TIMMO
Timing Model

Report type: Deliverable D6
Report name: TADL: Timing Augmented Description Language version 2
Report status: Public
Version number: Version 1.2
Date of preparation: 05.10.2009
TIMMO Partners

Audi Electronics Venture GmbH
Chalmers University
Continental Automotive GmbH
Commissariat à l’Énergie Atomique LIST
DENSO Europe B.V.
ETAS Entwicklungs- und Applikationswerkzeuge für elektronische Systeme GmbH
Mentor Graphics (Scand) AB
Paderborn University
Robert Bosch GmbH
Siemens AG - Siemens IT Solutions and Services
SymtaVISION GmbH
TTTech Computertechnik AG
Volkswagen AG
Volvo Technology AB
ZF Friedrichshafen AG
Authors

Rolf Johansson, Mentor Graphics
Patrick Frey, ETAS GmbH
Jan Jonsson, Chalmers University of Technology
Johan Nordlander, Chalmers University of Technology
Risat Mahmud Pathan, Chalmers University of Technology
Nico Feiertag, Symtavision GmbH
Martin Schlager, TTTech Computertechnik AG
Huascar Espinoza, CEA LIST
Kai Richter, Symtavision GmbH
Stefan Kuntz, Continental Automotive GmbH
Henrik Lönn, Volvo Technology Corporation
Ramin Tavakoli Kolagari, Volvo Technology Corporation
Hans Blom, Volvo Technology Corporation

Project Co-ordinator

Dr. Friedhelm Stappert

Continental Automotive GmbH
Systems & Technology Automotive
Siemensstrasse 12
93055 Regensburg, Germany
Tel.: +49 941 790-5778
Fax: +49 941 790-995778
Email: friedhelm.stappert@continental-corporation.com

Copyright 2009: The TIMMO Consortium
## Revision chart and history log

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>2009-07-10</td>
<td>Version 1.0</td>
</tr>
<tr>
<td>1.1</td>
<td>2009-09-02</td>
<td>Public version. An example added. Layout streamlining by IMC</td>
</tr>
<tr>
<td>1.2</td>
<td>2009-09-30</td>
<td>Corrected style w.r.t. italics, added page numbering</td>
</tr>
</tbody>
</table>
# Table of contents

1 Introduction .................................................................................................................. 3  
1.1 Restrictions and challenges .................................................................................. 3  
1.2 The development of TADL .................................................................................. 4  
2 Timing modeling......................................................................................................... 8  
2.1 AUTOSAR concepts with timing properties ....................................................... 8  
2.2 EAST-ADL2 for higher abstraction levels and requirements ......................... 11  
2.3 Exchanging Timing Models .............................................................................. 14  
2.4 The TADL editor ................................................................................................. 15  
2.5 Supporting tools ................................................................................................... 16  
2.6 Modeling guideline ............................................................................................... 17  
2.6.1 Vehicle Feature Model .............................................................................. 17  
2.6.2 Requirements modeling in EAST-ADL2 .................................................... 18  
2.6.3 Modeling of timing for functions ................................................................. 18  
2.6.4 Modeling of timing in the design phase ...................................................... 19  
3 Syntax ....................................................................................................................... 21  
3.1 Structural framework ............................................................................................ 21  
3.2 Events and event chains ....................................................................................... 22  
3.3 Age and reaction timing constraints ..................................................................... 23  
3.4 Input and output synchronization constraints ..................................................... 24  
3.5 Jitter as a property of constraints ......................................................................... 25  
3.6 Event Constraints ................................................................................................ 26  
3.6.1 Periodic Event Constraint .......................................................................... 27  
3.6.2 Sporadic Event Constraint .......................................................................... 27  
3.6.3 [Concrete] Pattern Event Constraint ......................................................... 29  
3.6.4 Arbitrary Event Constraint .......................................................................... 30  
4 Semantics ................................................................................................................ 32  
4.1 Events .................................................................................................................... 32  
4.2 Repetition rate constraints ................................................................................... 34  
4.2.1 Periodic repetition ...................................................................................... 36  
4.2.2 Sporadic repetition ...................................................................................... 38  
4.2.3 Pattern repetition ....................................................................................... 41  
4.2.4 Arbitrary repetition ..................................................................................... 43  
4.3 Delay constraints .................................................................................................. 45  
4.3.1 ReactionConstraint ................................................................................... 46  
4.3.2 AgeTimingConstraint .............................................................................. 48  
4.3.3 Output synchronization ............................................................................... 50
4.3.4 Input synchronization .................................................. 53
4.4 Delay composition and decomposition .......................... 55
  4.4.1 Sequential composition and decomposition ............... 56
  4.4.2 Parallel composition and decomposition .................. 57
5 Conclusions ........................................................................... 58
  5.1 TADL semantics in research ........................................... 58
References ................................................................................ 60
6 Annex A Meta-modeling guide.............................................. 61
  6.1.1 How the AUTOSAR meta-model was extended in
        TIMMO ........................................................................... 61
  6.2 Meta-modeling in Enterprise Architect ......................... 62
       6.2.1 Alignment of EAST-ADL2 with AUTOSAR M3.............. 62
7 Annex B UML model in XMI Format ........................................ 67
8 Annex C Specification of the model TADL XML exchange format ........................................ 68
9 Annex D TADL model editor ............................................... 69
    9.1 Snapshots of the TADL editor ..................................... 69
10 Annex E UML profile for TADL ........................................... 72
11 Annex F TADL meta-model ................................................. 75
### Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTOSAR</td>
<td>AUTomotive Open System ARchitecture</td>
</tr>
<tr>
<td>AR</td>
<td>AUTOSAR</td>
</tr>
<tr>
<td>BMT</td>
<td>Behavioral Modeling Tool</td>
</tr>
<tr>
<td>COTS</td>
<td>Commercial/Common-off-the-shelf</td>
</tr>
<tr>
<td>DM</td>
<td>Domain Model</td>
</tr>
<tr>
<td>EA</td>
<td>Enterprise Architect (Sparx Systems)</td>
</tr>
<tr>
<td>EAST-ADL2</td>
<td>Embedded Architectures and Software Technologies - Architecture Description Language 2</td>
</tr>
<tr>
<td>MARTE</td>
<td>Modeling and Analysis of Real Time and Embedded systems (OMG)</td>
</tr>
<tr>
<td>MM</td>
<td>Meta model</td>
</tr>
<tr>
<td>OMG</td>
<td>Object Management Group</td>
</tr>
<tr>
<td>RTE</td>
<td>RunTime Environment (AUTOSAR)</td>
</tr>
<tr>
<td>SW</td>
<td>Software</td>
</tr>
<tr>
<td>SWC</td>
<td>Software Component</td>
</tr>
<tr>
<td>TADL</td>
<td>Timing Augmented Description Language</td>
</tr>
<tr>
<td>TIMMO</td>
<td>Timing Model</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modeling Language (OMG)</td>
</tr>
<tr>
<td>VSA</td>
<td>Vehicle System Architect, Product from Mentor Graphics</td>
</tr>
<tr>
<td>XSD</td>
<td>XML Schema Definition</td>
</tr>
</tbody>
</table>
Executive summary

Deliverable D6 describes the development of the Timing Augmented Description Language (TADL) of the TIMMO project. The primary purpose of TADL is the enrichment of design models that are created by typical modeling languages like EAST-ADL2 and AUTOSAR with data that specify the required and/or existing timing behavior of the parts of the systems described. Hence, TADL enables the analysis of the given system to scrutinize the given system in terms of timing and dynamics.

The design of TADL draws inspiration from three main sources: existing design frameworks such as AUTOSAR, EAST-ADL2, and MARTE, existing analysis tools within the automotive domain, and typical use case scenarios from automotive applications where timing is important.

The syntax of TADL is compliant to the released AUTOSAR metamodel (3.1); the semantics of TADL is novel and extends and clarifies the original AUTOSAR semantics. A tool infrastructure has been developed that allows the user to address the system design on several abstraction levels. The definition of TADL is done using UML within the Enterprise Architect tool, while user model editing is done with an Eclipse-compliant plug-in in the Mentor VSA tool set. An exchange format between the user model editor and supporting analysis tools is defined.

The feasibility of the TADL tool chain infrastructure has been demonstrated with the aid of a simple, but non-trivial, use case example. Validation of TADL has been done by feedback from the TIMMO methodology work package and the modeling performed in the TIMMO validators. The TIMMO tool board has identified the exploitation of TADL in supporting tools.
1 Introduction

The main goal of TIMMO is to provide a solution of handling 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. The TIMMO project is organized in several work packages (WPs), and each work package has a specific goal that is aligned to this main project goal. The objectives of WP2 are to define a Timing Augmented Description Language (TADL) together with an exchange format that enables the collaborative work of future TADL-compliant design, development, and performance analysis tools.

The inspirational input in WP2 are the project requirements developed in WP1 (Conceptual Framework), a set of real-life scenarios defined in WP3 (Methodology), and existing concepts available in other frameworks, such as AUTOSAR [11], EAST-ADL2 [4], and MARTE [10]. The anticipated result from WP2 is therefore a language that is compliant with existing design frameworks and application requirements.

In order to reach the anticipated goals within the allocated time frame for the TIMMO project, it was deemed best that the development of TADL was

(1) imposed with reasonable restrictions regarding the scope of syntax and semantics, and

(2) guided by an efficient design process.

Section 1.1 summarizes the necessary restrictions on TADL and also describes how they give rise to some interesting design challenges. Section 1.2 describes how TADL has been developed using a milestone-driven process that interacts with WP3 and WP4, in combination with an iterative refinement process where new concepts and refined semantics are added on demand as new proof-of-concepts demonstrations require so.

1.1 Restrictions and challenges

During the initial work done within WP2, a “wish list” emerged for the TADL timing model. To that end, the following aspects were used for expressing the desired characteristics.

With respect to **compatibility**, the chosen timing model should …

… encompass existing automotive timing requirements,

… work within existing automotive design architectures and models,

… interface to existing automotive scheduling mechanisms, and

… interface to existing tool vendor timing models.

With respect to **generality**, the chosen timing model should …

… not be hard-coded to a certain scheduling mechanism,

… not be hard-coded to a certain tool vendor’s timing model,
be extendible to other state-of-the-art scheduling mechanisms, and

have a semantics that is consistent across abstraction levels. With respect to conciseness, the chosen timing model should

avoid multiple timing constructs with overlapping purposes and

have unambiguous semantics for each timing construct.

The compatibility aspect requires the use of existing modeling and analysis tools that are already available to the participants of TIMMO, such as Enterprise Architect, Eclipse and Mentor VSA. This aspect also indicates that general modeling languages and data exchange formats, such as UML and XML, should be used.

In the first phase of WP1 it was decided that WP2 should (1) use AUTOSAR as the reference architecture, (2) take inspiration from the extensions made to AUTOSAR within the EAST-ADL2 project, and (3) take inspiration from existing description languages, such as MARTE. While this decision provided WP2 with some very elaborate frameworks suitable for the TADL, it also raised two important design challenges:

Challenge 1: How could the scope of the work be restricted in such a way that a functional first version of TADL would be available within the relatively tight schedule for the first phase of WP2?

It soon became clear that considerable project time would have to be spent to fully understand the extensive functionality available in the AUTOSAR, EAST-ADL2, and MARTE frameworks. For this reason, it was decided that the development of TADL should start with a minimal set of concepts, triggered by the use case scenarios provided by WP3. The set of concepts should then be dynamically extended as more proof-of-concept testing is performed in later phase of the project.

Challenge 2: How could a clear, unambiguous semantics for the timing concepts be defined for the TADL?

An initial review of the AUTOSAR, EAST-ADL2, and MARTE frameworks revealed that there exists timing concepts in one or more of the frameworks where the semantics are unclear, mutually incompatible, or lacking expressiveness at certain design levels. For this reason, the development of a new "stand-alone" timing semantics was initiated. The purpose of the new semantics is to act as a stepping stone between higher-level functional design concepts and lower-level implementation detail, such as the scheduling of threads in an operating system. The creation of the new semantics is done in parallel with the development of the tool infrastructure, with the goal of gradually adopting the new semantics by the tools. To avoid complications in the adoption, the new semantics is being based on an object-oriented model similar to that of AUTOSAR and EAST-ADL2, but with a significantly more concise formalism.

1.2 The development of TADL

The work leading up to the proposed TADL was driven by two development processes. One process was coordinating the development work within WP2 and WP3 by means of a set of
milestones to be reached within the first phase of the work packages. The WP2 milestones were the following:

1. Evaluate and decide upon a tool infrastructure, in terms of meta-model, suitable semantics, modeling framework, and data interchange format.
2. Define the basic language structure, together with a first version of the model editor. This was based on preliminary versions of scenarios supplied by WP3.
3. The requirements collected in WP1 were analyzed and further organized into fundamental (non-redundant) language construct classes to be introduced in TADL. This included evaluating how the requirements map versus the capacities of existing tools.
4. Validate the TADL ("proof of concept") with respect to extended versions of the scenarios from WP3 and WP4.

A second design process concerns iterative refinement of the TADL using both modeling tools and scenario examples in the iterative loop. Below follows a summary of this process.

The model structure for TIMMO is based on the EAST-ADL2 meta-model, where the AUTOSAR meta-model is combined with the EAST-ADL2 meta-model on higher abstraction layers for functional definition, as shown in Figure 1.1. Existing timing constructs in AUTOSAR and EAST-ADL2 are reused where applicable, and new constructs are added to meet the TIMMO needs. As EAST-ADL2 is rather flexible, changes can be made to meet TIMMO needs, while the AUTOSAR R3.1 meta-model is fixed and modifications can only be proposed for the AUTOSAR R4.0 release.

![Figure 1.1. Model Structure of TIMMO](image)

The AUTOSAR meta-modeling approach is used for all aspects of the meta-model, in order to make the meta-model integration and tool chain smooth. This includes using the AUTOSAR M3 meta-meta-model, i.e., the template modeling guide, and using the Enterprise Architect UML Editor for the meta-model definition, compare Figure 1.2. This is also in line with the requirements set up by TIMMO in WP1.

The TADL meta-model in Enterprise Architect includes AUTOSAR, EAST-ADL2, and TADL. The definition of the exchange format for
TADL is in the TADL XML schema definition (XSD) defined based on this Enterprise Architect model.

**Formal mapping of AUTOSAR concepts on metalevels**

**M3**: An Autosar MMM that consist of an M3 profile and Template Modeling Guidelines [1].

**M2**: Autosar MM specifying templates following the rules set up at the M3 level, represented as an AUTOSAR XML Schema Definition (XSD).

**M1**: Autosar user model represented as standard AUTOSAR XML, conforming to the AUTOSAR XSD.

**M0**: The real world objects that are classified by the user model elements.

In order to define the TADL, the starting point has been an AUTOSAR subset that gave structural information for timing analysis. The AUTOSAR meta-model was imported as a base for the syntax definition in EA, see Annex B UML model in XMI Format.

In addition to the AUTOSAR meta-model, relevant parts of the EAST-ADL2 are included. The EAST-ADL2 was already defined in Enterprise Architect by the ATESST project and was integrated after certain modifications to comply with the AUTOSAR meta-modeling guidelines, cf. Annex A Meta-modeling.

In order to have feedback to the TADL meta-model definition, the language has been defined iteratively, with both modeling tool and examples in the loop. According to Figure 1.3, the iterations included:

- TADL concepts definitions based on TIMMO requirements and existing approaches. This is described in this deliverable in chapters 2, 3.
- TADL definition in Enterprise Architect, see Annex A Meta-modeling.
- TADL exchange format definition (XSD), see Annex C Specification of the model TADL XML exchange.
- TADL Editor, the AUTOSAR authoring tool update for TIMMO meta-model (TADL Editor), see Annex D TADL model.
- User modeling of example systems in the TADL Editor, see Annex D TADL model.
- Validation of TADL in supporting tools for schedulability analysis, communication configuration, allocation, etc. see section 2.5.

Definition of UML2 profile for TADL. See

- Annex E UML profile for TADL.
Figure 1.3. Development of the TADL and the different tools, refined incrementally.
This chapter describes the basis of timing modeling and how the concepts developed in TIMMO are based on modeling concepts available in EAST-ADL2 and AUTOSAR.

2.1 AUTOSAR concepts with timing properties

One of the primary objectives of the TIMMO project is to ensure that the concepts worked out are properly aligned with the AUTOSAR standards.

At the time the TIMMO project started, the AUTOSAR Release 2.1 was available for reference and analysis—the release 3.0 was under preparation at that time, and became available later. In a first step the contents of both releases—2.1 and 3.1—have been examined with regard to timing. In other words, all timing related elements and attributes have been identified in the AUTOSAR meta-model. The result of the comparison of both releases showed that not many additional elements have been introduced in release 3. Mainly more elements and attributes related to FlexRay have been added due to the integration of the Field Bus Exchange Format (FIBEX) standard. The table below shows an excerpt of elements currently contained in the AUTOSAR meta-model Release 3.0 that are related to timing.

<table>
<thead>
<tr>
<th>AUTOSAR 3.0</th>
<th>Element</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTOSAR VFB.VFBRunnable</td>
<td>Time</td>
</tr>
<tr>
<td>AUTOSAR Templates.GenericStructure.Infrastructure</td>
<td>ARObj</td>
</tr>
<tr>
<td>AUTOSAR Templates.GenericStructure.CommonPatterns.PrimitiveTypes</td>
<td>DateTime</td>
</tr>
<tr>
<td>AUTOSAR Templates.ECUResourceTemplate.BasicElements</td>
<td></td>
</tr>
<tr>
<td>---------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td><strong>BasicElements</strong></td>
<td></td>
</tr>
<tr>
<td>BootTime</td>
<td></td>
</tr>
<tr>
<td>TimeRange</td>
<td></td>
</tr>
<tr>
<td><strong>Peripherals</strong></td>
<td></td>
</tr>
<tr>
<td>AnaloguelIO</td>
<td></td>
</tr>
<tr>
<td><strong>ECUElectronics</strong></td>
<td></td>
</tr>
<tr>
<td>Clock</td>
<td></td>
</tr>
<tr>
<td>Oscillator</td>
<td></td>
</tr>
<tr>
<td>PowerDriverHWPort</td>
<td></td>
</tr>
<tr>
<td><strong>SensorActuator</strong></td>
<td></td>
</tr>
<tr>
<td>DisplayHW</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AUTOSAR Templates.SWComponentTemplate</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ApplicationAttributes</strong></td>
</tr>
<tr>
<td>FilterDebouncingEnum</td>
</tr>
<tr>
<td><strong>Communication</strong></td>
</tr>
<tr>
<td>ModeSwitchedAckRequest</td>
</tr>
<tr>
<td>UnqueuedReceiverComSpec</td>
</tr>
<tr>
<td>TransmissionAcknowledgementRequest</td>
</tr>
<tr>
<td><strong>SwcInternalBehavior.ServerCall</strong></td>
</tr>
<tr>
<td>ServerCallPoint</td>
</tr>
<tr>
<td><strong>SwcInternalBehavior.RTEEvents</strong></td>
</tr>
<tr>
<td>TimingEvent</td>
</tr>
<tr>
<td>WaitPoint</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SystemTemplate</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Fibex.Fibex4Can.CanTopology</strong></td>
</tr>
<tr>
<td>CanCluster</td>
</tr>
<tr>
<td>CanCommunicationController</td>
</tr>
<tr>
<td><strong>Fibex.Fibex4Flexray.FlexrayCommunication</strong></td>
</tr>
<tr>
<td>AbsolutelyScheduledTiming</td>
</tr>
<tr>
<td><strong>Fibex.Fibex4Flexray.FlexrayTopology</strong></td>
</tr>
<tr>
<td>FlexrayCluster</td>
</tr>
<tr>
<td>FlexrayCommunicationController</td>
</tr>
<tr>
<td><strong>Fibex.Fibex4Lin.LinCommunication</strong></td>
</tr>
<tr>
<td>AssignFrameIdTiming</td>
</tr>
<tr>
<td>AssignNadTiming</td>
</tr>
<tr>
<td>AssignmentTimingType</td>
</tr>
<tr>
<td>DataTiming</td>
</tr>
<tr>
<td>RelativelyScheduledTiming</td>
</tr>
<tr>
<td>UnassignFrameIdTiming</td>
</tr>
<tr>
<td><strong>Fibex.Fibex4Lin.LinTopology</strong></td>
</tr>
<tr>
<td>LinMaster</td>
</tr>
<tr>
<td><strong>Fibex.FibexCore.CoreCommunication.Timing</strong></td>
</tr>
<tr>
<td>CyclicTiming</td>
</tr>
<tr>
<td>EventControlledTiming</td>
</tr>
<tr>
<td>RequestControlledTiming</td>
</tr>
<tr>
<td>TimeRangeType</td>
</tr>
<tr>
<td>TimeRangeTypeTolerance</td>
</tr>
<tr>
<td>TransmissionModeFalseTiming</td>
</tr>
<tr>
<td><strong>Fibex.FibexCore.CoreCommunication</strong></td>
</tr>
<tr>
<td>IpduTiming</td>
</tr>
<tr>
<td>SignalPort</td>
</tr>
<tr>
<td><strong>TimingConstraints</strong></td>
</tr>
<tr>
<td>EndToEndTiming</td>
</tr>
<tr>
<td>TimingConstraint</td>
</tr>
<tr>
<td>[OrderedElement]</td>
</tr>
<tr>
<td>[RunnableExecutionOrder]</td>
</tr>
<tr>
<td><strong>TransportProtocols</strong></td>
</tr>
<tr>
<td>CanTpConnectionChannel</td>
</tr>
</tbody>
</table>
A second analysis has been conducted that compared the timing parameters the tool vendors (ETAS, SyntaVISION, and TTTech) already support in their available tools. The conclusion of this analysis is that the parameters describe mainly:

1. execution time of tasks/programs,
2. network latencies (network “execution time”) and parameters, and
3. scheduling parameters. The vast majority of these parameters are already defined in the AUTOSAR standards.

Further scrutiny showed that there is still a lack of precise definition of time, events and event models, timing constraints and properties.

Time essentials have been proposed to sketch out TIMMO’s idea of time and events. The time essentials also propose to make use of the Timing Model specified by OMG’s Modeling and Analysis of Real-Time Embedded systems (MARTE) profile. The decision taken was to make only use of a consistent subset of this Timing Model.

A number of event models has been defined to describe event occurrences that are common in [automotive] software-intensive-systems, like periodic events, sporadic events, events following a pattern (for example events generated by the crankshaft), and arbitrary patterns.

In contrast to AUTOSAR, TIMMO is also considering higher levels of abstraction (for example analysis and design, therefore the EAST-ADL2 is also included partly) and proposing means to capture timing relevant data on such higher levels of abstraction as well as a methodology to process such data. And AUTOSAR focuses on the implementation level.

In both cases, the idea is to be able to describe the timing properties of an existing system in order to:

1. perform analyses regarding dynamic behavior of the system and
2. to validate an existing system in terms of timing.

One of the challenges is to provide means to transform timing data from one level of abstraction to another level of abstraction—especially the design level (TIMMO) to the implementation level (AUTOSAR)—to ensure consistency between the two domains regarding timing data.
2.2 EAST-ADL2 for higher abstraction levels and requirements

EAST-ADL2 is an Architecture Description Language (ADL) originally defined in the EAST-EEA project [3]. It is an approach for describing automotive electronic systems through an information model that captures engineering information in a standardized way. Aspects covered include:

- (Vehicle) Features,
- Functions,
- Requirements,
- Variability,
- Software components,
- Hardware components, and
- Communication.

EAST-ADL2 contains several abstraction levels, see Figure 2.2.

The software- and electronics-based functions/features of the vehicle are described at different levels of abstraction. The abstraction levels are in a sense “views” on the EAST-ADL2 system model. Unlike views, however, the content of each abstraction level are fundamental building blocks of the EAST-ADL2 system model. The content of an abstraction level forms a complete representation of the vehicle embedded system, with respect to the concerns of that abstraction level:

- On vehicle level the features of the vehicle.
- On analysis level an abstract functional definition of the vehicle.
- The design level has a detailed functional representation of the vehicle including hardware.
• The implementation level represents the software and hardware architecture. This level includes AUTOSAR.

• The operational level corresponds to the actual vehicle embedded system with its memory contents, circuits, and wires.

To further elaborate on the concept of abstraction levels, a set of related and orthogonal concerns can be identified. The concerns include different levels of abstraction, detail, integration, decomposition, and model of computation. See the TIMMO deliverable D7.

The proposed abstraction levels and the contained artefacts provide separation of concerns and an implicit style for using the modeling elements, although it was tried to avoid forcing a certain process or method when applying the EAST-ADL2. In general, application software development should be made on the vehicle, analysis and design levels; the infrastructure software should already be available, or developed separately. Implementation and operational level aspects may to some extent be automatically generated by tools.

Figure 2.3 shows the artifacts that represent the system on each abstraction level. Grey arrows indicate that there is direct data exchange between constructs. In addition to the data exchange, parts of the model are linked with various traceability relations between abstraction levels. This makes it possible to trace an entity from abstract features down to components in hardware and software.

![Diagram of the EAST-ADL2 model](image)

**Figure 2.3.** The parts of the EAST-ADL2 model.

The electronic systems of a vehicle or the respective vehicle product lines (or vehicle product families, i.e., a set of systems developed from a common set of core assets in a prescribed way) and the features including the description of the feature variability are to be defined in the Vehicle Feature Model (VFM) at the vehicle level.
A complete representation of the electronic functionality on an abstract level is modeled in the Functional Analysis Architecture (FAA). One or several entities (analysis functions) of the FAA can be combined and reused to realize features. The FAA captures the principal interfaces and behavior of the subsystems of the vehicle. It allows validation and verification of the integrated system or its subsystems on a high level of abstraction.

The more implementation-oriented aspects are introduced while defining the Functional Design Architecture (FDA). The features are here realized in a more implementation-oriented manner. The system is structured and decomposed to suitable components taking efficiency, legacy and reuse, COTS availability, hardware allocation, etc, into account.

A Hardware Design Architecture is included on this level, to capture sensors and actuators and their properties. The ECUs, buses, power supply, etc. are also included to allow a definition of the target hardware for the system for reasoning about allocation.

The platform software should usually not have to be considered during the development of application functions. However, the AUTOSAR Basic Software contains infrastructures of two kinds: transparent functions including execution control and communication services, and services that directly influence the applications, such as error handling and mode control. Application development needs to take the functionality of these latter services into account, even on the Design Level. For this reason, the Middleware Abstraction is included in the Design Architecture.

The software architecture corresponding to the FDA is defined in the Implementation Architecture. This is a container element for the model that captures the AUTOSAR realization of the system. The contained AUTOSAR software components are defined to realize the functional definition as given on Design Level. It is organized to meet implementation constraints related to efficiency, allocation, procurement, legacy, etc.

The code may be obtained by automatic code generation out of FDA models, or by extracting the (optional) plain code out of the software functions of the FDA. The Implementation Architecture (IA) further contains the implementation of data exchange, i.e., the implementation representation of signals for frames and memory buffers.

Entities on the VFM, FAA and FDA are related. However there are seldom 1-1 relationships between features (VFM), analysis functions (FAA), and software functions (FDA).

The hardware architecture should be considered in parallel with application development. At Analysis Level, the hardware should not be considered at all in the early stages. When the development effort is iterated, however, implementation effects such as communication delays or computational inaccuracy will manifest on lower abstraction levels. These may have to be analyzed from a functional point of view even on the abstract representations. There is thus an indirect effect of hardware even for the FAA models. On the Design Level and below, the hardware architecture forms a natural constraint for development and the hardware and application software development needs to be iterated and performed together.
To verify and validate a feature across all abstraction levels, an environment model is needed early on. This plant model captures the behavior of the vehicle dynamics, driver, etc. The core part of the environment model can be the same for all abstraction levels. Alternate plant models would rather be needed to support different verification and validation scenarios. For example, a simulation to assess fuel consumption requires a simple vehicle dynamics mode, while stability assessment of electronic brakes needs a largely different representation of vehicle dynamics. The latter environment model should be the same regardless of whether the abstract algorithm or the final ECU implementation is verified. This approach can be modeled by means of the EAST-ADL2 variability constructs applied to the environment model. The environment model can also be used early to define timing constraints.

### 2.3 Exchanging Timing Models

In the TIMMO meta-model approach an XML exchange format is defined and allows a tool chain according to Figure 2.4:

- Specify language in Enterprise Architect, using AUTOSAR guidelines (M3 profile and template guidelines [1]).
- Generate the TADL Editor and XSD from the common source which is the XMI export from Enterprise Architect.
- The editing of a user model is supported by an AUTOSAR authoring tool extended for TIMMO.
- XML is the exchange format between the TADL Editor and between supporting tools. The format of the exchanged files is specified as the XML Schema definition (XSD) that resembles AUTOSAR XSD.

At a later stage, the TIMMO concepts can be implemented as UML2 profiles for AUTOSAR, EAST-ADL2 and TADL. This would allow a regular profile aware UML tool to be used as TADL editor. Some transformation is anticipated to make the model export (XMI) conform to the defined TADL and AUTOSAR XSD.

Figure 2.4. EA model (UML) as syntax, refined as XSD, semantics documented for each element.
Note that there are two kinds of tools represented in Figure 2.4: Infrastructure tools and supporting tools. The former are used to define the TIMMO meta-model, while the latter are used to edit and analyze the user models.

### 2.4 The TADL editor

The TADL has been defined in a meta-model that is used by VSA to generate the exchange format and the TADL editor, where the user modeling is done, see Figure 2.5.

![Figure 2.5. The specified syntax is the specification for the exchange format and the TADL editor.](image)

The TADL editor, see Figure 2.6, is an adaptation of an Eclipse-based plug in the VSA tool set from Mentor Graphics. One input to the tool generation at Mentor Graphics is the AUTOSAR Meta model modeled in Enterprise Architect (EA). In the context of TIMMO, AUTOSAR MM has been complemented with a TADL XMI and imported to the AUTOSAR Enterprise Architect file. This resulting Meta-model describes the AUTOSAR as well as the TADL. It is used in the tool generation. The EA Meta Model is converted to both an ECore Meta Model and an XSD schema. The ECore meta-model is the base for the generation of the Eclipse tool.
2.5 Supporting tools

The user models can be exchanged from the TADL editor to the supporting tools for analysis, see Figure 2.7. For exploitation of the TADL in tools from the different TIMMO partners we refer to the TIMMO tool board.

Figure 2.6. A ReactionConstraint has been created and refer to the port instances in their contexts. Additional snapshots of the TADL editor are available in Annex D TADL model.

Figure 2.7. The TADL is used for exchange of analysis information and is validated in supporting tools.
2.6  Modeling guideline

2.6.1  Vehicle Feature Model

Vehicle features are modeled in the Vehicle Feature Model with the concepts from EAST-ADL2. A vehicle feature is a (non-)functional characteristic, constraint or property that can be present or not in a (vehicle) product line describing the overall vehicle. Relationships between vehicle features are modeled with feature links, the basic dependencies between vehicle features are of type includes or excludes. Different vehicle features may be grouped and information on the cardinality of the owned features can be supplied. This way the different possible configurations of the described features are modeled.

Figure 2.8. Meta-model for vehicle feature modeling.

At this high level of abstraction it is also possible to model requirements that relate, e.g., to timing. This is done by the requirements modeling concepts from EAST-ADL2 that are aligned with SysML. On vehicle level, timing relates to solution-independent events like "brake request" and "brake activation" rather than concrete solutions like sensor sampling and signal arrivals. A "VehicleFeature" is linked to the requirement by the "ADLSatisfy" relationship. Verification and validation information can be linked to the requirement with an "ADLVerify" relation. More concrete requirements may be defined at lower abstraction levels and an "ADLDeriveReqt" relation provides traceability. Requirements have a text attribute for the requirement text and timing-related requirements can also be supported by a more formal "TimingConstraint" pointed out by the requirement. This provides jitter, upper, lower, and/or nominal values of the timing parameter at hand. The requirement text may suggest whether the requirement refers to age, reaction, rate, or synchrony.
2.6.2 Requirements modeling in EAST-ADL2

Requirements are the input information for any system design effort, and contain the constraints, demands, prescriptions and rules for the system and the system development process. To improve dependability and in particular safety, rigorous treatment of requirements is critical. Since the main modeling entities in the EAST-ADL2 are architectural, it is desirable to investigate requirements, and how they can be associated to the system modeling elements.

Requirements are often refined in several stages before they influence a concrete implementation. For tracing back to the original requirement, the relation DeriveReqt is used: the derived requirement points back to its source.

Information in the system model may represent both design choices with some degree of freedom, and strict requirements. The attribute ConstraintKind (Contract, Offered, Required) is used to explicitly state if a model entity is mandatory (required), a suggested property (offered) or agreed between stakeholders (contract).

Another aspect of information in the system model is that it may represent both properties and requirements, depending on the perspective. For example, a choice of control frequency is a property on the function from the control engineer’s perspective, but a requirement from the software engineer when the algorithm is turned into software.

2.6.3 Modeling of timing for functions

The Functional Analysis Architecture (FAA) of EAST-ADL2 is modeled with ADLFunctions that represent the vehicle functionality and are defined and structured according to the functional requirements, i.e., the functionalities provided to the user. A more optimized structure, from an implementation point of view, is defined in the lower abstraction level named the Functional Design Architecture.

The requirement support in EAST-ADL2 allows for tracing from solutions as modeled in the structural model to requirements, and from verification cases to requirements. The TADL constraints for timing modeling fit in this traceability support.

Further support in EAST-ADL2 includes the tracing between abstraction levels by realization of model elements in the highest abstraction layer. In the example given here the realization is between the Vehicle Feature in the Vehicle Feature Model and the functional description in the Functional Analysis Architecture.

In Figure 2.9 an example model is shown where a Vehicle Feature in the VFM has a requirement and the FAA has functions with connected ports. The Vehicle Feature is realized by the function occurrences in the lower abstraction level. The requirement on the Vehicle Feature is refined by a requirement satisfied by the FAA and is satisfied by the timing reaction constraint on the FAA functionality.
The relationships seen as dashed arrows in Figure 2.9 are defined in the meta-model as shown in Figure 2.10. Requirements may be refined by the ADLDeriveReqt relationship, and can be marked as satisfied by model elements by the ADLSatisfy relationship. The model elements in different abstraction levels are related and traceable by the ADLRelationship.

Figure 2.9. This is the example model.

Figure 2.10. This is an extract of the EAST-ADL2 meta-model, which shows the relationship and traceability support for model elements and requirements.

### 2.6.4 Modeling of timing in the design phase

A preliminary hardware architecture is defined using nodes (ECUs), bus segments, actuators, and sensors. Nodes can represent both CPUs and the electronics that interface peripherals.

The functional architecture from the analysis level is refined and traced by realization dependencies. This refined functional design architecture is eligible for allocation on the hardware architecture. Functional descriptions can also be supplied for the hardware.
elements as transfer functions. Time budgets in this phase are distributed over application functions, middleware functions and hardware elements.

Figure 2.11. An allocated function in the functional design architecture.
3 Syntax

The timing language, TADL, in Figure 3.1, is specified by semantics and syntax as described in the next chapters. The syntax is the input for the tool chain as described in the next section.

Figure 3.1. The formalization of the semantics is performed in Enterprise Architect where the syntax is defined, which specifies the exchange format.

This section gives an overview of the concepts defined in the TIMMO meta-model, which specifies the timing exchange format. See Annex C Specification of the model TADL XML exchange.

The syntax has been organized in the following packages. See also the Annex A Meta-modeling for a description on how the meta-model complies to the AUTOSAR meta-modeling guidelines.

- TIMMO
  - TADL-Infrastructure
  - TADL
  - TimingConstraints
  - EAST-ADL2 Timing
  - EAST-ADL2 Events

3.1 Structural framework

EAST-ADL2 and AUTOSAR provide the modeling concepts for structural and functional descriptions on which the Timing Augmented Description Language of TIMMO is based. EAST-ADL2 defines five abstraction levels called Vehicle, Analysis, Design, Implementation, and Operational level. On each abstraction level, a complete representation of the embedded system is defined. The content on each level is increasingly concrete and solution-oriented from vehicle level and down.

The model on implementation level uses the AUTOSAR constructs to define a software architecture and a system topology. AUTOSAR defines views on the model, the VFB view where allocation is
disregarded, the System view where the software architecture is allocated to a hardware architecture, but the model elements remain the same.

3.2 Events and event chains

On the higher abstraction levels (vehicle, analysis, and design) structural modeling is performed as defined within EAST-ADL. On the implementation level AUTOSAR modeling is performed. The TADL constraints are defined separately from the structural modeling and refer to structural elements via Event Chains and Events, see Figure 3.2. The events are tailored to refer to specific elements in the structural model, i.e., different sets of events are available for the EAST-ADL model and the AUTOSAR model.

![Diagram showing relationships between elements for modeling of constraints, event chains, events and structural elements.]

Additionally all TADL constraints carry a timing bound and a mode. Figure 3.3 shows an example of some of the timing constructs.

![Diagram showing a conceptual design modeled in the tree view.]

The requirement support in EAST-ADL allows for tracing from solutions as modeled in the structural model to requirements, and from verification cases to requirements. The TADL constraints fit in the requirement support as refinements of the requirements.
Further support in EAST-ADL includes the tracing between abstraction levels by realization between, e.g., a Vehicle Feature in the highest abstraction layer to functional descriptions in lower layers, and further down to software components and Runnables as found on the AUTOSAR level. Tracing between requirements is done by the SysML concept DeriveReqt. Figure 3.4. shows an example representing some of these concepts.

3.3 Age and reaction timing constraints

Multirate systems, which are common in automotive applications, have over- and under-sampling effects, which an end-to-end delay semantics has to consider. Very often register based communication is used where message loss (under sampling) or duplication (oversampling) occurs. TIMMO has identified two end-to-end delay semantics, which are interesting for real world situations. These delays are the reaction Reaction Delay and age Age Delay.

The reaction delay is utilized when the first reaction to a stimulus is of importance. This is usually the case in body electronics, but may also be the case in other domains. One example is the time it takes from a button is pressed to the light is switched on. Another example, from the chassis domain, is the time from the brake pedal is pressed until the brakes are activated. In both these cases the constrained time of importance is the delay from a given stimulus to the first corresponding response.

The age of data is mainly important in control engineering, but may appear in all domains. Here the focus is from the response perspective rather than from the stimulus perspective. In other words, the assumption is that last is best, i.e., it is accepted/tolerated that a value is overwritten along the path from stimulus to response. When for example an actuator value is periodically updated, it is of importance that the corresponding input values are not too old. It is...
not a problem if some stimuli are lost as long as the ones reaching the response are not too old.

The delay from any of these semantics can be composed of segments of the same kind (reaction or age) throughout the complete system and might span multiple ECUs and buses. This is part of the time budgeting and decomposition as described in the TIMMO methodology.

Given these semantics and the corresponding TADL elements, it is possible to describe and verify end-to-end delay constraints throughout the complete supplier and car manufacturer chain. Furthermore there is the possibility to explicitly define the kind of delay, which will reduce misunderstandings even before implementation or design stage.

3.4 Input and output synchronization constraints

Constraining timing is not always about delays between stimuli and responses. An important class of constraints deals with the synchronization between either stimuli or responses, respectively. One example is a central lock function. The ReactionConstraint between button pressed (stimuli) and door locked (response) could typically have a span between fastest and slowest reaction of several hundreds of milliseconds. However, the tolerated difference between when the different doors are locked is perhaps just some tenths of milliseconds. This difference between a number of responses related to the same stimuli, is expressed in TADL by an additional attribute to a dedicated specialization to the class ReactionConstraint called OutputSynchronizationConstraint. Figure 3.5 depicts a central lock system where the locks of the four doors are controlled by the remote control in the key. The locking of one door is defined as an event each. The OutputSynchronizationConstraint constrains the maximum difference in time between any of these four events. Note that this range can be much less than the tolerated range in the ReactionConstraint between the stimulus event defined as pressing the key and any of these four response events defined as locking of a door.

In a similar way, there is also a constraint called InputSynchronizationConstraint in the TADL specializing the AgeConstraint. This is to constrain the maximum difference between numbers of stimuli events, e.g., sampling of sensors.
3.5 **Jitter as a property of constraints**

The occurrence of events, for example the change of a state, receipt of data across a network, or the start of a task, may be jittery (sporadic) meaning that the point in time such event occurs is in a time range. The time range is given by a lower and upper bound. Only the range is known but not the actual time value—in the sense of point in time.

Jitter is the deviation from a given [ideal] point in time an event is expected to occur.

![Diagram of Central Lock System](image)

Figure 3.5. Central lock system.

---

**Occurrence of Event**

- **Event occurs before the expected point in time.**
  - $t_{\text{expected}}$: Absolute Time Instant
  - $jitter$: Value of time duration/interval

- **Event occurs after the expected point in time.**
  - $t_{\text{expected}}$: Absolute Time Instant
  - $jitter$: Value of time duration/interval

---

**Interval**

- Time duration/interval around the expected point in time ($t_{\text{expected}}$) during which an event occurs. The occurrence of the event is jittery, i.e., its occurrence varies in time. The start time instant ($t_{\text{start}}$) of the duration is relative to the expected point in time.

---

Figure 3.6. Jitter
3.6 Event Constraints

An event constraint describes the basic characteristics of the way an event occurs over time. The categories listed below are specified and each of these event models is described in one of the subsequent sections:

- Periodic Event Constraint
- Sporadic Event Constraint
- [Concrete] Pattern Event Constraint
- Burst Event Constraint
- Arbitrary Event Constraint

Offset

In addition an event constraint may specify an offset, which delays the start of the first period—the occurrence of the very first event—by the given amount of time. This is shown in Figure 3.8 where the first period starts after the offset.

Figure 3.7. Periodic Event Occurrence and Jitter

Figure 3.8. The Event Constraint and Offset
3.6.1 Periodic Event Constraint

The Periodic Event Constraint is characterized by the following parameters:

1. The period specifies the [ideal] time interval between two subsequent occurrences of the event.
2. The minimum inter-arrival time specifies the minimal possible time interval between two consecutive occurrences of an event.
3. The jitter specifies the maximal possible time interval the occurrence of an event can vary (formerly: be delayed).

The parameters described above are shown in Figure 3.10 below.

Figure 3.10. The Periodic Event Constraint

3.6.2 Sporadic Event Constraint

The Sporadic Event Constraint describes that an event occurs occasionally. In general it is supposed that the event eventually occurs. Indeed, it is also known that some of the events do not occur
for whatsoever reasons. The sporadic event constraint is characterized by the following parameters:

1. The period specifies the [ideal] time interval between two subsequent occurrences of the event.
2. The minimum inter-arrival time specifies the minimal possible time interval between two consecutive occurrences of an event.
3. The optional parameter maximum inter-arrival time specifies the maximal possible time interval between two consecutive occurrences of an event.
4. The optional parameter jitter specifies the maximal possible time interval the occurrence of an event can vary (formerly: be delayed). By nature, a sporadic event is supposed to occur at any time, thus it is one of the characteristic that the occurrence is jittery.

Note! The parameters minimum inter-arrival time and maximum inter-arrival time must reference the same point in time. Typically, this is the point in time that specifies the beginning of the period subject to consideration.

The parameters described above are shown in Figure 3.11 below.

Figure 3.11. The Sporadic (Aperiodic) Event Constraint

As shown in the figure above the maximum inter-arrival time may be used to describe different cases:

- The maximum inter-arrival time is equal to the duration of the period. (1)
- The maximum inter-arrival time is used to specify a point in time within the period that immediately follows the period subject to consideration. (2)
- The maximum inter-arrival time is used to specify a point in time within one of the subsequent periods that follow the period subject to consideration. (3)
3.6.3 [Concrete] Pattern Event Constraint

The [Concrete] Pattern Event Constraint describes that an event occurs following a known pattern. The pattern event constraint is characterized by the following parameters:

- The period specifies the time interval within the event occurs any number of times following a pattern.
- The minimum inter-arrival time specifies the minimal possible time interval between two consecutive occurrences of the event within the given period.
- The jitter specifies maximal possible time interval the occurrence of the events within the given period can vary (formerly: can be delayed).
- The set occurrence[1..n] specifies the offset for each occurrence of the event in the specified period. Each occurrence is specified from the beginning of the period.

The parameters described above are shown in Figure 3.12 and Figure 3.13 below.

**Observation**

The parameters described above do not provide the means to describe a specific pattern of event occurrences. Using this
information one can calculate the maximum number of event occurrences using the period and the minimum inter-arrival time. Assumed the period is 20 ms and the minimum inter-arrival time is 1.25ms than the possible maximum number of event occurrences within the given period may be 16.

In order to specify a specific pattern, one must define the occurrences of the event within the period. As shown in Figure 3.9 the pattern event constraint aggregates an order list of event occurrences—the points in time the event occurs at. For example, assumed the period is 20ms, the minimum inter-arrival time is 1ms, and the jitter is 0.5ms. The event occurs at the following points in time during the period: 0, 3, 4, 5, 6, 10, 13, 14, 15, and 16ms. This means the event occurs ten times.

Note that the each point in time is a relative time instant referring to the point in time the period begins which is 0ms. During simulation one wants to analyze the system's dynamic behavior and lets repeat the specified pattern over time. This means that the occurrence of the first pattern starts at 0ms, the occurrence of the second pattern at 20ms, the occurrence of the third pattern start at 40ms, and so forth. The particular events occur at 0, 3, 4, 5, 6, 10, 13, 14, 15, 16, 20, 23, 24, 25, 26, 30, 33, 34, 35, 36, 40ms, and so forth.

3.6.4 Arbitrary Event Constraint

The Arbitrary Event Constraint describes that an event occurs occasionally, singly, irregular or randomly. The primary purpose of this event constraint is to abstract event occurrences captured by data acquisition tools (background debugger, trace analyzer, etc.) during the operation of a system. The arbitrary event constraint is characterized by the following parameters:

- The set of minimum inter-arrival times specifies the minimum inter-arrival time between two and more subsequent occurrences of the event. The first element in the set specifies the minimum inter-arrival time between two subsequent occurrences of the event among the given occurrences. The second element in the set specifies the minimum inter-arrival time between three subsequent occurrences of the event among the given occurrences; and so forth.

- The set of maximum inter-arrival times specifies the maximum inter-arrival time between two and more subsequent occurrences of the event. The first element in the set specifies the maximum inter-arrival time between two subsequent occurrences of the event among the given occurrences. The second element in the set specifies the maximum inter-arrival time between three subsequent occurrences of the event among the given occurrences; and so forth.

Note! The number of elements in the sets minimum inter-arrival time and maximum inter-arrival time must be the same.

The parameters described above are shown in Figure 3.14 below.
Figure 3.14. The Arbitrary Event Constraint

Legend

\( \Delta n \) = Difference between \( n \) subsequent occurrences of the event
4 Semantics

In this chapter we define the semantics of the different TADL constraints on events and event chains. By the semantics of a constraint we mean the unambiguous interpretation of the conditions that must hold for the constraint to be satisfied.

Because TADL constraints are allowed to refer to arbitrary EAST-ADL or AUTOSAR structural models, a clear separation needs to be made between the semantics of these constraints and the semantics of the structural models. While the former issue is the core subject of interest in this report, the latter issue is inherently complex and goes far beyond the scope of the TIMMO project. The issues are nevertheless related, in the sense that the overall goal of TADL is to allow the timing behavior of a structural model to be predicted and checked against the TADL constraints by static analysis tools. However, it is only the prediction part of such analyses that requires knowledge of EAST-ADL/AUTOSAR semantics; once a predicted behavior exists, the constraint checking part should be possible to carry out without reference to anything but the TADL constraint semantics.

As is evident from the TADL syntax, the natural border of responsibilities in this respect follows the definition of events. We therefore begin this chapter with a section on the semantics of events, before we move on to the different constraints and their interpretations. Then, at the end of the chapter, the concept of delay constraint composition and decomposition is discussed.

A note on notation: Syntactic objects like events, constraints and time will be referenced by simple variable names in this chapter; for example an event e, a constraint c, or a time value t. To denote attributes and associations of such an object we use a uniform functional notation, where for example jitter(c) means the attribute jitter of object c, and event(c) denotes the object reached by following association event from object c. When sequences or other groups of objects are referenced we write expressions like <t1, t2, t3> or <t1, ..., t,n> for a sequence of known length, and <t, ...> for a sequence of unspecified length but beginning with t. We sometimes also need to express intervals of values, for which we use the [t1 .. t2] notation.

4.1 Events

An event is supposed to denote a distinct form of state change in a running system, taking place at distinct points in time called occurrences 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.

To this end, the semantics of TADL only concerns the identity of events; that is, the particular form of state change a distinct event
refers to. The actual dynamics of an event is merely an assumption — for a particular event the semantics cannot be more specific than to assume it occurs / have occurred / is predicted to occur at some unknown times \( t_1, t_2, \ldots \). Instead, the role of the semantics is to show how these assumptions must relate to each other for a particular set of timing constraints to be satisfied.

**Identity:** We define an semantic function \( identity() \) on events, that associates every event with an identifying list of elements from the underlying structural model.

For an \( \text{EventADLClientPort} \) event \( e \):

\[
identity(e) = \langle \text{eventKind}(e), \text{clientPort}(e) \rangle
\]

For an \( \text{EventADLFunction} \) event \( e \):

\[
identity(e) = \langle \text{function}(e) \rangle
\]

For an \( \text{EventADLInFlowPort} \) event \( e \):

\[
identity(e) = \langle \text{inFlowPort}(e) \rangle
\]

For an \( \text{EventADLInOutFlowPort} \) event \( e \):

\[
identity(e) = \langle \text{inOutFlowPort}(e) \rangle
\]

For an \( \text{EventADLOutFlowPort} \) event \( e \):

\[
identity(e) = \langle \text{outFlowPort}(e) \rangle
\]

For an \( \text{EventADLServerPort} \) event \( e \):

\[
identity(e) = \langle \text{eventKind}(e), \text{serverPort}(e) \rangle
\]

For an \( \text{EventDataElement} \) event \( e \):

\[
identity(e) = \langle \text{eventDataElementType}(e), \text{dataElement}(e) \rangle
\]

For an \( \text{EventOperation} \) event \( e \):

\[
identity(e) = \langle \text{eventOperationType}(e), \text{operation}(e) \rangle
\]

For an \( \text{EventInternalBehavior} \) event \( e \):

\[
identity(e) = \langle \text{eventInternalBehaviorType}(e), \text{runnable}(e), \text{component}(e) \rangle
\]

For an \( \text{EventFrame} \) event \( e \):

\[
identity(e) = \langle \text{eventType}(e), \text{frame}(e), \text{physicalChannel}(e) \rangle
\]

For an \( \text{EventFrClusterCycleStart} \) event \( e \):

\[
identity(e) = \langle \text{frCluster}(e) \rangle
\]

For an \( \text{EventPdu} \) event \( e \):
For an EventISignal event $e$:

\[
\text{identity}(e) = \langle \text{eventType}(e), \text{physicalChannel}(e), \text{iPdu}(e) \rangle
\]

For an EventBsw event $e$:

\[
\text{identity}(e) = \langle \text{tdEventBswType}(e), \text{bswModuleEntry}(e), \text{bswModuleDescription}(e) \rangle
\]

For an EventBswBehavior event $e$:

\[
\text{identity}(e) = \langle \text{eventBswBehaviorType}(e), \text{bswModuleEntity}(e) \rangle
\]

Equality: Two events $e_1$ and $e_2$ are equal ($e_1 = e_2$) if and only if

\[
\text{identity}(e_1) = \text{identity}(e_2)
\]

That is, two events are equal only if they are identified by exactly the same list of structural elements; it is generally not enough that they have just some structural elements in common.

Dynamics: We then capture the assumed dynamic behavior of structural elements by means of a semantic function $\text{dynamics}()$, which associates every list of structural elements that is an event identity according to the definition above, with a sequence of time values. That is, if $\text{identity}(e) = s$ for some event $e$, then we assume

\[
\text{dynamics}(s) = \langle t_1, \ldots \rangle
\]

where $\langle t_1, \ldots \rangle$ is a sequence of strictly increasing time values.

Finally we define the short-hand notation $\text{dynamics}(e)$ for an event $e$ as

\[
\text{dynamics}(e) = \text{dynamics}(\text{identity}(e))
\]

In the sections that follow, we will use this short-hand throughout.

4.2 Repetition rate constraints

All repetition rate constraints place restrictions on the distribution of the occurrences of an event, forming a so called event model. TADL includes the two most common examples of such models: the periodic event, which prescribes an even distribution of occurrences, and the sporadic model, whose distribution in non-deterministic but characterized by a minimum distance between occurrences (thus reducing to the periodic model in the worst case). To these TADL also adds two more elaborate forms: a pattern model defined by a recurring sequence of offsets between occurrences, and a model termed arbitrary because of its ability to put arbitrary statistical restrictions on event distributions.
However, the underlying TADL foundation for these event models is the *generic* repetition rate constraint, which allows the other forms of constraints to be defined as special cases — either by making certain attribute choices, or by combining constraints in a certain way. We will therefore begin by giving the semantics for this foundational constraint, before we show how it is applied to define the semantics of the other event models in subsequent sections.

**Syntax:** A generic repetition rate constraint \( c \) is syntactically characterized by the following attributes and associations:

\[
\begin{align*}
\text{event}(c) & \quad \text{the event that is constrained} \\
\text{lower}(c) & \quad \text{a lower bound on the distance between occurrences} \\
\text{upper}(c) & \quad \text{an upper bound on the distance between occurrences} \\
\text{jitter}(c) & \quad \text{the deviation from an ideal point accepted at each occurrence} \\
\text{span}(c) & \quad \text{a count indicating whether it is subsequent occurrences or occurrences farther apart that are constrained}
\end{align*}
\]

All attributes except for \( \text{event} \) are optional, where absence of an attribute means that the following default values apply:

- \( \text{lower} \): default: 0
- \( \text{upper} \): default: infinity
- \( \text{jitter} \): default: 0
- \( \text{span} \): default: 1

**Semantics:** A generic repetition constraint \( c \) is satisfied for some given event behavior \( \text{dynamics()} \) if and only if

- given \( \langle t_1, t_2, \ldots \rangle = \text{dynamics(\text{event}(c))} \)
- there are times \( \langle x_1, x_2, \ldots \rangle \) such that
  - for all \( i > 1 \),
    \[ x_i \leq t_i \leq x_i + \text{jitter}(c) \]
  - and for all \( i \geq \text{span}(c) \),
    \[ \text{lower}(c) \leq x_i - x_{i-\text{span}(c)} \leq \text{upper}(c) \]

That is, a generic repetition rate constraint is satisfied if there exists *ideal* points in time not too far from the actual event occurrences, that are also sufficiently close (distant) according to the *upper* (lower) constraint attributes. Note especially that for the default *span* value of 1, it is any two *subsequent* occurrences of an event that are constrained, whereas for a higher *span* count, the constraint instead specifies the allowed ranges containing a larger number of occurrences.

**Verification procedure:** To decide whether a particular event behavior \( \text{dynamics()} \) satisfies a generic repetition rate constraint \( c \), execute the following pseudo-code program:
let \(<t_1, t_2, \ldots> = \text{dynamics}(\text{event}(c))

\text{for all } 1 \leq i \leq \text{span}(c)

let a_i = t_i - \text{jitter}(c)

let b_i = t_i

\text{for all } i > \text{span}(c)

let a_i = \max(t_i - \text{jitter}(c), a_{i-\text{span}(c)} + \text{lower}(c))

let b_i = \min(t_i, b_{i-\text{span}(c)} + \text{upper}(c))

\text{check that } a_i \leq b_i

Intuitively, the \(a_i\) and \(b_i\) here constitute increasingly tight bounds of the ranges in which the unknown time points \(x_i\) must be found, taking more and more event history into account as \(i\) increases. The algorithm fails as soon as it discovers that a range is empty, in which case there cannot exist any \(x_i\) as required by the semantics. Conversely, if the algorithm succeeds, the ideal times \(x_i\) are known to exist.

4.2.1 Periodic repetition

\textbf{Syntax:} A periodic repetition constraint \(c\) is syntactically characterized by the following attributes and associations:

- \(\text{event}(c)\): the event that is constrained
- \(\text{period}(c)\): the recurring distance between each occurrence
- \(\text{jitter}(c)\): the deviation from an ideal point accepted at each occurrence
- \(\text{minimumInterArrivalTime}(c)\): an optional minimum distance between subsequent occurrences

\textbf{Semantics:} A periodic repetition constraint \(c\) is satisfied if and only if a generic repetition rate constraint \(d\) of the following definition is satisfied:

- \(\text{event}(d) = \text{event}(c)\)
- \(\text{lower}(d) = \text{period}(c)\)
- \(\text{upper}(d) = \text{period}(c)\)
- \(\text{jitter}(d) = \text{jitter}(c)\)
- \(\text{span}(d) = 1\)

If \(\text{minimumInterArrivalTime}(c)\) is defined, the corresponding TADL interpretation is a secondary generic repetition rate constraint \(d_2\) on \(\text{event}(c)\), defined as follows:

- \(\text{event}(d_2) = \text{event}(c)\)
- \(\text{lower}(d_2) = \text{minimumInterArrivalTime}(c)\)
- \(\text{upper}(d_2) = \text{absent}\)
- \(\text{jitter}(d_2) = 0\)
- \(\text{span}(d_2) = 1\)
Example: Consider a periodic constraint $c$ where
\[
\begin{align*}
event(c) & = A \\
period(c) & = 5 \\
\text{jitter}(c) & = 2
\end{align*}
\]
According to the definition above, this constraint is equivalent to a generic repetition rate constraint $d$ defined as follows:
\[
\begin{align*}
event(d) & = A \\
\text{lower}(d) & = 5 \\
\text{upper}(d) & = 5 \\
\text{jitter}(d) & = 2 \\
\text{span}(d) & = 1
\end{align*}
\]
Assume that the actual occurrences of event $A$ is given as follows:
\[
dynamics(A) = <t_1, t_2, t_3, t_4> = <1, 7, 12, 15.5>
\]
For $c$ to be satisfied, $d$ must be satisfied, which in turn requires the existence of ideal times $<x_1, x_2, x_3, x_4>$ placed exactly 5 time units apart and no more than 2 units below the corresponding $t_i$. Referring to the verification procedure, the increasingly tight ranges $[a_i .. b_i]$ become:
\[
\begin{align*}
[a_1 .. b_1] &= [-1 .. 1] \\
[a_2 .. b_2] &= [5 .. 6] \\
[a_3 .. b_3] &= [10 .. 11] \\
[a_4 .. b_4] &= [15 .. 15.5]
\end{align*}
\]
Since none of these ranges are empty, verification succeeds. One example of a sequence of ideal times $x_i$ that satisfy the constraint semantics is $<0, 5, 10, 15>$, another one is $<0.5, 5.5, 10.5, 15.5>$. Figure 4.1 depicts both these cases.

![Figure 4.1 Four actual occurrences of event A, and the existence of corresponding four ranges in which the ideal occurrences of the event A exist. Two example ideal sequences are shown.](image_url)

In figure 4.1, the four occurrences of event A is denoted by vertical line at time 1, 7, 12 and 15.5. The ideal occurrences of the event is
predicted to occur within the range \([a_i .. b_i]\) denoted by arrow heads in which the four occurrences of the event is expected to occur. Since none of the ranges is nonempty, at least one ideal sequence exists. Two examples of ideal event occurrences is also shown in figure 4.1.

For contrast, changing the jitter attribute to 1 would render the constraint unsatisfiable for the assumed event dynamics, as illustrated by the sequence of ranges then computed by the verification procedure:

\[
\begin{align*}
[a_1 .. b_1] &= [0 .. 1] \\
[a_2 .. b_2] &= [6 .. 6] \\
[a_4 .. b_4] &= [16 .. 15.5] \quad \text{(empty!)}
\end{align*}
\]

Figure 4.2 shows this scenario.

![Figure 4.2](image)

In Figure 4.2, the fourth occurrence of event A does not have a valid range in which the forth ideal occurrence of the event could occur. Therefore, the periodic constraint is not satisfied. Note that shifting the windows of allowed event occurrences in either direction cannot make them cover all times in \(\text{dynamics}(A)\); i.e., there are no ideal times \(x_i\) placed 5 units apart that are at most 1 unit below the actual times \(t_i\).

### 4.2.2 Sporadic repetition

**Syntax:** A sporadic repetition constraint \(c\) is syntactically characterized by the following attributes and associations:

- \(\text{event}(c)\) the event that is constrained
- \(\text{period}(c)\) the ideal distance between each occurrence
- \(\text{jitter}(c)\) the deviation from an ideal point accepted at each occurrence
- \(\text{minimumInterArrivalTime}(c)\) an optional minimum distance between subsequent occurrences
- \(\text{maximumInterArrivalTime}(c)\) an optional maximum distance between subsequent occurrences
**Semantics**: A sporadic repetition constraint $c$ is satisfied if and only if a generic repetition rate constraint $d$ of the following definition is satisfied:

- $\text{event}(d) = \text{event}(c)$
- $\text{lower}(d) = \text{period}(c)$
- $\text{upper}(d) = \text{absent}$
- $\text{jitter}(d) = \text{jitter}(c)$
- $\text{span}(d) = 1$

That is, unlike a periodic event, a sporadic event can allow any distance between the ideal occurrences as long as it is no less than the lower constraint attribute. Obviously, a sporadic constraint reduces to a periodic one in the worst case (i.e., when the distances are as short as possible).

If $\text{minimumInterArrivalTime}(c)$ or $\text{maximumInterArrivalTime}(c)$ are defined, the corresponding TADL interpretation involves a secondary generic repetition rate constraint $d_2$ on $\text{event}(c)$, defined as follows:

- $\text{event}(d_2) = \text{event}(c)$
- $\text{lower}(d_2) = \text{minimumInterArrivalTime}(c)$
- $\text{upper}(d_2) = \text{maximumInterArrivalTime}(c)$
- $\text{jitter}(d_2) = 0$
- $\text{span}(d_2) = 1$

**Example**: Consider a sporadic constraint $c$ where

- $\text{event}(c) = A$
- $\text{period}(c) = 5$
- $\text{jitter}(c) = 2$

According to the definition above, this constraint is equivalent to a generic repetition rate constraint $d$ defined as follows:

- $\text{event}(d) = A$
- $\text{lower}(d) = 5$
- $\text{upper}(d) = \text{absent}$
- $\text{jitter}(d) = 2$
- $\text{span}(d) = 1$

Assume that the actual occurrences of event $A$ is given as follows:

$$\text{dynamics}(A) = <t_1, t_2, t_3, t_4> = <1, 9, 12, 18>$$

For $c$ to be satisfied, $d$ must be satisfied, which in turn requires the existence of ideal times $<x_1, x_2, x_3, x_4>$ placed at least 5 time units apart and no more than 2 units below the corresponding $t_i$. Referring to the verification procedure, the increasingly tight ranges $[a_i .. b_i]$ become:

- $[a_1 .. b_1] = [-1 .. 1]$
- $[a_2 .. b_2] = [7 .. 9]$
- $[a_3 .. b_3] = [12 .. 12]$
Since none of these ranges are empty, verification succeeds. One example of a sequence of ideal times \( x_i \) that satisfy the constraint semantics is \( <0, 7, 12, 17> \), another one is \( <1, 7, 12, 18> \). Figure 4.3 depicts both these cases.

\[
[a_4 .. b_4] = [17 .. 18]
\]

Figure 4.3 Four actual occurrences of event A, and the existence of corresponding four ranges in which the ideal occurrences of the event A exist.

In figure 4.3, the four occurrences of event A is denoted by vertical line at time 1, 9, 12 and 18. The ideal occurrences of the event is predicted to occur within the range \([a_i .. b_i]\) denoted by arrow heads in which the four occurrences of the sporadic event is expected to occur. Since none of the ranges is nonempty, at least one ideal sequence exists.

Note that the allowance for jitter makes it possible for two subsequent event occurrences to appear closer together than what is indicated by the lower constraint attribute; more precisely, the minimum distance possible between any two occurrences of an event constrained by a sporadic constraint \( c \) is \( \text{lower}(c) - \text{jitter}(c) \). In our example, the distance between \( t_2 \) and \( t_3 \) is only 3, although the lower attribute is 5.

However, such a deviation can only be local, as the distance between the windows of allowed occurrences must be at least what is required by the lower attribute. For example, the dynamic behavior \( <1, 9, 12, 15> \) for event A is not accepted by the above constraint, as the verification procedure then will encounter an empty range at \( i = 4 \).

\[
[a_1 .. b_1] = [-1 .. 1]
[a_2 .. b_2] = [7 .. 9]
[a_3 .. b_3] = [12 .. 12]
[a_4 .. b_4] = [17 .. 15] \quad \text{(empty!)}
\]

Figure 4.4 illustrates this failure.
Figure 4.4 Four actual occurrences of event A, and the non-existence of corresponding forth range (no range exist for the real occurrence inside the circle).

In Figure 4.4, the forth occurrence of event A does not have a valid range in which the forth ideal occurrence of the event could occur. Therefore, the sporadic constraint is not satisfied.

### 4.2.3 Pattern repetition

**Syntax:** A pattern repetition constraint $c$ is syntactically characterized by the following attributes and associations:

- $\text{event}(c)$: the event that is constrained
- $\text{period}(c)$: the distance between each repetition of the recurring pattern
- $\text{jitter}(c)$: the deviation from an ideal point accepted at each occurrence
- $\text{minimumInterArrivalTime}(c)$: an optional minimum distance between subsequent occurrences
- $\text{occurrence}(c)$: a sequence of offsets, one for each successive occurrence in the recurring pattern, measured from the start of each period

**Semantics:** A pattern repetition constraint $c$ is satisfied if and only if a generic repetition rate constraint $d_0$ and reaction time constraints $d_1, \ldots, d_n$ of the following definition are simultaneously satisfied:

- $\text{event}(d_0) = e$
- $\text{lower}(d_0) = \text{period}(c)$
- $\text{upper}(d_0) = \text{period}(c)$
- $\text{jitter}(d_0) = 0$
- $\text{stimulus}(d_i) = <e>$ for $1 \leq i \leq n$
- $\text{response}(d_i) = <\text{event}(c)>$ for $1 \leq i \leq n$
- $\text{lower}(d_i) = t_i - \text{jitter}(c)/2$ for $1 \leq i \leq n$
- $\text{upper}(d_i) = t_i + \text{jitter}(c)/2$ for $1 \leq i \leq n$

assuming $e$ is some reference event not equal to $\text{event}(c)$, and $\text{occurrence}(c) = <t_1, \ldots, t_n>$. That is, a pattern repetition constraint on an event $A$ is semantically reduced to an ordinary periodic constraint on some reference event $B$, combined with multiple reaction time constraints, all using $B$ as stimulus and event $A$ as a response.
Example: a crankshaft generating an interrupt event A each time a tap on an attached wheel passes a sensor would be the source of a recurring event pattern, if the taps are unevenly distributed on the wheel perimeter. If the sensor wheel has 4 equal taps and one gap, the revolution period is 20 and the tap sensor is accurate up to 2 time unit, a pattern repetition constraint $c$ of the following definition would capture the interrupt behavior:

\[
\begin{align*}
\text{event}(c) & = A \\
\text{period}(c) & = 20 \\
\text{jitter}(c) & = 2 \\
\text{occurrence}(c) & = \langle 0, 4, 8, 12 \rangle
\end{align*}
\]

According to the semantics, this constraint is equivalent to the generic repetition rate constraint $d_0$ and the reaction time constraints $d_1, d_2, d_3$ and $d_4$, defined as follows:

\[
\begin{align*}
\text{event}(d_0) & = B \\
\text{lower}(d_0) & = 20 \\
\text{upper}(d_0) & = 20 \\
\text{jitter}(d_0) & = 0 \\
\text{span}(d_0) & = 1 \\
\text{stimulus}(d_1) & = \langle B \rangle \\
\text{response}(d_1) & = \langle A \rangle \\
\text{lower}(d_1) & = -1 \\
\text{upper}(d_1) & = 1 \\
\text{stimulus}(d_2) & = \langle B \rangle \\
\text{response}(d_2) & = \langle A \rangle \\
\text{lower}(d_2) & = 3 \\
\text{upper}(d_2) & = 5 \\
\text{stimulus}(d_3) & = \langle B \rangle \\
\text{response}(d_3) & = \langle A \rangle \\
\text{lower}(d_3) & = 7 \\
\text{upper}(d_3) & = 9 \\
\text{stimulus}(d_4) & = \langle B \rangle \\
\text{response}(d_4) & = \langle A \rangle \\
\text{lower}(d_4) & = 11 \\
\text{upper}(d_4) & = 13
\end{align*}
\]

where $B$ is some internal event distinct from $A$ that signals the completion of one full crankshaft revolution.
For further information on the semantics of reaction time constraints, see section 4.3.1.

### 4.2.4 Arbitrary repetition

**Syntax:** An arbitrary repetition constraint \( c \) is syntactically characterized by the following attributes and associations:

\[
\begin{align*}
\text{event}(c) & \quad \text{the event that is constrained} \\
\text{minimumInterArrivalTime}(c) & \quad \text{a sequence of minimum distances between every 2, 3, etc, event occurrences} \\
\text{maximumInterArrivalTime}(c) & \quad \text{an optional maximum distances between every 2, 3, etc, event occurrences}
\end{align*}
\]

**Semantics:** An arbitrary repetition constraint \( c \) is satisfied if and only if a set of generic repetition rate constraints \( d_1, ..., d_n \) of the following definition are simultaneously satisfied:

\[
\begin{align*}
\text{event}(d_i) & = e & & \text{for } 1 \leq i \leq n \\
\text{lower}(d_i) & = t_i & & \text{for } 1 \leq i \leq n \\
\text{upper}(d_i) & = u_i & & \text{for } 1 \leq i \leq n \\
\text{jitter}(d_i) & = 0 & & \text{for } 1 \leq i \leq n \\
\text{span}(d_i) & = i & & \text{for } 1 \leq i \leq n
\end{align*}
\]

assuming

\[
\begin{align*}
\text{minimumInterArrivalTime}(c) & = <t_1, ..., t_n> \\
\text{maximumInterArrivalTime}(c) & = <u_1, ..., u_n>
\end{align*}
\]

That is, an arbitrary repetition rate constraint is semantically reduced to a combination of generic repetition rate constraints with increasing \( \text{span} \) attributes. The first constraint defines the minimum and maximum distance between each pair of subsequent occurrences \((\text{span} = 1)\), the second constraint one constrains the length of any interval containing two occurrences \((\text{span} = 2)\), etc, up to the limit given by the number of distances in the original arbitrary repetition constraint.

**Example 1:** Consider the following event models:

\[
\begin{align*}
\text{event}(c) & = A \\
\text{minimumInterArrivalTime}(c) & = <2, 10, 20> \\
\text{maximumInterArrivalTime}(c) & = <50, 90, 100>
\end{align*}
\]

According to the semantics, this constraint is equivalent to the generic repetition rate constraints \( d_1, d_2 \) and \( d_3 \), defined as follows:

\[
\begin{align*}
\text{event}(d_1) & = A \\
\text{lower}(d_1) & = 2 \\
\text{upper}(d_1) & = 50 \\
\text{span}(d_1) & = 1
\end{align*}
\]
In principle, each constraint represents a pair of values from the arbitrary event model from Section 4.6.5.

**Example 2:** It is important to note that the different minimum and maximum distances of an arbitrary repetition constraint need not be in mutual agreement. For example, even though the minimum distance between two occurrences of some event A is 2, it may very well be the case that there may not be more than 3 of those in an interval of 10, and no fewer than 4 on an interval of 50. In this case, there are two constraints for the minimum distance and one for the maximum distance. However, we have to provide numbers also for the other values. Therefore, we use zero as the default for the minimum and infinity as the default for the maximum. Obviously, such numbers do not constrain anything.

Now, the behavior can be captured by the following arbitrary repetition constraint c:

\[
\begin{align*}
event(c) &= A \\
minimumInterArrivalTime(c) &= <2, 10, 0> \\
maximumInterArrivalTime(c) &= <\infty, \infty, 50>
\end{align*}
\]

According to the semantics, this constraint is equivalent to the generic repetition rate constraints \(d_1\), \(d_2\) and \(d_3\), defined as follows:

\[
\begin{align*}
event(d_1) &= A \\
lower(d_1) &= 2 \\
upper(d_1) &= \infty \\
span(d_1) &= 1 \\

lower(d_2) &= 10 \\
upper(d_2) &= \infty \\
span(d_2) &= 2 \\

lower(d_3) &= 0 \\
upper(d_3) &= 50
\end{align*}
\]
Note that without constraint $d_2$, an interval of 10 could fit up to 5 event occurrences, and without $d_3$, it would be ok for an interval of any length to contain no occurrences at all. However, taken together, $d_1$, $d_2$ and $d_3$ (or more succinctly, the equivalent arbitrary repetition rate constraint $c$) capture the desired event occurrence behavior at the desired level of precision.

4.3 Delay constraints

A delay constraint restricts the occurrences of one group of event to match the occurrences on another, for example by saying that for each occurrence of an input event A there must be an occurrence of output event B within a maximum of T time units. TADL introduces the concept of an event-chain for this purpose, which references two groups of events called stimulus and response. The intuition behind an event-chain is that each event in the stimulus group somehow causes, or at least affects the value of all events in the response group. However, since questions about causality and value influence clearly involve the semantics of the underlying structural model, this aspect of an event-chain is semantically outside the scope of TADL. Instead, TADL defines its delay constraint semantics solely in terms of the times at which the stimulus and response events occur, independently of whether there actually exists a causal connection between these events in the structural model.

There are two fundamentally different perspectives on delays expressible in TADL: (a) the reaction time constraint, which for each occurrence of a stimulus event requires a response occurrence at some constrained point in the future, and symmetrically the (b) age constraint, which looks at each response event occurrence and expects to find a stimulus occurrence at some constrained point in the history. Both these views are most easily understood when only a single stimulus event and a single response event are considered, although the full generalization to groups of events is intuitively just a replication of the same constraint to all pairwise combinations of an event in the stimulus group with an event from the response group.

Syntactically, a delay constraint $c$ is characterized by the following attributes and associations:

- **scope(c)** the constrained event chain
- **lower(c)** a time offset indicating the near edge of a time window
- **upper(c)** a time offset indicating the far edge of a time window

An event chain $E$, in turn, is characterized by

- **stimulus(E)** the group of events acting as stimuli
- **response(E)** the group of events acting as responses
- **segment(E)** a sequence of event-chains refining $e$
- **strand(E)** a group of parallel event-chains refining $e$

A discussion on the latter two associations will deferred until section 4.4. Moreover, the stimulus and response of an event-chain will only
be used when following a scope association from a delay constraint. We can therefore avoid the explicit mentioning of event-chains in our semantic definitions, by introducing the following notational shorthand of stimulus and response associations for a delay constraint $c$:

$$
\text{stimulus}(c) = \text{stimulus}(\text{scope}(c)) \\
\text{response}(c) = \text{response}(\text{scope}(c))
$$

The reaction time and age constraints are both specializations of a delay, adding no attributes of their own but being defined with two different (but very symmetrical) semantics. Sections 4.3.1 and 4.3.2 contain the semantics of these forms of constraints, respectively.

Moreover, TADL also defines a *synchronization constraint* as a delay constraint that adds the attribute

$$
\text{width}(c) \quad \text{a sliding window size}
$$

to those inherited from the delay constraint syntax. The intuition behind a synchronization constraint is that it strengthens the requirements of a delay to also impose a limit on how much the constrained occurrences can differ from one another — typically a range much more narrow than the window in which the individual events are expected to occur. Two specializations are defined by TADL: *output synchronization* strengthens a reaction constraint, and *input synchronization* is semantically a strengthening of an age constraint. The details of these constraint forms will be further described in sections 4.3.3 and 4.3.4.

### 4.3.1 ReactionConstraint

**Syntax:** A reaction time constraint $c$ is syntactically characterized by the following attributes and associations (all inherited from the delay constraint syntax, including notational shorthand):

- $\text{stimulus}(c)$ a group of events acting as stimuli
- $\text{response}(c)$ a group of events acting as responses
- $\text{lower}(c)$ a time offset indicating the near edge of a time window
- $\text{upper}(c)$ a time offset indicating the far edge of a time window

**Semantics:** A reaction time constraint $c$ is satisfied for some given event behavior $\text{dynamics}()$ if and only if

for all events $s$ in $\text{stimulus}(c)$,
for all times $t$ in $\text{dynamics}(s)$,
for all events $r$ in $\text{response}(c)$,
there exists at least one time $u$ in $\text{dynamics}(r)$ such that

$$
t + \text{lower}(c) \leq u \leq t + \text{upper}(c)
$$

**Verification procedure:** To decide whether a particular event behavior $\text{dynamics}()$ satisfies a reaction time constraint $c$, execute the following pseudo-code program:

for all events $s$ in $\text{stimulus}(c)$
for all times $t$ in $\text{dynamics}(s)$
let $T = [t + \text{lower}(c) .. t + \text{upper}(c)]$
for all events $r$ in $\text{response}(c)$
let $U = \text{dynamics}(r) \cap T$
check that $U = <u, ...>$

**Example:** Consider the reaction time constraint $c$ defined by

$\text{stimulus}(c) = <S>$
$\text{response}(c) = <R>$
$\text{lower}(c) = 3$
$\text{upper}(c) = 6$

Assume that the actual occurrences of events S and R are given as follows:

$\text{dynamics}(S) = <5, 12>$
$\text{dynamics}(R) = <1, 4, 9, 10, 14, 17>$

The actual occurrences of the events S and R are given below in Figure 4.5, together with the windows in which the response event R must occur.

![Figure 4.5: Actual occurrence of S and R and the window for reaction time constraint](image-url)

In this example, there is only one event in $\text{stimulus}(c)$ (namely event S) and one event in $\text{response}(c)$ (namely event R). In figure 4.5, for each occurrence of the event S, there is a *reference window* forward in time in which at least once the response event R must occur. There are two occurrences of the event S. For the first occurrence of the event S at time 5, the reference window is from $5 + \text{lower}(c) = 5 + 3 = 8$ to $5 + \text{upper}(c) = 5 + 6 = 11$. There are two occurrences of event R at time 9 and 10 that occur within the reference window that exists from time 8 to 11. For the second occurrence of the event S at time 12, the reference window is from $12 + \text{lower}(c) = 12 + 3 = 15$ to $12 + \text{upper}(c) = 12 + 6 = 18$. There is one occurrence of event R at time 17 that occurs within the reference window that exists from time 15 to 18. Since for all the occurrences of the stimulus event S, there is at least
one occurrence of response event $R$ in the corresponding reference window, the constraint is satisfied.

**Prohibiting undersampling:** A periodic event producing data which in the underlying structural model is consumed by another periodic event will give rise to data loss if the consumption period is greater than the period of the producer. Such a scenario is called undersampling, and is sometimes undesirable.

In TADL, undersampling can be prohibited by defining a reaction time constraint that lists the producer event as its sole stimulus and the consumer event as the single response. Constraining these events is perfectly reasonable in TADL, even though the consumer is a periodic event in its own right and not technically a response to the producer event in the causality sense. The reaction time constraint simply dictates that for each occurrence of the producer event there is at least one occurrence of the consumer event within the prescribed time window. Moreover, by making the length of this time window strictly smaller than the period of the producer, all reactions must be unique and undersampling thus becomes impossible.

Technically, for some periodic constraint $c$ and a distinct consumption event $e \neq \text{event}(c)$, undersampling can be prohibited by the definition of a reaction time constraint $d$ with

$$
\text{stimulus}(d) = \langle \text{event}(c) \rangle \\
\text{response}(d) = \langle e \rangle \\
\text{lower}(d) = \text{some } l \\
\text{upper}(d) = \text{some } u
$$

such that $u - l < \text{nominal}(c)$.

Note that the introduction of a reaction time constraint still allows the consumption event to occur more than once for each producer occurrence; that is, *oversampling* is still allowed after the addition of constraint $d$ above. For a method to prohibit oversampling as well, see the next section.

### 4.3.2 AgeTimingConstraint

**Syntax:** An age constraint $c$ is syntactically characterized by the following attributes and associations (all inherited from the delay constraint syntax, including notational shorthand):

- $\text{stimulus}(c)$ a group of events acting as stimuli
- $\text{response}(c)$ a group of events acting as responses
- $\text{lower}(c)$ a time offset indicating the near edge of a time window
- $\text{upper}(c)$ a time offset indicating the far edge of a time window

**Semantics:** An age constraint $c$ is satisfied for some given event behavior $\text{dynamics}(\cdot)$ if and only if

- for all events $r$ in $\text{response}(c)$,
- for all times $t$ in $\text{dynamics}(r)$,
- for all events $s$ in $\text{stimulus}(c)$,
there exists at least one time \( u \) in \( \text{dynamics}(s) \) such that
\[
t - \text{upper}(c) \leq u \leq t - \text{lower}(c)
\]

**Verification procedure:** To decide whether a particular event behavior \( \text{dynamics}() \) satisfies an age constraint \( c \), execute the following pseudo-code program:

for all events \( r \) in \( \text{response}(c) \)
for all times \( t \) in \( \text{dynamics}(r) \)
let \( T = [t - \text{upper}(c) .. t - \text{lower}(c)] \)
for all events \( s \) in \( \text{stimulus}(c) \)
let \( U = \text{dynamics}(s) \cap T \)
check that \( U = <u, ...> \)

**Example:** Consider the age constraint defined by
\[
\begin{align*}
\text{stimulus}(c) &= <S> \\
\text{response}(c) &= <R> \\
\text{lower}(c) &= 3 \\
\text{upper}(c) &= 5
\end{align*}
\]
Assume that the actual occurrences of the events S and R are as follows:
\[
\begin{align*}
\text{dynamics}(S) &= <1, 6, 8, 10, 16, 17, 19> \\
\text{dynamics}(R) &= <5, 12, 20>
\end{align*}
\]
The actual occurrences of the events S and R are given below in Figure 4.6, together with the windows in which the stimulus event S must occur.

![Figure 4.6: Actual occurrence of S and R and the reference window for age constraint](image)

In this example, there is only one event in \( \text{stimulus}(c) \) (namely event S) and one event in \( \text{response}(c) \) (namely event R). In figure 4.6, for each occurrence of the event R, there is a reference window backward in time in which at least once the stimulus event S must occur. There are three occurrences of the event R. For the first
occurrence of the event R at time 5, the reference window is from 5-
upper(c)=5-5=0 to 5-lower(c)=5-3=2. There is one occurrence of
event S at time 1 that occurs within the reference window that exists
from time 0 to 2. For the second occurrence of the event R at time 12,
the reference window is from 12-upper(c)=12-5=7 to 12-lower (c)=12-
3=9. There is one occurrence of event S at time 8 that occurs within
the reference window that exists from time 7 to 9. For the third
occurrence of the event R at time 20, the reference window is from
20-upper(c)=20-5=15 to 20-lower(c)=20-3=17. There are two
occurrences of event S at time 16 and 17 that occur within the
reference window that exists from time 15 to 17. Since for all the
occurrences of the response event R, there is at least one occurrence
of stimulus event S in the corresponding reference window, the
constraint is satisfied.

Prohibiting oversampling: A periodic event consuming data which
in the underlying structural model is produced by another periodic
event will lead to data duplication if the consumption period is smaller
than the period of the producer. Such a scenario is called
oversampling, and is sometimes undesirable.

In TADL, oversampling can be prohibited by defining an age
constraint that lists the producer event as its sole stimulus and the
consumer event as the single response. Constraining these events is
perfectly reasonable in TADL, even though the producer is a periodic
event in its own right and not technically a stimulus to the consumer
event in the causality sense. The age constraint simply dictates that
for each occurrence of the consumer event there is at least one
occurrence of the producer event within the prescribed history
window. Moreover, by making the length of this window strictly
smaller than the period of the consumer, all stimuli must be unique
and oversampling thus becomes impossible.

Technically, for some periodic constraint c and a distinct producer
event e ≠ event(c), oversampling can be prohibited by the definition of
an age constraint d with

\[
\begin{align*}
\text{stimulus}(d) & = <e> \\
\text{response}(d) & = <\text{event}(c)> \\
\text{lower}(d) & = \text{some } l \\
\text{upper}(d) & = \text{some } u
\end{align*}
\]

such that \( u - l < \text{nominal}(c) \).

Note that the introduction of an age constraint still allows the
production event to occur more than once for each consumer
occurrence; that is, undersampling is still allowed after the addition of
constraint d above. For a discussion on how to prohibit
undersampling as well, see the previous section.

### 4.3.3 Output synchronization

**Syntax:** An output synchronization constraint c is syntactically
characterized by the following attributes and associations (all
inherited from the synchronization constraint syntax, including
notational shorthand):

\[
\begin{align*}
\text{stimulus}(c) & \quad \text{a group of events acting as stimuli}
\end{align*}
\]
response\((c)\) a group of events acting as responses
lower\((c)\) a time offset indicating the near edge of a time window
upper\((c)\) a time offset indicating the far edge of a time window
width\((c)\) the size of a sliding time window

**Semantics:** An output synchronization constraint \(c\) is satisfied for some given event behavior \(dynamics()\) if and only if

for all events \(s\) in \(stimulus(c)\),
for all times \(t\) in \(dynamics(s)\),
there exists a time \(x\) such that
for all events \(r\) in \(response(c)\),
there exists exactly one \(u\) in \(dynamics(r)\) such that
\[
t + lower(c) \leq u \leq t + upper(c)
\]
and moreover, for this \(u\),
\[
x \leq u \leq x + width(c)
\]

**Verification procedure:** To decide whether a particular event behavior \(dynamics()\) satisfies an output synchronization constraint \(c\), execute the following pseudo-code program:

for all events \(s\) in \(stimulus(c)\)
for all times \(t\) in \(dynamics(s)\)
   let \(T = [t+lower(c) \ldots t+upper(c)]\)
for all events \(r\) in \(response(c)\)
   let \(U = dynamics(r) \cap T\)
   check that \(U = <u>\)
   let \(a_s = u - width(c)\)
   let \(b_s = u\)
   let \(a = \max(\text{all } a_s)\)
   let \(b = \min(\text{all } b_s)\)
   check that \(a \leq b\)

**Example:** Consider the output synchronization constraint defined by

\[
\begin{align*}
stimulus(c) &= <S> \\
response(c) &= <R_1, R_2, R_3> \\
lower(c) &= 3 \\
upper(c) &= 6 \\
width(c) &= 1
\end{align*}
\]

Assume that the actual occurrences of the events \(S\) and \(R\) are as follows:

\[
\begin{align*}
dynamics(S) &= <2.5, 12> \\
dynamics(R_1) &= <0.5, 2.5, 7.5, 10, 13, 16>
\end{align*}
\]
The actual occurrences of the events in the synchronization and reference group are given in Figure 4.7, together with the reference window for each occurrences of the event S.

Figure 4.7. Actual occurrences of the events, and the position of the reference and sliding window.

In this example, there is only one event in stimulus$(c)$ namely event S and three events in response$(c)$ namely event R1, R2 and R3. In figure 4.7, for each occurrence of the event S, there is a reference window forward in time in which exactly each one of the three response events must occur. For the stimulus event S occurred at time instant $t$, we must have a unique occurrence of each of the response events R1, R2 and R3 within $t+\text{lower}(c)$ and $t+\text{upper}(c)$. Moreover, the distance between any two of the three response events must not be separated by a distance greater than $\text{width}(c)=1$. There are two occurrences of the event S.

For the reference event S that occurs at time $t=2.5$, there are unique occurrences of the three response events between time instants $2.5+\text{lower}(c)=2.5+3=5.5$ and $2.5+\text{upper}(c)=2.5+6=8.5$. The time instant from 5.5 to 8.5 is the reference window for the first occurrence of event S. The events R1, R2 and R3 occurs uniquely within the reference window at time 7.5, 8, and 7.5, respectively (see the left-most oval in Figure 4.7). These three occurrences of the synchronization events are within a sliding window of length 1.

For the reference event S that occurs at time $t=12$, there are unique occurrences of the three response events between time instants $12+\text{lower}(c)=12+3=15$ and $12+\text{upper}(c)=12+6=18$. The time instant from 15 to 18 is the reference window for the second occurrence of event S. The events R1, R2 and R3 occur uniquely within reference
window at time 15.5, 16 and 15.5, respectively (see the right-most oval in Figure 4.7). These three occurrences of the synchronization events are within a sliding window of length 1.

Since for all the two occurrences of the stimulus event S, there are unique occurrences of all three response events within the sliding window that is within the reference window, the output synchronization constraint is satisfied.

### 4.3.4 Input synchronization

**Syntax:** An output synchronization constraint \( c \) is syntactically characterized by the following attributes and associations (all inherited from the synchronization constraint syntax, including notational shorthand):

- \( \text{stimulus}(c) \): a group of events acting as stimuli
- \( \text{response}(c) \): a group of events acting as responses
- \( \text{lower}(c) \): a time offset indicating the near edge of a time window
- \( \text{upper}(c) \): a time offset indicating the far edge of a time window
- \( \text{width}(c) \): the size of a sliding time window

**Semantics:** An input synchronization constraint \( c \) is satisfied for some given event behavior \( \text{dynamics}(r) \) if and only if

- for all events \( r \) in \( \text{response}(c) \)
- for all times \( t \) in \( \text{dynamics}(r) \)
- there exists a time \( x \) such that
- for all events \( s \) in \( \text{stimulus}(c) \)
- there exists exactly one \( u \) in \( \text{dynamics}(s) \) such that
  \[
  t - \text{upper}(c) \leq u \leq t - \text{lower}(c)
  \]
  and moreover, for this \( u \),
  \[
  x \leq u \leq x + \text{width}(c)
  \]

**Verification procedure:** To decide whether a particular event behavior \( \text{dynamics}(r) \) satisfies an input synchronization constraint \( c \), execute the following pseudo-code program:

- for all events \( r \) in \( \text{response}(c) \)
  - for all times \( t \) in \( \text{dynamics}(r) \)
    - let \( T = [t - \text{upper}(c),.., t - \text{lower}(c)] \)
    - for all events \( s \) in \( \text{stimulus}(c) \)
      - let \( U = \text{dynamics}(s) \cap T \)
      - check that \( U = \{u\} \)
      - let \( a_u = u - \text{width}(c) \)
      - let \( b_u = u \)
      - let \( a = \max(\{a_u\}) \)
let $b = \min(\text{all } b_i)$
check that $a \leq b$

**Example:** Consider the input synchronization constraint defined by

- $\text{stimulus}(c) = \langle S_1, S_2, S_3 \rangle$
- $\text{response}(c) = \langle R \rangle$
- $\text{lower}(c) = 3$
- $\text{upper}(c) = 5$
- $\text{width}(c) = 1$

Assume that the actual occurrences of the events S and R are as follows:

- $\text{dynamics}(S_1) = \langle 1, 3, 7.5, 10, 12, 14.5, 16, 19.5 \rangle$
- $\text{dynamics}(S_2) = \langle 1.5, 4, 8, 11, 13.5, 15.5, 18, 20 \rangle$
- $\text{dynamics}(S_3) = \langle 0.5, 2.5, 7.5, 10.5, 11.5, 14, 16 \rangle$
- $\text{dynamics}(R) = \langle 5, 12, 20 \rangle$

The actual occurrences of the events in the synchronization and reference groups are given in Figure 4.8, together with the reference window for each occurrences of the event R.

![Figure 4.8](image)

In this example, there are three events in $\text{stimulus}(c)$ namely event $S_1$, $S_2$, and $S_3$. And there is one event in $\text{response}(c)$ namely event R. In figure 4.8, for each occurrence of the response event R, there is a reference window backward in time in which exactly each one of the three stimulus events must occur. For the response event R occurred at time instant $t$, we must have a unique occurrence of each of the stimulus events $S_1$, $S_2$, and $S_3$ within $t-\text{upper}(c)$ and $t-\text{lower}(c)$. Moreover, the distance between any two of the three stimulus events must not be separated by a distance greater than $\text{width}(c)=1$. There are three occurrences of the event R.
For the reference event R that occurs at time $t=5$, there are unique occurrences of the three stimulus events between time instants $5\text{-upper}(c)=5-5=0$ and $5\text{-lower}(c)=5-3=2$. The time instant from 0 to 2 is the reference window for the first occurrence of event R. The events $S_1$, $S_2$ and $S_3$ occurs uniquely within the reference window at time 1, 1.5, and 0.5, respectively (see the left-most oval in Figure 4.8). These three occurrences of the synchronization events are within a sliding window of length 1.

For the reference event R that occurs at time $t=12$, there are unique occurrences of the three stimulus events between time instants $12\text{-upper}(c)=12-5=7$ and $12\text{-lower}(c)=12-3=9$. The time instant from 7 to 9 is the reference window for the third occurrence of event R. The events $S_1$, $S_2$ and $S_3$ occur uniquely within reference window at time 7.5, 8, and 7.5, respectively (see the middle oval in Figure 4.8). These three occurrences of the synchronization events are within a sliding window of length 1.

For the reference event R that occurs at time $t=20$, there are unique occurrences of the three stimulus events between time instants $20\text{-upper}(c)=20-5=15$ and $20\text{-lower}(c)=20-3=17$. The time instant from 15 to 17 is the reference window for the first occurrence of event R. The events $S_1$, $S_2$ and $S_3$ occur uniquely within reference window at time 16, 15.5, and 16, respectively (see the right-most oval in Figure 4.8). These three occurrences of the synchronization events are within a sliding window of length 1.

Since for all the three occurrences of the response event R, there are unique occurrences of each of the three stimulus events within the sliding window which is within the reference window, the input synchronization constraint is satisfied.

### 4.4 Delay composition and decomposition

In this section, we show how the TADL delay concept adds up sequentially, so that a set of delay constraints on multiple interconnected event-chains can be replaced by a single constraint describing the end-to-end delay of the composed event-chain; or conversely, an end-to-end delay can be broken down into smaller intervals with preserved overall semantics.

Such sequential composition and decomposition of delay constraints is an important activity in the negotiation process between automotive OEMs and their suppliers. For example, suppliers may provide their components as TADL models that are like black boxes from a functional point of view. Such a model characterized by a few timing parameters still allow the OEM to calculate the overall delay imposed by a chain of such components. Or the OEM may divide the timing constraints of a complex functionality into multiple constraints, to be supplied by separate suppliers.

Moreover, we also show how TADL supports parallel composition and decomposition of delay constraints. While this notion is intuitively simple (each part in a parallel decomposition is expected to satisfy the prescribed end-to-end delay), it is nevertheless important when a
modeler wishes to express several alternative designs in one model, while resting assured that any choice taken will indeed satisfy the given overall timing requirements.

In both the sequential and the parallel case, it is meaningless to mix age and reaction time constraints in a particular constraint composition/decomposition. We therefore only consider composing or decomposing similar timing constraints — either just ages or just reaction times — and make this an implicit syntactic requirement in the descriptions that follow.

In the general case, it is allowed to constrain the same event-chain with several age constraints and several reaction constraints. An event chain has a number of other event chains each playing the role of a segment. Each of these segments can of course also have as well age constraints as reaction constraints. The above statement of not mixing age and reaction time, relates to whether it is possible to draw any conclusion between the delay constraint of an event chain and the sum of the delay constraints of its segments. This kind of conclusions can only be drawn when all the considered constraints are of the same kind (age or reaction).

4.4.1 Sequential composition and decomposition

A sequence of event chains \(<E_1, \ldots, E_n>\) is a sequential decomposition of an event chain \(E\), and \(E\) is a sequential composition of \(<E_1, \ldots, E_n>\), if the following conditions hold:

\[
\begin{align*}
\text{stimulus}(E) &= \text{stimulus}(E_1) \\
\text{stimulus}(E_i) &= \text{response}(E_{i-1}) & \text{for } 1 < i \leq n \\
\text{response}(E) &= \text{response}(E_n)
\end{align*}
\]

The fact that \(<E_1, \ldots, E_n>\) is a sequential decomposition of \(E\) can be explicitly marked in \(E\) by setting

\[
\text{segment}(E) = <E_1, \ldots, E_n>
\]

A sequence of age (reaction time) constraints \(<c_1, \ldots, c_n>\) is a sequential decomposition of an age (reaction time) constraint \(c\) if

\[
<\text{scope}(c_1), \ldots, \text{scope}(c_n)>
\]

is a sequential decomposition of \(\text{scope}(c)\)

(and conversely for composition).

A sequence of age (reaction time) constraints \(<c_1, \ldots, c_n>\) is stronger (accepts no more variations in event dynamics) than its sequential composition \(c\) if

\[
\begin{align*}
\text{upper}(c_1) + \ldots + \text{upper}(c_n) &\leq \text{upper}(c) \\
\text{lower}(c_1) + \ldots + \text{lower}(c_n) &\geq \text{lower}(c)
\end{align*}
\]

An age (reaction time) constraint \(c\) is stronger than its sequential decomposition \(<c_1, \ldots, c_n>\) if

\[
\begin{align*}
\text{upper}(c) &\leq \text{upper}(c_1) + \ldots + \text{upper}(c_n) \\
\text{lower}(c) &\geq \text{lower}(c_1) + \ldots + \text{lower}(c_n)
\end{align*}
\]
Taken together, these relations state that an age (reaction time) constraint \( c \) and its sequential decomposition \(<c_1, \ldots, c_n>\) are equivalent (accept the same variations in event dynamics) if

\[
\begin{align*}
\text{upper}(c) &= \text{upper}(c_1) + \ldots + \text{upper}(c_n) \\
\text{lower}(c) &= \text{lower}(c_1) + \ldots + \text{lower}(c_n)
\end{align*}
\]

### 4.4.2 Parallel composition and decomposition

A group of event chains \(<E_1, \ldots, E_n>\) is a parallel decomposition of an event chain \( E \), and \( E \) is a parallel composition of \(<E_1, \ldots, E_n>\), if the following conditions hold:

\[
\begin{align*}
\text{stimulus}(E) &= \text{stimulus}(E_i) \quad \text{for } 1 \leq i \leq n \\
\text{response}(E) &= \text{response}(E_i) \quad \text{for } 1 \leq i \leq n
\end{align*}
\]

The fact that \(<E_1, \ldots, E_n>\) is a parallel decomposition of \( E \) can be explicitly marked in \( E \) by setting

\[
\text{strand}(E) = <E_1, \ldots, E_n>
\]

A group of age (reaction time) constraints \(<c_1, \ldots, c_n>\) is a parallel decomposition of an age (reaction time) constraint \( c \) if

\[
<\text{scope}(c_1), \ldots, \text{scope}(c_n)>
\]

is a parallel decomposition of \( \text{scope}(c) \)

(and conversely for composition).

A group of age (reaction time) constraints \(<c_1, \ldots, c_n>\) is stronger (accepts no more variations in event dynamics) than its parallel composition \( c \) if

\[
\begin{align*}
\text{upper}(c) &\leq \text{upper}(c_i) \quad \text{for } 1 \leq i \leq n \\
\text{lower}(c) &\geq \text{lower}(c_i) \quad \text{for } 1 \leq i \leq n
\end{align*}
\]

An age (reaction time) constraint \( c \) is stronger than its parallel decomposition \(<c_1, \ldots, c_n>\) if

\[
\begin{align*}
\text{upper}(c) &\leq \text{upper}(c_i) \quad \text{for } 1 \leq i \leq n \\
\text{lower}(c) &\geq \text{lower}(c_i) \quad \text{for } 1 \leq i \leq n
\end{align*}
\]

Taken together, these relations state that an age (reaction time) constraint \( c \) and its parallel decomposition \(<c_1, \ldots, c_n>\) are equivalent (accept the same variations in event dynamics) if

\[
\begin{align*}
\text{upper}(c) &= \text{upper}(c_i) \quad \text{for } 1 \leq i \leq n \\
\text{lower}(c) &= \text{lower}(c_i) \quad \text{for } 1 \leq i \leq n
\end{align*}
\]
5 Conclusions

The feasibility of the TADL tool chain infrastructure has been demonstrated with the aid of a simple, but non-trivial, use case example. Additional proof-of-concept examples are needed to further test the correctness of the tool interaction. A complete validation of TADL has not yet been done with respect to the supplied use case scenarios. These scenarios have, however, been instrumental in the identification of timing concept issues which has led to the development of the novel TADL semantics.

To achieve the TIMMO goals within scheduled time frame, we have chosen a strategy where the set of syntactic and semantic building blocks of the TADL should be kept as small as possible. While identifying relevant AUTOSAR concepts in use cases and tool interfaces, it is important to define an action plan in case the project runs into one or more modeling issues that would reveal infeasibility of the chosen TADL approach, or imply the need for a much more complicated TADL model.

5.1 TADL semantics in research

TADL introduces two distinct forms of delay constraints, age and reaction time, and a very general form of repetition rate constraint that can be specialized to capture the standard periodic/sporadic event models as well as recurring event patterns and arbitrary event distributions. The semantics of the different constraint forms has been formalized within the TIMMO project, and algorithmic verification procedures have also been defined.

As a first embryonic example of a high-level logic of time constraints, TADL gives rise to several interesting research challenges. One group of questions directly relate to the consistency of the current version of the language: Are there classes of constraints expressible in TADL that are always true / always false, independently of the given event dynamics? Can consistency checks be defined that would help identify these extreme cases?

Another type of question concern the expressiveness of TADL: are there meaningful timing constraints on automotive system models that cannot easily be captured? Are the current constraint forms the right ones? While the TIMMO project has demonstrated that there are good reasons to believe TADL is a step in the right direction, more experience is needed in order to properly validate the design.

In a more general outlook, there is the prospect of developing TADL into a more fully-fledged timing logic by allowing constraints to appear as assumptions as well. A particular operation that would thus be supported is to verify that a system meets some timing constraint, under the assumption that its environment behaves in a certain way.

Another line of research that directly relates to TADL is the search for a semantic definition of the underlying structural models; i.e., EAST-ADL and AUTOSAR. In relation to the scope of TADL, this is a highly complex issue that involves aspects of concurrency, real-time scheduling, and the semantics of computation in general. Nevertheless, it is a perspective that cannot be ignored by producers...
of static analysis tools, and even partial advances in structural model semantics may be beneficial.
References

[3] EAST-EEA project
6 Annex A Meta-modeling guide

6.1.1 How the AUTOSAR meta-model was extended in TIMMO

Supporting constructs have been defined in TADL to be an infrastructure used by the XSD and TADL editor generator from the meta-model, these are seen in Figure 6.1.

- TADL
- TADLPackage
- TADLPackageableElement

Figure 6.1. The infrastructure of the meta-model defines the concepts TADL, TADLPackage, and TADLPackageableElement. All elements in TADL inherit from the TADLPackageableElement.

Note that associations are allowed over subset boundaries without requiring that both associated concepts are included in the same subset. This includes the use of dependencies stereotyped by «instanceRef» from TADL to AUTOSAR elements.

Dependencies with the «instanceRef» stereotype are modeled with a detailed structure in packages named “_instanceRef”. Additional classes point out the target, identified by the dependency, and the context of the target. Note that instanceRef are only used for reusable concepts, the context identification is not necessary in a hierarchy of non-reusable concepts. This is clearly seen in the example shown in Figure 6.2 where the hierarchy from System down on the right side is reusable, while the left side is not reusable.
6.2 Meta-modeling in Enterprise Architect

The following table shows how concepts defined in the AUTOSAR Template Modeling Guideline [1] are represented in the Enterprise Architect UML tool.

<table>
<thead>
<tr>
<th>AUTOSAR Modeling construct</th>
<th>Enterprise Architect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Meta classes with attributes, but without operations</td>
<td>Classes, with attributes</td>
</tr>
<tr>
<td>New class stereotypes supporting types and prototypes</td>
<td>AUTOSAR M3 Profile, «atpType» and «atpPrototype» applied on classes</td>
</tr>
<tr>
<td>Constraints</td>
<td>In EA fields</td>
</tr>
<tr>
<td>Inheritance, single and multiple</td>
<td>Generalization</td>
</tr>
<tr>
<td>Aggregation</td>
<td>Association with composite end</td>
</tr>
<tr>
<td>Association</td>
<td>Association with direction</td>
</tr>
<tr>
<td>New association stereotypes «isOfType» and «instanceRef»</td>
<td>«isOfType» on association «instanceRef» on dependency</td>
</tr>
<tr>
<td>Packages</td>
<td>Packages</td>
</tr>
<tr>
<td>Enumerations</td>
<td>Enumerations</td>
</tr>
</tbody>
</table>

6.2.1 Alignment of EAST-ADL2 with AUTOSAR M3

An action on WP2 from the STC meeting 2007-12-05 raised the question on whether AUTOSAR and EAST-ADL2 meta-models fit together. This section describes the modifications that have been
done on a subset of the EAST-ADL2 meta-model to make it conform to the AUTOSAR M3 rules. The resulting meta-model has been used in a case study where an XML schema and an editor were generated.

First some comparison of terminology, as used by AUTOSAR vs. what is used within this document has been identified:

<table>
<thead>
<tr>
<th>Term</th>
<th>AUTOSAR</th>
<th>This document</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subset</td>
<td>The meta-classes that are included in an AUTOSAR template.</td>
<td>Subset is used more generally, e.g., “subset of UML2”, meaning a subset of UML2. Note that subset is also a UML2 term that means relation between inherited properties, this is not considered here.</td>
</tr>
<tr>
<td>Derived</td>
<td>A meta-class that specializes another meta-class by a generalization.</td>
<td>Not used. In UML2 terms this has a special meaning w.r.t. relation between inherited properties.</td>
</tr>
<tr>
<td>Baseclass</td>
<td>The general meta-class, by a generalization.</td>
<td>Not used.</td>
</tr>
</tbody>
</table>

Regarding tooling for modeling the AUTOSAR Metamodel and the EAST-ADL2 Domain Model are both defined in Enterprise Architect. In the ATESSST project a relevant subset of the AUTOSAR Metamodel was imported to make it possible to refer from EAST-ADL2 concepts to AUTOSAR concepts. The AUTOSAR Metamodel was not modified within the ATESSST project.

Regarding modeling style and alignment to a common M3 level there is a difference:

1) The EAST-ADL2 Domain Model relies on OMG MOF/Infrastructure [6] to define the model.

2) The AUTOSAR Metamodel relies on a subset of UML2, an AUTOSAR M3 Profile and AUTOSAR Template Modeling Guideline [1].

The rather strict way the AUTOSAR Metamodel is defined is utilized in the tool chain to generate XSD, and also to generate the AUTOSAR editor.

The current EAST-ADL2 Domain Model is not in line with the AUTOSAR Template Modeling Guideline. This means that the AUTOSAR tool chain will not correctly interpret the Domain Model and an “AUTOSAR looking” XSD will not be generated. The EAST-ADL2 Domain Model has to be aligned with the AUTOSAR template rules.

The details on how to do this include:

- The AUTOSAR Metamodel uses a subset of UML, while the EAST-ADL2 Domain Model is based on the OMG MOF/Infrastructure. The similarities are quite large, e.g., only class diagrams are used in both approaches. The following table describes the differences, with the UML subset used by AUTOSAR [1] §5.1 as starting point:
<table>
<thead>
<tr>
<th>AUTOSAR Modeling construct</th>
<th>EAST-ADL2 compliance</th>
<th>Comments and stereotypes to apply (implicit stereotypes do not have to be applied)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Meta classes with attributes, but without operations</td>
<td>Aligned</td>
<td>«atpClass» (implicit) «atpAttribute» (implicit)</td>
</tr>
<tr>
<td>New class stereotypes supporting types and prototypes</td>
<td>Basically aligned <strong>review needed</strong>, stereotypes have to be added. Compare the EAST-ADL2 profile where UML Class and Property are extended respectively.</td>
<td>«atpType» , «atpPrototype» both extending Class</td>
</tr>
<tr>
<td>Comments</td>
<td>Not used, aligned.</td>
<td>Not stereotyped</td>
</tr>
<tr>
<td>Constraints</td>
<td>Not formalized as OCL</td>
<td>«atpConstraint» (implicit) Unclear if AUTOSAR has formalized all constraints as OCL.</td>
</tr>
<tr>
<td>Inheritance, single and multiple</td>
<td>Aligned</td>
<td></td>
</tr>
<tr>
<td>Aggregation</td>
<td>Used, <strong>review needed</strong>, further restrictions might have to be considered.</td>
<td>Only <strong>composite</strong> aggregations are allowed</td>
</tr>
<tr>
<td>Associations</td>
<td>Aligned, further restrictions might have to be considered. Consider naming conventions for role names and navigability (only one navigable end)</td>
<td>«atpAssociation» (implicit)</td>
</tr>
<tr>
<td>New association stereotypes «isOfType» and «instanceRef»</td>
<td>To be added where needed. Model details of «instanceRef» dependencies in _instanceRef packages.</td>
<td>«isOfType» used with «atpType» and «atpPrototype»; «instanceRef» used with «atpPrototype» and «atpStructureElement»</td>
</tr>
<tr>
<td>Dependencies</td>
<td>Not used. To define a subset, i.e., template for EAST-ADL2 the «import» should be used. Stereotypes defined in UML may be used, «trace», «use», and «import». Dependencies without stereotypes are only used to document the model.</td>
<td></td>
</tr>
<tr>
<td>Packages</td>
<td>Aligned</td>
<td>Imports and merges are not required. «atpPackage» (implicit)</td>
</tr>
<tr>
<td>Tagged values</td>
<td>Not used</td>
<td>Used to control XSD definition, see [5]</td>
</tr>
<tr>
<td>Primitive types and literals</td>
<td>Aligned, primitive types used primitive types: String Boolean Integer</td>
<td>Not stereotyped, AUTOSAR uses the primitive types and their corresponding literals: String Boolean Integer UnlimitedNatural</td>
</tr>
<tr>
<td>New primitive type Float</td>
<td>To be aligned</td>
<td>With literal extending LiteralString</td>
</tr>
<tr>
<td>Enumerations</td>
<td>Aligned, enumeration literals are also stereotyped.</td>
<td>Not stereotyped with AUTOSAR stereotype, see §6.2.3 of [1]</td>
</tr>
<tr>
<td>Mixed content</td>
<td>Not used</td>
<td>Documentation including HTML elements. Unclear if this is really used.</td>
</tr>
</tbody>
</table>
Apply the AUTOSAR M3 Profile on the ATESST Domain Model. This for example includes adding the stereotypes «instanceRef» and «isOfType» on some associations spread over the Domain Model. Other stereotypes that are to be added on the EAST-ADL2 concepts are shown in Figure 6.3. Similar to how this is done in the AUTOSAR Metamodel most of these stereotypes will only be necessary to apply on general concepts; the semantics will be inherited by specializations. Also some of the stereotypes are considered as implicit by AUTOSAR, i.e., these stereotypes do not need to be applied as they are implicit from the used UML2 concept, see table above.

Figure 6.3. The AUTOSAR M3 Profile [1] the stereotypes defined here extends the UML2 subset stereotyped by «metaclas» here. ATP=AUTOSAR Template Profile

Make detailed modeling in instanceRef packages of the «instanceRef» stereotyped associations. See Figure 6.4.
The AUTOSAR naming conventions §6.1.2 of [1] might require updates in the EAST-ADL2 Domain Model.

Names like ADLFunctionType are allowed and will according to [5] §3.6 be ADL-FUNCTION-TYPE as XML element.

Role names should be reviewed, only singular form is allowed.

Definition of subsets see §8 of [1], the EAST-ADL2 Domain Model is suitable to be divided in subsets according to abstraction level:

- Vehicle Feature Model
- Functional Analysis Architecture
- Functional Design Architecture and relation to AUTOSAR concepts.

Further subsets may be defined according to:

- Requirements and Safety Cases
- Artifact Level Variation Management
- Behavior with Error Behavior
- Verification and Validation
7 Annex B UML model in XMI Format

The XMI will not be included here, but is provided as a separate part of this deliverable. This file contains the meta-model as defined in Enterprise Architect and used as input to the XSD and editor generator. The meta-model documentation in Annex F TADL is an extract of the UML model with documentation of classes, attributes and associations.

The XMI file can be imported on top of an AUTOSAR meta-model R3.1 in Enterprise Architect. To import the XMI file follow these instructions:

1) Download the TIMMO XMI (TIMMO_MM_YYYY-MM-DD.xml) from https://www.carmeqxchange.com/TIMMO/repos/10_Deliverables/TIMMO_D6/.

2) Open the AUTOSAR Meta-model R3.1 in Enterprise Architect.

8 Annex C Specification of the model TADL XML exchange format

The UML model from Enterprise Architect of the TADL meta-model, which specifies the exchange format, is documented in Annex F TADL. How the XML Schema is generated from the meta-model is described by the AUTOSAR Model Persistence Rules [5].
9 Annex D TADL model editor

This annex contains snapshots of VSA showing examples of modeling using the TADL as an extension of AUTOSAR.

9.1 Snapshots of the TADL editor

Figure 9.1. An ADLFunctionType with ADLFlowPorts has been created.

Figure 9.2. A FunctionalAnalysisArchitecture with two parts typed by the ADLFunctionType.
Figure 9.3. The FAATiming for the FAA timing information has been created. The events have been created and refer to the port occurrences in their contexts.

Figure 9.4. An EventChain with a response event (stimulus event not shown).
Figure 9.5. A ReactionConstraint valid in a mode and referring to the scope event chain.
The meta-model of TADL can be viewed as a domain model and can be concretized not only as a XML schema, but as an extension to UML, implemented as a UML profile, as well. This would allow the TADL to be used in a UML editor, as compared to the domain specific editor selected for the TIMMO project.

One of the key aspects to define a UML profile is the full compliance with the UML specification. A UML profile can extend the UML syntax and semantics, but cannot modify them. A very useful source to understand the conformance with UML is looking at the MARTE profile. The paper, co-authored with Symtavision, [7] represents a good starting point towards defining the UML profile for TADL.

As with the definition of the TADL Domain Model the basis of a TADL Profile need profiles for EAST-ADL2 and AUTOSAR. A UML profile has been defined for EAST-ADL2 within the ATESST project. This profile is readily available as a plug-in for the UML tool Papyrus. For AUTOSAR the process of defining a profile was early outlined [13], however there is no publicly available implementation of an AUTOSAR profile.

The basic syntax of TADL can however easily be defined within Papyrus as a profile. It is even possible to import the EAST-ADL2 profile and make the necessary references to the structural concepts available there. The EAST-ADL2 Profile is prepared for references to an AUTOSAR profile, even though such a profile is not available. These references are performed via the extended UML meta-class rather than explicit references to stereotypes.

So stereotypes can be defined for the TADL basic syntax, and the events referring to EAST-ADL2 structural concepts, however not for events explicitly referring to AUTOSAR structural concepts.

The selection of UML2 elements to extend still has to be decided, as a starting point UML2 NamedElement can be used. This allows the stereotypes to be applied on any element in a UML model. Note that the relationships between stereotypes are typed by the specified stereotype. Hence the value of a stereotype attribute need to have a specific stereotype applied.

Installation of the TADL profile, require:

1. Papyrus, available from [www.papyrusuml.org](http://www.papyrusuml.org) (version 1.11)
2. UML4EAST-ADL2 as plugin, available from [www.papyrusuml.org](http://www.papyrusuml.org) (version 1.9.8)
4. Follow the instructions on [www.papyrusuml.org](http://www.papyrusuml.org) on how to use profiles.
Figure 10.1. TIMMO concepts implemented as UML stereotypes in Papyrus.

Figure 10.2. TIMMO constraints implemented as stereotypes.
Figure 10.3. TIMMO constraints and their relation to the Event.

Figure 10.4. Events for the EAST-ADL2 structure.
11 Annex F TADL meta-model

TADL-Infrastructure
TADL
TimingConstraints
EAST-ADL2Timing
EAST-ADL Events

11.1 TADL

Timing Augmented Description Language.

![Diagram showing TADL Example]

Figure 5. TADL Example.
11.1.1 Event (from TADL) {abstract}

Generalizations

TimingDescription (from TADL)

Description

An Event (E) is the specification of some occurrence. Such an occurrence either stimulates an execution, or is caused by an execution [as a response to another event that occurred before]. In the first case the event is called Stimulus (S) and in the latter case it is called Response (R). Stimuli always precede responses; and responses in turn always succeed stimuli.

An event either indicates that a state change occurred, or reports a [current] state. In the latter case, the event periodically occurs. For example, the "driver door has been opened" is an event indicating a state change; whereas the "driver door is open" is an event reporting a state.

An event occurs instantaneous, which means that an event occurs at a time instant without any duration. In addition, an event can appear any number of times and the subsequent occurrences may follow a specific pattern, like periodic, sporadic, or in sudden bursts. Each of these occurrences has a unique time instant.

The distinction between an event and its occurrence is usually obvious from the considered context (causal and temporal). The event is not defined by its occurrences, but rather by a description expressing its purpose.
Attributes

sStateChange : Boolean = true [1]

This attribute indicates whether the event reports a state change or a [current] state. If the boolean value is TRUE, then the event reports a state change (no over-/undersampling).
If the boolean value is FALSE, then the event reports a [current] state.
By default, the value of this attribute is TRUE.

Associations

No additional associations

Dependencies

No additional dependencies

Constraints

[1] In the case that the event reports a [current] state (isStateChange is FALSE), then the event must have a periodic event model [or a pattern model].

11.1.2 EventChain (from TADL)

Generalizations

TimingDescription (from TADL)

Description

The purpose of event chains is to describe the temporal behavior of a number of steps to be taken to respond to one or more events accordingly. [A time chain is also used to express that a temporal requirement/constraint is imposed on a number of steps to respond to one or more events accordingly -> requirement.] Such events could be observed in a given system and are categorized into stimuli and responses.
Time chains can refer to other time chains which are then called time chain segments. A time chain respectively time chain segment can be atomic which means it is not refined to other time chains.

Attributes

No additional attributes

Associations

segment : EventChain [*] {ordered} Referred EventChains that are not parallel and in sequence refines this EventChain.

response : Event [1..*] The Response element is the entity to describe an event that is a response to a stimulus that occurred before.
stimulus : Event [1..*]  The Stimulus element is the entity to describe an event that stimulates the steps to be taken to respond to this event.

strand : EventChain [*]  Parallel EventChains refining this EventChain.

Dependencies
No additional dependencies

Constraints
No additional constraints

11.1.3 Mode (from TADL) «atpIdentifiable»

Generalizations
Identifiable (from GenericStructure:: Identifiable)

Description
Modes can appear to determine the state of the system, an hardware entity, or an application.
The use of modes can be used to filter different views of the model.

Attributes
condition : String [1]

Associations
No additional constraints

Dependencies
No additional constraints

Constraints
No additional constraints

11.1.4 ModeGroup (from TADL)

Generalizations
TADLPackageableElement (from TADL-Infrastructure)

Description
The set of Modes in a ModeGroup are mutually exclusive. This means that only one Mode of a ModeGroup is active at any point in time.

Attributes
precondition : String [1]

Associations
mode : Mode [1..*]  The modes in this group.
### Dependencies
No additional dependencies

### Constraints
No additional constraints

#### 11.1.5 Time (from TADL)

**Generalizations**
None

**Description**

<table>
<thead>
<tr>
<th>CseCodeType</th>
<th>Unit</th>
<th>Referred To</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1 µsec</td>
<td>Time</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>10 µsec</td>
<td>Time</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>100 µsec</td>
<td>Time</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1 msec</td>
<td>Time</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>10 msec</td>
<td>Time</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>100 msec</td>
<td>Time</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1 sec</td>
<td>Time</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>10 sec</td>
<td>Time</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1 min</td>
<td>Time</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1 h</td>
<td>Time</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1 d</td>
<td>Time</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>Angular degrees</td>
<td>Angle</td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>Revolutions 360</td>
<td>Angle</td>
<td>degrees</td>
</tr>
<tr>
<td>102</td>
<td>Cycle 720</td>
<td>Angle</td>
<td>degrees e.g. in case of IC engines</td>
</tr>
<tr>
<td>103</td>
<td>Cylinder segment</td>
<td>Combustion</td>
<td>e.g. in case of IC engines</td>
</tr>
<tr>
<td>998</td>
<td>When frame</td>
<td>Time</td>
<td>available Source defined in the ASAP 2 keyword, FRAME</td>
</tr>
<tr>
<td>999</td>
<td>Always if</td>
<td>Time</td>
<td>there is new value Calculation of a new upper range limit after receiving a new partial value, e.g. when calculating a complex trigger condition</td>
</tr>
<tr>
<td>1000</td>
<td>Non deterministic</td>
<td>Without fixed scaling</td>
<td></td>
</tr>
</tbody>
</table>

Deliverable D6 Version 1.2
If, for example, the value in <swCseCodeFactor> is 360 and the value in <swCseCode> is 100, this is equivalent to the value 1 in <swCseCodeFactor> and the value 101 in <swCseCode>.

CseCodeType is from AUTOSAR and MSR/ASAM.

Note that we have set the cseCodeType for 1 µsec to 0 (error in AUTOSAR). And have changed cseCodeType 2 to 100 µsec (error in MSR).

Attributes

- **cseCode**: CseCodeType [1] Within TIMMO this is normally time, note that when it is expressed as angle it can be converted to time.
- **cseCodeFactor**: Int [1] Is normally equal to 1.
- **value**: Float [1] The actual value complemented with the cseCode.

Associations

- No additional associations

Dependencies

- No additional dependencies

Constraints

- No additional constraints

### 11.1.6 Timing (from TADL) «atpIdentifiable»

Generalizations

- TADLPackageableElement (from TADL-Infrastructure)

Description

The collection of timing constraints and their descriptions in the form of events and event chains.

Attributes

- No additional attributes

Associations

- timingConstraint : TimingConstraint [*]
- timingDescription : TimingDescription [*]

Dependencies

- No additional dependencies

Constraints

- No additional constraints

### 11.1.7 TimingConstraint (from TADL) {abstract} «atpIdentifiable»

Generalizations

- ADLRequirement (from Requirements)
Description

The TimingConstraint metaclass denotes a requirement entity that allows giving bounds on system timing attributes, i.e., end-to-end delays, periods, etc.

Attributes

No additional attributes

Associations

mode : Mode [*]  The mode where the TimingConstraint is valid.

Dependencies

No additional dependencies

Constraints

No additional constraints

11.1.8 TimingDescription (from TADL) {abstract} «atpIdentifiable»

Generalizations

ADLEntity (from ADLCoreConstructs)

Description

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

Attributes

No additional attributes

Associations

No additional associations

Dependencies

No additional dependencies

Constraints

No additional constraints

11.2 TimingConstraints

This section describes the timing constraints as defined by TIMMO as shown in the figure below.
Figure 7. EventConstraint. TADL provides two categories of timing constraints: these impose constraints on the Event.

Figure 8. DelayConstraints. TADL provides two categories of timing constraints: these category imposes constraints on Event Chain.
11.2.1 AgeTimingConstraint (from TimingConstraints) «atpIdentifiable»

Generalizations

DelayConstraint (from TimingConstraints)

Description

Different tolerances on over-/undersampling can be identified when the solution has been modeled.

An age constraint is of interest in control engineering when looking back through the system.

In case of over- or undersampling, there is no one-to-one relation possible between the occurrences of stimuli and responses of the associated timing chain. Thus, the age constraint defines the semantic of which delay must be constrained.

The attribute upper is applicable in worst-case analysis.

The attribute lower is applicable in best-case analysis.

Attributes

No additional attributes

Associations

scope : EventChain [1] The EventChain on which this constraint is applied.

Dependencies

No additional dependencies

Constraints

No additional constraints
## 11.2.2 ArbitraryEventConstraint (from TimingConstraints) «atpIdentifiable»

### Generalizations

EventConstraint (from TimingConstraints)

### Description

The Arbitrary Event Constraint describes that an event occurs occasionally, singly, irregular or randomly. The primary purpose of this event model is to abstract event occurrences captured by data acquisition tools (background debugger, trace analyzer, etc.) during the operation of a system.

Note! The number of elements in the sets minimum inter-arrival time and maximum inter-arrival time must be the same.

### Attributes

No additional attributes

### Associations

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>minimumInterArrivalTime : Time [1..*]</td>
<td>The set of minimum inter-arrival times specifies the minimum inter-arrival time between two and more subsequent occurrences of the event. The first element in the set specifies the minimum inter-arrival time between two subsequent occurrences of the event among the given occurrences. The second element in the set specifies the minimum inter-arrival time between three subsequent occurrences of the event among the given occurrences; and so forth.</td>
</tr>
<tr>
<td>maximumInterArrivalTime : Time [1..*]</td>
<td>The set of maximum inter-arrival times specifies the maximum inter-arrival time between two and more subsequent occurrences of the event. The first element in the set specifies the maximum inter-arrival time between two subsequent occurrences of the event among the given occurrences. The second element in the set specifies the maximum inter-arrival time between three subsequent occurrences of the event among the given occurrences; and so forth.</td>
</tr>
</tbody>
</table>

### Dependencies

No additional dependencies

### Constraints

No additional constraints
11.2.3 DelayConstraint (from TimingConstraints) {abstract} «atpIdentifiable»

Generalizations
TimingConstraint (from TADL)

Description
The DelayConstraint metaclass denotes a requirement entity that allows giving bounds on system timing attributes, i.e., end-to-end delays, periods, etc.

A DelayConstraint can specify one or several of an upper bound, a lower bound or a nominal value and jitter. The jitter is evenly distributed around the nominal (nominal - jitter/2 .. nominal + jitter/2). The bound will be measured in a given unit.

The kind attribute from ADLEntity is either of:
OFFERED, property
REQUIRED, constrained

Attributes
No additional attributes

Associations
lower : Time [0..1] A lower bound on the distance between occurrences.
jitter : Time [0..1] Jitter is the range within which a value varies.
upper : Time [0..1] An upper bound on the distance between occurrences.
nominal : Time [0..1] The recurring distance between each occurrence.

Dependencies
No additional dependencies

Constraints
No additional constraints

11.2.4 EventConstraint (from TimingConstraints) {abstract} «atpIdentifiable»

Generalizations
TimingConstraint (from TADL)

Description
An event model describes the basic characteristics of the way an event occurs over time.

Attributes
No additional attributes
In addition an event model may specify an offset, which delays the start of the first period - the occurrence of the very first event - by the given amount of time.

**11.2.5 GenericEventConstraint (from TimingConstraints) «atpIdentifiable»**

**Generalizations**
EventConstraint (from TimingConstraints)

**Description**
A generic constraint describing the basic characteristics of the way an event occurs over time.

**Attributes**
- span : Int [1] A count indicating whether it is subsequent occurrences or occurrences farther apart that are constrained.

**Associations**
- lower : Time [0..1] A lower bound on the distance between occurrences.
- jitter : Time [0..1] Jitter is the range within which a value varies.
- upper : Time [0..1] An upper bound on the distance between occurrences.

**Dependencies**
No additional dependencies

**Constraints**
No additional constraints

**11.2.6 InputSynchronizationConstraint (from TimingConstraints) «atpIdentifiable»**

**Generalizations**
AgeTimingConstraint (from TimingConstraints)

**Description**
The InputSynchronizationConstraint metaclass is a language entity that expresses a timing requirement/constraint on the input synchronization among the set of stimulus events.
The parameters of InputSynchronizationConstraint, see TimingConstraint, constrain the time from the first stimulus until last stimulus (i.e., maximum skew between the stimuli). A join point is identified by the response event in the scope EventChain.

Attributes

No additional attributes

Associations


Dependencies

No additional dependencies

Constraints

[1] The set of ADLFlowPorts referenced by the events should contain only ADLInFlowPorts. The rationale for this is that the events shall relate to data on ADLFlowPorts which is considered (or shall be) temporally consistent.

[2] The scope EventChain shall only have one response Event.

[3] The semantics of this constraint require that there is more than one stimulus Events in the scope EventChain.

11.2.7 OutputSynchronizationConstraint (from TimingConstraints) «atplIdentifiable»

Generalizations

ReactionConstraint (from TimingConstraints)

Description

The OutputSynchronizationConstraint metaclass is a language entity that expresses a timing requirement/constraint on the output synchronization among the set of response events.

The parameters of OutputSynchronizationConstraint, see TimingConstraints, constrain the time from the first response until last response (i.e., maximum skew between the responses). A fork point is identified by the stimulus event in the scope EventChain.

Attributes

No additional attributes

Associations


Dependencies

No additional dependencies

Constraints

[1] The set of ADLFlowPorts referenced by the events should contain only ADLOutFlowPorts. The rationale for this is that the events shall relate to data on ADLFlowPorts which is considered (or shall be) temporally consistent.

[2] The scope EventChain shall only have one stimulus Event.
[3] The semantics of this constraint require that there is more than one response Events in the scope EventChain.

11.2.8 PatternEventConstraint (from TimingConstraints) «atpIdentifiable»

Generalizations

EventConstraint (from TimingConstraints)

Description

The [Concrete] Pattern Event Constraint describes that an event occurs following a known pattern. The pattern event model is characterized by the following parameters:

Attributes

No additional attributes

Associations

period : Time [1]

The period specifies the time interval within the event occurs any number of times following a pattern.

minimumInterArrivalTime : Time [1]

The minimum inter-arrival time specifies the minimal possible time interval between two consecutive occurrences of the event within the given period.

occurrence : Time [1..*]

{ordered}

The set occurrence[1..n] specifies the offset for each occurrence of the event in the specified period. Each occurrence is specified from the beginning of the period.

jitter : Time [1]

The jitter specifies maximal possible time interval the occurrence of the events within the given period can vary (formerly: can be delayed).

Dependencies

No additional dependencies

Constraints

No additional constraints

11.2.9 PeriodicEventConstraint (from TimingConstraints) «atpIdentifiable»

Generalizations

EventConstraint (from TimingConstraints)

Attributes

No additional attributes
Associations

- jitter : Time [1]  The jitter specifies the maximal possible time interval the occurrence of an event can vary (formerly: be delayed).
- period : Time [1]  The period specifies the [ideal] time interval between two subsequent occurrences of the event.
- minimumArrivalTime : Time [1]  The minimum inter-arrival time specifies the minimal possible time interval between two consecutive occurrences of an event.

Dependencies

No additional dependencies

Constraints

No additional constraints

11.2.10 ReactionConstraint (from TimingConstraints) «atpIdentifiable»

Generalizations

DelayConstraint (from TimingConstraints)

Description

ReactionConstraint is of interest when we look forward through the system. Compare AgeTimingConstraint.

Attributes

No additional attributes

Associations

- scope : EventChain  The EventChain on which this constraint is applied.

Dependencies

No additional dependencies

Constraints

No additional constraints

11.2.11 SporadicEventConstraint (from TimingConstraints) «atpIdentifiable»

Generalizations

EventConstraint (from TimingConstraints)

Description

The Sporadic Event Constraint describes that an event occurs occasionally. In general it is supposed that the event eventually occurs. Indeed, it is also known that some of the events do not occur for whatsoever reasons.
Note! The parameters minimum inter-arrival time and maximum inter-arrival time must reference the same point in time. Typically, this is the point in time that specifies the beginning of the period subject to consideration.

Attributes

No additional attributes

Associations

- **jitter** : Time [0..1]
  
  The optional parameter jitter specifies the maximal possible time interval the occurrence of an event can vary (formerly: be delayed). By nature, a sporadic event is supposed to occur at any time, thus it is one of the characteristic that the occurrence is jittery.

- **period** : Time [1]
  
  The period specifies the [ideal] time interval between two subsequent occurrences of the event.

- **maximumInterArrivalTime**:
  
  The optional parameter maximum inter-arrival time specifies the maximal possible time interval between two consecutive occurrences of an event.

  The maximum inter-arrival time may be used to describe different cases:

  (1) The maximum inter-arrival time is equal to the duration of the period.

  (2) The maximum inter-arrival time is used to specify a point in time within the period that immediately follows the period subject to consideration.

  (3) The maximum inter-arrival time is used to specify a point in time within one of the subsequent periods that follow the period subject to consideration.

- **minimumInterArrivalTime** :
  
  The minimum inter-arrival time specifies the minimal possible time interval between two consecutive occurrences of an event.

Dependencies

No additional dependencies

Constraints

No additional constraints

**11.3 EAST-ADL2Timing**

Events for EAST-ADL2 as defined by TIMMO. The TIMMO/TADL supports the abstraction levels of EAST-ADL with regard to timing.
Timing descriptions and constraints can be structured such that these timing informations are associated with a particular level of abstraction.

Figure 10. EAST-ADL2Extensions.

### 11.3.1 FAATiming (from EAST-ADL2Timing) «atpIdentifiable»

**Generalizations**

Timing (from TADL)

**Description**

The FAATiming contains all timing information (constraints and descriptions) relevant for the Functional Analysis Architecture.

**Attributes**

No additional attributes

**Associations**

architecture : FunctionalAnalysisArchitecture [1]

**Dependencies**

No additional dependencies

**Constraints**

No additional constraints

### 11.3.2 FDATiming (from EAST-ADL2Timing) «atpIdentifiable»

**Generalizations**

Timing (from TADL)

**Description**

The FDATiming contains all timing information (constraints and descriptions) relevant for the Functional Design Architecture.

**Attributes**

No additional attributes

**Associations**

architecture : FunctionalDesignArchitecture [1]
Dependencies
No additional dependencies

Constraints
No additional constraints

11.3.3 VFMTiming (from EAST-ADL2Timing) «atplIdentifiable»

Generalizations
Timing (from TADL)

Description
The VFMTiming contains all timing information (constraints and descriptions) relevant for the Vehicle Feature Model.

Attributes
No additional attributes

Associations
model : VehicleFeatureModel [1]

Dependencies
No additional dependencies

Constraints
No additional constraints
11.4 Events

This section describes the events that are defined in EAST-ADL.

Figure 11. EAST-ADL2Events.

Figure 12. instanceRef.
11.4.1 EASTEvent (from Events) {abstract}

Generalizations
Event (from TADL)

Description
An abstract metaclass that is specialized by the events applicable on the EAST-ADL2 model.

Attributes
No additional attributes

Associations
No additional associations

Dependencies
No additional dependencies

Constraints
No additional constraints

11.4.2 EventADLClientPort (from Events)

Generalizations
EventADLPort (from Events)

Attributes

eventKind : EventADLClientPortKind [1]

Associations

clientPort : EventADLClientPort_clientPort [1]

Dependencies

clientPort : ADLClientPort [1] «instanceRef»

Constraints
No additional constraints

11.4.3 EventADLClientPortKind (from Events) «enumeration»

Generalizations
None

Enumeration Literals

• receivedResponse
• sendRequest

Associations
No additional associations

Dependencies
No additional dependencies
## 11.4.4 EventADLFUncton (from Events)

### Generalizations
- EASTEvent (from Events)

### Description
An event of an ADLFUncton refers to the triggering of the ADLFUncton, i.e., when the input data is consumed, data transformation is performed on that input data by the function, and output data is produced.

Compare categories of AUTOSAR runnables:

- 1a triggering only on start and finish (this type of event)
- 1b triggering allowed anytime during the execution (events on ports, see EventADLInFlowPort)

Note: Adapt description of ADLFUncton in EAST.

### Attributes
- No additional attributes

### Associations
- function : EventADLFUncton_function [1]

### Dependencies
- function : ADLFUnctonPrototype [1] «instanceRef»

### Constraints
- No additional constraints

## 11.4.5 EventADLInFlowPort (from Events)

### Generalizations
- EventADLPort (from Events)

### Description
An event of an ADLInFlowPort refers to the consumption of data on that port.

Compare categories of AUTOSAR runnables:

- 1a triggering only on start and finish (events on function, see EventADLFUncton)
- 1b triggering allowed anytime during the execution of the ADLFUncton (this type of event)

### Attributes
- No additional attributes
11.4.6 EventADLInOutFlowPort (from Events)

Generalizations
EventADLPort (from Events)

Attributes
No additional attributes

Associations

Dependencies

Constraints
No additional constraints

11.4.7 EventADLOutFlowPort (from Events)

Generalizations
EventADLPort (from Events)

Description
An event of an ADLOutFlowPort refers to the creation of data on that port.

Attributes
No additional attributes

Associations
outFlowPort : EventADLOutFlowPort_outFlowPort [1]

Dependencies

Constraints
No additional constraints

11.4.8 EventADLPort (from Events) {abstract}

Generalizations
EASTEvent (from Events)

Attributes
No additional attributes
11.4.9 EventADLServerPort (from Events)

Generalizations
EventADLPort (from Events)

Attributes
eventKind : EventADLServerPortKind [1]

Associations
serverPort : EventADLServerPort_serverPort [1]

Dependencies
serverPort : ADLServerPort [1]  «instanceRef»

Constraints
No additional constraints

11.4.10 EventADLServerPortKind (from Events) «enumeration»

Generalizations
None

Enumeration Literals
  • receivedRequest
  • sendResponse

Associations
No additional associations

Dependencies
No additional dependencies

Constraints
No additional constraints

11.5 TADL-Infrastructure

Infrastructure required to align TIMMO with AUTOSAR M3.
11.5.1 TADL (from TADL-Infrastructure)

Generalizations
None

Attributes
No additional attributes

Associations
adminData : AdminData [0..1] (from MSR)
topLevelPackage : TADLPackage [0..*] «splitable»

Dependencies
No additional dependencies

Constraints
No additional constraints

11.5.2 TADLPackage (from TADL-Infrastructure)

Generalizations
Identifiable (from GenericStructure::Identifiable)

Attributes
No additional attributes

Associations
element : TADLPackageableElement [*] «splitable»

Dependencies
No additional dependencies

Constraints
No additional constraints
### 11.5.3 TADLPackageableElement (from TADL-Infrastructure) {abstract}

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generalizations</td>
<td>Identifiable (from GenericStructure::Identifiable)</td>
</tr>
<tr>
<td>Attributes</td>
<td>No additional attributes</td>
</tr>
<tr>
<td>Associations</td>
<td>No additional associations</td>
</tr>
<tr>
<td>Dependencies</td>
<td>No additional dependencies</td>
</tr>
<tr>
<td>Constraints</td>
<td>No additional constraints</td>
</tr>
</tbody>
</table>