time base declaration of the form above, the meaning of time base identifier TB_1 in some given variable assignment, is a

function that maps every value *r* to (*r***m*) / (k_1*k_2), where m is the meaning of *texp* in the given variable assignment, k_1 is the literal denoted by *LIT*, and k_2 is the factor of *UN* in dimension *DI*.

5.2.3 TimeBaseRelation

Expressing relation between time bases is mandatory to build a global perception of time. When timing constraints refer to multiple time bases, it results in a partially ordered set of instants from these time bases and corresponds to the global temporal perception of system behavior.

Metamodel for TimeBaseRelation

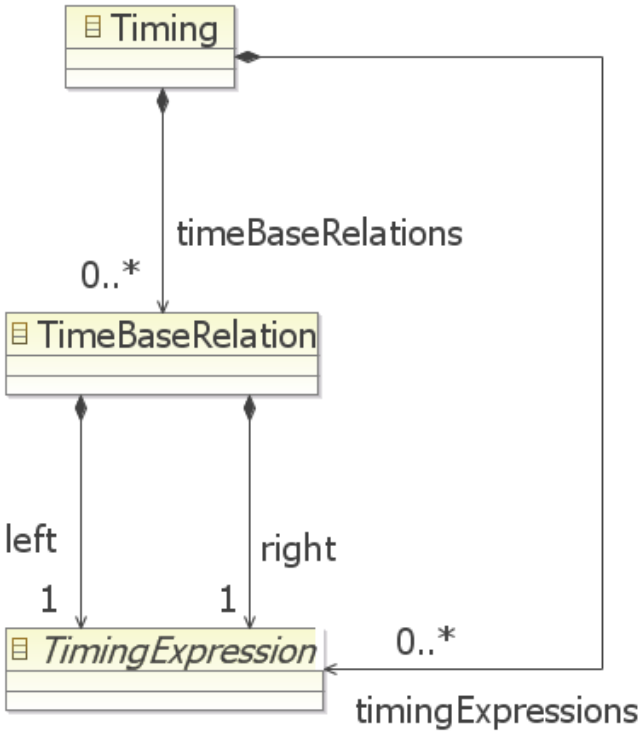

Figure 23: Metamodel Showing TimeBase Relations.

TimeBaseRelation is used to give equivalence between different time bases. More precisely, it specifies equality between *left* and *right timing expressions*.

Syntax Description

See TimingExpression section 5.3

5.3 TimingExpression

Timing Expression Capabilities

A Symbolic Timing Expression (STE) is a way to specify parameterized expressions between different time bases as motivated in the previous section. TimingExpression provides free

variables, constants, values and operators to express timing constraints made of a suitable set of arithmetic operators mixing constant values and symbolic identifiers. The language integrates basic arithmetic operators with timing values represented either by constant values or variables.

Timing expressions are used by the timing constraints in TADL2 in order to express the duration such as maximum/minimum delay, period, jitter and tolerance duration.

There are three different timing expressions: *ValueTimingExpression*, *VariableTimingExpression* and *SymbolicTimingExpression*. Figure 24 shows the part of the TADL2 metamodel representing timing expression.

Metamodel for Timing Expressions

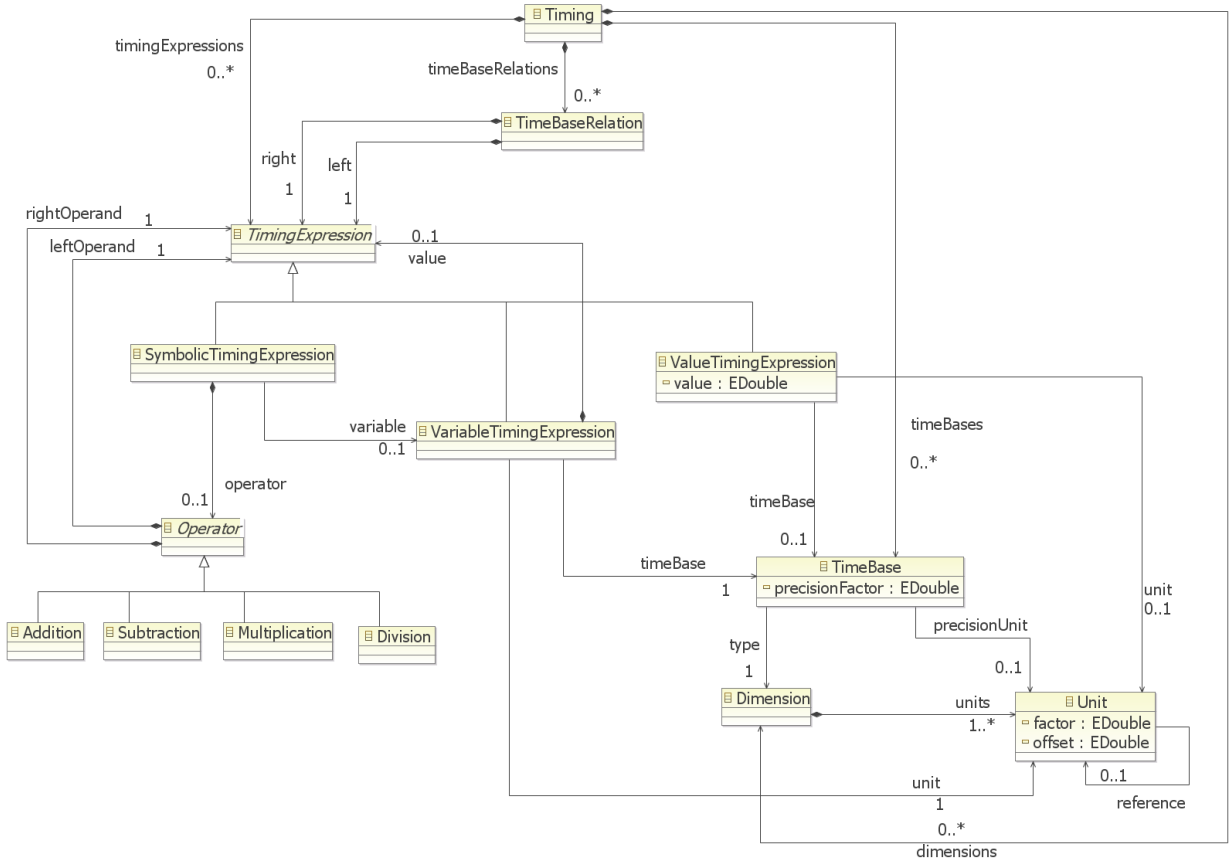

Figure 24 Part of the TADL2 metamodel Representing Timing Expressions.

ValueTimingExpression can have a unit and a time base as type. TADL2 is aimed to be a declarative language. Therefore, we have only free variables, constants and values. Please note that *ValueTimingExpression* does not have a name.

VariableTimingExpression stands for free variables and constants. When a value is assigned to a variable, the variable becomes a constant.

In *SymbolicTimingExpression*, the language integrates basic arithmetic operators such as *addition*, *subtraction*, and *multiplication* associated with timing values. There are some implicit constraints in the TADL2 metamodel which are not shown in Figure 24. The

constraints can be written in OCL [9] form in order to check them in the metamodel. These constraints are the following.

- *SymbolicTimingExpression* cannot have both an *Operator* and a reference to *VariableTimingExpression* (the association variable in Figure 24). It is not allowed to have an expression like {((X + Y), Z)}.
- The left hand side of *TimeBaseRelation* cannot be *SymbolicTimingExpression* with *Operator*. It can only be *VariableTimingExpression* or *ValueTimingExpression* with *Unit* and *TimeBase*.

Syntax Description

A *Timing Expression*, denoted by *texp*, is a term built from an arithmetic expression by applying an optional unit and referencing an optional time base. It stands for a value in the real number system extended with positive and negative infinity.

Grammar

texp	::=	*aexp*
	|	*aexp UN*
	|	*aexp* **on** *TB*
	|	*aexp UN* **on** *TB*

Semantics

Given a particular variable assignment, the meaning of a timing expression *texp* in that assignment is a value in the real number system extended with positive and negative infinity. Depending on the form of *texp*, this value is defined as follows:

- If *texp* is of the form *aexp*, its meaning is the meaning of *aexp* in the given variable assignment.
- If *texp* is of the form *aexp UN*, its meaning is *r* * *k*, where *r* is the meaning of *aexp* in the given variable assignment, and *k* is the factor of *UN* in the *Universal* time base.
- If *texp* is of the form *aexp* **on** *TB*, its meaning is *f* (*r*), where *f* is the meaning of *TB* in the given variable assignment, and *r* is the meaning of *aexp* in the same assignment.
- If *texp* is of the form *aexp UN* **on** *TB*, its meaning is *f* (*r* * *k*), where *f* is the meaning of *TB* in the given variable assignment, *r* is the meaning of *aexp* in the same assignment, *k* is the factor of *UN* in *DI*, and *DI* is the dimension of *TB*.

Examples

Declaration of an angular dimension:

Dimension *angle* { *degree*: 1,
rotation: 360 }

Declaration of the required *universal* dimension:

Dimension *universal* { *sec*: 1000000000*,*
ns: *1*,

micros: 1000,
ms: 1000000 }

Declaration of time bases with different scalings:

TimeBase *Ecu1 : universal*
{ 100 *micros* **on** *Ecu1* = 96 *micros* }

TimeBase *Ecu2 : universal*
{ 1 *ms* **on** *Ecu2* = 1500 *micros* **on** *Ecu1* }

TimeBase *Slow : universal*
{ 1 *ms* **on** *Slow* = 1 *sec* }

Examples of equivalent expressions:

3 *ms* **on** *Ecu1* = 3 * 1000 * 96 / *100* = 2880 = 2880 *micros*

3 *ms* **on** *Ecu2* = 3 * 1500 micros **on** *Ecu1* = 4320 *micros*

3 *ms* **on** *Slow* = 3 * 1000 * 1000000 / 1000 = 3000000 *micros*

An angular time base, connected to the *universal* dimension via *Ecu1*:

TimeBase *Crank: angle*
{ 1 *rotation* **on** *Crank* = *speed ms* **on** *Ecu1* }

An angular clock running at twice the speed of *Crank*:

TimeBase *Cam: angle*
{ 2 *degree* **on** *Cam* = 1 *degree* **on** *Crank* }

Equivalent expressions:

3 *degree* **on** *Crank* = 3 * *speed* / 360 *ms* **on** *Ecu1*
= (3 * speed / 360) * 1000 * 99 / 100
= *speed* * 8 *micros*

3 *degree* **on** *Cam* = 2 * 3 *degree* **on** *Crank*
= (6 * speed / 360) * 1000 * 99 / 100
= *speed* * 16 *micros*

Alternative declaration of *universal* with twice the semantic precision:

Dimension *universal* { *sec*: 2000000000,
ns: *2,*
micros: 2000,
ms: 2000000 }

Ill-formed timing expressions:

~~*3 degree* **on** *Slow*~~

~~*2 micros* **on** *Cam*~~

6 Probabilistic Timing Constraints

This section presents the extension of the basic timing constraints of TADL2 with probabilistic parameters, which can be either based on distributions or follow the weakly-hard approach, which was introduced to express that not more than a given number of deadlines may be missed within a time window [12][13]. The goal of these additional parameters is to allow the expression of more fine-grain information than the usual interval between the best case and the worst case.

6.1 Constructs for Probabilistic Constraints

Probabilistic timing information can be used in different ways to represent different abstractions. For example, one may be interested in probabilistic information based on distributions or instead in weakly-hard constraints which express that not more than a given number of deadlines may be missed within a time window.

In the following paragraphs we present separately time distributions and weakly-hard expressions. Then we show how basic constraints can be extended with such a probabilistic expression and what their semantics then becomes.

Grammar

ProbabilisticExpression	*::=*	*TimeDistribution*
	|	*WeaklyHardExpression*

6.1.1 Definition of Time Distributions

In order to offer flexibility to the designer, we propose two different representations for distributions. First, it is possible to define a distribution function by choosing it from a predefined set which contains the standard distribution functions, namely the uniform, normal (also known as Gaussian), Gumbel, Fréchet and Weibull distribution functions. The other option is to provide a histogram representation of distributions, where only the intervals defined by a given partition of [*lower, upper*] are associated with probability values.

Grammar

TimeDistribution	*::=*	**uniform**
	|	**normal** *Real Real*
	|	**gumbel** *Real LIT*
	|	**frechet** *LIT LIT*
	|	**weibull** *Real LIT LIT*
	|	*ProbaInterval+*
ProbaInterval	*::=*	**pr [** *TimingExpression* **;** *TimingExpression* **] =** *ProbaValue*

ProbaValue	*::=*	*LIT[0;1]*
Real	*::=*	*LIT*
	|	*-LIT*

A ProbaValue is a real number between 0 and 1 used to express probability values.

Note that all parameters of the predefined distributions are unitless. This means that a conversion must be performed from quantities of type TimingExpression to Float. This can be done by converting any TimingExpression into the universal TimeBase and then remove the unit.

Semantics

Time distributions can only be defined within constraints (such as StrongDelayConstraint) having as attributes two TimingExpression denoted mostly *lowe*r and *upper*, with *lower* ≤ *upper*. These attributes are used to define the definition domain of the distribution. In other words, the semantics of a time distribution is parameterized by *lower* and *upper*. It is important to note here that we will also use Time distributions to describe jitter, and in this case the bounds will be 0 and *jitter* instead of *lower* and *upper*. The following explanations also apply to these distributions.

The semantic of the predefined distributions is as usual. For example, the semantics of a 'uniform' distribution between *lower* and *upper* (which have been rendered unitless as explained in the previous paragraph) is a function that associates with every interval $[t_1;t_2]$ included in [*lower*, *upper*] a probaValue equal to (t_2-t_1) / (*upper-lower*). The only difference concerns situations where the predefined distributions may have values outside the bounds defined by *lower* and *upper*. In that case it is assumed that these values will be ignored and therefore to ensure that the sum of all remaining probability values P (formally defined as the definite integral of the probability distribution between *lower* and *upper*) is equal to 1, we adapt the standard semantics as follows: the probability of any interval between *lower* and *upper* is divided by P.

We now give the semantics of distributions defined by a histogram. Consider a list of probaInterval { pr $[t_0;t_1] = p_1$, … , pr $[t_{n-1};t_n] = p_n$ } where $t_0 =$ *lower*, $t_n =$ *upper* and $t_0 \leq t_1 \leq \ldots \leq t_n$.

If the sum of all probability values $P = p_1 + ... + p_n$ is equal to 1, as must be the case by definition of a probability distribution, then our distribution is such that the probability to obtain a value in the interval $[t_{i-1};t_i]$ (for $i \in [1, n]$) is equal to p_i.

If the sum of all probability values $P = p_1 + ... + p_n$ is larger than 1, then we consider the distribution to be an over-approximation of the exact distribution. That is, the probability to obtain a value in the interval $[t_{i-1};t_i]$ (for $i \in [1, n]$) is ***smaller than or equal to*** p_i. We proceed similarly if P is smaller than 1.

6.1.2 Constructs for Weakly-Hard Constraints

The timing constraints defined in TADL2 are ***strongly-hard*** in the sense that they must hold for *each* occurrence of some designated

event. For example, the DelayConstraint requires that for each occurrence of the source event, there is at least one occurrence of the target event within a fixed interval relative to the source. It only takes one absent response occurrence to render the whole DelayConstraint violated.

In many situations, a system may in fact work correctly even if a strongly-hard constraint is not satisfied for a bounded number of occurrences. Therefore, TADL2 generalizes the concept of ***weakly-hard*** constraints (which was originally introduced for describing allowed deadline misses) to formalize scenarios in which a bounded number of occurrences are allowed to violate the constraint requirements.

Grammar

WeaklyHardExpression ::= Int Int

The parameters of a weakly-hard expression are denoted *m* and *k.*

Semantics

The semantics of a weakly-hard expression *(m, k)* is that the behavior must satisfy the given constraint ***at least*** *m* times out of *k* consecutive occurrences.

6.2 Probabilistic Extension of Timing Constraints

We offer in TADL2 probabilistic extensions for the following timing constraints: the ExecutionTimeConstraint, the StrongDelayConstraint, the RepeatConstraint, and the RepetitionConstraint (therefore also the PeriodicConstraint).

6.2.1 Probabilistic Extension of the ExecutionTimeConstraint

Description

An ExecutionTimeConstraint limits the time between the starting and stopping of an executable entity (task, function), not counting the intervals when the executable entity (task, function) has been interrupted. Its probabilistic extension allows a quantitative description of the repartition of the execution time.

Attributes

start	: *Event*
stop	: *Event*
preempt	: *Event*
resume	: *Event*
lower	: *TimingExpression*
upper	: *TimingExpression*
probabilisticExecutionTime	: *ProbabilisticExpression*

Semantics

Remember the non-probabilistic semantics of this constraint:
ExecutionTimeConstraint (
start, *stop*, *preempt*, *resume*, *lower*, *upper*)

⇔

$\forall x \in start : lower \leq |[x..stop] \setminus [preempt..resume]| \leq upper$

The semantics of its probabilistic extension depends on whether the *probabilisticExecutionTime* parameter is defined as a TimeDistribution or a WeaklyHardExpression.

In the first case (*probabilisticExecutionTime* is a TimeDistribution), $\forall x \in start$, the probability of the corresponding execution time, namely
$|[x..stop] \setminus [preempt..resume]|$, to be in a given time interval is defined by the distribution *probabilisticExecutionTime.*

In the second case (i.e., if *probabilisticExecutionTime* is a WeaklyHardExpression with parameters *m* and *k*), out of *k* consecutive occurrences $x \in start$, at least *m* satisfy

$lower \leq |[x..stop] \setminus [preempt..resume]| \leq upper$

6.2.2 Probabilistic Extension of the StrongDelayConstraint

A probabilistic StrongDelayConstraint will typically be used for describing the probabilistic information on response time delays obtained by analysis of probabilistic information on execution times.

Description

A StrongDelayConstraint imposes limits between the occurrences of an event called *source* and an event called *target*. Only one-to-one occurrence patterns are allowed, and no stray *target* occurrences are accepted.

Attributes

source	: *Event*
target	: *Event*
lower	: *TimingExpression* = 0
upper	: *TimingExpression* = **infinity**
probabilisticDelay	: *ProbabilisticExpression*

Semantics

Remember the non-probabilistic semantics of this constraint:

StrongDelayConstraint (*source*, *target*, *lower*, *upper*)

⇔

$|source| = |target| \wedge$
$\forall i \leq |source| : \exists x = source(i) : \exists y = target(i) : lower \leq y - x \leq upper$

The semantics of its probabilistic extension depends on whether the *probabilisticDelay* parameter is defined as a TimeDistribution or a WeaklyHardExpression.

In the first case (*probabilisticDelay* is a TimeDistribution), $\forall x \in source$ such that $x = source(i)$ for some $i \leq |source|$, the probability of the corresponding delay *target*(*i*) - x to be in a given time interval is defined by the distribution *probabilisticDelay.*

In the second case (*probabilisticDelay* is a WeaklyHardExpression with parameters *m* and *k*), out of *k* consecutive occurrences *x* of *source*, that is, *source*(*i*) to *source*(*i*+*k*), at least *m* satisfy

$lower \leq target(i) - x \leq upper$

6.2.3 Probabilistic Extension of the RepeatConstraint

Description

A RepeatConstraint describes the distribution of the occurrences of a single event without jitter.

Attributes

event	: *Event*
lower	: *TimingExpression* = 0
upper	: *TimingExpression* = **infinity**
span	: *int* = 1
probabilisticDistance	: *ProbabilisticExpression*

Semantics

Remember the non-probabilistic semantics of this constraint:

RepeatConstraint (*event*, *lower*, *upper*, *span*)

$\Leftrightarrow$

$\forall X \leq event : |X| = span+1 \Rightarrow lower \leq |[X]| \leq upper$

The semantics of its probabilistic extension depends on whether the *probabilisticDistance* parameter is defined as a TimeDistribution or a WeaklyHardExpression.

In the first case (*probabilisticDistance* is a TimeDistribution), $\forall x \in$ *event* the probability that the length of the sub-sequence *X* of *event* starting at *x* and that has *span*+1 occurrences is in a given interval is defined by the distribution *probabilisticDistance.*

In the second case (that is, if the *probabilisticDistance* is a WeaklyHardExpression with parameters *m* and *k*), out of *k* consecutive occurrences *x* of *event*, at least *m* sequences *X* of *event* starting at *x* and containing *span*+1 occurrences of *event* have a length bounded by *lower* and *upper.*

6.2.4 Probabilistic Extension of the RepetitionConstraint

Description

A RepetitionConstraint describes the distribution of the occurrences of a single event including jitter.

Attributes

event	: *Event*
lower	: *TimingExpression* = 0
upper	: *TimingExpression* = **infinity**
span	: *int* = 1
jitter	: *TimingExpression* = 0
probabilisticDistance	: *ProbabilisticExpression*
probabilisticJitter	: *ProbabilisticExpression*

Semantics

The probabilistic semantics of this constraint is directly derived from its non-probabilistic semantics.

RepetitionConstraint (*event*, *lower*, *upper*, *span*, *jitter, probabilisticDistance, probabilisticJitter*)

$\Leftrightarrow$

$\exists X$:

RepeatConstraint (*X*, *lower*, *upper, span, probabilisticDistance*)

$\wedge$

StrongDelayConstraint (*event*, *X*, *0*, *jitter, probabilisticJitter*)

Remember that the jitter distribution is bounded by 0 and *jitter* while the distance distribution is bounded by *lower* and *upper.*

7 Example User Model

The following subsections provide some example systems and illustrate how TADL2 is used to provide timing models for those systems.

7.1 Brake-By-Wire Example

A distributed Brake-By-Wire (BBW) application with anti-lock braking functionality is given to illustrate the use of TADL2.

7.1.1 The Functional Decomposition of the Braking Functionality

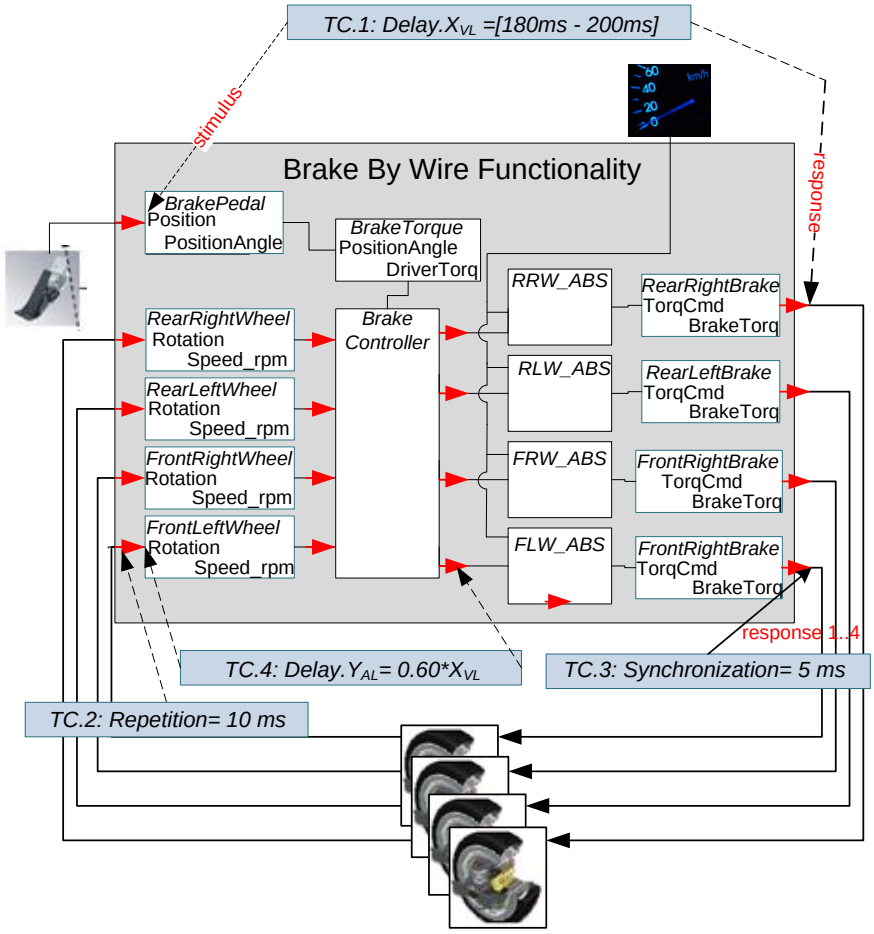

Figure 25. Brake-By-Wire Functional View.

The BBW is composed of two mains functions. Firstly, a brake controller reads wheel speed sensors and a brake pedal sensor. The brake controller computes the desired brake torque to be applied at the four wheels. In addition to this basic brake controller functionality, a second function Anti blocking System ABS adapts the brake force on each wheel if the speed of one wheel is significantly smaller than the estimated vehicle speed. In this case, the brake force is reduced on that wheel until it regains speed that is comparable with the estimated vehicle speed. The ABS takes as inputs the sensors values on each wheel and the estimated vehicle speed.

7.1.2 Hardware Architecture and Allocation

The hardware architecture and the allocation of BBW functions on this architecture are represented in Figure 26.

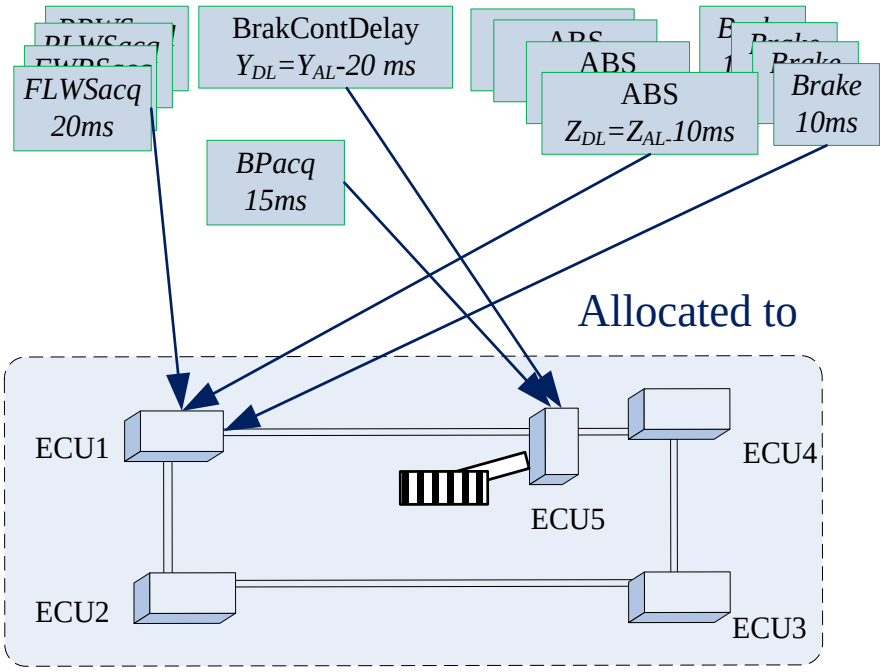

Figure 26. Hardware Architecture and Allocation of BBW Functions.

The hardware platform consists of sensors/actuators and computing parts (five electronic control units connected by a communication bus). Each ECU runs independently with their own temporal references (time base), which is not necessarily (well) synchronized with the other one and the communication between them is still mainly asynchronous (despite the use of Time Triggered buses).

7.1.3 Timing Constraints Applied on the BBW System

Figure 27 gives examples of timing constraints (TC) applied to this functional description:

TC. 1*: A* ***Delay*** *constraint X_{VL} is bounded with a minimum value of 180ms and a maximum value of 200ms. This delay is measured from brake pedal stimulus to brakes response. Here, activation of the brake pedal sensor is the stimulus and brake actuation is the response.*

TC. 2: *A Periodic acquisition of wheel sensors must be done with a* ***Repetition*** *constraint of 10 ms.*

TC. 3: *The tolerated maximum* ***Synchronization*** *constraint between first and last wheel brake actuation is 5 ms.*

TC. 4: *The* ***Delay*** *constraint applied on sensor acquisitions and brake controller is a percentage of the initial time budget X_{VL}.*

In the design process based on EAST-ADL and AUTOSAR, the functional description is refined while passing different development levels.

Figure 27 shows a complemented view of BBW timing constraints that follows the functional decomposition through the levels.

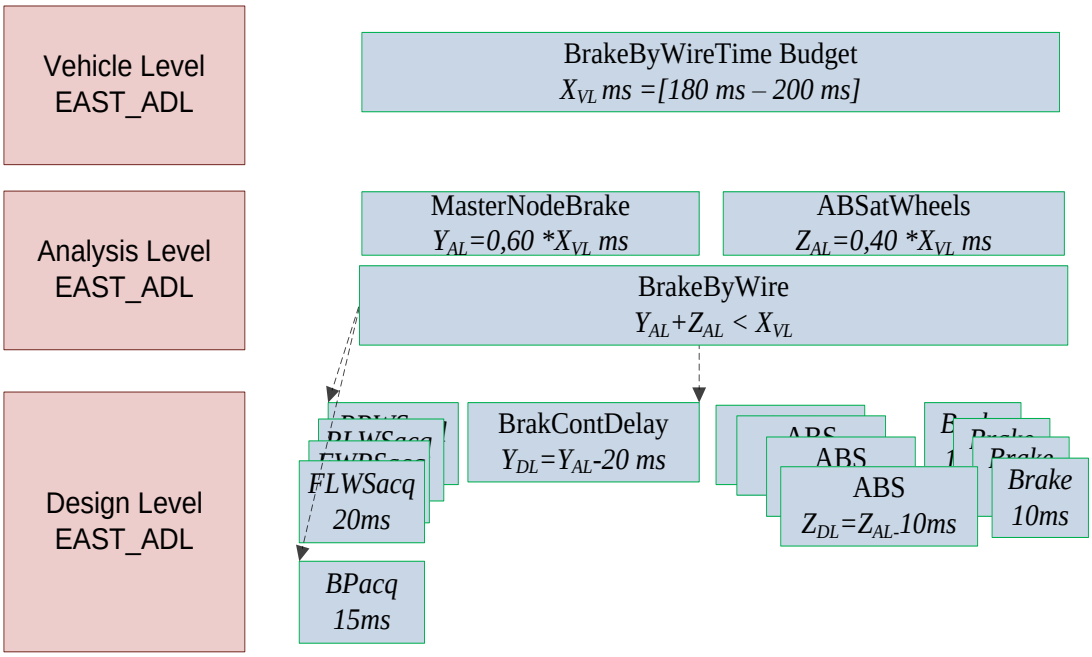

Figure 27. Brake-By-Wire Time Budgeting.

TC. 5: *At the vehicle level, a timing budget of X_{VL} ms is assigned by the supplier. An interval value of [180ms-200ms] should be assigned to X_{VL}.*

TC. 6: *At the analysis and design levels time budgets are split into segments that could be values or percentages of the initial time budget (X_{VL}).*

Additional constraints coming from the hardware characteristic (see Figure 26) should be part of the design such *as the* potential drifts between time bases of computing hardware parts (ECU clocks) or latencies in communication parts (bus, memory access, etc.):

TC. 7: *ECU5 has a drift of 0.02 millisecond for each second compared to the universal time.*

TC. 8: *ECU5 time base goes 2 times faster than time bases of ECU 1 to 4.*

7.1.4 Dimension, Time Base and Time Base Relation Declarations in TADL2 for the BBW System

Referring to the timing constraints TC. 1 to TC. 8, there are up to three different time bases in the system. The time bases are based on a common Dimension (physicalTime).

```
1 TimingSpecification ts1 {
2
3   Dimension physicalTime {
4     Units {
```

```
5           micros{factor 1.0 offset 0.0},
6           ms{factor 1000.0 offset 0.0 reference micros}
7           second{factor 1000000.0 offset 0.0 reference micros}
8       }
9   }
10……..
```

Listing 1. Example Dimension in the BBW System.

Listing 1 shows the dimension declaration. A list of units and attributes for their conversion expression are given saying that there is a conversion for *micros*, *millisecond* (*ms*) and *second* in the *physicalTime* dimension (see lines 5 - 7).

Based on this dimension type, the *TimingSpecification* declares three *TimeBases* (see Listing 2).

```
10   ……
11   TimeBase chrono_time {
12      dimension physicalTime
13      precisionFactor 0.1
14      precisionUnit micros
15   }
16
17   TimeBase universal_time {
18      dimension physicalTime
19      precisionFactor 0.1
20      precisionUnit micros
21   }
22
23   TimeBase Ecu<x> {
24      dimension physicalTime
25      precisionFactor 0.1
26      precisionUnit micros
27   }
28
```

Listing 2. Example TimeBases in the BBW System.

The *chrono_time* time base is declared with a type *physicalTime* (see lines 11-15). The second time base is *universal_time* time base which is the reference time base for the whole system (see lines 17-21). The third time base model the ECUs' clocks.

For all *time bases*, a *precisionFactor* and a *precisionUnit* are given. For the *universal_time* time base, the *precision* means that this time base is able to specify value with a precision of 0.1 micro second.

Time bases can be related to each other by either constant values or dynamic relations (example dependency between °CRK and engine round per minute speed). Expressing relationship between time bases is mandatory for building a global perception of time and ensuring a time safe cooperation over the platform.

```
28
29   TimeBaseRelation tbr2 {
30      (1.0 second on universal_time) = (1.00002 second on ecu5)
31   }
32
33   TimeBaseRelation tbr3 {
34      (1.0 ms on ecu1) = (2.0 ms on ecu5)
35  }
36
```

Listing 3. Example TimeBase Relations in the BBW System.

Listing 3 shows the time base relations. As stated in Listing 1 and Listing 2, *ecu5* has a drift of 0.02 ms for each second compared to

the universal time. Also, the *ecu5* time base goes 2 times faster than time bases of *ecu1* to *4*.

7.1.5 Timing Expressions in TADL2 for the BBW System

Listing 4 extends the timing specification *ts1* with examples of timing expressions for the time budgeting in Figure 27.

```
37
38    var X_VL ms on universal_time    // variable timing expression
39   { (X_VL < (200 ms on universal_time)) }
40   { (X_VL > (180 ms on universal_time)) }
41
42   var Y_AL ms on universal_time    // variable timing expression
43   { (Y_AL := 0.60* X_VL) }              // STE
44
45   var Z_AL ms on universal_time   // variable timing expression
46   { (Z_AL := 0.40* X_VL) }              // STE
47
48   { (Y_AL + Z_AL ≤ X_VL) }             // STE
49
50   var Y_DL ms on universal_time  // variable timing expression
51   { (Y_DL := Y_AL - (20 ms on universal_time)) }       // STE
52
53    var Z_DL ms on universal_time  // variable timing expression
54    { (Z_DL := Z_AL - (10 ms on universal_time)) }       // STE
55
56    var FLWSacq ms on universal_time := 20
57    var BPacq ms on universal_time := 15
58    var Brake ms on universal_time := 10
59
```

Listing 4. Example Timing Expressions in the BBW System.

The *var* keyword is used for defining both free variables and constants. Free variables are useful for characterizing parameters or variant in timing expression or when referring to already existing timing expression. Line 42 gives the variable Y_{AL} declared and accessed in the symbolic Timing expression *line 43*.

A Symbolic Timing Expression allows the assignment of intervals to variables The variable X_{VL} comes from the timing constraint TC. 1, TC. 5, TC. 6. X_{VL} is defined in line 38 with a value interval which comprises between 180 and 200 ms on universal time. Please note that different time bases can be used in upper and lower bounds of the value interval. In this case, the time base relations are used to calculate the time interval for a single time base. *FLWSacq*, *BPacq* and *Brake* used for allocation of functions to ECUs are expressed as constants in Listing 4. The scope of all free variables and constants is the *ts1* timing specification.

Listing 5 gives an example synchronization constraint (see the timing constraint *TC.3*).

```
59
60     Event firstWheelBrakeActuation { }
61     Event secondWheelBrakeActuation { }
62     Event thirdWheelBrakeActuation { }
63     Event fourthWheelBrakeActuation { }
64
65     SynchronizationConstraint sc1 {
66        events firstWheelBrakeActuation,
67                secondWheelBrakeActuation,
68                thirdWheelBrakeActuation,
```

```
69                  fourthWheelBrakeActuation
70
71      tolerance = (5 ms on universal_time)
72
73    }
74  }// end of the timing specification ts1
```

Listing 5. Example Synchronization Constraint in the BBW System.

The constraint is about the maximum tolerated time difference between the first and last wheel brake actuation. The brake actuation is defined for each wheel as an event (see lines 60-63). For these events, the synchronization constraint *sc1* has the attribute *tolerance* which is *ValueTimingExpression* (see line 71).

7.2 BSG-E Example

We take as example an industrial application provided by Delphi: a Box Service Generic-External (BSG-E). This industrial use case illustrates timing constraints coming from both hardware and software parts of the system. BSG-E means in French “Boîtier de Servitude Externe” (Box Service Generic - External). One of the main functions of the product is the management of vehicle front fog lights which is a critical functionality. These lights are also used as cornering lights. Moreover, the BSG-E covers the following main functions:

- ***Function 1.*** Ensure the dialogue with the main car ECU BSI (Box Servitude Internal) by a CAN low speed communication network
- ***Function 2.*** Ensure the internal and output diagnostic
- ***Function 3.*** Management and storage of local defects
- ***Function 4.*** The electrical protection of downstream wires (not loads).

These functions require handling of real-time performance and some timing characteristics of the system.

7.2.1 Functional/Hardware Architecture of the BSG-E

At the Design Level, two different views are proposed to separate the competency concerns: the Functional Design Architecture (see Figure 28) and the Hardware Design Architecture (see Figure 29).

The Functional Design Architecture focuses on the Software (SW) part of the system. It shows components and their interfaces (input and output ports).

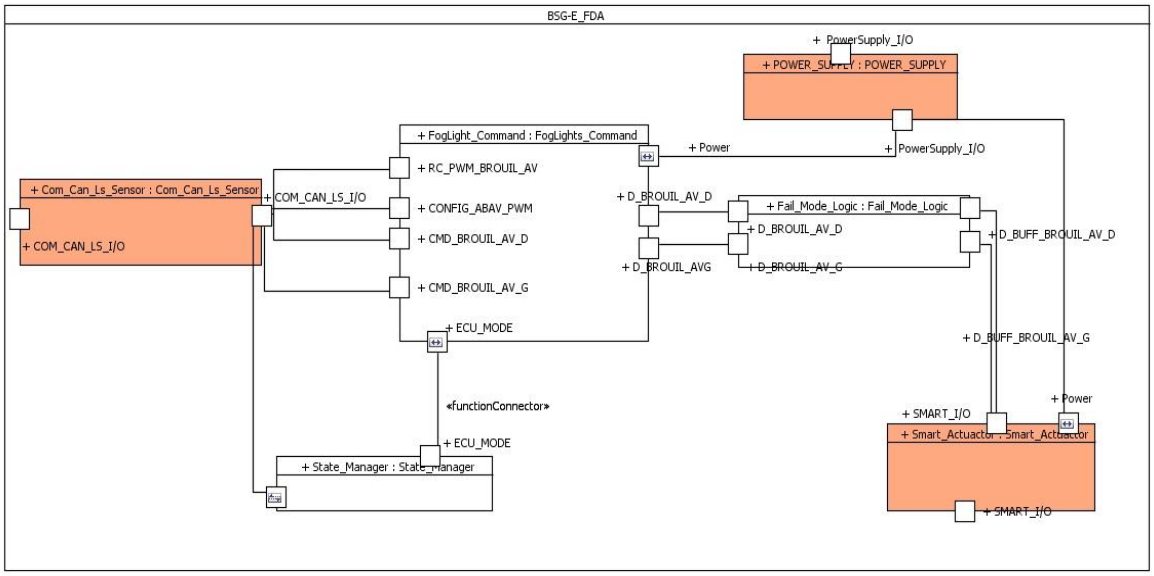

Figure 28. Functional Design Architecture of the BSG-E System.

The BSG-E receives orders from the BSI (Box Servitude Internal) which is the main ECU that communicates with the BSG-E via CAN bus. Communication with the BSI is handled, at the software level, by the Com_Can_Ls_Sensor component (see Function 1). The POWER_SUPPLY component in Figure 28 ensures the acquisition of the alimentation. The FogLights_Command component is the main software component. It receives all messages from the main ECU (BSI) through the CAN frames and manages them for executing the functionalities of the system. Starting from it, the Fail_Mode_Logic component can manage the protection and diagnostic functions (see Function 2) and the Smart_Actuator component receives orders for activating the front fog lights. The State_Manager component handles the internal mode changes of the system.

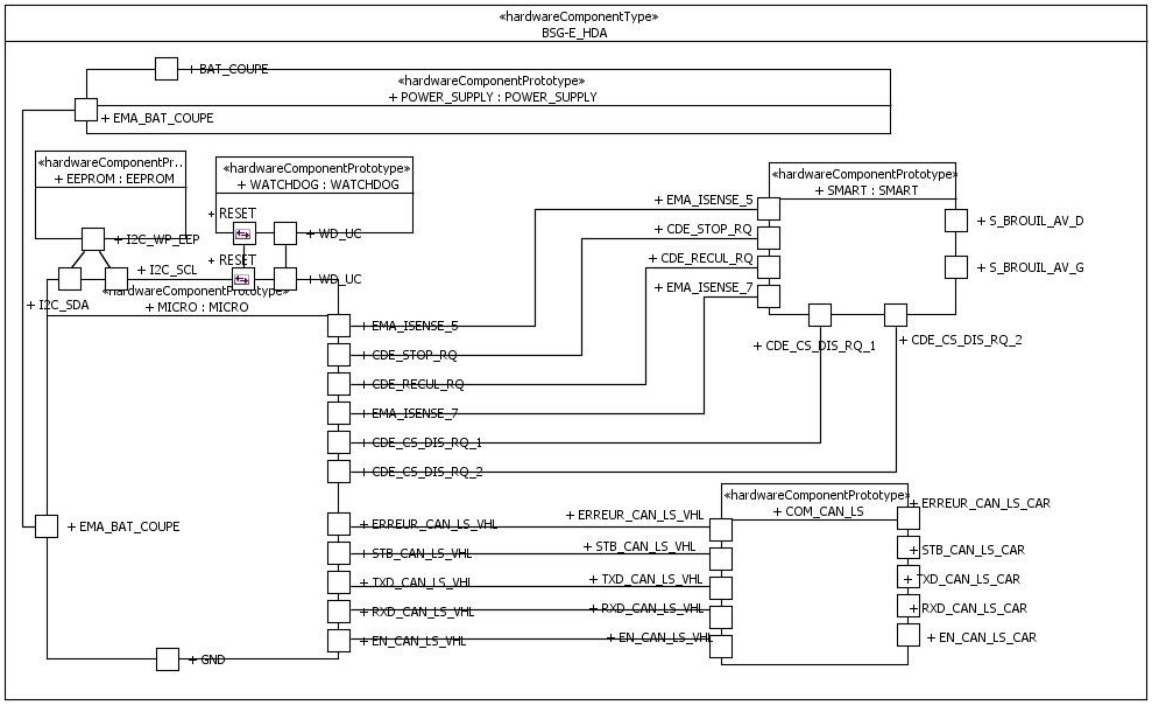

Figure 29. Hardware Design Architecture of the BSG-E System

The Hardware Design Architecture in Figure 29 represents the physical architecture of the system. Each element in the Functional Design Architecture is allocated to one element in the Hardware Design Architecture. One or many SW components are allocated on each Hardware (HW) component.

The MICRO component realizes the FogLights_Command, State_Manager and the Fail_Mode_Logic functions which appear in Figure 28. The SMART, POWER_SUPPLY and Com_Can_Ls are hardware components. The SMART is a driver to complete output command control and the POWER_SUPPLY ensures the alimentation distribution. The Com_Can_Ls is a bus used for the network management and control. Thus, these components manage the first two functions of the BSG-E (see Function 1 and Function 2). Another two functions are specifically managed at the HW level by the EEPROM and WATCHDOG components. The EEPROM, a memory component, is used to manage the SMART defect counter memorization and also to store the configuration data of the BSG-E (see Function 3). The WATCHDOG is an ASIC that triggers the system reset if the MICRO quits the “normal” mode operation (see Function 4).

Connectors between components are also refined regarding the system architecture. Output and input lines in the Functional and Hardware Design Architectures are submitted to timing requirements.

7.2.2 BSG-E Requirements Including Timing Characteristics

Some hardware components (together with the software functions they realized) are subjects of timing constraints. The BSG-E system contains timing constraints of different nature such as delay, synchronization and arbitrary constraints. In this section, we present the textual timing requirements for the BSG-E system obtained during the requirements analysis phase and the formalization of these requirements in TADL2. We use the TADL2 textual concrete syntax.

Timing Requirements for the POWER_SUPPLY

When the vehicle is under tension, all the components including the BSG-E are switched on. The internal power supply acquisition is done periodically through the EMA_PERM3 line after filtering of the initial voltage read (see Figure 29). Requirements PWS_1 and PWS_2 are about timing characteristics of the power supply acquisition.

Requirement ID	Description
PWS_1	PERM3 (+BAT_COUPE) - Analog Input $EMA\_PERM3 \approx \frac{+PERM3}{5}$ The power supply needs to be monitored to manage the diagnostics link with its value.

Requirement ID	Description
PWS_2	The acquisition period should be 5ms with a filtering done on 3 samples. So the EMA_PERM3 voltage value must be evaluated every 15ms to determine its level.

In Listing 6, we give the TADL2 specification for the PWS_1 and PWS_2 requirements. The specification has two periodic constraints. Please note that we use the dimension and time base declarations given in the BBW example.

```
1   var AcqPerm  ms on universal_time :=5.0
2
3   Event HAD_PowerSupply_PERM3 { }
4   Event HAD_PowerSupply_EMA_PERM3 { }
5
6   PeriodicConstraint pc1{
7       event HAD_PowerSupply_PERM3
8       period = AcqPerm
9       minimum = 0.0
10      jitter = 0.0
11  }
12
13  PeriodicConstraint pc2{
14      event HAD_PowerSupply_EMA_PERM3
15      period = (3*AcqPerm)
16      minimum = 0.0
17      jitter = 0.0
18  }
```

Listing 6. TADL2 Specification for the PWS_1 and PWS_2 Requirements.

We have two periodic constraints (the pc1 for the PWS_1 and the pc2 for the PWS_2). The events HAD_PowerSupply_PERM3 and HAD_PowerSupply_EMA_PERM3 are declared for power supply monitoring and acquisition (see lines 3 and 4). These events are attached to the corresponding input and/or output ports of the FDA/HDA.

The period value for the events is declared as a constant (see line 1). The pc1 and pc2 periodic constraints describe periodic occurrence of the events for power supply monitoring and acquisition with periods AcqPerm and 3*AcqPerm. The variable AcqPerm is used twice in two different constraints.

Timing Requirements for the MICRO

The MICRO is the component which realizes the State_Manager whose role is to handle internal mode changes of the system. After power is switched ON, the BSG-E is initialized and it gets into the transitory mode INIT. When the system gets into a stable mode, it carries out its associated functions. It can also get into the DEGRADED or RESET mode if an abnormal operation is detected. The following MICRO_1 requirement is the timing requirement for the mode transitions.

Requirement ID	Description
MICRO_1	- When the BSGE enters into the **INIT** mode, its initialization must be performed. - BSG_E must stay in the **INIT** mode for a maximum time of **T_init**. **- T_init** represents the time for the following transitions: OFF=>INIT=>NORMAL or RESET=>INIT=>NORMAL. - The BSG_E initialization time **T_init** corresponds to the time between the detection of rising edge of power supply present on EMA_PERM3 (EMA_BAT_COUPE) and the consumption of the first frame CAR_CDE_BSE. - This must be lower than **40 ms.** - In case of reset, **T_init** is the duration calculated between the reset activation and the consumption of the first frame CAR_CDE_BSE.

isting 7 gives the TADL2 specification for the MICRO_1 timing requirement.

```
1   Event EMA_PERM3 { }
2   Event CAR_CDE_BSE { }
3   Event RESET { }
4
5   var T_init ms on universal_time := 40.0
6
7   DelayConstraint dc1_a {
8      source EMA_PERM3
9      target CAR_CDE_BSE
10     lower = 0.0
11     upper = T_init
12  }
13
14  DelayConstraint dc1_b {
15     source RESET
16     target CAR_CDE_BSE
17     lower = 0.0
18     upper = T_init
19  }
```

Listing 7. TADL2 Specification for the MICRO_1 Requirement.

The specification has two delay constraints with three events. The minimum and maximum duration between the occurrences of target and source events are given by the attributes lower and upper. The dc1_a delay constraint states that the duration between the detection of the rising edge of the power supply in the EMA_PERM3 and the consumption of the first frame CAR_CDE_BSE should be less than 40 ms. The dc1_b delay constraint states the same timing constraint between the RESET activation and the consumption of the first frame CAR_CDE_BSE.

Timing Requirements for the SMART

The SMART driver is the component that completes the output control commands S_BROUIL_AV_D and S_BROUIL_AV_G (see Figure 29). In case of normal operation, i.e. the system is in the NORMAL mode, the fog lights are activated with the outputs S_BROUIL_AV_D and S_BROUIL_AV_G.

Requirement ID	Description
SMART_1	The BSG_E outputs (*S_BROUIL_AV_D and S_BROUIL_AV_G*) have to be activated or deactivated in less than **10 ms** for the CAR_CDE_BSE frame reception. This time is calculated between the end of the reception of the frame and the real output commutation.

In Listing 8, we give the TADL2 specification for the SMART_1 timing requirement. The specification has two delay constraints with three events.

```
1   var BSG_E_O_Delay  ms on universal_time := 10.0
2
3   Event CAR_CDE_BSE { }
4   Event S_BROUIL_AV_D { }
5   Event S_BROUIL_AV_G { }
6
7   DelayConstraint dc2_a{
8       source CAR_CDE_BSE
9       target S_BROUIL_AV_D
10      lower = 0.0
11      upper = BSG_E_O_Delay
12  }
13
14  DelayConstraint dc2_b{
15      source CAR_CDE_BSE
16      target S_BROUIL_AV_G
17      lower = 0.0
18      upper = BSG_E_O_Delay
19  }
```

Listing 8. Example TADL2 Specification for the SMART_1 Requirement.

The dc2_a delay constraint is for the activation of the S_BROUIL_AV_D. After the consumption of the first frame CAR_CDE_BSE (see lines 3 and 8 for the event CAR_CDE_BSE), the S_BROUIL_AV_D should be activated in less than 10 ms. The dc2_b is a similar delay constraint for the activation of the S_BROUIL_AV_G.

The two outputs S_BROUIL_AV_D and S_BROUIL_AV_G correspond to the left and right fog lights respectively. When they are commuted, the driver must see them simultaneously activated. The minimum dephasing time between the two signals should be very low (see the SMART_2 requirement).

Requirement ID	Description
SMART_2	For S_BROUIL_AV (“Brouillards AV allumés”), the dephasing time between right and left outputs must be lower than **25 ms**

Listing 9 gives the TADL2 specification for the SMART_2 timing requirement with a synchronization constraint.

```
1   var dephasing_GD ms on  universal_time  := 25.0
2
3   SynchronizationConstraint sc1 {
4       events S_BROUIL_AV_G, S_BROUIL_AV_D
5       tolerance = dephasing_GD
6   }
```

Listing 9. TADL2 Specification for the SMART_2 Requirement.

The sc2 constraint is about the maximum tolerated time difference between the activation of left and right fog lights (the S_BROUIL_AV_D and the S_BROUIL_AV_G). The activation of left and right fog lights is defined by two events (see line 4). For these

events, the synchronization constraint sc1 has the attribute tolerance with the constant dephasing_GD which is 25 ms (see line 5).

Timing Requirements for the WATCHDOG

The WATCHDOG drives the following operations:

- Drive to specific value of the buffer outputs in order to drive some specific BSG outputs using the WD_UC line (see Figure 29).
- The WATCHDOG safe mode: reset the BSG µC through the input line RESET.

The WD_UC line is being triggered periodically. It is falling edge sensitive, i.e. the signal on the line is read only at the low state. Furthermore, this signal must be present for a minimum time. Otherwise, it is too short to be handled correctly by the WATCHDOG.

Requirement ID	Description
WD_1	The WD_UC line is falling edge sensitive.

Requirement ID	Description
WD_2	The WD_UC signal must be present at low state for at least 6µs to be taken into account by the WATCHDOG.

In Listing 10, we give the TADL2 specification for the WD_1 and WD_2 requirements with a delay constraint.

```
1   var WD_UC_Hold micros on universal_time := 6.0
2   var infinity second on universal_time := 100000000000.0
3
4   Event WD_UC_fallingEdge { }
5   Event WD_UC_risingEdge { }
6
7   DelayConstraint dc4 {
8       source WD_UC_fallingEdge
9       target WD_UC_risingEdge
10      lower = WD_UC_Hold
11      upper = infinity
12  }
```

Listing 10. TADL2 Specification for the WD_1 and WD_2 Requirements.

The dc4 delay constraint states that the WD_UC line should be maintained at a lower state for at least 6 microsecond (see lines 7 - 12).

7.3 Timing Constraint and Symbolic Timing Expressions

Consider time budgeting as an example of a model where event chains and timing constraints from chapter 3 are used together with timing expressions from chapter 5.

Short Description

A time budget is the breakdown of an end-to-end latency constraint according to the internal structure of the constrained system. If the signal path from stimulus to response consists of a sequence of subsystems, a time budget assigns latency constraints to each of these in such a way that the original end-to-end latency constraint is not violated.

For the time budget concept to be meaningful, the sequence of subsystems involved must be constructed such that the response event of one subsystem is the stimulus of the next one, and so on.

Example

The logical connection between a stimulus and a response event can be expressed using an *EventChain* construct:

```
c = EventChain {
      stimulus = EventFunctionFlowPort { port = PedalIn },
      response = EventFunctionFlowPort { port = BrakeOut }
}
```

To express a maximal reaction time of *200 ms* for the event chain *c* one writes:

```
r = ReactionConstraint {
      scope = c,
      maximum = 200 ms
}
```

To express that an event chain *c* actually consists of internal two segments *c1* and *c2* joined by a common event, one may write:

```
c = EventChain {
      stimulus = EventFunctionFlowPort { port = PedalIn },
      response = EventFunctionFlowPort { port = BrakeOut },
      segment  = < c1, c2 >
}

c1 = EventChain {
      stimulus = EventFunctionFlowPort { port = PedalIn },
      response = EventFunctionFlowPort { port = TorqueOut },
}

c2 = eventChain {
      stimulus = EventFunctionFlowPort { port = TorqueOut },
      response = EventFunctionFlowPort { port = BrakeOut },
}
```

If the end-to-end reaction time for chain *c* is given by constraint *r* above, a valid time budget for *c* might consist of the following reaction constraints:

```
r1 = ReactionConstraint {
      scope = c1,
      maximum = 120 ms
}

r2 = ReactionConstraint {
      scope = c2,
      maximum = 80 ms
}
```

This sequence of reaction constraints is equivalent to the end-to-end constraint *r* because the sum *r1.maximum* + *r2.maximum* equals *r.maximum*. A similar equivalence condition applies if the lower attribute is used as well. The values in the *maximum* attributes are modeled by instances of ValueTimingExpression with the unit *ms*.

By using symbolic time expressions and the order constraint, it is possible to express the dependencies between attributes that makes a time budget valid:

```
r1 = ReactionConstraint {
      scope = c1,
      maximum = T1
}

r2 = ReactionConstraint {
      scope = c2,
      maximum = T2
}

o = ComparisonConstraint {
      leftOperand = r1.maximum + r2.maximum,
      rightOperand = r.maximum,
      operator = LessThanOrEqual
}
```

T1 and *T2* are variables standing for unknown (yet to be negotiated) timing parameters, these are modeled by instances of VariableTimingExpression. They may be replaced with any values as long as constraint *o* is satisfied. Fixing just one of the variables results in a more direct constraint on the other one – information that may be crucial during time budget negotiations.

The *leftOperand* of the constraint *o* is a SymbolicTimingExpression with an Addition operator and the operands are references to the same VariableTimingExpression instances already modeled in the reaction constraints *r1* and *r2*.

8 Language Modeling Environment

TADL2 is formalized in a metamodel that is connected to the combined metamodels of EAST-ADL and AUTOSAR. This provides the basis for the definition of an exchange format in the same way as is defined for AUTOSAR [14]. As a part of this deliverable you find the metamodel XMI, which is an export from the Enterprise Architect UML tool. The documentation of each metaclass in the model has been exported and is found in Appendix A. An XML schema (XSD) can be generated from the metamodel and this serves as a description of the format of the exchanged timing information between tools using XML containing TADL2. The TADL2 semantics is instructing the supporting tools on how to interpret the information, see the schematic overview in Figure 30.

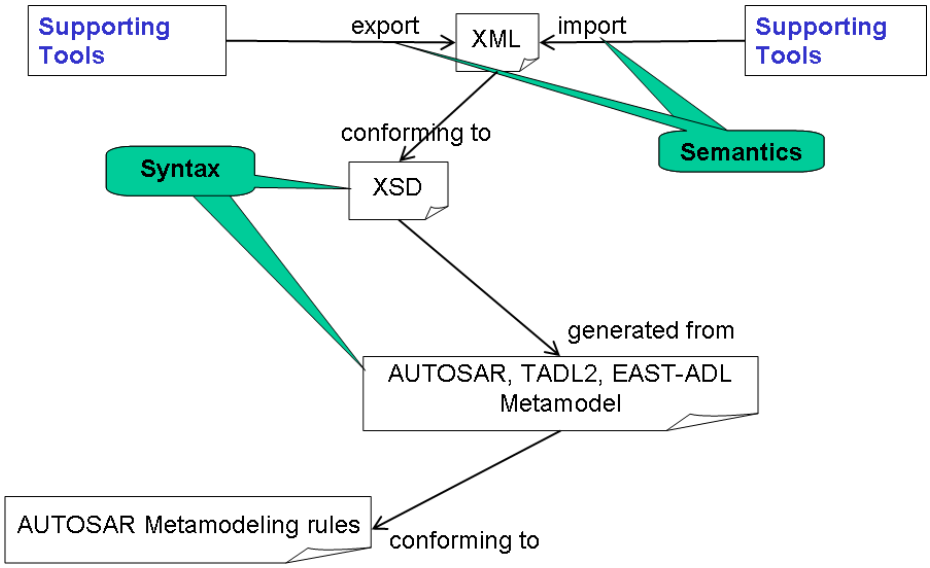

Figure 30. Tools operating on TADL2 use XML as exchange format.

9 References

[1] TIMMO D6 TADL: Timing Augmented Description Language, https://svn-vu-4.c-lab.de/svn/T2U/20_TIMMO/01_Deliverables/TIMMO_D6.pdf

[2] ATESST2 2008—2010 http://www.atesst.org/

[3] D4.1.1 EAST-ADL2 Language definition, http://www.atesst.org/home/liblocal/docs/ATESST2_D4.1.1_EAST-ADL2-Specification_2010-06-02.pdf

[4] AUTOSAR 4 Timing extension, http://www.autosar.org/download/R4.0/AUTOSAR_TPS_TimingExtensions.pdf

[5] Björn Lisper and Johan Nordlander. “A Simple and Flexible Timing Constraint Logic”. In 5th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (ISoLA), 15-18 October 2012, Amirandes, Heraklion, Crete.

[6] Walicki, Michał (2011), *Introduction to Mathematical Logic*, Singapore. World Scientific Publishing, ISBN 978-981-4343-87-9.

[7] Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman (2006). Compilers: Principles, Techniques, and Tools (2nd Edition). Prentice Hall, ISBN 978-0321486813.

[8] AUTOSAR 4 Metamodel, http://www.autosar.org/download/R4.0/AUTOSAR_MMOD_MetaModel.zip

[9] Object Constraint Language: OMG Available Specification Version 2.0 formal/06-05-01

[10] Maria Victoria Cengarle and Alexander Knapp. Towards OCL/RT. In Lars-Henrik Eriksson and Peter Lindsay, editors, Proc. 11th Int. Symp. Formal Methods Europe, volume 2391 of Lecture Notes in Computer Science, pages 390-409. Springer- Verlag, 2002.

[11] AUTOSAR: Generic Structure Template (V3.0.0 R4.0 Rev 1), http://www.autosar.org/download/R4.0/AUTOSAR_TPS_GenericStructureTemplate.pdf

[12] G. Bernat, “Specification and Analysis of Weakly Hard Real-Time Systems,” PhD thesis, Departament de Ciències Matemàtiques i Informàtica. Universitat de les Illes Balears. Spain, Jan. 1998.

[13] Guillem Bernat, Alan Burns, and Albert Llamosi. 2001. “Weakly Hard Real-Time Systems”. *IEEE Trans. Comput.* 50, 4 (April 2001), 308-321.

[14] AUTOSAR Metamodeling rules and exchange format, http://www.autosar.org/download/R4.0/AUTOSAR_TR_XMLPersistenceRules.pdf

[15] “ACC: Adaptive Cruise Control – The Bosch Yellow Jackets” Edition 2003

10 Appendix A - Current modeling of timing

10.1 EAST_ADL and AUTOSAR

Current Definition of Time Bases

EAST-ADL and AUTOSAR give the possibility to model different units and types of units in a TimeDuration class for EAST-ADL and in the MultidimensionalTime for AUTOSAR. References to these units are attributes of this class (cf. Figure 31).

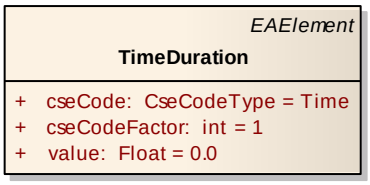

Figure 31: EAST-ADL metamodel for Units in timing constraints.

It is possible to have in a single design, multiple constraints with lower/upper values expressed by a *TimeDuration/* MultidimensionalTime element of different types: *cseCodeType*

- *cseCodeType are enumerations.*
- cseCodeType are units referring ***implicit*** clocks, namely universal time and the engine rotation angle.
- Relation between units can be obtained via the *cseCodeFactor* attribute for units of the same type.

Main Issues with this Definition

The notion of unit is **not easily extendable**. Current time bases are the chronometric time and, the angular degree. For any other time bases (distance, temperature …) new enumerations lists should be added which will modify the metamodel.

The relation between second and millisecond is very simple to express, because we manage types (and not explicit timebases) but a big issue is that **relation** between these units **should be computed for each TimeDuration/MultidimensionalTime expression** in the model by using the cseCodeFactor.

Main Issues concerning the Relation between Time Bases

The last issue concerns the relation between cseCodeType of different nature. For example s, ms, and angular degree. In this case, due to the implicit nature of time base, it is not possible to express relation between these different time bases.

Timing Expression with AUTOSAR

AUTOSAR has a concept for writing formulas [11] which relates to other model elements in an unambiguous way. This makes dependencies explicit and ensures that the right parameter is

referenced also in a type-prototype hierarchy. For example, one timing constraint may be based on parameters of the left door-lock control while another timing constraint uses the same parameter of the right door lock control. These may have different values, although having the same type definition.

The Formula Language is based on the atpMixedString concept which allows mixing strings (operators and arguments) and model elements (references).

Its specialization to strings containing formulas is called FormulaExpression. The syntax and semantics of FormulaExpression is compliant with C expressions and defined using the ANTLR grammar notation. The FormulaExpression supports arithmetic and logic expressions such as `+, -, *,/, &&, ||, &, |, sin, cos, floor`, etc.

Parameters in the expression are identified using references according to the AUTOSAR structural concepts. The role name of the association is used in the formula and the path to the referenced element is inlined.

10.2 UML_OCL

The Object Constraint Language (OCL) [9] is a declarative and semi-formal specification language for describing rules that apply to any MetaObject Facility (MOF) model or metamodel that cannot otherwise be expressed by diagrammatic notation. For UML, OCL supplements the language by providing expressions that have neither the ambiguities of natural language nor the inherent difficulty of using complex mathematics.

OCL is a precise text language based on first-order predicate logic and using a syntax similar to programming languages that provides constraint and object query expressions. These expressions typically specify invariant conditions that must hold for the system being modeled, queries over objects described in a model and the transformation of system states by means of pre- and post-conditions for operations. An invariant is a Boolean expression that must be true for every instance of some type at any time. A pre-condition is a condition that must always be true just prior to the execution of an operation in order to get the expected effect. A post-condition is a condition that must always be true just after the execution of an operation. The language also allows the definition of variables and operations that can be used within expressions and action clauses, indicating that actions will be taken when some condition becomes satisfied.

OCL statements are constructed in four parts:

- a context that defines the limited situation in which the statement is valid
- a property that represents some characteristics of the context (e.g., if the context is a class, a property might be an attribute)
- an operation (e.g., arithmetic, set-oriented) that manipulates or qualifies a property, and
- Keywords (e.g., if, then, else, and, or, not, implies) that are used to specify conditional expressions.

Examples

Consider the example where the system must enforce that an operation opX never takes longer than 5 seconds to execute. This timing constraint is invariant.

```
Context System::opX()
Inv: self.durT <= 5*Sec
```

Consider an ATM system. Every time the operation *spend* is invoked, the ATM must not be on an error state, it must have a card, the *amount* of money to be withdrawn must be positive and the *depot* must cover the withdrawal. After spend has been executed, the right amount of money must have been spent or some error has occurred.

```
context ATM::spend(amount : Integer)
pre: (state = #ok) and (cardId <> 0) and (amount > 0) and (depot > amount+100)
post: (depot = depot@pre-amount) or (state = #error)
```

We can also define time expressions in guards. For example, we could define a transition with a guard that rejects all eventXs that are dispatched less than 3 seconds after the previous event that entered the source state:

```
[distT >= 3*Sec] eventX / action
```

Another example is to specify that a service be finished within certain time bounds. An OCL solution to such a requirement would be to define an explicit clock attribute requiring the clock to be reset to zero in the pre-condition of the operation and putting a constraint on the clock's value in the post-condition:

```
context System
pre: (clock = 0) and (state = #ok) and : …
post: (clock <= T) and : …
```

So far, OCL shows distinct limitations for formulating temporal constraints as the language does not feature time or signal handling constructs, nor is capable of expressing general liveness properties of systems conveniently. An example is to specify when, the other way round, the occurrence of some (external) signal is waited for during a certain period, and if nothing happens the system has to react in a predefined way. But OCL does not over any convenient means to handle the occurrence of signals and thus to specify the intended behavior.

Nevertheless, different approaches have taken OCL as a basis and developed temporal extensions to enable modelers to specify temporal constraints. One of them it is a temporal extension named OCL for Real Time (OCL/RT) [10].

OCL/RT

OCL/RT is based on the notion of traces of events with timestamps, which allows specifying the temporal behavior of a system. The constraints are evaluated over sequences of system states (system

execution) instead of just on a given state as OCL does. In this context, new operators are defined extending the expressiveness of the original language. These are modal operators **always** and **sometime** (by abbreviating not (always (not c)) to sometime c) over event occurrences. These can be used for specifying deadlines and timeouts of operations and reactions on received signals.

10.3 PSL Property Specification Language

The Property Specification Language PSL, is a language for specifying properties. It is typically used for specifying temporal properties of systems, i.e., properties that deal with the behavior of a system over time. The assumption is that the system has some definition of time points, which may be points at which a system clock ticks (if the system is synchronous), or points at which certain chosen events occur.

PSL includes a type of regular expression called SERE (*Sequential Extended Regular Expression)*. SEREs are used to describe scenarios. The simplest type of SERE is a sequence of Boolean expressions separated by semicolons, such as `{req; !ack; ack}`. This SERE describes a scenario spanning three time points, in which `req` holds at the first time point, `ack` does not hold at the second, and `ack` holds at the third.

Generally, a SERE may describe a set of scenarios. For example, the operator [*] indicates an interval of zero or more time points, in which anything may occur. Therefore, the SERE `{start;[*];done}` describes any scenario that begins with start and ends with done. The `[*]` operator may also be attached to a Boolean expression. The expression `busy[*]` describes an interval of zero or more time points in which busy is true. Additional operators serve as shorthand for longer constructions. For example, `{busy[*4]}` is equivalent to `{busy; busy; busy; busy}`. For any constant number `n`, the expression `busy[*n]` describes a sequence of exactly `n` time points

SEREs may be used as building blocks of PSL properties. Typically, a property may be composed of SEREs using the temporal (suffix) implication operator `|=>`. For example:

```
{[*]; req; ack} |=> {start; busy[*]; done}
```

This property states that any occurrence of the left-hand side scenario must be followed by an occurrence of the right-hand side scenario. In this particular case, `{[*]; req; ack}` describes a sequence of req followed immediately by ack, which may occur at any time point (due to the `[*]` at the beginning of the SERE).

The property states that such a sequence must immediately be followed (starting at the next time point) by a scenario matching `{start; busy[*]; done}`. This property makes a requirement for any occurrence of a `{req; ack}` sequence, at any time point, including overlapping occurrences.

10.4 Expressing TADL2 Constrints using PSL

PSL offers temporal operators like `always` or `next`, which define when a Boolean expression must be valid. These temporal operators combined with SEREs and their corresponding temporal implication operators enable PSL to express and specify timing-related properties.

Furthermore, PSL offers the expressiveness to handle timing constraints and features of TADL2. Therefore the formal timing requirements expressed in PSL can be transformed to TADL2 constraints.

The following examples show the expressiveness of PSL to formalize textual requirements and how these expressions are related to TADL2 constraints. The given timing related requirements are real world examples for an Adaptive Cruise Control (ACC) system [15].

The first textual requirement defines the possible delays (range) for messages from or to the ACC system integrated in the overall vehicle network:

„The transmission of a message from or to the ACC may need 0,5 ms to 10 ms and the transmission latency must not exceed 10 ms.“

Considering a synchronous periodic system with a clock tick rate of 1 us this example can be formalized and expressed with PSL as follows:

```
property MsgDelay;

MsgDelay = always ({MsgSent} |=> {[*500:10000]; MsgRcv});

assert MsgDelay;
```

The property `MsgDelay` defines, that the reception of a message (`MsgRcv`) can and must always occur 500 to 10.000 ticks (0,5...10ms) after the message has been sent (`MsgSent`). The assertion of this property is utilized to verify the property against the implementation (e.g. during simulation).

This PSL property corresponds with the TADL Delay Constraint i.e. can be transformed to a TADL2 Delay Constraint. Furthermore it implicitly defines an order of the events `MsgSent` and `MsgRcv`.

The RADAR sensor of the considered ACC system scans new data with a rate of 10Hz. To guarantee a certain controller quality and correct system behavior we define the textual requirement for the data age:

„The RADAR sensor scans new data with a rate of 10Hz. To ensure calculations based on the most recent values, the control loop cycle duration must not exceed 100 ms.”

This example can be formalized and expressed with PSL as follows (clock tick rate 1 us):

```
property CalcFinish;

CalcFinish = always ({DataScan} |=> {[*0:100000]; CalcResult});

assert CalcFinish;
```

The property `CalcFinish` defines, that the calculation of a control loop result (`CalcResult`) has to be the finished within 100 ms after the input from the RADAR sensor (`DataScan`) to ensure calculations based on most recent values. This means, that the age of an input for a calculation may not exceed 100ms.

This PSL property expresses a TADL2 Age Constraint.

An ACC system has to perform two basic actions to fulfill its functionality: detection and reaction. The combined time consumption of these actions may not exceed a certain value, which depends on the distance to an object, the current speed, scanning rate, computation rate, and the possible value for acceleration and deceleration. The resulting textual requirement can be formulated as:

„The acceleration/deceleration level of the ACC is limited and takes time to build. Additionally to the acceleration/deceleration a detection time, and a reaction time have to be considered. For a correct "distance keeping" the sum of acceleration/deceleration time (depending on relative speed), detection time, and reaction time (depending on scanning rate, computation time, …) must not exceed x ms."

This textual requirement can be formalized and expressed with PSL as:

```
property AdjustSpeed;

AdjustSpeed = always ({detection} |=>
{{[*0:MaxReact]; reaction; [*0:MaxDecel];
acceleration/deceleration}});

assert AdjustSpeed;
```

The property `AdjustSpeed` defines, that after the detection of an object (`detection`) a corresponding reaction (`reaction`) has to be performed within given time bounds (`[*0:MaxReact]`) offering sufficient remaining time (`[*0:MaxDecel]`) for the system to perform the necessary calculation of acceleration/deceleration to set the desired speed (`acceleration/decelaration`) for the "distance keeping".

This PSL property contains TADL2 Delay Constraints for the individual actions. Since `MaxReact` and `MaxDecel` influence each other, it also describes the concept of time budgeting for the different actions as described above. Furthermore, the available time budgets depend on (relative) speed and distance, which results in "implicit" multiform (symbolic timing) expressions.

An additional (optional) feature of ACC systems is the "emergency mode" where the vehicle is decelerated with the maximum possible value to avoid a collision with a "dangerous" obstacle (emergency braking). In this case the ACC system has to detect the dangerous situation, switch from "normal mode" to "emergency mode" and stop the vehicle. The resulting textual requirement can be formulated as:

„In case of a detection of a "dangerous" obstacle the maximum value for deceleration is switched from "normal mode" (2.5 m/s²) to „emergency mode" (8,0 m/s²). Depending on the relative speed and the distance the switching must not take longer than x ms."

Integrated in the “detection and reaction”-requirement shown above, this requirement can be expressed with PSL as:

```
Property EmergencyBrake;

Sequence SwitchMode;

SwitchMode = {isEmergency} |=> {[*0:MAX]; (MODE
==EMERGENCY)}

EmergencyBrake = always ({detection} |=>
{{SwitchMode}; [*0:MaxReact]; reaction;
[*0:MaxDecel]; deceleration}});

assert EmergencyBrake;
```

Here, a sequence `SwitchMode` is integrated in the property `EmergencyBrake`, which corresponds to the property `AdjustSpeed`. This means that this PSL property also covers TADL2 delay contraints, time budgeting and “implicit” multiform (symbolic timing) expressions. Additionally this property supports the concept of mode dependency, since the (timing) behavior of the ACC system depends on the current operational mode.

The examples above showed that PSL offers the expressiveness to handle the timing constraints and features of TADL2. Therefore, textual requirements can be formalized with PSL and transformed to TADL2. The sequences specified in PSL correspond to events and event chains in TADL2 and PSL expressions can be transformed to TADL2 constraints/features and vice versa.

10.5 MARTE-Clock Constraint Specification Language

The UML Profile MARTE for Modeling and Analysis of Real-Time and Embedded (RTE) systems has recently been adopted by the OMG. Its Time Model extends the informal and simplistic Simple Time Package proposed by UML2 and offers capabilities to model explicit discrete/dense and chronometric/logical time both in a common design.

MARTE OMG specification introduces a Time Structure inspired from time models of the concurrency theory and proposes a new clock constraint specification language (CCSL) to specify, within the context of UML, usual logical and chronometric time constraints.

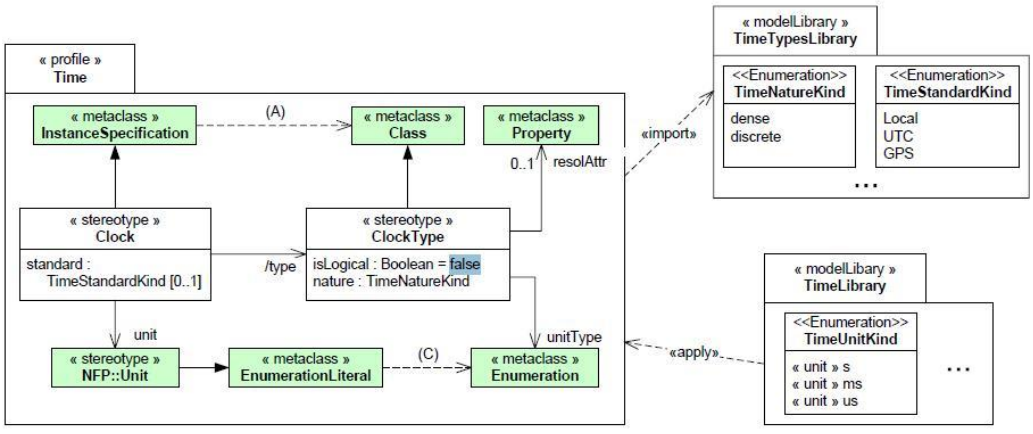

Figure 32: The MARTE Time Profile.

The profile core consists of two stereotypes (ClockType and Clock). These stereotypes provide mechanisms to create multiple time bases of different nature (within a clock type) and to put together features common to several clocks. By this way it is possible to create library of time units. Relations between units (s and ms) can be expressed with OCL. Relation between time bases of different nature (for example ms and angle degree) is modeled with the CCSL language.

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

Synchronous clock constraints rely on coincidence. Subclocking is such a constraint: each instant of the subclock must coincide with one instant of the superclock. Of course, the mapping must be order-preserving. The former discretizes a dense clock. It is mainly used to derive a discrete chronometric clock from IdealClk. IdealClk is a dense chronometric clock, predefined in the MARTE Time Library, and supposed to follow "physical time" faithfully.

For instance:

Clock c10 = IdealClk ***discretizedBy*** *0:0001* *(1)*

Eq. 1 specifies that c10 is a discrete chronometric clock whose period is 0.0001 second, where second is the time unit associated with IdealClk, therefore c10 is a 10 kHz clock.

From this clock others can be derived others:

Clock ECU1 ***isPeriodicOn*** *Idealclk period 10* *(2)*

Clock ECU2 ***isPeriodicOn*** *Idealclk period 12* *(3)*

Eq. 2 reads that there is a tick of the ECU1 every 10th ticks of IdealClk (i.e., the ECU1 has a 1 kHz clock).

In Eq.3 we have a clock ECU2 which has a small drift with the ECU1 E.g., if 1 ms elapsed time on ECU1 represents 1.2 ms on ECU2.

These two CCSL relations create relations between 3 timebases (idealclk, ECU1 and ECU2. These equations are solved using algorithms that provide as a solution partial ordered of clocks instants.

Asynchronous clock constraints are based on *precedence*, which may appear in a strict or anon-strict form.

From *precedence* are derived four new instant relations: *Coincidence, Strict precedence, Independence and Exclusion*

To express clock relations, one can then use these instant relations. For instance, a strict clock *precedence* relation (denoted $\prec$) between two clocks a and b is asynchronous and specifies that for all natural number k, the k^{th} instant of a occurs before the k^{th} instant of *b*:

Such relation could be used for example to express data dependencies and to describe end to end paths.

The ***coincidence*** relation (denoted =) between two clocks imposes a stronger synchronous dependency: the k^{th} instant of *a* must be coincident with the k^{th} instant of *b*:

The same mechanism applies for all relations. Informally, the ***exclusion*** relation (denoted #) between two clocks *a* and *b* specifies that no instants of the clock *a* coincide with one of the clock b.

The ***alternatesWith*** relation (denoted ~) between two clocks *a* and b specifies that instants of the clock *b* are interleaving instants of the clock *a*.

Non Functional Property constraints apply to any time base. While IdealClk is supposed to be perfect, an actual clock may have flaws. CCSL introduces special constraints to specify *stability*, *drift*, *offset* of chronometric clocks.

For example Eq 4 and 5 model the ***repetition rate*** for a function of 5 ms with a jitter of 1ms. F_Start stands for the activation event of the function.

F_Start is Periodic On IdealClock period 5 *(5)*

F_start hasStability 1E-3 *(6)*

Stochastic parameters are available in CCSL Non determinism introduced by such parameters may reflect a partial knowledge about the actual constraints. It may also be a deliberate choice for hiding unnecessary details. Several probability distributions are provided.

The uniform distribution is often used to represent a tolerance interval on a duration.

For example an ***input synchronization constraint*** means that when the first acquisition is done for a data the others acquisitions should be done within a certain delay. The same is for ***output synchronization*** constraint where a maximum delay should elapse between the first actuator setting and the last one.

These delays are generally a random duration. CCSL represent a random duration with the Uniform(0::5).

Example

Finally we give an example of using CCSL for expressing relationship between two time bases of different nature.

The relation between *angle* and *RPM* depends on modes and is given by the two following rules:

- In mode 1: RPM=6000 => 1 angle°=27 us
- In mode 2: RPM=1000 => 1 angle°=167 us.

The introduction of mode in TADL2 is presented in 3.1.

The modeling of these relations is given by the two following expressions:

In mode 1:

In mode 2:


crkAngle isPeriodicOn ChronoTime period 167

11 Appendix B – TADL2 Metamodel

The metamodel of TADL2 is available as an XMI file as a part of this deliverable. The model is an export from Enterprise Architect and should be imported in an EAST-ADL metamodel [3] (in turn depending on the AUTOSAR 4.0.3 metamodel [8]).

The XMI contains only TADL2. The basis is the final language specification from the ATESST2 project (2010-06-02) [2]. The package Timing has been modified in a few ways to avoid clash with TADL2:

1. The EAST-ADL Event has been renamed EventEAST-ADL
2. The EAST-ADL concepts for organize the timing information have been replaced by concepts in TADL2 and have been removed. These metaclasses Timing, TimingDescription, TimingConstraint and EventChain were removed.
3. The EAST-ADL TimingConstraints package has been removed. That package contained AgeTimingConstraint, ArbitraryEventConstraint, DelayConstraint, EventConstraint, InputSynchronizationConstraint, OutputSynchronizationConstraint, PatternEventConstraint, PeriodicEventConstraint, ReactionConstraint, and SporadicEventConstraint. These constraints have been replaced by concepts in TADL2 with updated semantics and syntax.

List of helper timing constraints that are not defined in the metamodel

- RepeatConstraint, instead the RepetitionConstraint is available, where jitter = 0 corresponds to RepeatConstraint.

11.1 TADL2

11.1.1 Overview

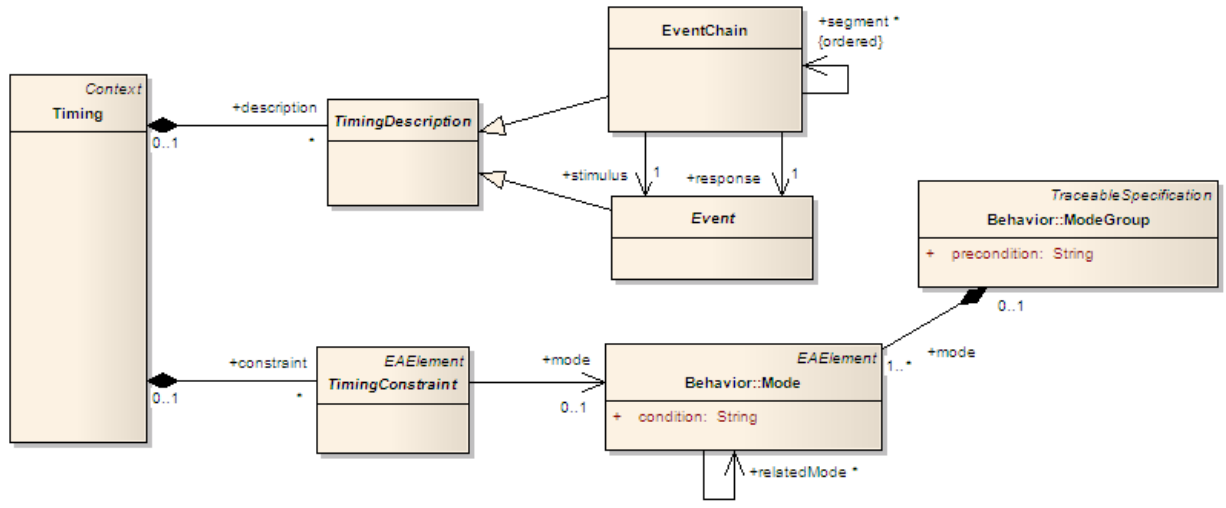

Figure 33: Basic TADL2 elements organized in Timing, with TimingConstraints referring to EAST-ADL Mode.

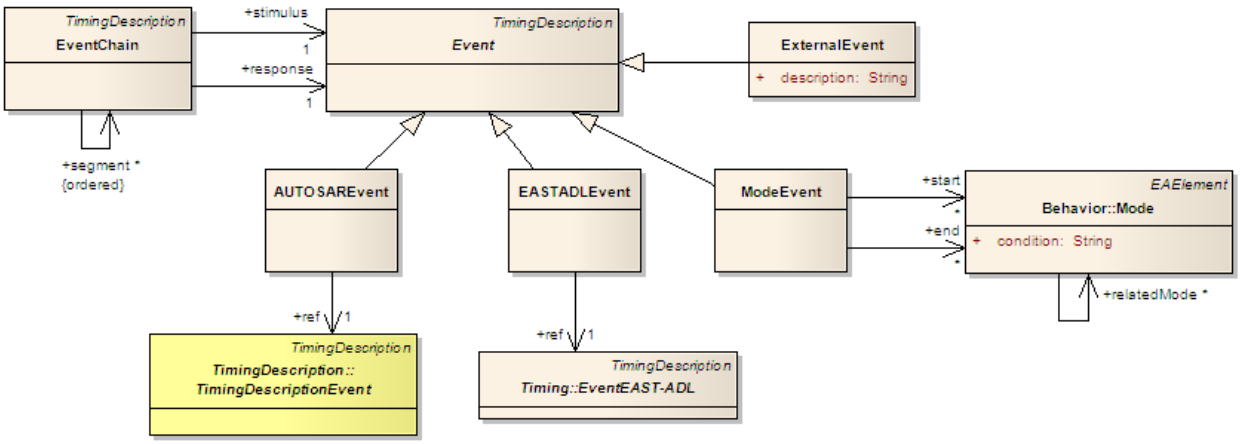

Figure 34: The Events are defined within AUTOSAR and EAST-ADL. These events refer to the structural models of AUTOSAR and EAST-ADL respectively.

11.1.2 Element Descriptions

11.1.2.1 AUTOSAREvent (from TADL2)

Generalizations

- Event (from TADL2)

Description

An AUTOSAREvent instance refers to an event of the form defined by AUTOSAR.

Attributes

No additional attributes

Associations

- ref : TimingDescriptionEvent [1]

Constraints

No additional constraints

11.1.2.2 EASTADLEvent (from TADL2)

Generalizations

- Event (from TADL2)

Description

An EASTADLEvent instance refers to an event of the form defined by EAST-ADL.

Attributes

No additional attributes

Associations

- ref : EventEAST-ADL [1]

Constraints

No additional constraints

11.1.2.3 Event (from TADL2) {abstract}

Generalizations

- TimingDescription (from TADL2)

Description

The Event class stands for all the forms of identifiable state changes that are possible to constrain with respect to timing using TADL2.

Attributes

No additional attributes

Associations

No additional associations

Constraints

No additional constraints

Semantics

An event denotes a distinct form of state change in a running system, taking place at distinct points in time called occurrence of the event. That is, a running system can be observed by identifying certain forms of state changes to watch for, and for each such observation point, noting the times when changes occur. This notion of observation also applies to a hypothetical predicted run of a system or a system model - from a timing perspective, the only information that needs to be in the output of such a prediction is a sequence of times for each observation point, indicating the times that each event is predicted to occur.

In system models, events appear syntactically as names indicating the state changes of interest. Semantically, an event name is a variable standing for some statically unknown set of occurrences. Note that this connection is purely conceptual; occurrences never exist concretely in any system model as they are a purely semantic notion representing the state changes that can be observed when a system is executed, or simulated, or perhaps only mathematically predicted.

TADL2 assumes that occurrences are characterized by two pieces of information: a timestamp indicating when the corresponding state change occurred, and a color that partitions different event occurrences into groups that should be understood as being causally related. The timestamp is a real value of SI unit seconds, whereas the color value is drawn from some abstract, possibly infinite type whose only restriction is that must support an equality test on its values.

11.1.2.4 EventChain (from TADL2)

Generalizations

- TimingDescription (from TADL2)

Description

An EventChain is a container for a pair of events that must be causally related.

Attributes

No additional attributes

Associations

- stimulus : Event [1]

  The event that stimulates the steps to be taken to respond to this event.

- response : Event [1]

  The event that is a response to a stimulus that occurred before.

- segment : EventChain [*] {ordered}

  Referred EventChains in sequence refine this EventChain.

Constraints

No additional constraints

Semantics

A system behavior is consistent with respect to an event chain ec if and only if

for each occurrence x in ec.stimulus,

for each occurrence y in ec.response,

if x.color = y.color then x < y

11.1.2.5 ExternalEvent (from TADL2)

Generalizations

- Event (from TADL2)

Description

An ExternalEvent instance stands for some particular form of state change.

It is implied that the attribute description uniquely identifies the intended form of state change. It is also assumed that a description string is sufficiently informative to determine an unambiguous set of occurrences for each observation.

Attributes

- description : String [1]

Associations

No additional associations

Constraints

No additional constraints

11.1.2.6 ModeEvent (from TADL2)

Generalizations

- Event (from TADL2)

Description

A mode that identifies when the mode starts or ends.

Attributes

No additional attributes

Associations

- start : Mode [*]

  The mode that is started.

- end : Mode [*]

  The mode that ends.

Constraints

No additional constraints

11.1.2.7 Timing (from TADL2)

Generalizations

- Context (from Elements)

Description

The collection of timing descriptions, namely events and event chains, and the timing constraints imposed on these events and event chains. This collection can be done across the EAST-ADL abstraction levels.

Attributes

No additional attributes

Associations

- constraint : TimingConstraint [*]
- description : TimingDescription [*]
- timingExpression : TimingExpression [*]
- timeBase : TimeBase [*]
- dimension : Dimension [*]
- timeBaseRelation : TimeBaseRelation [*]

Constraints

No additional constraints

11.1.2.8 TimingConstraint (from TADL2) {abstract}

Generalizations

- EAElement (from Elements)

Description

This abstract element references a mode in order to indicate that the corresponding TimingConstraint is only valid when the specified mode is active.

Attributes

No additional attributes

Associations

- mode : Mode [0..1]

Reference to the mode in which the timing constraint is valid.

Constraints

No additional constraints

Semantics

The TimingConstraint does not describe what is classically referred to as a "design" constraint but has the role of a property, requirement, or a validation result. It is a requirement if this TimingConstraint refines a Requirement (by the Refine relationship). The TimingConstraint is a validation result if it realizes a VVActualOutcome, it is an intended validation result if it realizes a VVIntendedOutcome, and in other cases it denotes a property.

11.1.2.9 TimingDescription (from TADL2) {abstract}

Generalizations

None

Description

An abstract metaclass describing the timing events and their relations by event chains within the timing model.

Attributes

No additional attributes

Associations

No additional associations

Constraints

No additional constraints

11.2 TimingConstraints

11.2.1 Overview

TADL2 offers a palette of means to constrain the time occurrences of events. These can roughly be grouped into restrictions on the recurring delays between a pair of events, restrictions on the repetitions of a single event, and restrictions on the synchronicity of a set of events. All constraints provided by TADL2 are defined in this package.

The semantics of some timing constraint is described by references to other timing constraints in this package. Default attribute values, which apply in a right-to-left manner whenever a constraint argument list is too short to match all defined attributes, are given when applicable.

A helper constraint RepeatConstraint is defined in TADL2, in modeling a RepetitionConstraint with jitter = 0 is used instead.

A system behavior satisfies a RepeatConstraint c if and only if

for each subsequence X of c.event,

if X contains span + 1 occurrences then

e is the distance between the outermost

occurrences in X

and

c.lower <= e <= c.upper

The RepeatConstraint 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 span is still 1 but lower is strictly less than upper, the pattern may deviate from a periodic one in an accumulating fashion, making the window within which occurrence number N may appear as wide as N(upper-lower) time units. A span attribute greater than 1 similarly constrains every sequence of span+1 occurrences, but places no restriction on the distances within shorter sequences.

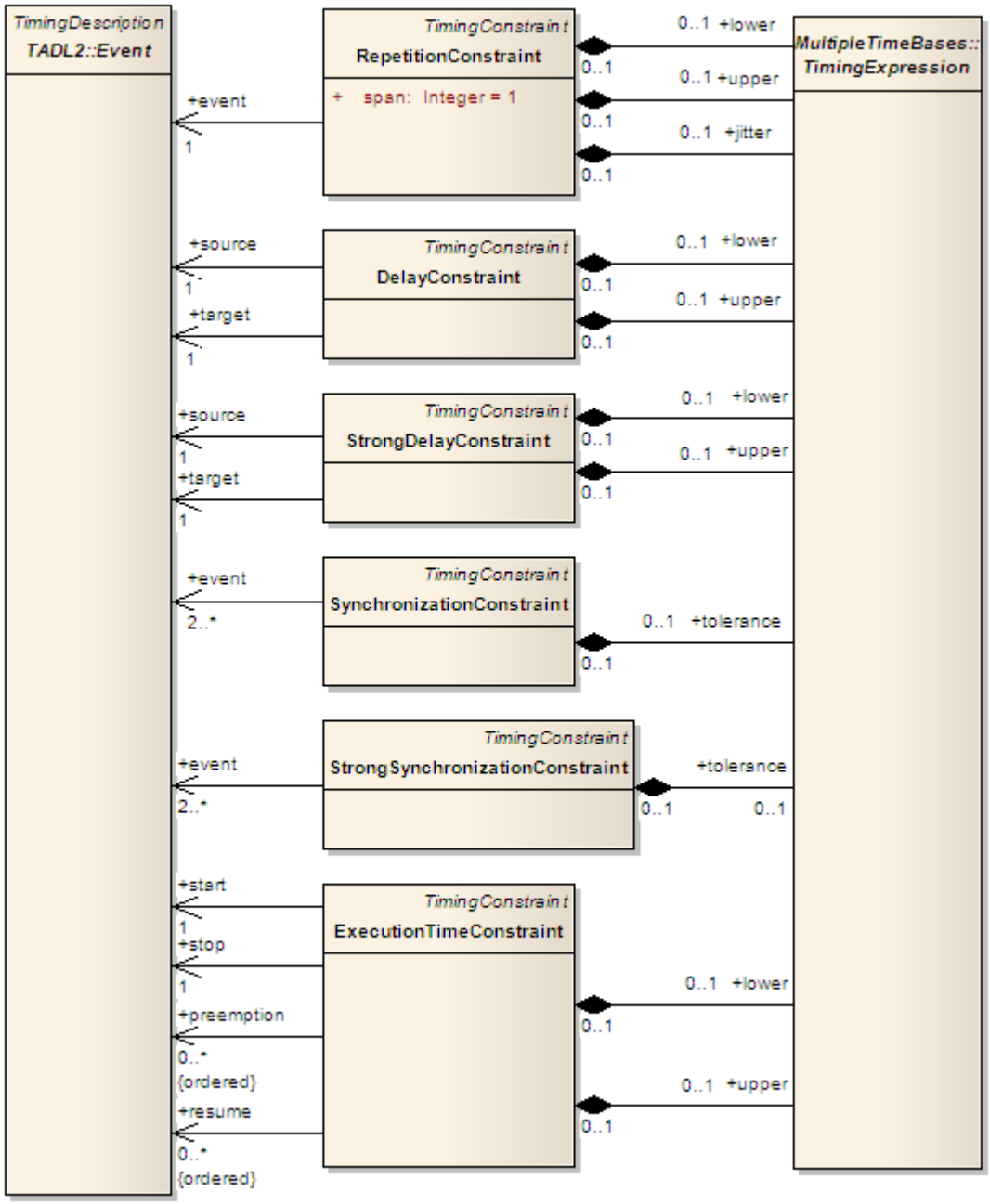

Figure 35: The first of two sets with TADL2 constraints with attributes of type TimingExpression and references to events.

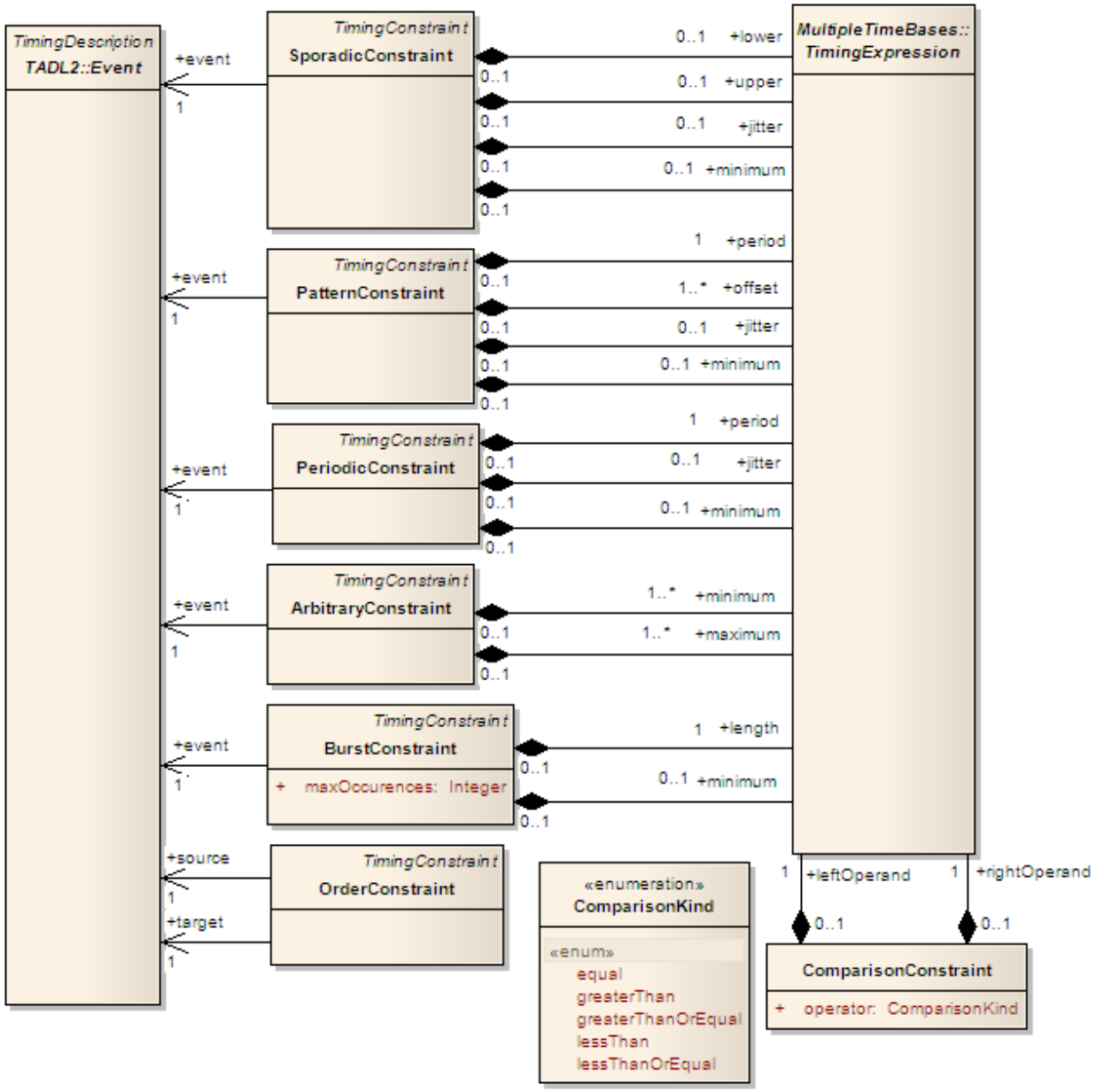

Figure 36: The second of two sets with TADL2 constraints with attributes of type TimingExpression and references to events. Also shown is the ComparisonConstraint with attributes of type TimingExpression.

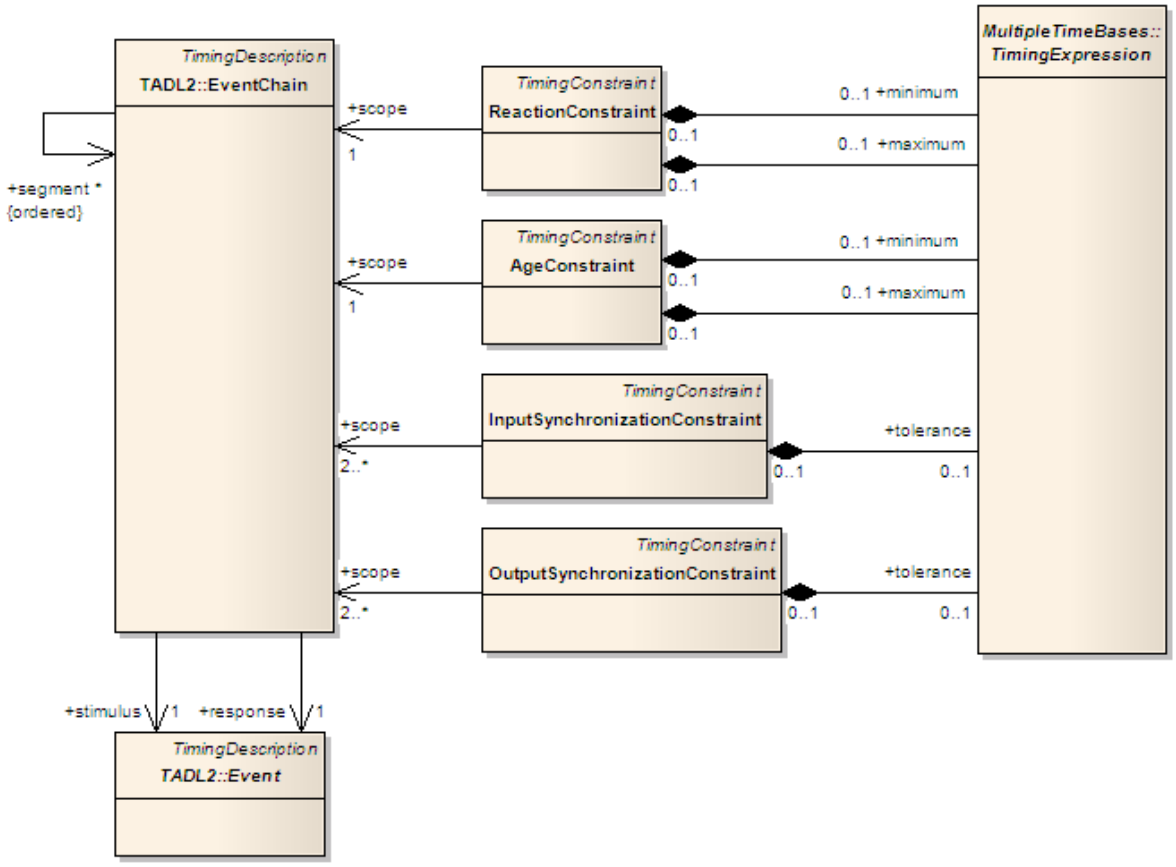

Figure 37: The TADL2 constraints that refer to EventChain, and have attributes of type TimingExpression.

11.2.2 Element Descriptions

11.2.2.1 AgeConstraint (from TimingConstraints)

Generalizations

- TimingConstraint (from TADL2)

Description

An AgeConstraint defines how long before each response a corresponding stimulus must have occurred.

This constraint provides an alternative to the ordinary DelayConstraint for situations where the causal relation between event occurrences must be taken into account. It differs from the DelayConstraint in that it applies to an event chain, and only looks at the stimulus occurrences that have the same color as each particular response occurrence. It is the latest of these stimulus occurrences that is required to lie within the prescribed time bounds. If the roles of stimulus and response are swapped, and the time bounds negated, a ReactionConstraint is obtained.

Attributes

No additional attributes

Associations

- scope : EventChain [1]
- maximum : TimingExpression [0..1]

  Default: infinity
- minimum : TimingExpression [0..1]

  Default: 0

Constraints

No additional constraints

Semantics

A system behavior satisfies an AgeConstraint c if and only if

for each occurrence y in c.scope.response,

there is an occurrence x in c.scope.stimulus such that

x.color = y.color

and

x is maximal in c.scope.stimulus with that color

and

c.minimum <= y - x <= c.maximum

11.2.2.2 ArbitraryConstraint (from TimingConstraints)

Generalizations

- TimingConstraint (from TADL2)

Description

An ArbitraryConstraint describes an event that occurs irregularly.

An ArbitraryConstraint is equivalent to a combination of Repeat constraints, each one constraining sequences of i+1 occurrences (that is, i repetition spans), with i ranging from 1 to some given n.

Attributes

No additional attributes

Associations

- event : Event [1]
- maximum : TimingExpression [1..*]
- minimum : TimingExpression [1..*]

Constraints

[1] The number of elements in minimum and maximum must be equal.

Semantics

A system behavior satisfies an AribtraryConstraint c if and only if

for each c.minimum index i, the same system behavior satisfies

RepeatConstraint { event = c.event,

lower = c.minimum(i),

upper = c.maximum(i),

span = i }

11.2.2.3 BurstConstraint (from TimingConstraints)

Generalizations

- TimingConstraint (from TADL2)

Description

A BurstConstraint describes an event that occurs in semi-regular bursts.

A BurstConstraint expresses the maximum number of event occurrences that may appear in any interval of a given length, which is equivalent to constraining the same number of repeat spans (which count one extra occurrence at the end) to have a minimum width of length.

Attributes

- maxOccurences : Integer [1]

Associations

- event : Event [1]
- length : TimingExpression [1]
- minimum : TimingExpression [0..1]

  Default: 0

Constraints

No additional constraints

Semantics

A system behavior satisfies a BurstConstraint c if and only if

the same system behavior concurrently satisfies

RepeatConstraint { event = c.event,

lower = c.length,

upper = infinity,

span = c.maxOccurrences }

and

RepeatConstraint { event = c.event,

lower = c.minimum }

11.2.2.4 ComparisonConstraint (from TimingConstraints)

Generalizations

None

Description

A ComparisonConstraint states that a certain ordering relation must exist between two timing expressions.

This constraint is special in that it does not reference any events. Its main purpose is to express relations between arithmetic variables used in other

constraint; for example, stating that the sum of the variables denoting segment delays in a time-budgeting scenario must be less than the maximum end-to-end deadline allowed.

Attributes

- operator : ComparisonKind [1]

Associations

- rightOperand : TimingExpression [1]
- leftOperand : TimingExpression [1]

Constraints

No additional constraints

Semantics

A system behavior satisfies a ComparisonConstraint c if and only if

c.leftOperand and c.rightOperand are related according to the ordering relation given by c.operator.

11.2.2.5 ComparisonKind (from TimingConstraints) «enumeration»

Generalizations

None

Enumeration Literals

- equal
- greaterThan
- greaterThanOrEqual
- lessThan
- lessThanOrEqual

Associations

No additional associations

Constraints

No additional constraints

11.2.2.6 DelayConstraint (from TimingConstraints)

Generalizations

- TimingConstraint (from TADL2)

Description

A DelayConstraint imposes limits between the occurrences of an event called source and an event called target.

This notion of delay is entirely based on the distance between source and target occurrences; whether a matching target occurrence is actually caused by the corresponding source occurrence is of no importance. This means that one-to-many and many-to-one source-target patterns are allowed, and so are stray target occurrences that are not within the prescribed distance of any source occurrence.

Attributes

No additional attributes

Associations

- target : Event [1]
- source : Event [1]
- lower : TimingExpression [0..1]

  Default: 0
- upper : TimingExpression [0..1]

  Default: infinity

Constraints

No additional constraints

Semantics

A system behavior satisfies a DelayConstraint c if and only if

for each occurrence x of c.source,

there is an occurrence y of c.target such that

c.lower <= y - x <= c.upper

11.2.2.7 ExecutionTimeConstraint (from TimingConstraints)

Generalizations

- TimingConstraint (from TADL2)

Description

An ExecutionTimeConstraint limits the time between the starting and stopping of an executable entity (function), not counting the intervals when the execution of such an executable entity (function) has been interrupted.

Attributes

No additional attributes

Associations

- preemption : Event [0..*] {ordered}
- stop : Event [1]
- start : Event [1]
- resume : Event [0..*] {ordered}
- upper : TimingExpression [0..1]
- lower : TimingExpression [0..1]

Constraints

No additional constraints

Semantics

A system behavior satisfies an ExecutionTimeConstraint c if and only if

for each occurrence x of event c.start,

E is the set of times between x and the next c.stop

occurrence, excluding the times between any c.preempt occurrence and its next c.resume occurrence,

and

c.lower <= length of all continuous intervals in E <= c.upper

11.2.2.8 InputSynchronizationConstraint (from TimingConstraints)

Generalizations

- TimingConstraint (from TADL2)

Description

An InputSynchronizationConstraint defines how far apart the responses that belong to a certain stimulus may occur.

This constraint provides an alternative to the ordinary SynchronizationConstraint for situations where the causal relation between event occurrences must be taken into account. It differs from the SynchronizationConstraint in that it applies to a set of event chains, and only looks at the stimulus occurrences that have the same color as each particular response occurrence. It is the latest of these stimulus occurrences for each chain that are required to lie no more than tolerance time units apart. If the roles of stimuli and responses are swapped, an OutputSynchronizationConstraint is obtained.

Attributes

No additional attributes

Associations

- scope : EventChain [2..*]
- tolerance : TimingExpression [0..1]

  Default: infinity

Constraints

[1] All scopes must reference one common response event.

Semantics

A system behavior satisfies an InputSynchronizationConstraint c if and only if

for each occurrence y in c.scope(1).response,

there is a time t such that for each c.scope index i,

there is an occurrence x in c.scope(i).stimulus such that

y.color = x.color

and

x is maximal in c.scope(i).stimulus with that color

and

0 <= x - t <= c.tolerance

11.2.2.9 OrderConstraint (from TimingConstraints)

Generalizations

- TimingConstraint (from TADL2)

Description

An OrderConstraint imposes an order between the occurrences of an event called source and an event called target.

The OrderConstraint is a minor variant of an application of StrongDelayConstraint with lower set to 0 and upper to infinity; the difference being that the OrderConstraint does not allow matching target and source occurrences to coincide.

Attributes

No additional attributes

Associations

- source : Event [1]
- target : Event [1]

Constraints

No additional constraints

Semantics

A system behavior satisfies an OrderConstraint c if and only if

c.source and c.target have the same number of occurrences,

and for each index i,

if there is an i:th occurrence of c.source at time x, there is

also an i:th occurrence of c.target at time y such that

x < y

11.2.2.10 OutputSynchronizationConstraint (from TimingConstraints)

Generalizations

- TimingConstraint (from TADL2)

Description

An OutputSynchronizationConstraint defines how far apart the responses that belong to a certain stimulus may occur.

This constraint provides an alternative to the ordinary SynchronizationConstraint for situations where the causal relation between event occurrences must be taken into account. It differs from the SynchronizationConstraint in that it applies to a set of event chains, and only looks at the response occurrences that have the same color as each particular stimulus occurrence. It is the earliest of these response occurrences for each chain that are required to lie no more than tolerance time units apart. If the roles of stimuli and responses are swapped, an InputSynchronizationConstraint is obtained.

Attributes

No additional attributes

Associations

- scope : EventChain [2..*]
- tolerance : TimingExpression [0..1]

Default: infinity

Constraints

[1] All scopes must reference one common stimulus event.

Semantics

A system behavior satisfies an OutputSynchronizationConstraint c if and only if

for each occurrence x in c.scope(1).stimulus,

there is a time t such that for each c.scope index i,

there is an occurrence y in c.scope(i).response such that

y.color = x.color

and

y is minimal in c.scope(i).response with that color

and

0 <= y - t <= c.tolerance

11.2.2.11 PatternConstraint (from TimingConstraints)

Generalizations

- TimingConstraint (from TADL2)

Description

A PatternConstraint describes an event that exhibits a known pattern relative to the occurrences of an imaginary event.

A PatternConstraint requires the constrained event occurrences to appear at a predetermined series of offsets from a sequence of reference points in time that are strictly periodic. The exact placement of these reference points is irrelevant; if one placement exists that is periodic and allows the event occurrences to be reached at the desired offsets, the constraint is satisfied.

Attributes

No additional attributes

Associations

- event : Event [1]
- jitter : TimingExpression [0..1]

  Default: 0
- minimum : TimingExpression [0..1]

  Default: 0
- offset : TimingExpression [1..*]
- period : TimingExpression [1]

Constraints

No additional constraints

Semantics

A system behavior satisfies a PatternConstraint c if and only if

there is a set of times X such that the same system behavior concurrently satisfies

PeriodicConstraint { event = X,

period = c.period }

and for each c.offset index i,

DelayConstraint { source = X,

target = c.event,

lower = c.offset(i),

upper = c.offset(i) + c.jitter }

and

RepeatConstraint { event = c.event,

lower = c.minimum }

11.2.2.12 PeriodicConstraint (from TimingConstraints)

Generalizations

- TimingConstraint (from TADL2)

Description

A PeriodicConstraint describes an event that occurs periodically.

Attributes

No additional attributes

Associations

- event : Event [1]
- minimum : TimingExpression [0..1]

  Default: 0
- period : TimingExpression [1]
- jitter : TimingExpression [0..1]

  Default: 0

Constraints

No additional constraints

Semantics

A system behavior satisfies a PeriodicConstraint c if and only if

the same system behavior satisfies

SporadicConstraint { event = c.event,

lower = c.period,

upper = c.period,

jitter = c.jitter,

minimum = c.minimum }

11.2.2.13 ReactionConstraint (from TimingConstraints)

Generalizations

- TimingConstraint (from TADL2)

Description

A ReactionConstraint defines how long after the occurrence of a stimulus a corresponding response must occur.

This constraint provides an alternative to the ordinary DelayConstraint for situations where the causal relation between event occurrences must be taken into account. It differs from the DelayConstraint in that it applies to an event chain, and only looks at the response occurrences that have the same color as each particular stimulus occurrence. It is the earliest of these response occurrences that is required to lie within the prescribed time bounds. If the roles of stimulus and response are swapped, and the time bounds negated, an AgeConstraint is obtained.

Attributes

No additional attributes

Associations

- scope : EventChain [1]
- maximum : TimingExpression [0..1]

  Default: infinity
- minimum : TimingExpression [0..1]

  Default: 0

Constraints

No additional constraints

Semantics

A system behavior satisfies a ReactionConstraint c if and only if

for each occurrence x in c.scope.stimulus,

there is an occurrence y in c.scope.response such that

y.color = x.color

and

y is minimal in c.scope.response with that color

and

c.minimum <= y - x <= c.maximum

11.2.2.14 RepetitionConstraint (from TimingConstraints)

Generalizations

- TimingConstraint (from TADL2)

Description

A RepetitionConstraint describes the distribution of the occurrences of a single event, including the allowance for jitter.

The RepetitionConstraint extends the basic notion of repeated occurrences by allowing local devitions from the ideal repetitive pattern described by a RepeatConstraint. Its jitter, lower and upper attributes all contribute to the

width of the window in which occurrence number N is accepted, according to the formula N(upper-lower) + jitter. That is, with lower = upper, the uncertainty of where occurrence N may be found does not grow with an increasing N, unlike the case when lower differs from upper by a similar amount and jitter is 0. By adjusting all three attributes, a desired balance between accumulating and non-accumulating uncertainties can be obtained.

Attributes

- span : Integer = 1 [1]

Associations

- event : Event [1]
- jitter : TimingExpression [0..1]

  Default: 0
- lower : TimingExpression [0..1]

  Default: 0
- upper : TimingExpression [0..1]

  Default: infinity

Constraints

No additional constraints

Semantics

A system behavior satisfies a RepetitionConstraint c if and only if

the same system behavior concurrently satisfies

RepeatConstraint { event = X,

lower = c.lower,

upper = c.upper,

span = c.span }

and

StrongDelayConstraint { source = X,

target = c.event,

lower = 0,

upper = c.jitter }

11.2.2.15 SporadicConstraint (from TimingConstraints)

Generalizations

- TimingConstraint (from TADL2)

Description

A SporadicConstraint describes an event that occurs sporadically.

The SporadicConstraint is just an application of the RepetitionConstraint with a default span attribute of 1, combined with an additional requirement that the effective minimum distance between any two occurrences must be at least the value given by minimum (even if lower-jitter would suggest a smaller value).

Attributes

No additional attributes

Associations

- event : Event [1]
- lower : TimingExpression [0..1]

  Default: 0
- minimum : TimingExpression [0..1]

  Default: 0
- upper : TimingExpression [0..1]

  Default: infinity
- jitter : TimingExpression [0..1]

Default: 0

Constraints

No additional constraints

Semantics

A system behavior satisfies a SporadicConstraint c if and only if

the same system behavior concurrently satisfies

RepetitionConstraint { event = c.event,

lower = c.lower,

upper = c.upper,

jitter = c.jitter }

and

RepeatConstraint { event = c.event,

lower = c.minimum }

11.2.2.16 StrongDelayConstraint (from TimingConstraints)

Generalizations

- TimingConstraint (from TADL2)

Description

A StrongDelayConstraint imposes limits between each indexed occurrence of an event called source and the identically indexed occurrence of an event called target.

The strong delay notion requires source and target occurrences to appear in lock-step. Only one-to-one source-target patterns are allowed, and no stray target occurrences are accepted.

Strong synchronization differs from the ordinary form of SynchronizationConstraint by grouping event occurrences into synchronization clusters strictly according to their index. This means that multiple occurrences of a single event cannot belong to a single cluster, and clusters may not share occurrences. Strong synchronization tightens the

requirements compared to ordinary synchronization in much the same way as StrongDelayConstraint refines the ordinary DelayConstraint.

Attributes

No additional attributes

Associations

- source : Event [1]
- target : Event [1]
- lower : TimingExpression [0..1]

  Default: 0
- upper : TimingExpression [0..1]

  Default: infinity

Constraints

No additional constraints

Semantics

A system behavior satisfies a StrongDelayConstraint c if and only if

c.source and c.target have the same number of occurrences,

and for each index i,

if there is an i:th occurrence of c.source at time x

there is also an i:th occurrence of c.target at time y

such that

c.lower <= y - x <= c.upper

11.2.2.17 StrongSynchronizationConstraint (from TimingConstraints)

Generalizations

- TimingConstraint (from TADL2)

Description

A StrongSynchronizationConstraint describes how tightly the occurrences of a group of events follow each other.

Attributes

No additional attributes

Associations

- event : Event [2..*]
- tolerance : TimingExpression [0..1]

Default: infinity

Constraints

No additional constraints

Semantics

A system behavior satisfies a StrongSynchronizationConstraint c if and only if

there is a set of times X such that for each c.event index i, the same system behavior satisfies

StrongDelayConstraint { source = X,

target = c.event(i),

lower = 0,

upper = c.tolerance }

11.2.2.18 SynchronizationConstraint (from TimingConstraints)

Generalizations

- TimingConstraint (from TADL2)

Description

A SynchronizationConstraint describes how tightly the occurrences of a group of events follow each other.

This form of synchronization only takes the width and completeness of each occurrence cluster into account; it does not care whether som events occur multiple times within a cluster or whether some clusters overlap and share occurrences. In particular, event occurrences are not partitioned into clusters according to their role or what has caused them. Stray occurrences of single events are not allowed, though, since these would just count as incomplete clusters according to this constraint.

Attributes

No additional attributes

Associations

- event : Event [2..*]
- tolerance : TimingExpression [0..1]

Default: infinity

Constraints

No additional constraints

Semantics

A system behavior satisfies a SynchronizationConstraint c if and only if

there is a set of times X such that for each c.event index i, the same system behavior concurrently satisfies

DelayConstraint { source = X,

target = c.event(i),

lower = 0,

upper = c.tolerance }

and

DelayConstraint { source = c.event(i),

target = X,

lower = -c.tolerance,

upper = 0}

11.3 MultipleTimeBases

11.3.1 Overview

This work on symbolic time expression is twofold: firstly, it concerns the concepts of TADL2 to manage in a same design, time bases of multiple types (universal time - i.e. chronometric time, angular time, etc.). The second aspect concerns with the extension of constant time expressions and the possibility to define time as an algebraic expression that is able to manipulate symbolic identifiers. So, a value expression in a TADL2 time constraint may refer to an expression made of a suitable set of arithmetic operators mixing symbolic identifiers and referring to different time bases.

A typical use for this feature is to capture unknown configuration parameters; another one is to relate constraints in different time-bases to each other.

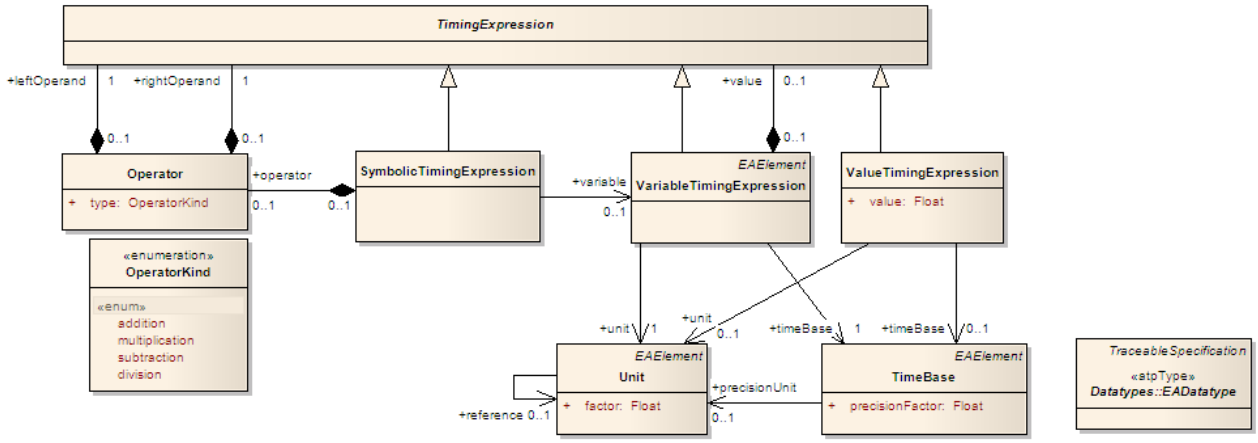

Figure 38: TimingExpression.

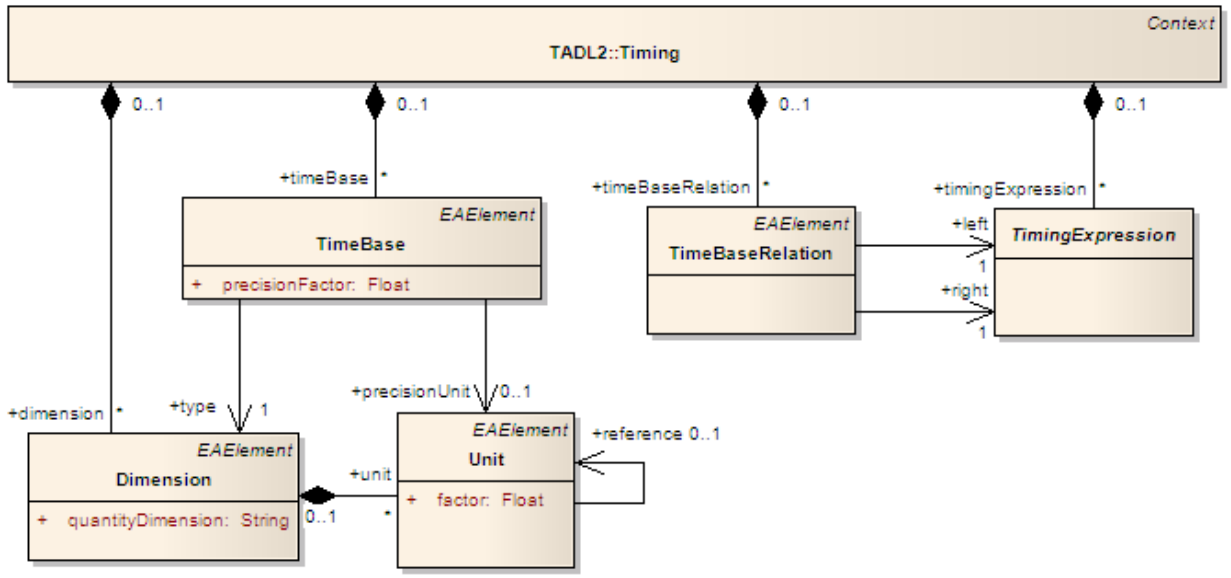

Figure 39: TimeBase with Dimension and Unit.

11.3.2 Element Descriptions

11.3.2.1 Dimension (from MultipleTimeBases)

Generalizations

- EAElement (from Elements)

Description

A Dimension defines a set of units of the same quantity dimension.

Some examples of Dimension are:

name = "Length" and quantityDimension = "L"

name = "Angle" and quantityDimension = "", i.e. the empty string as angle is without dimension.

name = "Acceleration" and quantityDimension = "LT-2", the exponent for length is 1 and for time it is -2.

Valid symbols in the quantityDimension attribute and their quantity names are:

L, length

M, mass

T, time

I, electric current

K, thermodynamic temperature

J, luminous intensity

N, amount of substance

Attributes

- quantityDimension : String [1]

Associations

- unit : Unit [*]

Constraints

No additional constraints

11.3.2.2 Operator (from MultipleTimeBases)

Generalizations

None

Description

An Operator used between two TimingExpressions in a SymbolicTimingExpression.

Attributes

- type : OperatorKind [1]

Associations

- leftOperand : TimingExpression [1]
- rightOperand : TimingExpression [1]

Constraints

No additional constraints

11.3.2.3 OperatorKind (from MultipleTimeBases) «enumeration»

Generalizations

None

Description

An enumeration of operators.

Enumeration Literals

- addition
- division
- multiplication
- subtraction

Associations

No additional associations

Constraints

No additional constraints

11.3.2.4 SymbolicTimingExpression (from MultipleTimeBases)

Generalizations

- TimingExpression (from MultipleTimeBases)

Description

In SymbolicTimingExpression, the language integrates basic arithmetic operators such as addition, subtraction, and multiplication associated with timing values.

Attributes

No additional attributes

Associations

- variable : VariableTimingExpression [0..1]
- operator : Operator [0..1]

Constraints

[1] SymbolicTimingExpression cannot have both an Operator and a reference to VariableTimingExpression.

11.3.2.5 TimeBase (from MultipleTimeBases)

Generalizations

- EAElement (from Elements)

Description

TimeBase has been introduced to cope with the need of modeling various temporal referential used in an automotive distributed systems design (clocks from different ECUs, motor position, etc.).

TADL2 timing expressions may contain an explicit TimeBase which 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 a system. Events may refer even to "classical" time dimension or to some evolution of a mechanical part like the rotation of crankshaft, distance, etc.

Attributes

- precisionFactor : Float [1]

Because a TimeBase is a discrete set of instants, a discretization step is specified with the precisionFactor attribute which rely on a precisionUnit.

Associations

- type : Dimension [1]
- precisionUnit : Unit [0..1]

Constraints

[1] Every TimeBase declaration must introduce a unique timebase identifier.

[2] A TimeBase declaration with the name universal must exist.

11.3.2.6 TimeBaseRelation (from MultipleTimeBases)

Generalizations

- EAElement (from Elements)

Description

Expressing relation between time bases is mandatory to build a global perception of time. When timing constraints refer to multiple time bases, it results in a partially ordered set of instants from these time bases and corresponds to the global temporal perception of system behavior.

Attributes

No additional attributes

Associations

- right : TimingExpression [1]
- left : TimingExpression [1]

Constraints

No additional constraints

11.3.2.7 TimingExpression (from MultipleTimeBases) {abstract}

Generalizations

None

Description

A Timing Expression, denoted by texp, is a term built from an arithmetic expression by applying an optional unit and referencing an optional time base. It stands for a value in the real number system extended with positive and negative infinity.

Grammar:

texp ::= aexp

| aexp UN

| aexp on TB

| aexp UN on TB

Attributes

No additional attributes

Associations

No additional associations

Constraints

No additional constraints

Semantics

Given a particular variable assignment, the meaning of a timing expression texp in that assignment is a value in the real number system extended with positive and negative infinity. Depending on the form of texp, this value is defined as follows:

- If texp is of the form aexp, its meaning is the meaning of aexp in the given variable assignment.

- If texp is of the form aexp UN, its meaning is r * k, where r is the meaning of aexp in the given variable assignment, and k is the factor of UN in the Universal time base.

- If texp is of the form aexp on TB, its meaning is f (r), where f is the meaning of TB in the given variable assignment, and r is the meaning of aexp in the same assignment.

- If texp is of the form aexp UN on TB, its meaning is f (r * k), where f is the meaning of TB in the given variable assignment, r is the meaning of aexp in the same assignment, k is the factor of UN in DI, and DI is the dimension of TB.

11.3.2.8 Unit (from MultipleTimeBases)

Generalizations

- EAElement (from Elements)

Description

Each Unit relates to another unit by the factor attribute to enable conversions.

As a unit conversion example:

second = 1000 * millisecond

has factor = 1000.

Attributes

- factor : Float [1]

Associations

- reference : Unit [0..1]

Constraints

No additional constraints

11.3.2.9 ValueTimingExpression (from MultipleTimeBases)

Generalizations

- TimingExpression (from MultipleTimeBases)

Description

A ValueTimingExpression may have a unit and a time base as type. TADL2 is aimed to be a declarative language. Therefore, we have only free variables, constants and values. Please note that ValueTimingExpression does not have a name.

Attributes

- value : Float [1]

Associations

- timeBase : TimeBase [0..1]
- unit : Unit [0..1]

Constraints

No additional constraints

11.3.2.10 VariableTimingExpression (from MultipleTimeBases)

Generalizations

- EAElement (from Elements)
- TimingExpression (from MultipleTimeBases)

Description

The VariableTimingExpression stands for free variables and constants. If a value is assigned to a variable, then the variable becomes a constant.

Attributes

No additional attributes

Associations

- timeBase : TimeBase [1]
- unit : Unit [1]
- value : TimingExpression [0..1]

Constraints

No additional constraints

11.4 ProbabilisticTiming

11.4.1 Overview

This section presents the extension of the basic timing constraints of TADL2 with probabilistic parameters, which can be either based on distributions or follow the weakly-hard approach, which was introduced to express that not more than a given number of deadlines may be missed within a time window. The goal of these additional parameters is to allow the expression of more fine-grain information than the usual interval between the best case and the worst case.

Probabilistic timing information can be used in different ways to represent different abstractions. For example, one may be interested in probabilistic information based on distributions or instead in weakly-hard constraints which express that not more than a given number of deadlines may be missed within a time window.

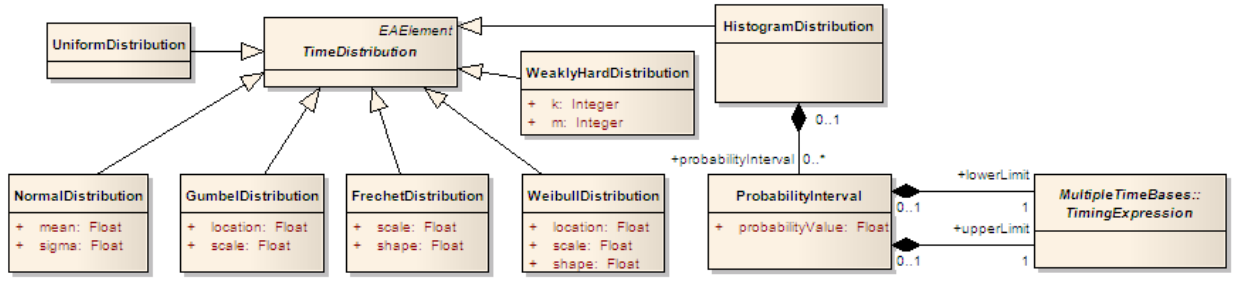

Figure 40: Distributions for probabilistic timing.

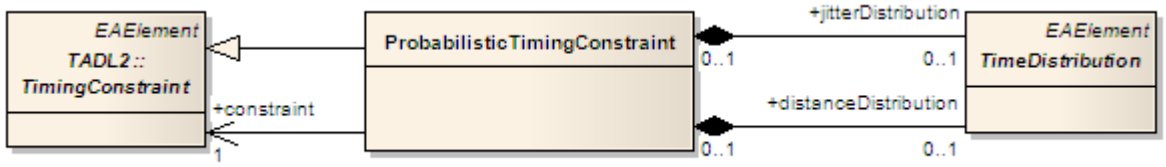

Figure 41: Constraints for probabilistic timing modifies timing constraints and add a distribution.

11.4.2 Element Descriptions

11.4.2.1 FrechetDistribution (from ProbabilisticTiming)

Generalizations

- TimeDistribution (from ProbabilisticTiming)

Description

A FrechetDistribution, defined only for positive values. To evaluate the probability density from this distribution the argument shall be in the universal timebase, to correspond to the float parameters of this distribution.

Attributes

- scale : Float [1]
- shape : Float [1]

Associations

No additional associations

Constraints

No additional constraints

11.4.2.2 GumbelDistribution (from ProbabilisticTiming)

Generalizations

- TimeDistribution (from ProbabilisticTiming)

Description

A Gumbel distribution, defined on the entire real axis. To evaluate the probability density from this distribution the argument shall be in the universal timebase, to correspond to the float parameters of this distribution.

Attributes

- location : Float [1]
- scale : Float [1]

Associations

No additional associations

Constraints

No additional constraints

11.4.2.3 HistogramDistribution (from ProbabilisticTiming)

Generalizations

- TimeDistribution (from ProbabilisticTiming)

Description

The HistogramDistribution is The probability interval describes the probability for each interval if the distribution is discretized.

Attributes

No additional attributes

Associations

- probabilityInterval : ProbabilityInterval [0..*]

Constraints

No additional constraints

Semantics

Consider a given list of probaInterval of the form {pr[t_0;t_1]=p_1, ... , pr[t_n-1;t_n]=p_n} where t_0=lower, t_n=upper and t_0 <= t_1 <= ... <= t_n.

If the sum of all probability values P=p_1+...+p_n is equal to 1, as must be the case by definition of a probability distribution, then our distribution is such that the probability to obtain a value in the interval [t_i-1, t_i] (for i in [1, n]) is equal to p_i.

If the sum of all probability values P=p_1+...+p_n is larger than 1, then we consider the distribution to be a overapproximation of the exact distribution, that is, the probability to obtain a value in the interval [t_i-1, t_i] (for i in [1, n]) is smaller than or equal to p_i. We proceed similarly if P is smaller than 1.

11.4.2.4 NormalDistribution (from ProbabilisticTiming)

Generalizations

- TimeDistribution (from ProbabilisticTiming)

Description

A Normal (Gaussian) distribution.

Attributes

- mean : Float [1]

  The mean value.

- sigma : Float [1]

  The standard deviation.

Associations

No additional associations

Constraints

No additional constraints

11.4.2.5 ProbabilisticTimingConstraint (from ProbabilisticTiming)

Generalizations

- TimingConstraint (from TADL2)

Description

A timing constraints that modifies the associated constraints and adds a probabilistic distribution. Valid for constraints that has a jitter and/or a lower and upper parameter. Separate distributions can be provided for the jitter or distance.

The semantic of the predefined distributions is as usual. For example, the semantics of a 'uniform' distribution between lower and upper is a function that associates with every interval [t1;t2] included in [lower, upper] a probaValue equal to (t2-t1) / (upper-lower). The only difference concerns situations where the predefined distributions may have values outside the bounds defined by lower and upper. In that case it is assumed that these values will be ignored and therefore to ensure that the sum of all remaining probability values P (formally defined as the definite integral of the probability

distribution between lower and upper) is equal to 1, we adapt the standard semantics as follows: the probability of any interval between lower and upper is divided by P.

It is important to note here that we will also use Time distributions to describe jitter, and in this case the bounds will be 0 and jitter instead of lower and upper. The following explanations also apply to these distributions.

Attributes

No additional attributes

Associations

- constraint : TimingConstraint [1]
- jitterDistribution : TimeDistribution [0..1]

  Distribution within the window defined by the jitter parameter of the constraint.

- distanceDistribution : TimeDistribution [0..1]

  Distribution within the window defined by the lower and upper parameter of the constraint.

Constraints

[1] constraint must have a jitter or constraint must have a lower and upper parameter.

11.4.2.6 ProbabilityInterval (from ProbabilisticTiming)

Generalizations

None

Description

The attribute probabilityValue is the probability for the interval from lowerLimit to upperLimit.

Attributes

- probabilityValue : Float [1]

Associations

- lowerLimit : TimingExpression [1]
- upperLimit : TimingExpression [1]

Constraints

No additional constraints

11.4.2.7 TimeDistribution (from ProbabilisticTiming) {abstract}

Generalizations

- EAElement (from Elements)

Description

The abstract concept of distributions in time, see the concrete specializations.

Attributes

No additional attributes

Associations

No additional associations

Constraints

No additional constraints

11.4.2.8 UniformDistribution (from ProbabilisticTiming)

Generalizations

- TimeDistribution (from ProbabilisticTiming)

Description

A Uniform distribution.

Attributes

No additional attributes

Associations

No additional associations

Constraints

No additional constraints

11.4.2.9 WeaklyHardDistribution (from ProbabilisticTiming)

Generalizations

- TimeDistribution (from ProbabilisticTiming)

Description

The timing constraints defined in TADL2 are strongly-hard in the sense that they must hold for each occurrence of some designated event. For example, the DelayConstraint requires that for each occurrence of the source event, there is at least one occurrence of the target event within a fixed interval relative to the source. It only takes one absent response occurrence to render the whole DelayConstraint violated.

In many situations, a system may in fact work correctly even if a strongly-hard constraint is not satisfied for a bounded number of occurrences. Therefore, TADL2 generalizes the concept of weakly-hard constraints (which was originally introduced for describing allowed deadline misses) to formalize scenarios in which a bounded number of occurrences are allowed to violate the constraint requirements.

Attributes

- k : Integer [1]
- m : Integer [1]

Associations

No additional associations

Constraints

No additional constraints

Semantics

The semantics of a weakly-hard expression (m, k) is that the behavior must satisfy the given constraint at least m times out of k consecutive occurrences.

11.4.2.10 WeibullDistribution (from ProbabilisticTiming)

Generalizations

- TimeDistribution (from ProbabilisticTiming)

Description

A Weibull distribution, defined only for positive values. To evaluate the probability density from this distribution the argument shall be in the universal timebase, to correspond to the float parameters of this distribution.

Attributes

- location : Float [1]
- scale : Float [1]
- shape : Float [1]

Associations

No additional associations

Constraints

No additional constraints

12 Appendix C – Relationships

Relationships between TADL2 and the AUTOSAR Timing Extension [4], and the EAST-ADL Timing package [3] are described. This is to facilitate the transformation of existing models and describe the differences between the metamodels. TADL1 has been integrated in EAST-ADL, for comparison between TADL2 and TADL1 see the comparison with EAST-ADL.

12.1 Relation to AUTOSAR 4.0.3 Timing Extension

AR.PeriodicEventTriggering
(*event*, *period*, *jitter*, *minimumInterArrivalTime*)

↔

PeriodicConstraint
(*event*, *period*, *jitter*, *minimumInterArrivalTime*)

AR.SporadicEventTriggering
(*event*, *period*, *maximumInterArrivalRate*, *jitter*, *minimumInterArrivalRate*)

↔

SporadicConstraint
(*event*, *period*, *maximumInterArrivalTime*, *jitter*, *minimumInterArrivalTime*)

AR.ConcretePatternEventTriggering
(*event*, $offset_1$, …, $offset_n$, *patternLength*)

↔

PatternConstraint
(*event*, *patternLength*, $offset_1$, …, $offset_n$, 0)

Note: The AR.ConcretePatternEventTriggering constraint lacks a *jitter* attribute, which is here by default interpreted as a tolerated jitter of 0.

AR.BurstPatternEventTriggering
(*event*, *patternLength*, *maxNumberOfOccurrences*, *minimumInterArrivalTime*)

↔

BurstConstraint
(*event*, *patternLength*, *maxNumberOfOccurrences*, *minimumInterArrivalTime*)

AR.ArbitraryEventTriggering
(*event*, *minimumDistance*$_1$, …, *minimumDistance*$_n$,
maximumDistance$_1$, …, *maximumDistance*$_n$, *confidenceInterval*)

↔

ArbitraryConstraint
(*event*, *minimumDistance*, *maximumDistance*)

Note: the AR.ArbitraryEventTriggering constraint attribute *confidenceInterval* is ignored in this translation. For further discussions on the use of probablistic distributions in TADL2, see chapter 6

AR.LatencyTimingConstraint (*scope*, *minimum*, *maximum*, **reaction**)

↔

ReactionConstraint (*scope*, *minimum*, *maximum*)

Note: the *nominal* attribute of an AR.LatencyTimingConstraint is ignored in the TADL2 translation, as it plays no role in the semantics of an AR.LatencyTimingConstraint.

AR.LatencyTimingConstraint (*scope*, *minimum*, *maximum*, **age**)

↔

AgeConstraint (*scope*, *minimum*, *maximum*)

Note: the *nominal* attribute of an AR.LatencyTimingConstraint is ignored in the TADL2 translation, as it plays no role in the semantics of an AR.LatencyTimingConstraint.

AR.SynchronizationTimingConstraint
(*scope*$_1$, …, *scope*$_n$, tolerance, **responseSynchronization**)

↔

OutputSynchronizationConstraint
(*scope*$_1$, …, *scope*$_n$, *tolerance*)

AR.SynchronizationTimingConstraint
(*scope*$_1$, …, *scope*$_n$, tolerance, **stimulusSynchronization**)

↔

InputSynchronizationConstraint
($scope_1$, …, $scope_n$, *tolerance*)

AR.OffsetTimingConstraint (*source*, *target*, *minimum*, *maximum*)

↔

DelayConstraint (*source*, *target*, *minimum*, *maximum*)

AR.ExecutionOrderConstraint
($orderedElement_1$, …, $orderedElement_n$)

↔

OrderConstraint
($orderedElement_1$.*triggerEvent*, $orderedElement_2$.*triggerEvent*)

∧ … ∧

OrderConstraint
($orderedElement_{n-1}$.*triggerEvent*, $orderedElement_n$.*triggerEvent*)

Note: which exact event that constitutes the *triggerEvent* for each *orderedElement* is dependent on the type of element entity referenced.

12.1.1 Comparison

Generally

- Time values are typed by TimingExpression in TADL2, in AUTOSAR these are typed by MultidimensionalTime.
- The TADL2 generalizes the notion of AUTOSAR Timing DescriptionEvent and TimingDescriptionEventChain; and the notion of EAST-ADL Events and EventChain by the metamodel elements Event and EventChain. The specific AUTOSAR and EAST-ADL Events are referenced via the specializations AUTOSAREvent and EASTADLEvent.

New concepts in TADL2 as alignment with AUTOSAR are:

- BurstConstraint
- OrderConstraint

TADL2	AUTOSAR Timing R4.0.3	Notes
Event (from TADL2) {abstract}	**TimingDescriptionEvent (from TimingDescription) {abstract}**	
Generalizations TimingDescription (from TADL2) **Attributes**	**Generalizations** TimingDescription (from TimingDescription) **Attributes**	

Associations

Associations
occurrenceExpression : TDEventOccurrenceExpression [0..1]

EventChain (from TADL2)

Generalizations
TimingDescription (from TADL2)
Attributes
Associations
segment : EventChain [*] {ordered}
response : Event [1]
stimulus : Event [1]

TimingDescriptionEventChain (from TimingDescription)

Generalizations
TimingDescription (from TimingDescription)
Attributes
Associations
segment : TimingDescriptionEventChain [1..*]
response : TimingDescriptionEvent [1]
stimulus : TimingDescriptionEvent [1]

TimingConstraint (from TADL2) {abstract}

Generalizations
EAElement (from Elements)

Attributes
Associations
mode : Mode [0..1]

TimingConstraint (from TimingConstraint) {abstract}

Generalizations
Identifiable (from Identifiable)
Traceable (from RequirementsTracing)
Attributes
Associations

DelayConstraint (from TimingConstraints)

Generalizations
TimingConstraint (from TADL2)
Attributes

Associations
lower : TimingExpression [0..1] = 0
upper : TimingExpression [0..1] = infinity
target : Event [1]
source : Event [1]
Inherited Associations
mode : Mode [0..1]
TADL2 DelayConstraint is based on AUTOSAR.

OffsetTimingConstraint (from OffsetConstraint)

Generalizations
TimingConstraint (from TimingConstraint)
Attributes
Associations

minimum : MultidimensionalTime [1]
maximum : MultidimensionalTime [1]
target : TimingDescriptionEvent [1]
source : TimingDescriptionEvent [1]

LatencyConstraintTypeEnum (from LatencyTimingConstraint) «enumeration»

Generalizations
Enumeration Literals
age
reaction
Associations

AgeConstraint (from TimingConstraints)

Generalizations
TimingConstraint (from TADL2)

AgeConstraint (from AgeConstraint)

Generalizations
TimingConstraint (from TimingConstraint)

Attributes
Associations
scope : EventChain [1]
minimum : TimingExpression [0..1] = 0
maximum : TimingExpression [0..1] = infinity
Inherited Associations
mode : Mode [0..1]

Attributes
Associations
scope : TDEventVariableDataPrototype [1] **2**
minimum : MultidimensionalTime [0..1] **3**
maximum : MultidimensionalTime [0..1] **3**

ReactionConstraint (from TimingConstraints)

Generalizations
TimingConstraint (from TADL2)
Attributes
Associations

scope : EventChain [1]
minimum : TimingExpression [0..1] = 0
maximum : TimingExpression [0..1] = infinity

Inherited Associations
mode : Mode [0..1]

LatencyTimingConstraint (from LatencyTimingConstraint)

Generalizations
TimingConstraint (from TimingConstraint)
Attributes
latencyConstraintType : LatencyConstraintTypeEnum [1]
Associations
scope : TimingDescriptionEventChain [1] **2**
minimum : MultidimensionalTime [1] **3**
maximum : MultidimensionalTime [1] **3**
nominal : MultidimensionalTime [1] **4**

PatternConstraint (from TimingConstraints)

Generalizations
TimingConstraint (from TADL2)

Attributes
Associations
minimum : TimingExpression [0..1] = 0
jitter : TimingExpression [0..1] = 0
offset : TimingExpression [1..*]

event : Event [1]
ref : Event [1]
Inherited Associations
mode : Mode [0..1]

ConcretePatternEventTriggering (from EventTriggeringConstraint)

Generalizations
EventTriggeringConstraint (from EventTriggeringConstraint)
Attributes
Associations
patternLength : MultidimensionalTime [1] **4**
offset : MultidimensionalTime [1..*] **2**
Inherited Associations
event : TimingDescriptionEvent [1] **2**

PeriodicConstraint (from TimingConstraints)

Generalizations
TimingConstraint (from TADL2)

Attributes
Associations
jitter : TimingExpression [0..1] = 0
period : TimingExpression [1]
minimum : TimingExpression [0..1] = 0

event : Event [1]
Inherited Associations
mode : Mode [0..1]

PeriodicEventTriggering (from EventTriggeringConstraint)

Generalizations
EventTriggeringConstraint (from EventTriggeringConstraint)
Attributes
Associations
jitter : MultidimensionalTime [1]
period : MultidimensionalTime [1]
minimumInterArrivalTime : MultidimensionalTime [1]
Inherited Associations
event : TimingDescriptionEvent [1] **2**

SporadicConstraint (from TimingConstraints)

Generalizations
TimingConstraint (from TADL2)

Attributes
Associations
jitter : TimingExpression [0..1] = 0

minimum : TimingExpression [0..1] = 0

upper : TimingExpression [0..1] = infinity
lower : TimingExpression [0..1] = 0

event : Event [1]
Inherited Associations
mode : Mode [0..1]

SporadicEventTriggering (from EventTriggeringConstraint)

Generalizations
EventTriggeringConstraint (from EventTriggeringConstraint)

Attributes
Associations
jitter : MultidimensionalTime [0..1]
period : MultidimensionalTime [0..1]
maximumInterArrivalTime : MultidimensionalTime [1]
minimumInterArrivalTime : MultidimensionalTime [1]

Inherited Associations
event : TimingDescriptionEvent [1] **2**

SynchronizationTypeEnum (from SynchronizationTimingConstraint) «enumeration»

Generalizations
Enumeration Literals
responseSynchronization
stimulusSynchronization

Associations

InputSynchronizationConstraint (from TimingConstraints)

Generalizations
TimingConstraint (from TADL2)
Attributes

Associations
tolerance : TimingExpression [0..1] = infinity
scope : Event [2..*]
Inherited Associations
mode : Mode [0..1]

SynchronizationTimingConstraint (from SynchronizationTimingConstraint)

Generalizations
TimingConstraint (from TimingConstraint)
Attributes
synchronizationConstraintType : SynchronizationTypeEnum [1]
Associations
tolerance : MultidimensionalTime [1]
scope : TimingDescriptionEventChain [2..*]

New concept in TADL2 aligned with AUTOSAR concept with synchronizationConstraintType = stimulusSynchronization.

OutputSynchronizationConstraint (from TimingConstraints)

Generalizations
TimingConstraint (from TADL2)
Attributes

SynchronizationTimingConstraint (from SynchronizationTimingConstraint)

Generalizations
TimingConstraint (from TimingConstraint)
Attributes

synchronizationConstraintType : SynchronizationTypeEnum [1]

Associations

tolerance : TimingExpression [0..1] = infinity

scope : EventChain [2..*]

Inherited Associations

mode : Mode [0..1]

New concept in TADL2 aligned with AUTOSAR concept with synchronizationConstraintType = responseSynchronization.

Associations

tolerance : MultidimensionalTime [1]

scope : TimingDescriptionEventChain [2..*]

EOCExecutableEntityRef (from ExecutionOrderConstraint)

Generalizations

Identifiable (from Identifiable)

Attributes

Associations

executable : ExecutableEntity [1]

component : ComponentInCompositionInstanceRef [0..1]

successor : EOCExecutableEntityRef [0..*]

OrderConstraint (from TimingConstraint)

Generalizations

TimingConstraint (from TADL2)

Attributes

Associations

target : Event [1]

source : Event [1]

Inherited Associations

mode : Mode [0..1]

New concept in TADL2 aligned with AUTOSAR concept.

ExecutionOrderConstraint (from ExecutionOrderConstraint)

Generalizations

TimingConstraint (from TimingConstraint)

Attributes

Associations

orderedElement : EOCExecutableEntityRef [2..*]

EventTriggeringConstraint (from EventTriggeringConstraint) {abstract}

Generalizations

TimingConstraint (from TimingConstraint)

Attributes

Associations

event : TimingDescriptionEvent [1]

ConfidenceInterval (from EventTriggeringConstraint)

Generalizations

Attributes

propability : Float [1]

Associations

lowerBound : MultidimensionalTime [1]

upperBound : MultidimensionalTime [1]

ArbitraryConstraint (from TimingConstraints)

Generalizations
TimingConstraint (from TADL2)

Attributes
Associations
minimum : TimingExpression [1..*]
maximum : TimingExpression [1..*]
event : Event [1]
Inherited Associations
mode : Mode [0..1]

ArbitraryEventTriggering (from EventTriggeringConstraint)

Generalizations
EventTriggeringConstraint (from EventTriggeringConstraint)

Attributes
Associations
minimumDistance : MultidimensionalTime [1..*]
maximumDistance : MultidimensionalTime [1..*]
confidenceInterval : ConfidenceInterval [0..*] **1**
Inherited Associations
event : TimingDescriptionEvent [1] **2**

BurstConstraint (from TimingConstraints)

Generalizations
TimingConstraint (from TADL2)

Attributes
maxOccurences : int [1]

Associations
length : TimingExpression [1]
minimum : TimingExpression [0..1] = 0
event : Event [1]
Inherited Associations
mode : Mode [0..1]
New concept in TADL2, based on AUTOSAR.

BurstPatternEventTriggering (from EventTriggeringConstraint)

Generalizations
EventTriggeringConstraint (from EventTriggeringConstraint)

Attributes
maxNumberOfOccurrences : PositiveInteger [1]
minNumberOfOccurrences : PositiveInteger [0..1]

Associations
patternLength : MultidimensionalTime [1]
minimumInterArrivalTime : MultidimensionalTime [1]
patternPeriod : MultidimensionalTime [0..1]
patternJitter : MultidimensionalTime [0..1]
Inherited Associations
event : TimingDescriptionEvent [1]

Notes relationships to AUTOSAR 4 Timing Extension

These notes are referred from the table by number.

1 Compare TADL2 Probabilistic Timing.

2 Multiplicity aligned.

3 Name aligned.

4 See semantics of TADL2.

12.2 Relationship to EAST-ADL

As described in the introduction we compare the constraints in TADL2 with the constraints in the current EAST-ADL Timing package.

General changes:

- Time values are typed by TimingExpression instead of TimeDuration.

New concepts in TADL2 are:

- BurstConstraint
- RepetitionConstraint
- ExecutionTimeConstraint
- OrderConstraint
- StrongDelayConstraint
- TimingExpression, see chapter 5 and section 11.3.
- Probabilistic Timing Constraints, see chapter 6 and section 11.4.

TADL2	EAST-ADL Timing	Note
Event (from TADL2) {abstract}	**Event (from Timing) {abstract}**	
Generalizations	**Generalizations**	
TimingDescription (from TADL2)	TimingDescription (from Timing)	
Attributes	**Attributes**	
	isStateChange : Boolean = true [1]	**12**
Associations	**Associations**	
EventChain (from TADL2)	**EventChain (from Timing)**	
Generalizations	**Generalizations**	
TimingDescription (from TADL2)	TimingDescription (from Timing)	
Attributes	**Attributes**	
Associations	**Associations**	
segment : EventChain [*] {ordered}	segment : EventChain [*] {ordered}	
response : Event [1]	response : Event [1..*]	**5**
stimulus : Event [1]	stimulus : Event [1..*]	**5**
	strand : EventChain [*]	**13**
TimingConstraint (from TADL2) {abstract}	***TimingConstraint (from Timing) {abstract}***	
Generalizations	**Generalizations**	
EAElement (from Elements)	EAElement (from Elements)	
Attributes	**Attributes**	
Associations	**Associations**	
mode : Mode [0..1]	mode : Mode [*]	
	upper : TimeDuration [0..1]	**12**
	lower : TimeDuration [0..1]	**12**

	EventConstraint (from TimingConstraints) {abstract}	
	Generalizations	
	TimingConstraint (from Timing)	
	Attributes	
	Associations	
	offset : TimeDuration [0..1]	
	event : Event [0..1]	
	Inherited Associations	
	mode : Mode [*]	
	upper : TimeDuration [0..1]	
	lower : TimeDuration [0..1]	

The abstract concept EventConstraint has been removed

DelayConstraint (from TimingConstraints)	***DelayConstraint (from TimingConstraints) {abstract}***	
Generalizations	**Generalizations**	
TimingConstraint (from TADL2)	TimingConstraint (from Timing)	
Attributes	**Attributes**	
Associations	**Associations**	
lower : TimingExpression [0..1] = 0	lower : TimeDuration [0..1]	**3**
upper : TimingExpression [0..1] = infinity	upper : TimeDuration [0..1]	**3**
	jitter : TimeDuration [0..1]	**15**
	nominal : TimeDuration [0..1]	**15**
	scope : EventChain [0..1]	**14**
target : Event [1]		**1**
source : Event [1]		**1**
Inherited Associations	**Inherited Associations**	
mode : Mode [0..1]	mode : Mode [*]	

DelayConstraint is now not abstract, it has target and source Event relationships instead of a scope EventChain.

AgeConstraint (from TimingConstraints)	**AgeTimingConstraint (from TimingConstraints)**	**10**
Generalizations	**Generalizations**	
TimingConstraint (from TADL2)	DelayConstraint (from TimingConstraints)	**11**
Attributes	**Attributes**	
Associations	**Associations**	
	Inherited Associations	
scope : EventChain [1]	scope : EventChain [0..1]	**7**
minimum : TimingExpression [0..1] = 0	lower : TimeDuration [0..1]	**8**
maximum : TimingExpression [0..1] = infinity	upper : TimeDuration [0..1]	**8**
	jitter : TimeDuration [0..1]	**15**
	nominal : TimeDuration [0..1]	**15**
Inherited Associations		
mode : Mode [0..1]	mode : Mode [*]	

ReactionConstraint (from TimingConstraints)

Generalizations
TimingConstraint (from TADL2)
Attributes
Associations

scope : EventChain [1]
minimum : TimingExpression [0..1] = 0

maximum : TimingExpression [0..1] = infinity

Inherited Associations
mode : Mode [0..1]

ReactionConstraint (from TimingConstraints)

Generalizations
DelayConstraint (from TimingConstraints) **11**
Attributes
Associations
Inherited Associations
scope : EventChain [0..1] **7**
lower : TimeDuration [0..1] **8**

upper : TimeDuration [0..1] **8**

jitter : TimeDuration [0..1] **15**
nominal : TimeDuration [0..1] **15**

mode : Mode [*]

PatternConstraint (from TimingConstraints)

Generalizations
TimingConstraint (from TADL2)
Attributes
Associations
period : TimingExpression [1] = 0
minimum : TimingExpression [0..1] = 0

jitter : TimingExpression [0..1] = 0

offset : TimingExpression [1..*]

event : Event [1]
ref : Event [1]
Inherited Associations
mode : Mode [0..1]

PatternEventConstraint (from TimingConstraints) 10

Generalizations
EventConstraint (from TimingConstraints) **11**
Attributes
Associations
period : TimeDuration [1]
minimumInterArrivalTime : TimeDuration [1] **9**
occurrence : TimeDuration [1..*] {ordered} **15**
jitter : TimeDuration [1] **3**
Inherited Associations
offset : TimeDuration [0..1] **6**

event : Event [0..1] **7**
2

mode : Mode [*]
upper : TimeDuration [0..1] **15**
lower : TimeDuration [0..1] **15**

PeriodicConstraint (from TimingConstraints)

Generalizations
TimingConstraint (from TADL2)
Attributes
Associations
jitter : TimingExpression [0..1] = 0
period : TimingExpression [1]
minimum : TimingExpression [0..1] = 0

event : Event [1]
Inherited Associations
mode : Mode [0..1]

PeriodicEventConstraint (from TimingConstraints) 10

Generalizations
EventConstraint (from TimingConstraints) **11**
Attributes
Associations
jitter : TimeDuration [1] **3**
period : TimeDuration [1]
minimumInterArrivalTime : TimeDuration [1] **9**
Inherited Associations
offset : TimeDuration [0..1] **15**
event : Event [0..1] **7**

mode : Mode [*]
upper : TimeDuration [0..1] **15**
lower : TimeDuration [0..1] **15**

SporadicConstraint (from TimingConstraints)	SporadicEventConstraint (from TimingConstraints)	10
Generalizations	**Generalizations**	
TimingConstraint (from TADL2)	EventConstraint (from TimingConstraints)	**11**
Attributes	**Attributes**	
Associations	**Associations**	
jitter : TimingExpression [0..1] = 0	jitter : TimeDuration [0..1]	**3**
	period : TimeDuration [1]	**15**
	maximumInterArrivalTime : TimeDuration [0..1]	**15**
minimum : TimingExpression [0..1] = 0	minimumInterArrivalTime : TimeDuration [1]	**9**
upper : TimingExpression [0..1] = infinity	upper : TimeDuration [0..1]	**3**
lower : TimingExpression [0..1] = 0	lower : TimeDuration [0..1]	**3**
	Inherited Associations	
	offset : TimeDuration [0..1]	**15**
event : Event [1]	event : Event [0..1]	**7**
Inherited Associations		
mode : Mode [0..1]	mode : Mode [*]	

InputSynchronizationConstraint (from TimingConstraints)	InputSynchronizationConstraint (from TimingConstraints)	
Generalizations	**Generalizations**	
TimingConstraint (from TADL2)	AgeTimingConstraint (from TimingConstraints)	
Attributes	**Attributes**	
Associations	**Associations**	
tolerance : TimingExpression [0..1] = infinity	width : TimeDuration [1]	**8**
	Inherited Associations	
scope : Event [2..*]	scope : EventChain [0..1]	**5**
	lower : TimeDuration [0..1]	**15**
	upper : TimeDuration [0..1]	**15**
	jitter : TimeDuration [0..1]	**15**
	nominal : TimeDuration [0..1]	**15**
Inherited Associations		
mode : Mode [0..1]	mode : Mode [*]	

OutputSynchronizationConstraint (from TimingConstraints)	OutputSynchronizationConstraint (from TimingConstraints)	
Generalizations	**Generalizations**	
TimingConstraint (from TADL2)	ReactionConstraint (from TimingConstraints)	**11**
Attributes	**Attributes**	
Associations	**Associations**	
tolerance : TimingExpression [0..1] = infinity	width : TimeDuration [1]	**8**
	Inherited Associations	
scope : Event [2..*]	scope : EventChain [0..1]	**5**
	lower : TimeDuration [0..1]	**15**
	upper : TimeDuration [0..1]	**15**
	jitter : TimeDuration [0..1]	**15**
	nominal : TimeDuration [0..1]	**15**

Inherited Associations
mode : Mode [0..1]

mode : Mode [*]

PrecedenceConstraint (from Timing)

Generalizations
TimingConstraint (from Timing)
Attributes
Associations
Dependencies
successive : FunctionPrototype [1..*]
«instanceRef»
preceding : FunctionPrototype [1]
«instanceRef»

Not included in TADL2, do not reference Events.

ArbitraryConstraint (from TimingConstraints)

Generalizations
TimingConstraint (from TADL2)
Attributes
Associations
minimum : TimingExpression [1..*]
maximum : TimingExpression [1..*]
event : Event [1]
Inherited Associations
mode : Mode [0..1]

ArbitraryEventConstraint (from TimingConstraints) 10

Generalizations
EventConstraint (from TimingConstraints) **11**
Attributes
Associations
minimumInterArrivalTime : TimeDuration [1..*] **10**
maximumInterArrivalTime : TimeDuration [1..*] **10**
Inherited Associations
offset : TimeDuration [0..1] **15**
event : Event [0..1] **7**
mode : Mode [*]
upper : TimeDuration [0..1] **15**
lower : TimeDuration [0..1] **15**

Notes Relationships to EAST-ADL

These notes are referred from the table by number.

1 Added

2 Added, see semantics of TADL2.

3 Default value added.

4 Multiplicity changed and default value added.

5 Multiplicity changed, alignment with AUTOSAR.

6 Multiplicity changed, see semantics of TADL2.

7 Multiplicity changed.

8 Name changed due to AUTOSAR alignment, multiplicity changed, default value added.

9 Name changed, default value added.

10 Name changed.

11 New specialization of abstract concept.

12 Removed from this abstract concept.

13 Removed, alignment with AUTOSAR.

14 Removed, replaced by target and source Events.

15 Removed, see semantics of TADL2.