Contents

References

[1] Meta Model
AUTOSAR_MMOD_MetaModel

[2] Glossary
AUTOSAR_TR_Glossary

[3] Methodology for Classic Platform
AUTOSAR_TR_Methodology

[4] Standardization Template
AUTOSAR_TPS_StandardizationTemplate

[5] Requirements on Timing Extensions
AUTOSAR_RS_TimingExtensions

[6] Virtual Functional Bus
AUTOSAR_EXP_VFB

[7] Generic Structure Template
AUTOSAR_TPS_GenericStructureTemplate

[8] Specification of Timing Extensions
AUTOSAR_TPS_TimingExtensions

[9] Methodology for Adaptive Platform
AUTOSAR_TR_AdaptiveMethodology

1 Introduction

1.1 Overview

This AUTOSAR document contains the specification of the AUTOSAR Timing Extensions and describes the elements of the AUTOSAR meta-model [1] used for creating timing models for the respective AUTOSAR Platform. It is a supplement to the formal definition of the Timing Extensions by means of the AUTOSAR meta-model. In other words, this document in addition to the formal definition provides introductory description and rationale for the part of the AUTOSAR meta-model relevant for the creation of timing models.

1.2 Terms and Abbreviations

The main list of terms and abbreviations are defined in [2]. The following table contains the list of terms and abbreviations used in the scope of this document which are not already defined in [2] along with the spelled-out meaning of each of the abbreviations.

Table 1.1: Abbreviations used in the scope of this Document

1.3 Glossary of terms

Table 1.2: Terms used in the scope of this Document

1.4 Template implications

All AUTOSAR templates use a common meta-model which is defined by using the Unified Modeling Language (UML). For the integration of timing information into the AUTOSAR meta-model we have to decide between two viable alternatives: on the one hand the extension of existing templates, and on the other hand the definition of a separate timing template.

Several discussions lead to the decision to explicitly NOT defining a separate timing template. The most valuable advantage of such an approach is addressed by the idea behind the current template composition. They are highly adapted to the AUTOSAR methodology (see [3] for more details about the AUTOSAR methodology) and the several templates handle specific process steps in the methodology. Since it is not our scope to provide a proposal for a timing augmented development process, it is as well not in our scope to define an isolated, new process step (e.g. a timing process step). For this reason, our project result has an impact to some of the existing templates. Therefore, the augmentation of the existing templates instead of the creation of a new timing template reduces dependencies in the meta-model among templates.

1.5 Scope

The primary purpose of the timing extensions is to support constructing embedded real-time systems that satisfy given timing requirements and to perform timing analysis/validations of those systems once they have built up.

The AUTOSAR Timing Extensions provide a timing model as specification basis for a contract based development process, in which the development is carried out by different organizations in different locations and time frames. The constraints entered in the early phase of the project (when corresponding solutions are not developed yet) shall be seen as extra-functional requirements agreed between the development partners. In such way the timing specification supports a top-down design methodology. However, due to the fact that a pure top-down design is not feasible in most of the cases (e.g. because of legacy code), the timing specification allows the bottom-up design methodology as well.

The resulting overall specification (AUTOSAR Model *and* Timing Extensions) shall enable the analysis of a system's timing behavior and the validation of the analysis results against timing constraints. Thus, timing properties required for the analysis shall be contained in the timing augmented system model. Such timing properties can be found all across AUTOSAR. For example the System Template provides means to configure and specify the timing behavior of the communication stack. Furthermore the execution time of an executable can be specified. In addition, the overall specification shall provide means to describe timing constraints. A timing constraint defines a restriction for the timing behavior of the system (e.g. bounding the maximum latency from sensor sampling to actuator access). Timing constraints are added to the system model using the AUTOSAR Timing Extensions. Constraints, together with the result of timing

analysis, are considered during the validation of a system's timing behavior, when a nominal/actual value comparison is performed.

Note: The timing specification shall enable the analysis and validation of an AUTOSAR system's timing behavior. However, the specification of analysis and validation **results** (e.g. the maximum resource load of an ECU, etc.) is not addressed in this document.

1.6 Document Conventions

Technical terms are typeset in mono spaced font, e.g. `PortPrototype`. As a general rule, plural forms of technical terms are created by adding "s" to the singular form, e.g. `PortPrototype`s. By this means the document resembles terminology used in the AUTOSAR XML Schema.

This document contains constraints in textual form that are distinguished from the rest of the text by a unique numerical constraint ID, a headline, and the actual constraint text starting after the ⌈ character and terminated by the ⌋ character.

The purpose of these constraints is to literally constrain the interpretation of the AUTOSAR meta-model such that it is possible to detect violations of the standardized behavior implemented in an instance of the meta-model (i.e. on M1 level).

Makers of AUTOSAR tools are encouraged to add the numerical ID of a constraint that corresponds to an M1 modeling issue as part of the diagnostic message issued by the tool.

The attributes of the classes introduced in this document are listed in form of class tables. They have the form shown in the example of the top-level element AUTOSAR:

Please note that constraints are not supposed to be enforceable at any given time in an AUTOSAR workflow. During the development of a model, constraints may legitimately be violated because an incomplete model will obviously show inconsistencies.

However, at specific points in the workflow, constraints shall be enforced as a safeguard against misconfiguration.

The points in the workflow where constraints shall be enforced, sometimes also known as the "binding time" of the constraint, are different for each model category, e.g. on the classic platform, the constraints defined for software-components are typically enforced prior to the generation of the RTE while the constraints against the definition of an Ecu extract shall be applied when the Ecu configuration for the Com stack is created.

For each document, possible binding times of constraints are defined and the binding times are typically mentioned in the constraint themselves to give a proper orientation for implementers of AUTOSAR authoring tools.

Let `AUTOSAR` be an example of a typical class table. The first rows in the table have the following meaning:

Class: The name of the class as defined in the UML model.

Package: The UML package the class is defined in. This is only listed to help locating the class in the overall meta model.

Note: The comment the modeler gave for the class (class note). Stereotypes and UML tags of the class are also denoted here.

Base Classes: If applicable, the list of direct base classes.

The headers in the table have the following meaning:

Attribute: The name of an attribute of the class. Note that AUTOSAR does not distinguish between class attributes and owned association ends.

Type: The type of an attribute of the class.

Mul.: The assigned multiplicity of the attribute, i.e. how many instances of the given data type are associated with the attribute.

Kind: Specifies, whether the attribute is aggregated in the class (`aggr` aggregation), an UML attribute in the class (`attr` primitive attribute), or just referenced by it (`ref` reference). Instance references are also indicated (`iref` instance reference) in this field.

Note: The comment the modeler gave for the class attribute (role note). Stereotypes and UML tags of the class are also denoted here.

Please note that the chapters that start with a letter instead of a numerical value represent the appendix of the document. The purpose of the appendix is to support the explanation of certain aspects of the document and does not represent binding conventions of the standard.

The verbal forms for the expression of obligation specified in [TPS_STDT_00053] shall be used to indicate requirements, see Standardization Template, chapter Support for Traceability ([4]).

The representation of requirements in AUTOSAR documents follows the table specified in [TPS_STDT_00078], see Standardization Template, chapter Support for Traceability ([4]).

1.7 Requirements Traceability

The following table references the requirements specified in AUTOSAR RS Timing Extensions [5] and denotes how each of them are satisfied by the meta-model.

Table 1.3: RequirementsTracing

2 Fundamentals

The AUTOSAR Timing Extensions provide some basic means to describe and specify timing information: Timing descriptions, expressed by *events* and *event chains*, and *timing constraints* that are imposed on these events and event chains. Both means, timing descriptions and timing constraints, are organized in *timing views* for specific purposes. By and large, the purpose of the Timing Extensions are two fold: The first purpose is to provide timing requirements that guide the construction of systems which eventually shall satisfy those timing requirements. And the second purpose is to provide sufficient timing information to analyze and validate the temporal behavior of a system.

Events: Events refer to locations in systems at which the *occurrences* of events are observed. The AUTOSAR Specification of Timing Extensions defines a set of predefined event types for such *observable locations*. Those event types are used in different *timing views* and each of these timing views correspond to one of the AUTOSAR platform views: *VFB Timing* and Virtual Functional Bus (VFB) View:

- *System Timing* and System View
- *Machine Timing* and Machine View
- *Executable Timing* and Executable View
- *Service Timing* and Service View

In particular, these events are used to specify:

- the usage and operation of services in timing views such VFB, System, Machine, Executable and Service Timing.

Event Chains: Event chains specify a causal relationship between events and their temporal occurrences. The notion of event chain enables one to specify the relationship between two events, for example when an event A occurs then the event B occurs, or in other words, the event B occurs if and only if the event A occurred before. In the context of an event chain the event A plays the role of the *stimulus* and the event B plays the role of the *response*. Event chains can be composed of existing event chains and decomposed into further event chains — in both cases the event chains play the role of *event chain segments*.

Timing Constraints imposed on Events: The notion of *Event* is used to describe that in a system, specific events occur and also at which locations in this system the occurrences are observed. In addition, an Event Triggering Constraint imposes a constraint on the occurrences of an event, which means that the event triggering constraint specifies the way an event occurs in the temporal space. The AUTOSAR Specification of Timing Extensions provides means to specify periodic and sporadic event occurrences, as well as event occurrences that follow a specific pattern (burst, concrete, and arbitrary pattern).

Timing Constraints imposed on Event Chains: Like event triggering constraints impose timing constraints on events and their occurrences; the latency and synchronization timing constraints impose constraints on event chains. In the former case, a constraint is used to specify a reaction and age, for example if a stimulus event occurs then the corresponding response event shall occur not later than a given amount of time. And in the latter case, the constraint is used to specify that stimuli or response events shall occur within a given time interval (tolerance) to be said to occur simultaneous and synchronous respectively.

Additional Timing Constraints: In addition to the timing constraints that are imposed on events and event chains, the AUTOSAR Timing Extensions provide timing constraints which are imposed on *Executable Entities*, namely the *Execution Order Constraint* and *Execution Time Constraint*.

These fundamental concepts sketch the representation in the meta-model and form the basis of the descriptions in the subsequent sections.

3 Modeling

This chapter shall walk through the meta-model representation of the timing extensions in the following sub-sections.

3.1 TimingExtensions

An AUTOSAR Timing Extension model starts with the meta-class `TimingExtension` or rather, one of the sub-classes of `TimingExtension` as the top-level element. This is the owning element for all other related elements. The sub-classes of `TimingExtension` define a set of timing views as shown in Figure 3.1 and detailed in the next sub-sections. The timing views are:

- **VfbTiming**: timing information related to the interaction of `AdaptiveApplicationSwComponentType`s at VFB level (3.1.1)
- **ExecutableTiming**: timing information related to an `Executable` (3.1.2)
- **SystemTiming**: timing information related to a `System`, utilizing information about topology, software deployment, and signal mapping (3.1.3)
- **ServiceTiming**: timing information related to a *service*, specifically `AdaptivePlatformServiceInstance` (3.1.4)
- **MachineTiming**: timing information related to a `Machine` (3.1.5)

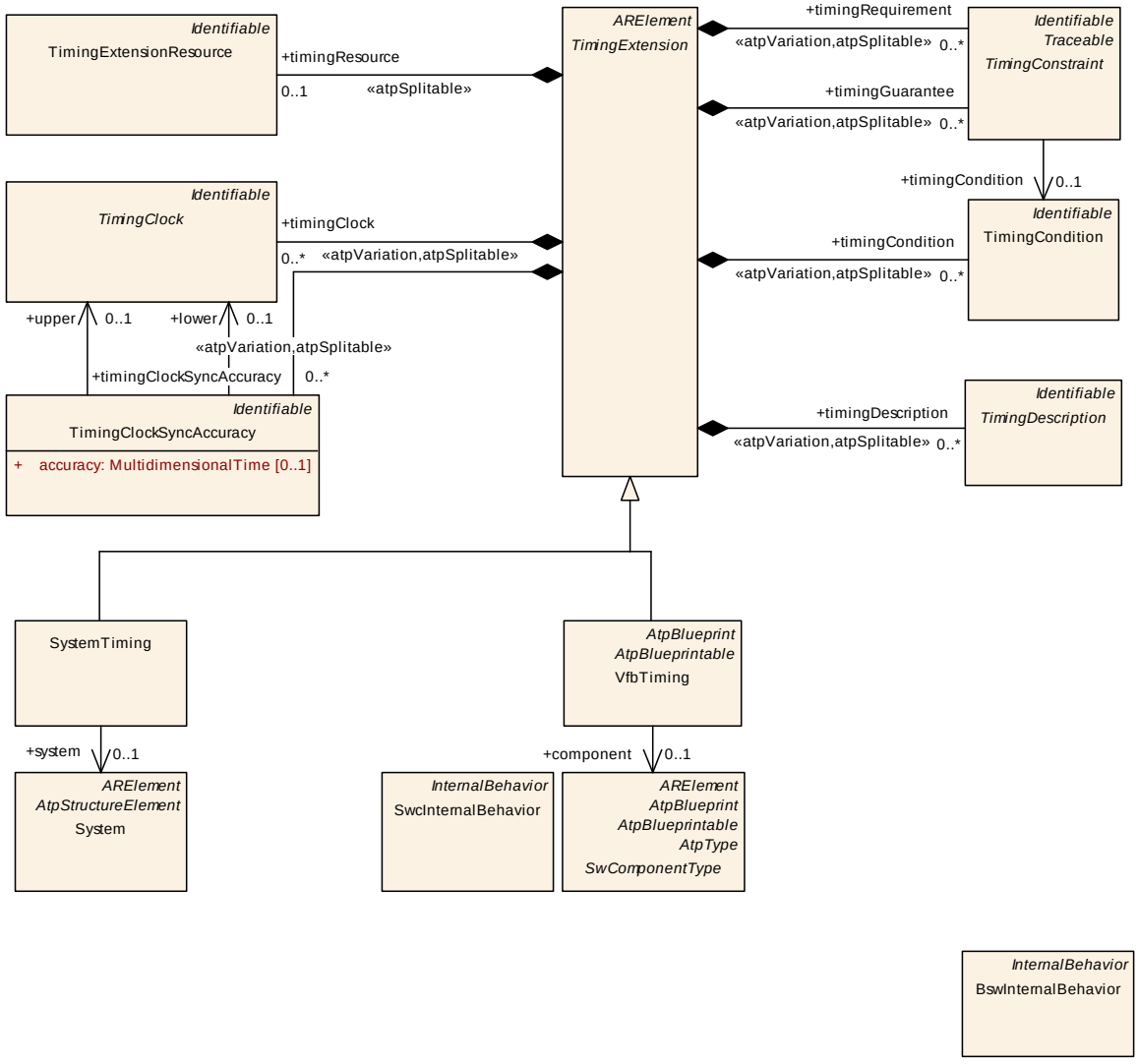


Figure 3.1: Timing Extensions top-level view

3.1.1 VfbTiming

AUTOSAR defines the *Virtual Functional Bus* [6] as a composition of SwComponentPrototypes at a logical level, regardless of their physical distribution. On this logical level a special view can be applied for timing specification. This section describes what kind of timing specification can be applied at VFB level for a system or sub-system. Typically, end-to-end timing constraints, including (physical) sensors and actuators, shall be captured in this view, allowing an early formalization of those constraints.

Neglecting the physical distribution means that the VfbTiming view does not deal with the question, in which system context the prototype of a CompositionSwComponentType shall be implemented. An additional restriction of the VfbTiming view is present due to the black box treatment of software components. For these mentioned restrictions (irrelevance of the physical distribution, black box view), TimingDescrip-

tions at VFB level should only refer to SwComponentTypes, PortPrototypes and their connections.

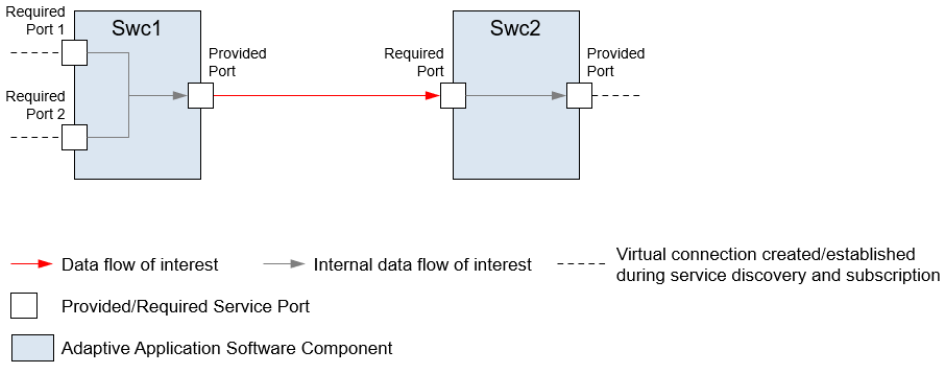


Figure 3.2: Example: Data flow in the scope of the VfbTiming view

The VfbTiming view is applicable for different system granularities. The smallest granularity is the investigation of a single SwComponentType without any contextual embedding. Here, a timing description can only refer to relations between a component's RPortPrototypes and the same component's PPortPrototypes.

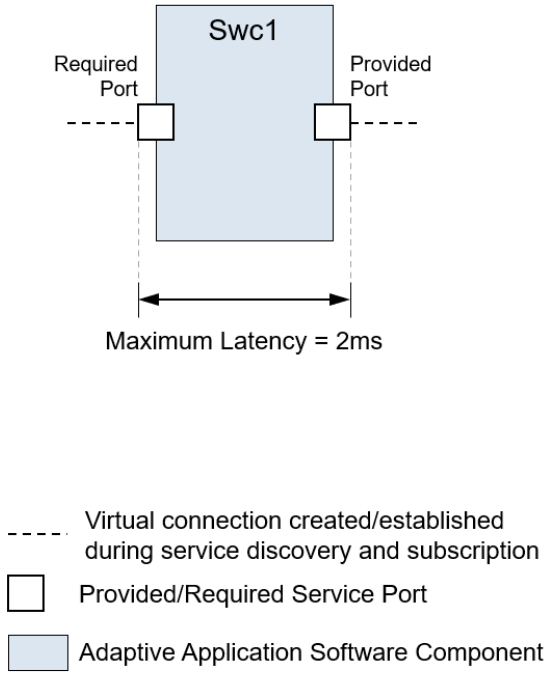


Figure 3.3: Example: Latency requirement

As an example, consider the timing constraint illustrated in Figure 3.3: "From the point in time, where the value is received by AA named *Swc1*, until the point in time, where the newly calculated data value is sent via the provided service port, there shall be a

maximum latency of 2 ms". This would be attached to the timing description that refers to an AdaptiveApplicationSwComponentType called *Swc1*.

In case of a CompositionSwComponentType that itself contains other SwComponentPrototypes, the timing interrelation between different components, e.g. from one component's PPortPrototype to another component's RPortPrototype, could be of interest.

[TPS_TIMEX_00087]{DRAFT} **Purpose of VfbTiming** ⌈The element VfbTiming aggregates all timing information, timing descriptions and timing constraints related to the VFB View.⌋*(RS_TIMEX_00001)*

Table 3.1: VfbTiming

3.1.2 ExecutableTiming

[TPS_TIMEX_00064]{DRAFT} **Purpose of ExecutableTiming** ⌈The element ExecutableTiming aggregates all timing information, timing descriptions and timing constraints, that is related to the Executable View.⌋*(RS_TIMEX_00001, RS_TIMEX_00024)*

Table 3.2: ExecutableTiming

3.1.3 SystemTiming

At system level a special prototype of a `CompositionSwComponentType`—the `RootSwCompositionPrototype`—is instantiated. This prototype, the chosen hardware topology and other artifacts are used as input to the task dealing with the deployment of software components onto machines in order to configure the system. The main configuration result is the mapping of software components to Machines and in further steps the resulting communication matrix is created. This information is aggregated in the `System` description.

The `SystemTiming` view is used to provide timing information at system level. As an extension, it can be attached to a `System`. As the `System` description aggregates all the information about `AdaptiveApplicationSwComponentType`s, it is possible to use the same concepts that are available in the view `VfbTiming` also in this timing view. The difference is the specific system context that defines the validity of timing information at system level. Without knowledge of the mapping of software components to a target hardware respectively ECU, only a generic platform independent description can be provided.

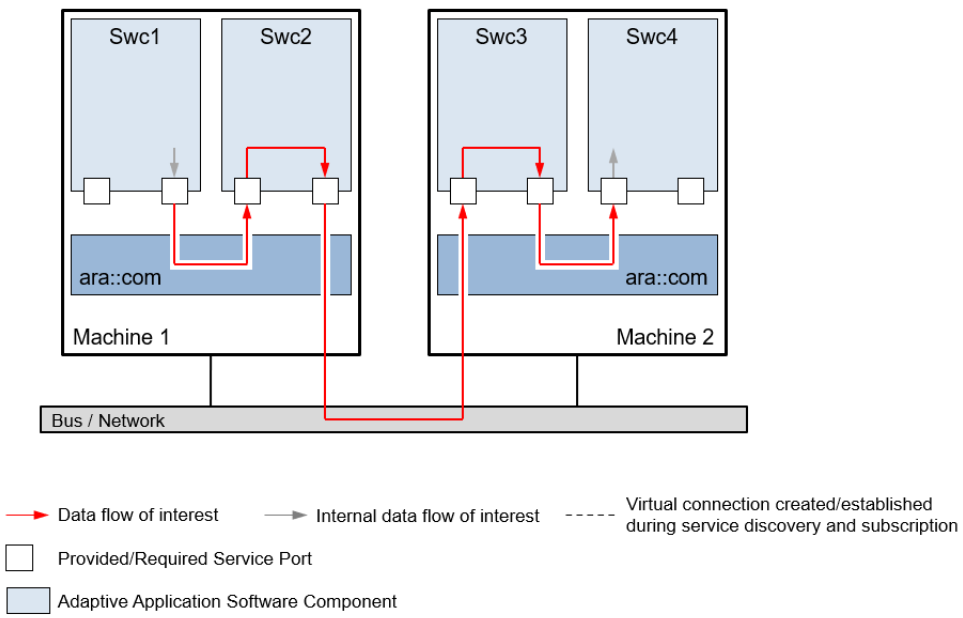


Figure 3.4: Example: Data flow in the scope of System Timing view

In addition, a timing description in system view refers to the concrete communication of software components that only was represented as abstract connectors in `VfbTiming` view. Due to the software mapping, now communication is either local communication within a machine, or remote communication between machines across a communication bus. A system-specific timing description thus can refer to signals and frames sent across a physical network.

[TPS_TIMEX_00088]{DRAFT} **Purpose of `SystemTiming`** ⌈The element `SystemTiming` aggregates all timing information, timing descriptions and timing constraints, that is related to the System View.⌋ *(RS_TIMEX_00001)*

Table 3.3: SystemTiming

3.1.4 ServiceTiming

[TPS_TIMEX_00065]{DRAFT} **Purpose of `ServiceTiming`** ⌈The element `ServiceTiming` aggregates all timing information, timing descriptions and timing constraints, that is related to the Service View.⌋ *(RS_TIMEX_00001, RS_TIMEX_00024)*

Table 3.4: ServiceTiming

3.1.5 MachineTiming

[TPS_TIMEX_00063]{DRAFT} **Purpose of `MachineTiming`** ⌈The element `MachineTiming` aggregates all timing information, timing descriptions and timing constraints, that is related to the Machine View.⌋ *(RS_TIMEX_00001, RS_TIMEX_00024)*

Table 3.5: MachineTiming

3.2 Formal specification of timing behavior

Compared to the specification of a system's functional behavior, the specification of its timing behavior requires additional information to be captured. Not only the eventual occurrence of events but also their exact timing or the concurrency of various events become important. Therefore, in the specification of timing extensions for AUTOSAR, the *event* is the basic entity. This event is used to refer to an observable behavior within a system at a certain point in time.

Having to deal with different abstraction levels and views (see chapter 3.1), and in order to avoid semantic confusion with existing concepts, a new abstract type `TimingDescriptionEvent` (see section 3.5.2) is introduced as a formal basis for the timing extensions. Depending on the model entity and the associated observable behavior, specific timing events are defined and linked to the different views.

For the analysis of a system's timing behavior usually not only single events but also the correlation of different events is of fundamental importance. To relate timing events to each other, a further concept called TimingDescriptionEventChain (see section 3.5.1) is introduced. Hereby, it is important to note that for the referenced events of an event chain a functional dependency is implicitly assumed. This means that an event of a chain somehow causes subsequent chain events.

Based on events and event chains, it is possible to express various specific timing constraints derived from the abstract type TimingConstraint. These timing constraints specify the expected timing behavior. As timing constraints shall be valid independently from implementation details, they are also expressed on a abstract level by referencing the above introduced formal basis of TimingDescriptionEvents and TimingDescriptionEventChains.

Thus, by means of events, event chains and timing constraints defined on top of these, a separate central timing specification can be provided, decoupling the expected timing behavior from the actually implemented behavior. This approach supports timing contracts for AUTOSAR systems in a top-down as well as bottom-up approach.

3.3 Specifying Time Sets

Sometimes it is necessary to specify that there are several alternatives with regard to timing requirements. For example, quite often it is reasonable to specify that a process shall be periodically activated either at 1ms, 2ms, 5ms, 8ms, or 10ms. In other words, it is perfectly fine to decide that the process is activated every 8ms. Indeed, it is allowed to activate the process either at 1ms, 2ms, 5ms, 8ms, or 10ms. Hence, there should be a means to specify such time sets which contain all allowed timings, like in case of activating a process at {1, 2, 5, 8, 10} ms.

For the purpose of specifying time sets the timing extensions utilize the ”Variant Handling” capabilities specified and described in [7].

3.4 Timing Conditions

Please refer to [8] chapter "Timing Conditions".

3.5 TimingDescription

The TimingDescription is an abstract class which provides the base for the two abstract sub-classes TimingDescriptionEventChain and TimingDescriptionEvent - which further provide the base for the respective concrete event types as shown in Figure 3.5. These are detailed in the next sections.

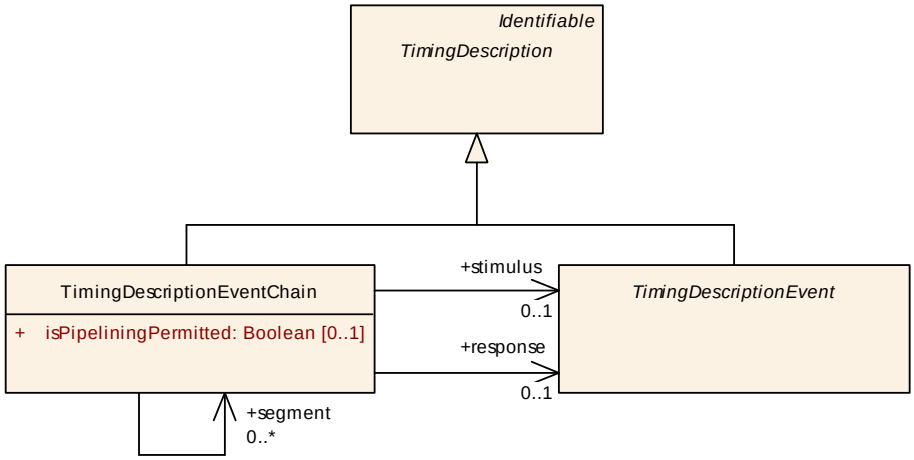


Figure 3.5: TimingDescription

3.5.1 TimingDescriptionEventChain

A timing event chain describes a causal order for a set of functionally dependent timing events. Each event chain defines at least the relationship between two differing events, its *stimulus* and *response* [constr_4515].

This means that if the stimulus event occurs then the response event occurs after or in other words the response event follows if and only if the stimulus event occurred before.

[TPS_TIMEX_00070]{DRAFT} **Purpose of `TimingDescriptionEventChain`** ⌈The element `TimingDescriptionEventChain` is used to specify a causal relationship between timing description events and their occurrences during the runtime of a system.⌋*(RS_TIMEX_00001, RS_TIMEX_00004, RS_TIMEX_00005)*

Thus, by means of an event chain, the correlation between a stimulation of a system and its corresponding response can be explicitly described, and used as a formalized definition of the scope for timing constraints. This is important, because timing constraints refer to a specific part of the overall system's timing and need clear validity semantics.

[constr_4581]{DRAFT} **Specifying stimulus and response in `TimingDescriptionEventChain`** ⌈The references between `TimingDescriptionEventChain` and `TimingDescriptionEvent` playing the role `stimulus` and `response` shall not reference the same `TimingDescriptionEvent`.⌋*()*

Depending on the value of the `category`s of the `TimingDescriptionEventChain`, it may be used in different use-cases.

[TPS_TIMEX_00095]{DRAFT} **Standardized `category`s of `TimingDescriptionEventChain` in Adaptive Platform** ⌈AUTOSAR standardizes the following `category`s of `TimingDescriptionEventChain` and their semantics:

- undefined: as per STANDARD
- STANDARD: No specific semantics are imposed on the TimingDescriptionEventChain. It indicates the standard behavior.
- SL_LET_INTERVAL: The TimingDescriptionEventChain represents a SL-LET interval

⌋()

Please note constraints: [constr_4515], [constr_4560] and specification items: [TPS_TIMEX_00111], [TPS_TIMEX_00114] in [8] shall apply here also.

3.5.1.1 Segments

[constr_4582]{DRAFT} **Specifying event chain segments** ⌈If a TimingDescriptionEventChain consists of further event chain segments then at least one sequence of event chain segments shall exist from the event chain's stimulus to the response.⌋()

[constr_4583]{DRAFT} **Referencing no further event chain segments** ⌈If a TimingDescriptionEventChain is not subdivided in further event chain segments, then the reference playing the role of segment shall reference this TimingDescriptionEventChain. In other words, an event chain without any event chain segments shall reference itself.⌋()

[constr_4584]{DRAFT} **Specifying stimulus event and response event of first and last event chain segment** ⌈The stimulus event of the first event chain segment and the response event of the last event chain segment shall reference the stimulus and response of the parent event chain the event chain segments directly belong to.⌋ ()

3.5.1.2 Approach

The following subsections describe how to structure event chains for systems. Depending on the pre-conditions two different approaches can be distinguished: top-down (decomposition) and bottom-up (composition).

The decomposition respectively composition of event chains can be performed according to the software component hierarchy, but does not necessarily have to follow this hierarchy. The primary purpose is to increase respectively decrease granularity of the timing descriptions.

Note that event chains are used in all AUTOSAR timing views and any composition and decomposition of event chains can be done across various AUTOSAR timing views.

3.5.1.2.1 Decomposition

In a first step the time critical path in the system is identified. This means that a causal relationship between a stimulus event and response event is described by an event chain. For this event chain a timing constraint is specified describing the time budget. The second step is to decompose this event chain into event chain segments which implies that the given time budget gets split — decomposed —, too.

Since event chain segments are event chains as well, these event chain segments can be subject to further decomposition.

Figure 3.6 shows a time critical path between the event "requesting the brake pedal position" (*Stimulus*) and the event "making available the determined vehicle speed" (*Response*). This event chain (*EC*) is subject to a timing constraint, namely a `LatencyTimingConstraint`, and is budgeted accordingly. For example, the time budget for the event chain *EC* is constrained by a maximum latency of 2 ms.
In subsequent steps of the development and with deeper knowledge about the system's dynamics, this event chain and its time budget can be split across the system's components. This results in the event chain segments *EC1*, *EC2* and *EC3* and their appropriate time budgets. The sum of these time budgets shall not exceed the given time budget of 2 ms.

3.5.1.2.2 Composition

In the first step the system is build up based on available software components including timing descriptions. In the second step available event chains are connected with each other. This results in a sequence of event chains where the response event of one event chain plays the role of the stimulus event of the subsequent event chain. In the third step, a high-level event chain is specified based on a sequence of available event chains which play the role of event chain *segments*. For this high-level event chain a time budget shall be specified. Finally, the aggregated time budget needs to be assessed if acceptable which means that the aggregated time budget shall be equal or less than the time budget of the high-level event chain.

Figure 3.6 shows the connected event chains *EC1*, *EC2* and *EC3*. For each event chain a time budget, using a `LatencyTimingConstraint`, is specified: The time budget of event chain *EC1* is 0.5 ms, of event chain *EC2* is 0.6 ms and of event chain *EC3* is 0.7 ms. The high-level event chain *EC* is a composition of the event chains *EC1*, *EC2* and *EC3*. The stimulus event of the high-level event chain is the event "requesting the brake pedal position" (*Stimulus*) and the response event of the high-level event chain is the event "making available the determined vehicle speed" (*Response*). Eventually, a time budget is assigned to the high-level event chain using a `LatencyTimingConstraint`, for example 2 ms. This value is consistent with the aggregated time budget of the event chain segments (0.5 ms + 0.6 ms + 0.7 ms = 1.8 ms).

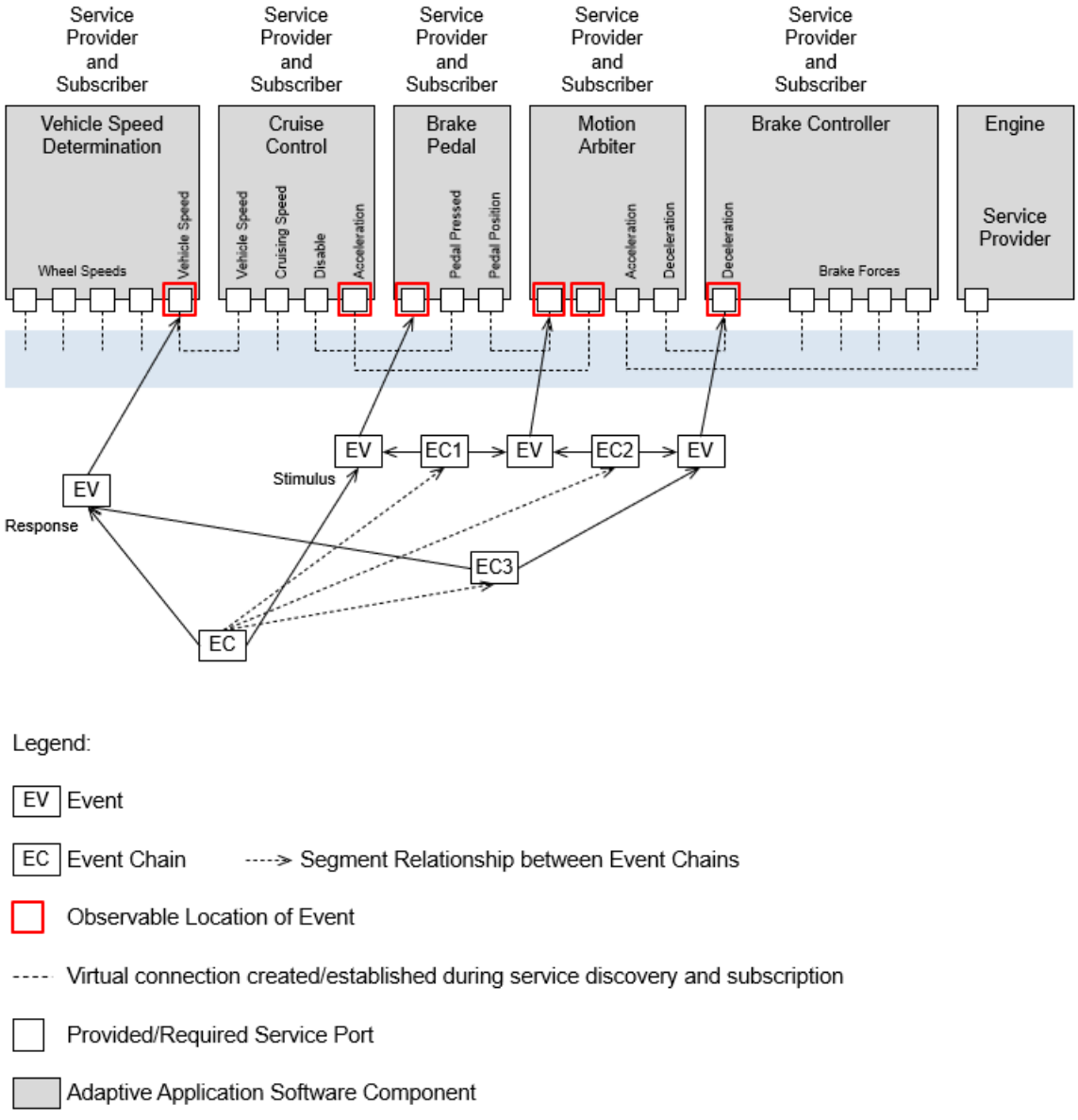


Figure 3.6: Example of a composed and decomposed event chain

3.5.1.3 Patterns

A sequence or hierarchy of event chains can form complex structures. However, if one of the aforementioned approaches is correctly followed then there is only a handful of patterns applicable. These patterns are introduced in the following with a simple example.

3.5.1.3.1 Sequence

The most frequently used pattern is the sequence of events. Such a sequence describes a succession of causally related events without an alternative path.

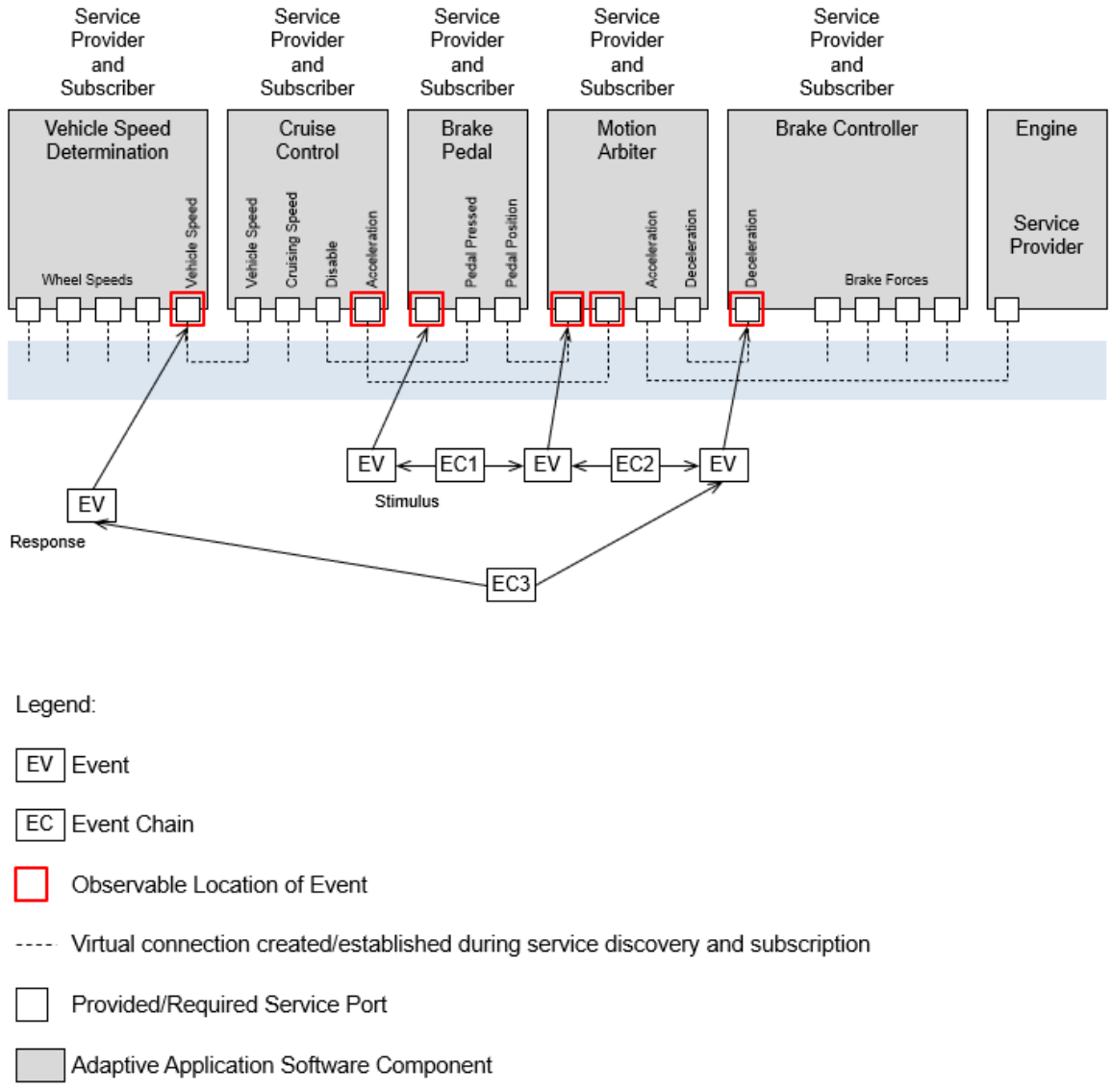


Figure 3.7: Example of the ”Sequence” pattern

An example for this pattern is depicted in Figure 3.7. The event chains *EC1* through *EC3* define a causal relationship of events observed at a port of the AA called *Brake Pedal* and a port of the AA called *Vehicle Speed Determination*.

3.5.1.3.2 Fork

The ”Fork” pattern describes the constellation where several event chains have one common stimulus event and different response events.

The pattern is illustrated in Figure 3.8, which shows a path that forks because the AA *Brake Controller* calculates the brake force value for each wheel (*EC5* through *EC8*).

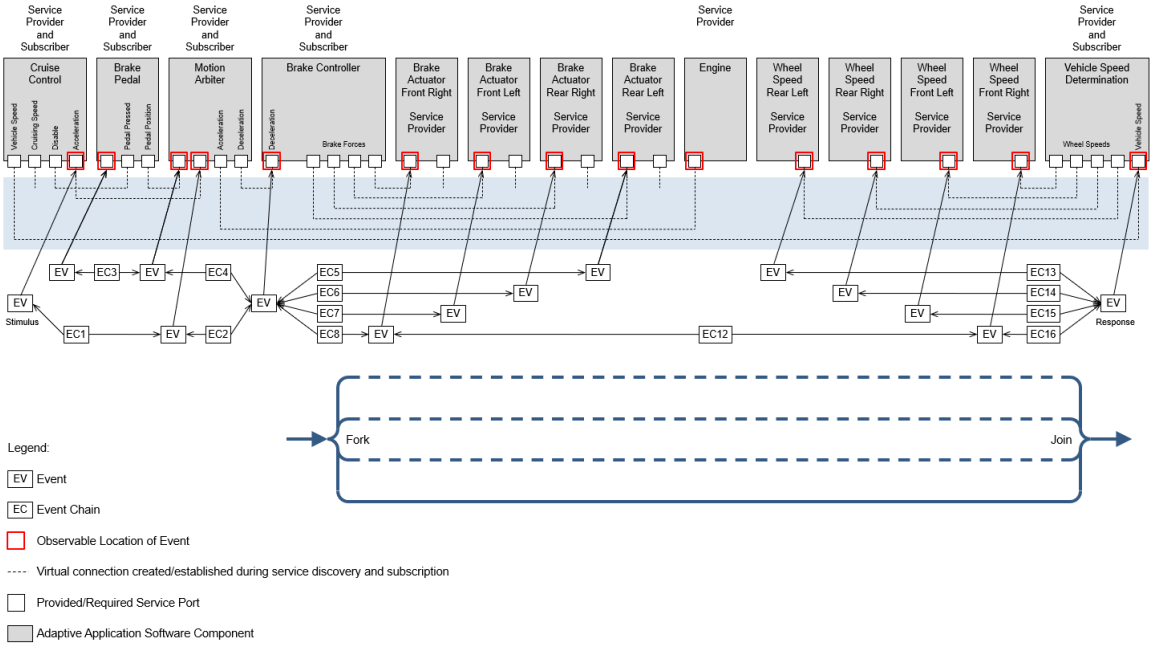


Figure 3.8: Example of the ”Fork” and ”Join” pattern

3.5.1.3.3 Join

The ”Join” pattern describes the constellation where several event chains have one common response event and different stimulus events.

The pattern is illustrated in Figure 3.8 which shows a path that joins because the AA *Vehicle Speed Determination* aggregates the wheel speed values from individual wheels (*EC13* through *EC16*).

3.5.1.3.4 Alternative

The ”Alternative” pattern describes the constellation where more than one path between a stimulus and response event exists. This implies that at least one ”Fork” is followed by at least one ”Join”.

The pattern is illustrated in Figure 3.9 which shows that an event observed at a required port of the AA *Motion Arbiter* leads to an occurrence of an event either at the port called

Deceleration of the AA *Brake Controller*, or at the port called *Acceleration* of the AA *Engine*. These alternative causal relationships are described by the event chains *EC2* and *EC4* in this figure. In either case, the deceleration or acceleration of the vehicle leads to the occurrence of an event at the provided port called *Vehicle Speed* of the AA *Vehicle Speed Determination* reporting the vehicle's speed. These alternative causal relationships are described by the event chains *EC3* and *EC5* which both reference the same response event. To fulfill the overall event chain, only one of the alternative paths shall have been occurred.

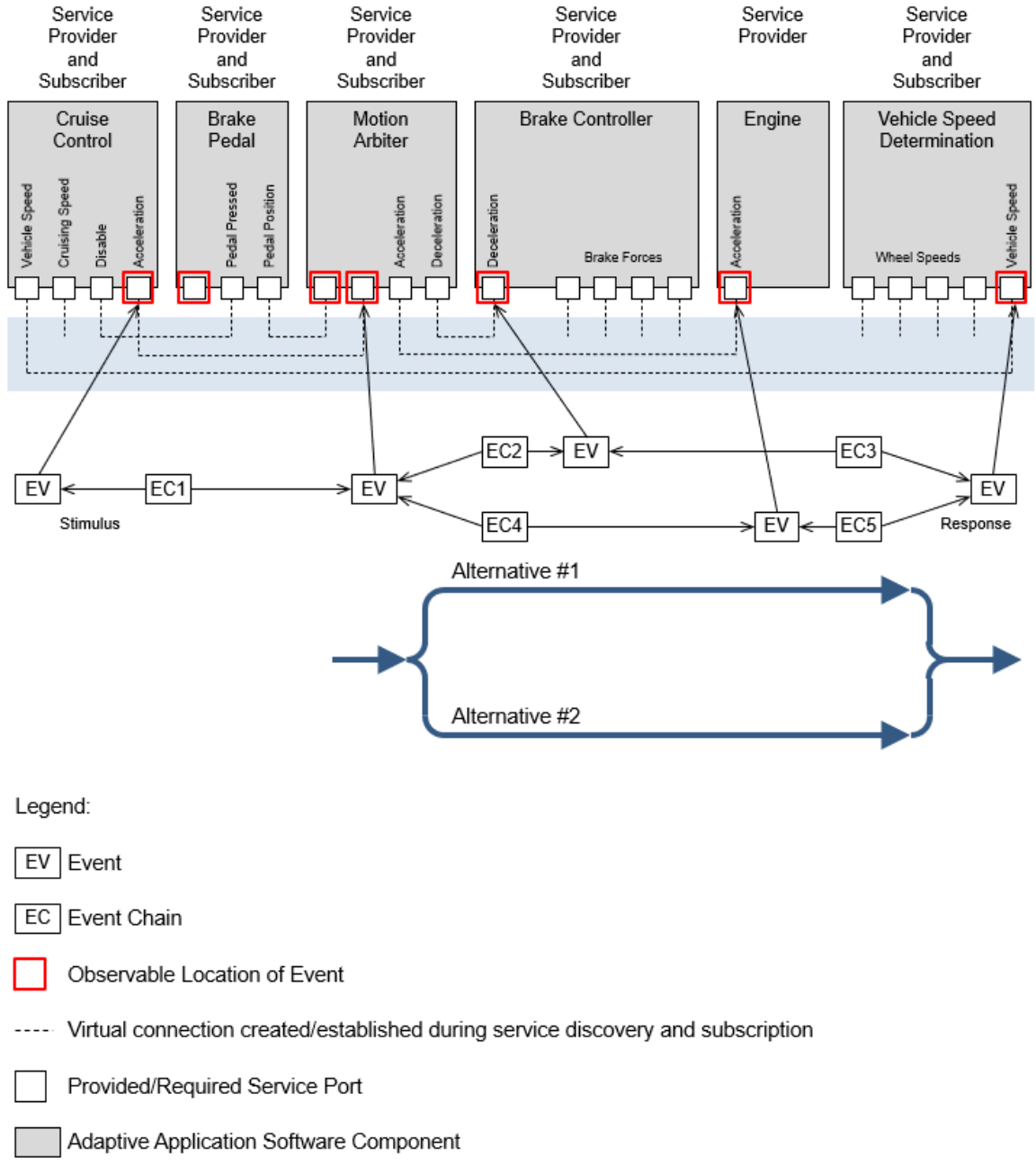


Figure 3.9: Example of the "Alternative" pattern

3.5.1.3.5 Cycle

The ”Cycle” pattern describes the constellation where a path from the response event of an event chain leads to the stimulus of this event chain.

The pattern is illustrated in Figure 3.10 which shows three event chains *EC8*, *EC12* and *EC17* forming a cycle. The stimulus event of event chain *EC8* is the response event of event chain *EC17*; and the response event of event chain *EC12* is the stimulus event of event chain *EC17*. Event chain *EC8* and *EC12* reference the same event in different roles, namely response event from event chain *EC8* perspective and stimulus event from the event chain *EC12* perspective.

Note that an event chain referencing the same event for its stimulus and its response is forbidden according to the constraint [constr_4581]. As a consequence a cycle consists of at least two event chains.

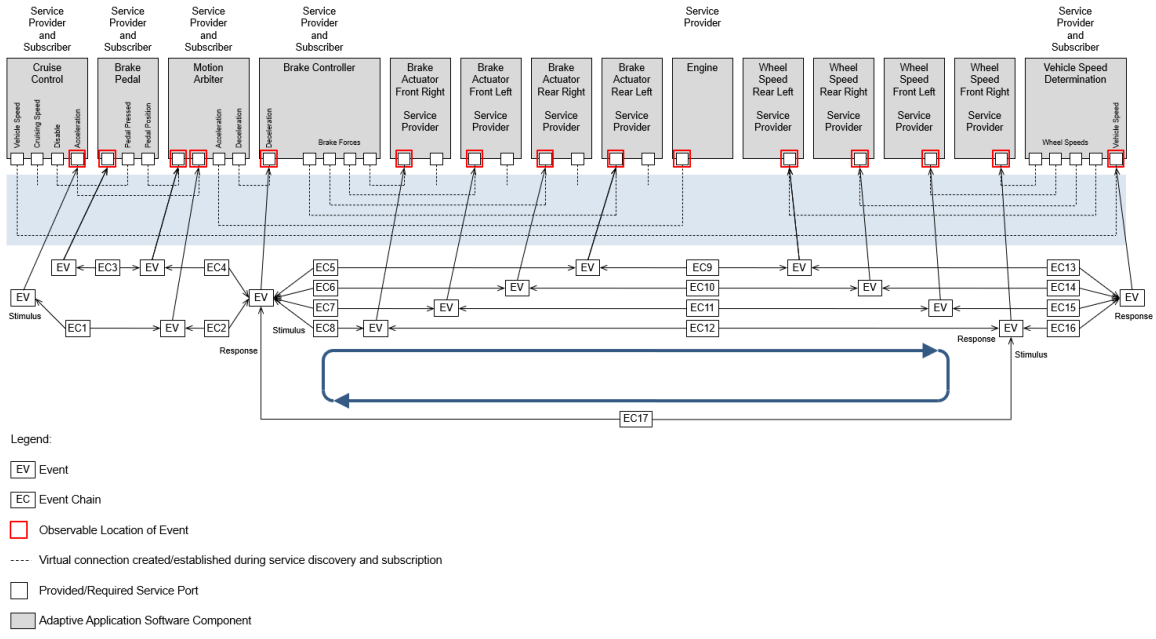


Figure 3.10: Example of the ”Cycle” pattern

3.5.2 TimingDescriptionEvent

[TPS_TIMEX_00069]{DRAFT} **Purpose of `TimingDescriptionEvent`** ⌈The element `TimingDescriptionEvent` and its specializations are used to describe the occurrences of an event which are observed at a specific location in a system during runtime respectively the operation of the system.⌋*(RS_TIMEX_00001)*

For example, this can be the start of a service or the different steps in executing an executable.

An overview of the different event types is given in Figure 3.11. These are described in more detail in the following sub-sections.

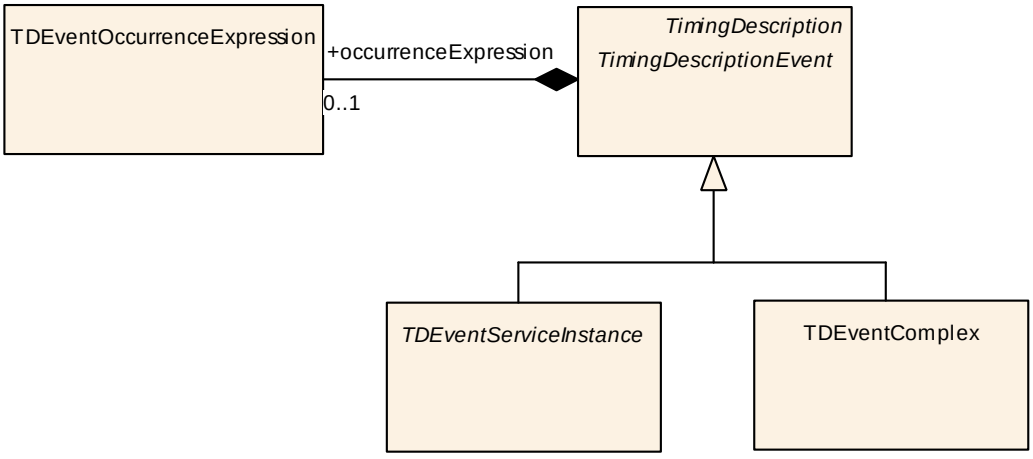


Figure 3.11: Overview of the different types of timing events

Depending on the value of the `category` of the `TimingDescriptionEvent`, it may be used in different use-cases.

[TPS_TIMEX_00094]{DRAFT} **Standardized `category`s of `TimingDescriptionEvent` in Adaptive Platform** ⌈AUTOSAR standardizes the following `category`s of `TimingDescriptionEvent` and their semantics:

- `undefined`: as per `STANDARD`
- `STANDARD`: No specific semantics are imposed on the `TimingDescriptionEvent`. It indicates the standard behavior.
- `SL_LET_RELEASE`: The `TimingDescriptionEvent` represents the release/start point of an SL-LET interval
- `SL_LET_TERMINATE`: The `TimingDescriptionEvent` represents the termination/end point of an SL-LET interval

⌋*()*

Please note constraint: [constr_4559] in [8] shall apply here also.

Also note that information regarding the occurrence of a `TimingDescriptionEvent` is described separately in 3.6.1.

3.5.2.1 TDEventVfb

[TPS_TIMEX_00082]{DRAFT} **Purpose of `TDEventVfb`** ⌈The element `TDEventVfb` and its specializations are used to describe the occurrences of an event which are observed at a specific location in the VFB view.⌋*(RS_TIMEX_00001)*

Events related to the VFB can be used during the specification of:

- VfbTiming 3.1.1
- SystemTiming 3.1.3

Table 3.6: TDEventVfb

[TPS_TIMEX_00092]{DRAFT} **Purpose of TDEventVfbPort** ⌈The element TDEventVfbPort and its specializations are used to describe the occurrences of an event which are observed at a specific location in the VFB view.⌋*(RS_TIMEX_00001, RS_TIMEX_00019)*

Table 3.7: TDEventVfbPort

[TPS_TIMEX_00093]{DRAFT} **Purpose of TDEventVfbReference** ⌈The element TDEventVfbReference is used to reference timing description events already specified in other timing views. In other words, it enables one to re-use existing timing models.⌋*(RS_TIMEX_00001, RS_TIMEX_00019)*

Table 3.8: TDEventVfbReference

[TPS_TIMEX_00083]{DRAFT} **`TDEventVariableDataPrototype` specifies events observable at sender/receiver ports** ⌈The element `TDEventVariable-DataPrototype` is used to specify events, namely the receipt and sending of variable data prototypes, observable at required and provided sender/receiver ports.⌋ *(RS_TIMEX_00001)*

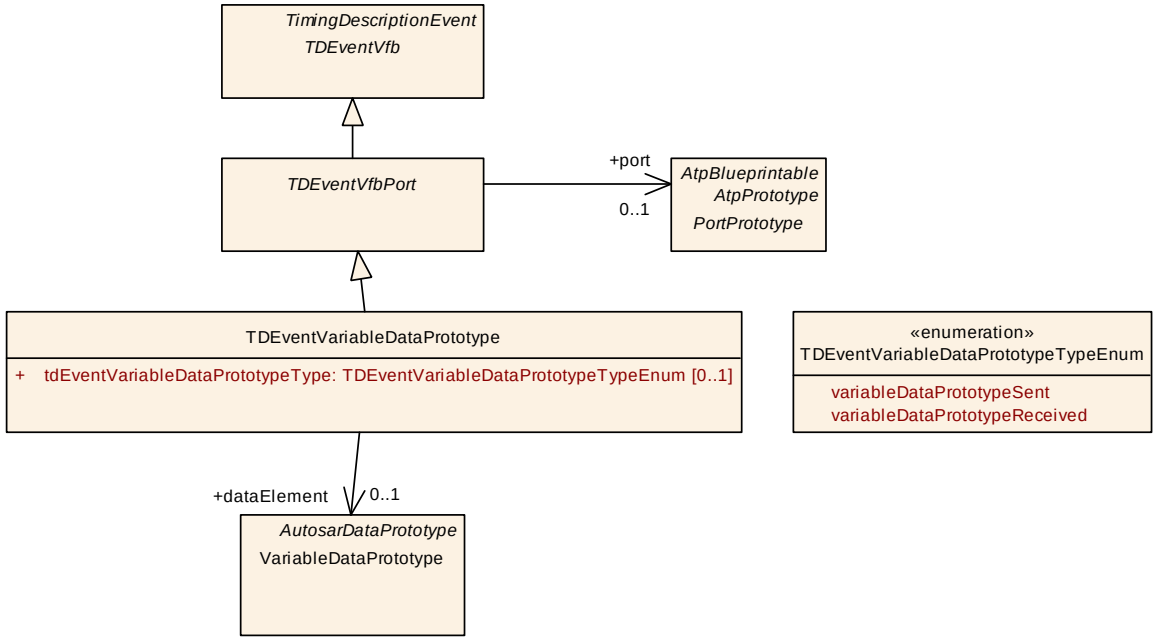


Figure 3.12: Variable Data Prototype

Table 3.9: TDEventVariableDataPrototype

Table 3.10: TDEventVariableDataPrototypeTypeEnum

[TPS_TIMEX_00084]{DRAFT} **`TDEventOperation` specifies events observable at client/server ports.** ⌈The element `TDEventOperation` is used to specify events, namely the invocation of operations and their completion, observable at required and provided client/server ports.⌋ *(RS_TIMEX_00001)*

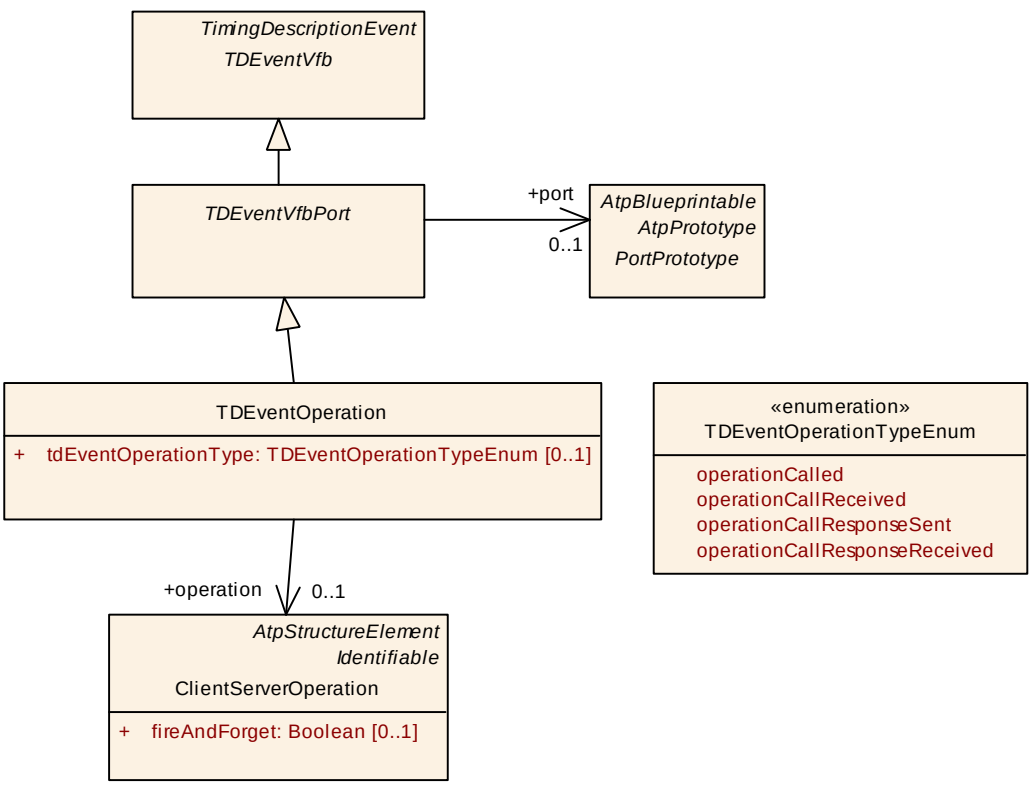


Figure 3.13: Operation

Table 3.11: TDEventOperation

Table 3.12: TDEventOperationTypeEnum

[TPS_TIMEX_00085]{DRAFT} **`TDEventModeDeclaration` specifies events observable at mode ports.** ⌈The element `TDEventModeDeclaration` is used to specify events, namely initiation and propagation of mode changes, observable at required and provided mode ports.⌋ *(RS_TIMEX_00001)*

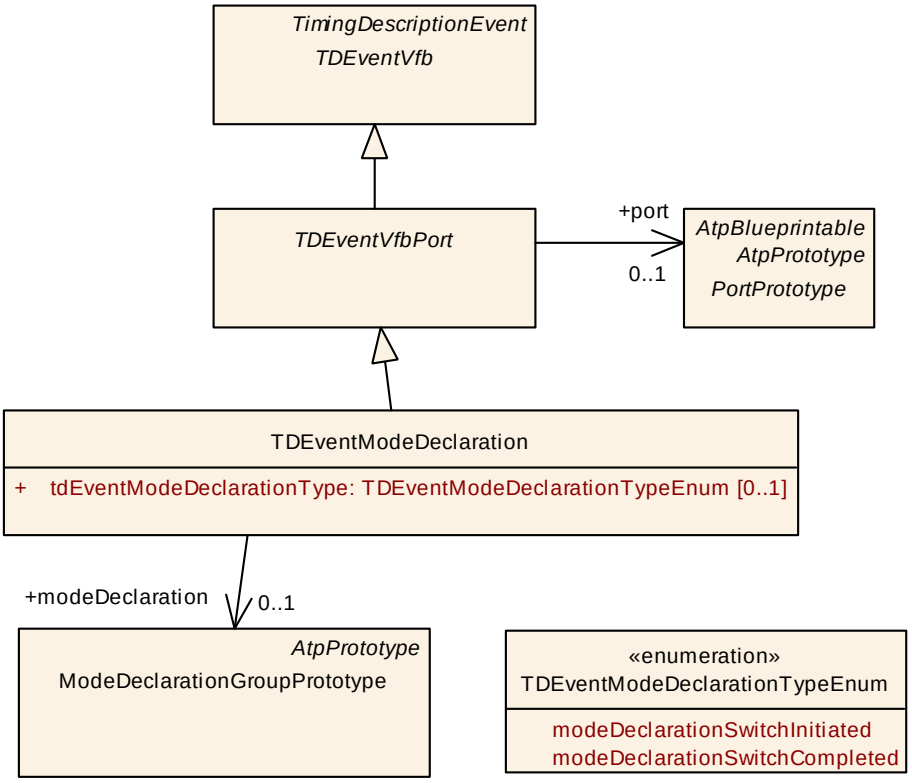


Figure 3.14: Mode Declaration

Table 3.13: TDEventModeDeclaration

Table 3.14: TDEventModeDeclarationTypeEnum

[TPS_TIMEX_00090]{DRAFT} **`TDEventTrigger` specifies events observable at trigger ports** ⌈The element `TDEventTrigger` is used to specify events, namely the activation and release of triggers, observable at required and provided trigger ports.⌋ *(RS_TIMEX_00001)*

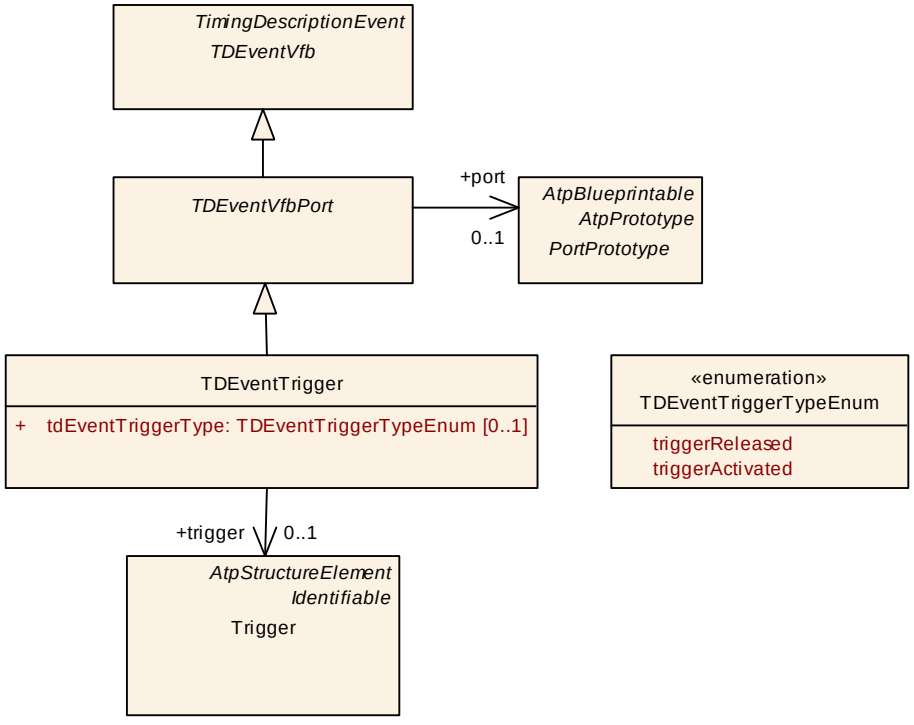


Figure 3.15: Trigger

Table 3.15: TDEventTrigger

Table 3.16: TDEventTriggerTypeEnum

3.5.2.2 TDEventServiceInstance

[TPS_TIMEX_00058]{DRAFT} **Purpose of `TDEventServiceInstance`** ⌈The element `TDEventServiceInstance` and its specializations are used to describe the occurrences of an event which are observed at a specific location in the Service view.⌋ *(RS_TIMEX_00001, RS_TIMEX_00024)*

Events related to the adaptive service can be used during the specification of:

- `VfbTiming` 3.1.1
- `SystemTiming` 3.1.3
- `ServiceTiming` 3.1.4

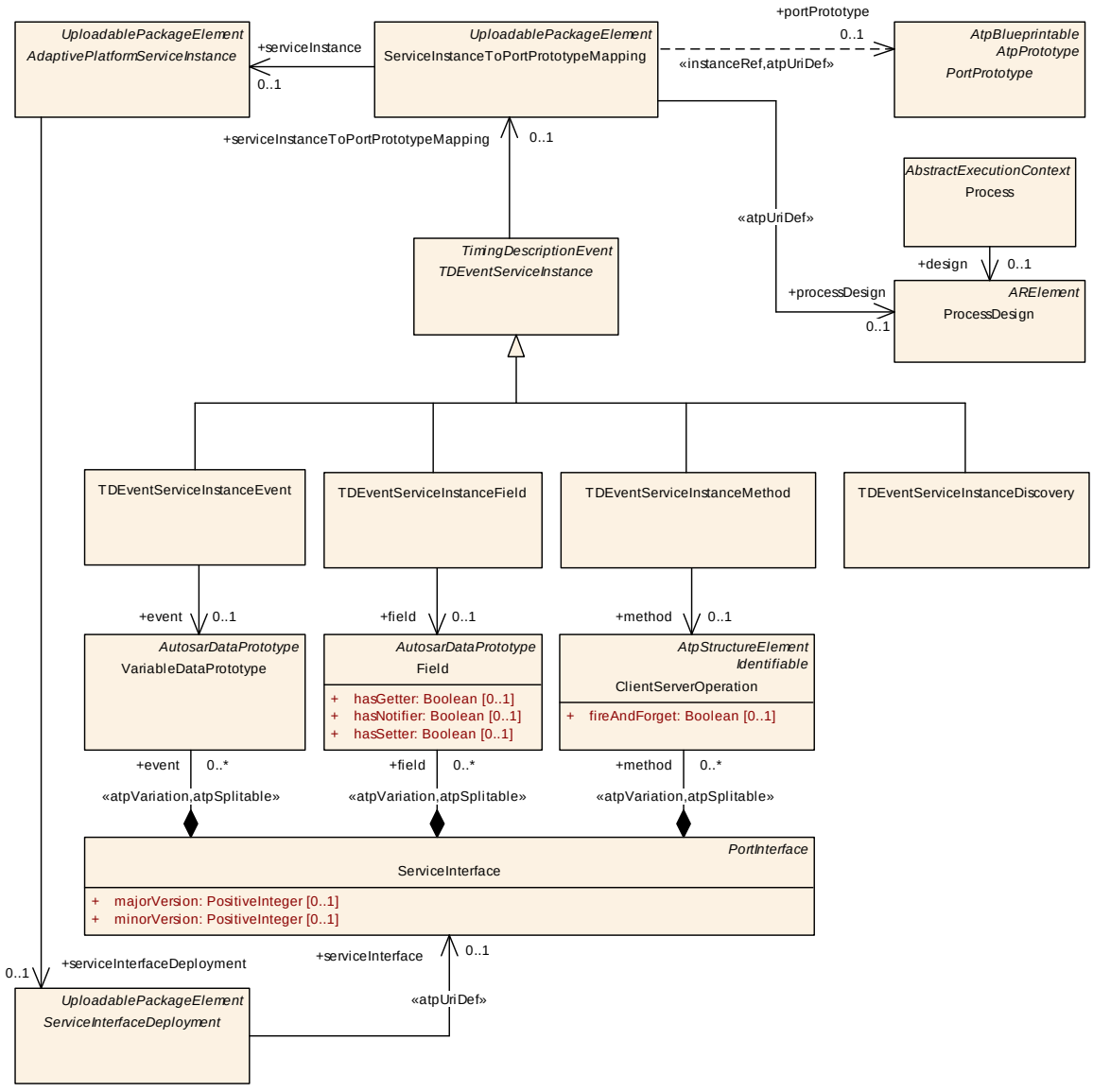


Figure 3.16: Adaptive Service events

Table 3.17: TDEventServiceInstance

[TPS_TIMEX_00059]{DRAFT} **Purpose of `TDEventServiceInstanceEvent`** ⌈The element `TDEventServiceInstanceEvent` is used to describe the occurrences of an event which are observed at a specific location in the Service view.⌋*(RS_TIMEX_00001, RS_TIMEX_00024)*

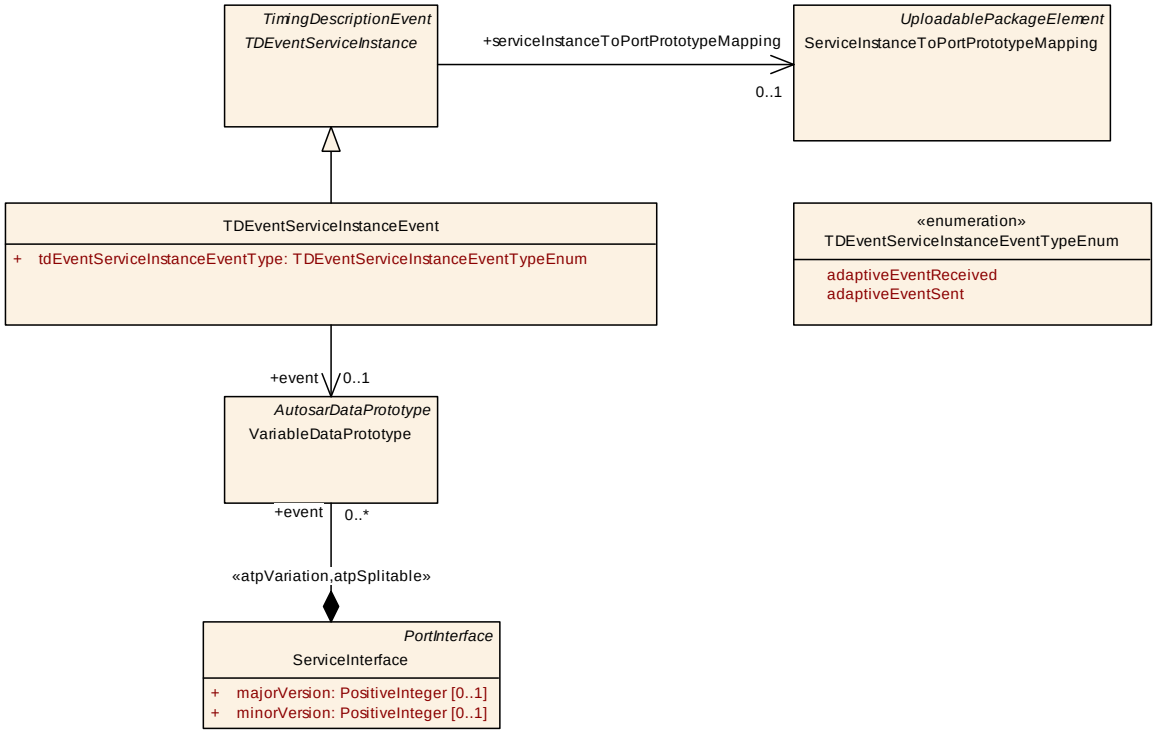


Figure 3.17: Adaptive Service Event

Table 3.18: TDEventServiceInstanceEvent

Table 3.19: TDEventServiceInstanceEventTypeEnum

[TPS_TIMEX_00060]{DRAFT} **Purpose of `TDEventServiceInstanceField`** ⌈The element `TDEventServiceInstanceField` is used to describe the occurrences of an event which are observed at a specific location in the Service view.⌋*(RS_TIMEX_00001, RS_TIMEX_00024)*

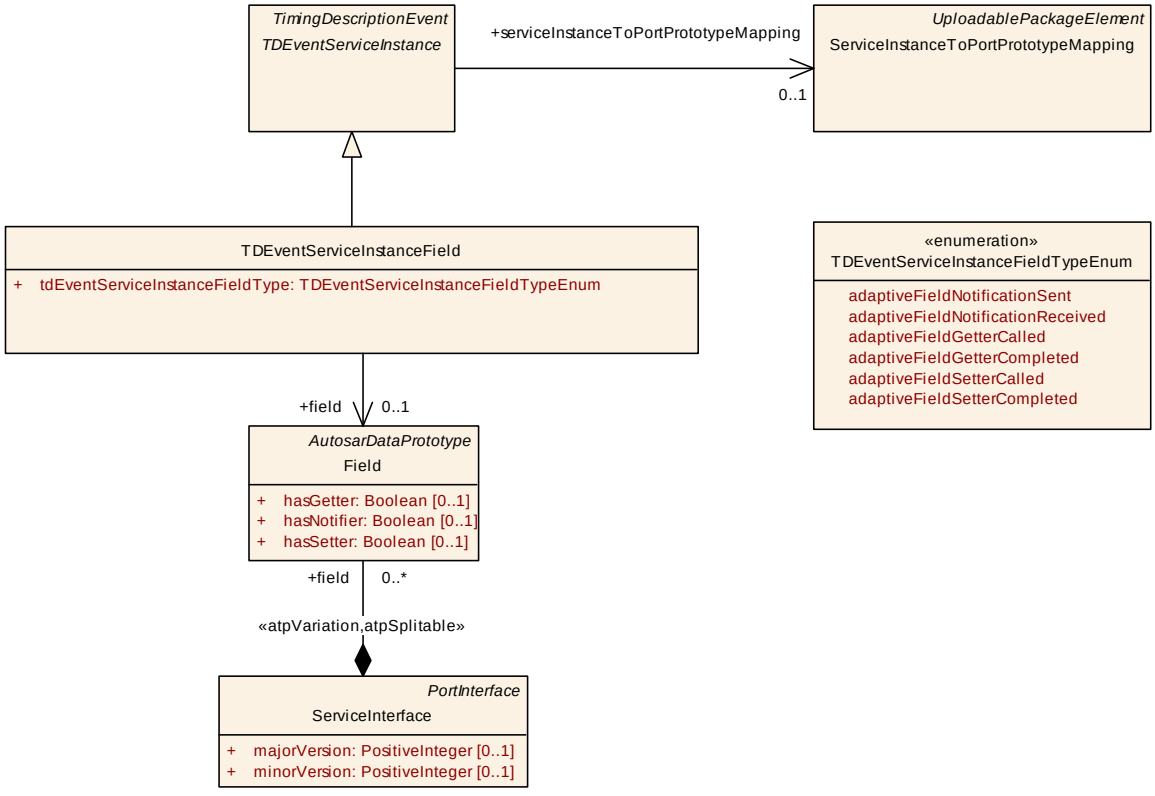


Figure 3.18: Adaptive Service Field

Table 3.20: TDEventServiceInstanceField

Table 3.21: TDEventServiceInstanceFieldTypeEnum

[TPS_TIMEX_00061]{DRAFT} **Purpose of `TDEventServiceInstanceMethod`**
⌈The element `TDEventServiceInstanceMethod` is used to describe the occurrences of an event which are observed at a specific location in the Service view.⌋
(RS_TIMEX_00001, RS_TIMEX_00024)

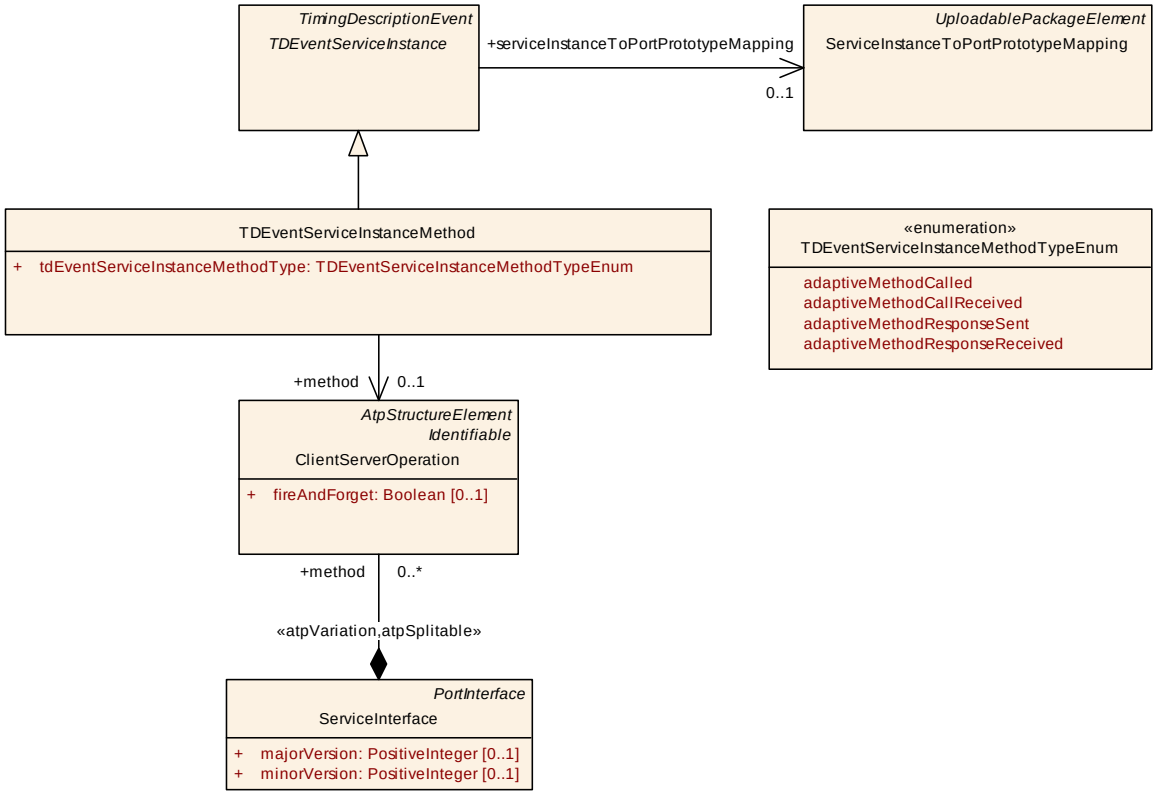


Figure 3.19: Adaptive Service Method

Table 3.22: TDEventServiceInstanceMethod

Table 3.23: TDEventServiceInstanceMethodTypeEnum

[TPS_TIMEX_00062]{DRAFT} **Purpose of `TDEventServiceInstanceDiscovery`**
⌈The element `TDEventServiceInstanceDiscovery` is used to describe the occurrences of an event which are observed at a specific location in the Service view.⌋
(RS_TIMEX_00001, RS_TIMEX_00024)

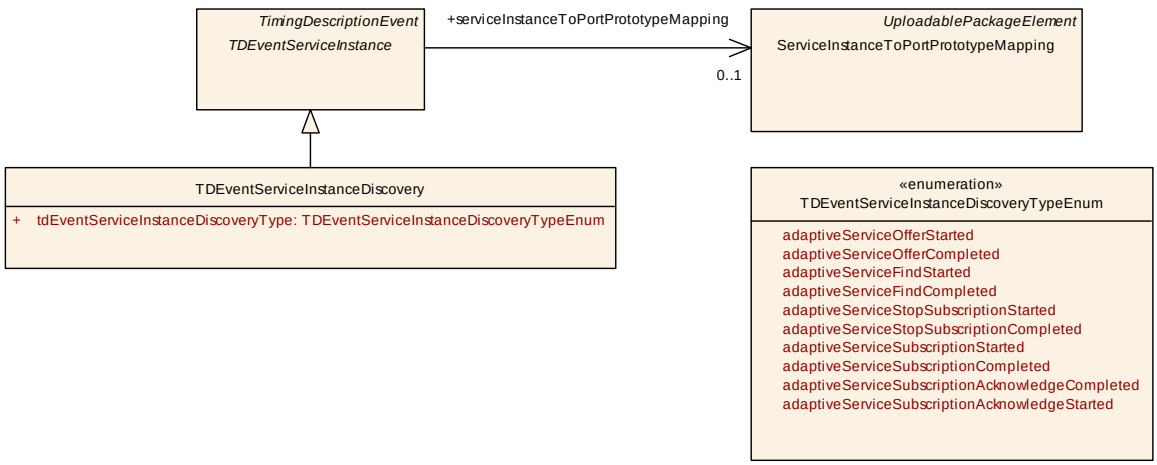


Figure 3.20: Adaptive Service Discovery

Table 3.24: TDEventServiceInstanceDiscovery

Table 3.25: TDEventServiceInstanceDiscoveryTypeEnum

3.5.2.3 TDEventComplex

[TPS_TIMEX_00086]{DRAFT} **Purpose of `TDEventComplex`** ⌈The element `TDEventComplex` is used to specify relationships between occurrences of events.⌋*(RS_TIMEX_00001)*

Complex timing events can be used during the specification of:

- `VfbTiming` 3.1.1
- `SystemTiming` 3.1.3

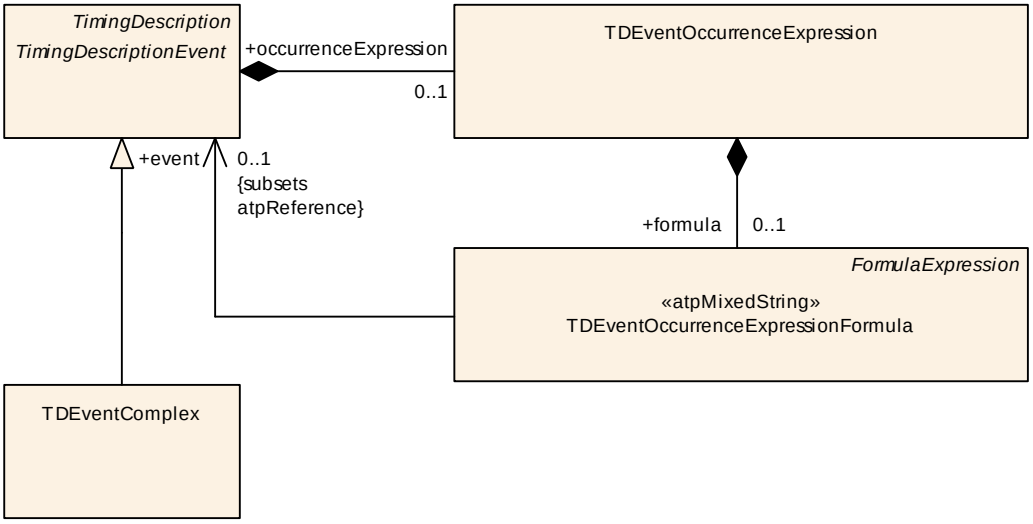


Figure 3.21: Complex timing event

Table 3.26: TDEventComplex

A complex timing event is a special observable event. In comparison to the ”atomic” events described above a complex event does not contain information about the context it references, like `VariableDataPrototype` in `TDEventVariableDataPrototype`. Instead, a complex event uses the occurrence expression to specify the context with regard to occurrences of `TimingDescriptionEvent`s as describe in the following section.

3.5.2.4 TDEventSLLET

SL-LET timing events can be used during the specification of:

- `VfbTiming` 3.1.1
- `ExecutableTiming` 3.1.2
- `SystemTiming` 3.1.3
- `MachineTiming` 3.1.5

For the remaining aspects, please refer to [8] chapter "TDEventSLLET". Specifically [TPS_TIMEX_00120] and [TPS_TIMEX_00124] apply.

3.5.2.5 Occurrence Expression Language for Timing Events

The `TimingDescriptionEvent`s mentioned in the previous sections allow to specify observable events with a well-defined context. However, sometimes the context information of the events is not sufficient, because additional conditions, like a value filter or additional stimuli, influence the occurrence. Thus, the occurrence expression provides means to overcome the limitations of atomic events.

The occurrence expression provides the ability to refine the context specification of a timing event for the following cases:

Content Filter filters occurrences of an atomic event based on the *value* of exchanged data or operation arguments.

Complex Event combines any number of atomic and complex event to specify a new timing event.

3.5.2.5.1 Specifying an Occurrence Expression

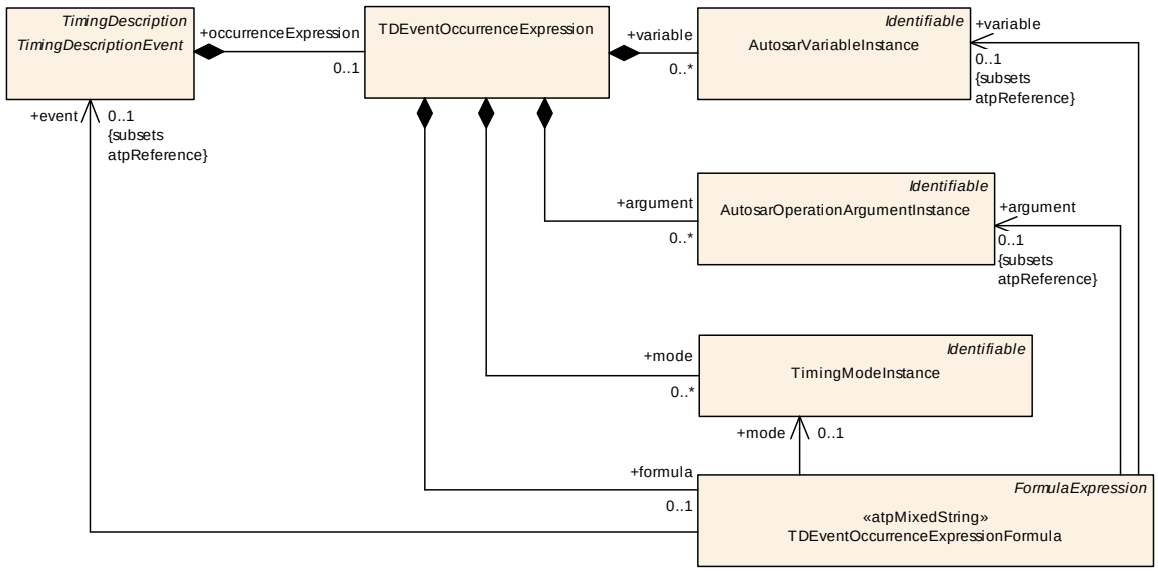


Figure 3.22: The occurrence expression

As shown in Figure 3.22, each `TimingDescriptionEvent` aggregates a `TDEventOccurrenceExpression` as optional parameter. A `TDEventOccurrenceExpression` is a container for all information required to formulate the expression. The expression itself is defined via `TDEventOccurrenceExpressionFormula` which is derived from `FormulaExpression` (see Generic Structure Template [7]). The `TDEventOccurrenceExpressionFormula` uses the capabilities of the `FormulaExpression` and adds the following functions to the expression language:

- The function *TIMEX_value*, which requires as operand either a reference to an `AutosarVariableInstance` or a reference to an `AutosarOperationArgumentInstance` whose value shall be evaluated. The return type of this function is `Numerical` (see constraint [constr_4591]).
- The function *TIMEX_occurs*, which requires as operand a reference to the `TimingDescriptionEvent` whose occurrence shall be evaluated. The return type of this function is `Boolean`. It returns TRUE if the referenced timing event occurs at the point in time the expression is evaluated.
- The function *TIMEX_hasOccurred*, which requires as operand a reference to the `TimingDescriptionEvent` whose occurrence shall be evaluated. The return type of this function is `Boolean`. It returns TRUE if the referenced timing event has occurred *at least once* before or at the same point in time the expression is evaluated.

- The function *TIMEX_timeSinceLastOccurrence*, which requires as operand a reference to the `TimingDescriptionEvent` whose occurrence shall be evaluated. The return type of this function is `Float` and the unit is seconds. It returns the time difference between the point in time of the last occurrence of the referenced event and the point in time the expression is evaluated.
- The function *TIMEX_angleSinceLastOccurrence*, which requires as operand a reference to the `TimingDescriptionEvent` whose occurrence shall be evaluated. The return type of this function is `Float` and the unit is degree. It returns the angle of the crank shaft between the point in time of the last occurrence of the referenced event and the point in time the expression is evaluated.
- The function *TIMEX_modeActive* queries the `TimingModeInstance` specified as argument. The return type of this function is `Boolean`. It returns TRUE if the specified mode declaration is *active* at the point in time the expression is evaluated, otherwise it returns FALSE.

The starting point of the time interval considered by the TIMEX functions is the point in time the measurement of the event occurrences has been started.

All operands required by the functions are references to model elements. Thus, `TDEventOccurrenceExpressionFormula` requires references to the respective elements of type `TimingDescriptionEvent`, `AutosarVariableInstance`, `AutosarOperationArgumentInstance`. Due to the atpMixedString nature of the `TDEventOccurrenceExpressionFormula` several references can be used within the occurrence expression.

[constr_4569]{DRAFT} **Restricted usage of functions** ⌈The functions *TIMEX_occurs*, *TIMEX_hasOccurred*, *TIMEX_timeSinceLastOccurrence*, *TIMEX_angleSinceLastOccurrence*, and *TIMEX_modeActive* can only be used for occurrence expressions, which are applied to events of type `TDEventComplex`.⌋*()*

[constr_4570]{DRAFT} **Application rule for the occurrence expression in `TDEventComplex`** ⌈The occurrence expression shall be specified such that it describes an *event* rather than a state. As a consequence the occurrence expression shall ensure that a complex timing event *could* only occur at the occurrence time of one of the referenced `TimingDescriptionEvent`s.⌋*()*

[constr_4571]{DRAFT} **Use references only as function operands** ⌈The references to model elements (e.g. the *timing event* reference targeting `TimingDescriptionEvent`) do have specific semantics. The usage of these references within the expression is *only* allowed as operand of the functions mentioned above.⌋*()*

[constr_4591]{DRAFT} **Use only Numericals in `TDEventOccurrenceExpression`** ⌈The target data prototype of the instance references of `variable` and `argument` shall be `Numerical`.⌋*()*

The example given below shows how to combine the functions introduced above in order to specify an occurrence expression for a complex event called *EC*.

Figure 3.23 sketches the AUTOSAR software component model of this example.

A software component named *Swc1* has a required port, called *RequiredPort*, and a provided port, called *ProvidedPort*. Both ports are sender-receiver ports. The sender-receiver port interface of the required port is called *SenderReceiverInterface1*, and consists of three data elements: The first data element is called *DE1*, the second data element is called *DE2*, and the third data element is called *DE3*. Note, that alternatively it would be also possible to define three required sender-receiver ports and the port interface of each of those ports consists of one of the data elements.

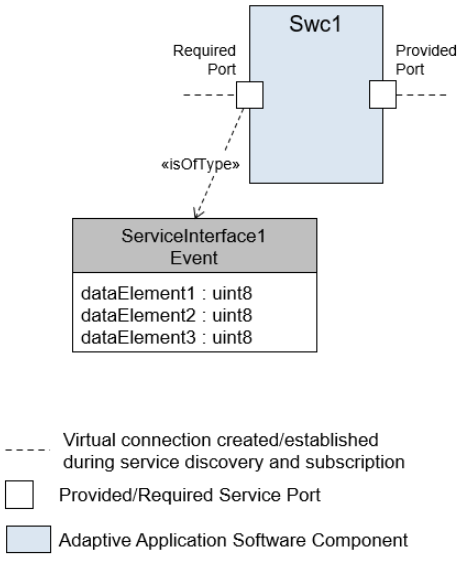


Figure 3.23: The SWC used by the Occurrence Expression Example

Since the timing is described for a software component in the Virtual Functional Bus view, the `VfbTiming` is used for specifying the corresponding timing model, namely the Virtual Functional Bus Timing View. And this timing model shall only contain timing description events related to the Virtual Functional Bus as described in section 3.5.2.1.

The complex event *EC* occurs when the following conditions are fulfilled:

Condition1 Either atomic timing event *E1* or *E2* shall occur. In this example, *E1* and *E2* are atomic timing events `TDEventVariableDataPrototype` which occur when the `VariableDataPrototype`s called *DE1* and *DE2* are received on `PortPrototype` called *Required Port* of the component called *Swc1*.

Condition2 The value of the `VariableDataPrototype` called *DE3* shall be greater than 3.

Condition3 The `VariableDataPrototype`s called *DE1* and *DE2* shall become available at the *required* `PortPrototype` called *RequiredPort* within a time interval of maximum 0.5 milliseconds.

The complex event *EC* would be described by the following occurrence expression:

```
1 // Condition 1
2 ( TIMEX_occurs( /example/expression/E1 )
```

```
3     || TIMEX_occurs( /example/expression/E2 ) )
4  // Condition 2
5  && TIMEX_value( /example/expression/EC/DE3 ) > 3
6  // Condition 3
7  && abs( TIMEX_timeSinceLastOccurrence( /example/expression/E1 ) -
8     TIMEX_timeSinceLastOccurrence( /example/expression/E2 ) ) <= 0.0005
```

Listing 3.1: Event Occurrence Filter

Due to the first condition the complex event *EC* can only occur when one of the atomic timing events *E1* or *E2* occurs at the point in time of evaluation. Thus, this expression satisfies the semantics constraint defined in [constr_4570]. Figure 3.26 shows a measurement of the event occurrences.

The corresponding AUTOSAR ARXML file fragment for the complex event *EC* has the following appearance:

Table 3.27: TDEventOccurrenceExpression

Table 3.28: TDEventOccurrenceExpressionFormula

Table 3.29: AutosarVariableInstance

Table 3.30: AutosarOperationArgumentInstance

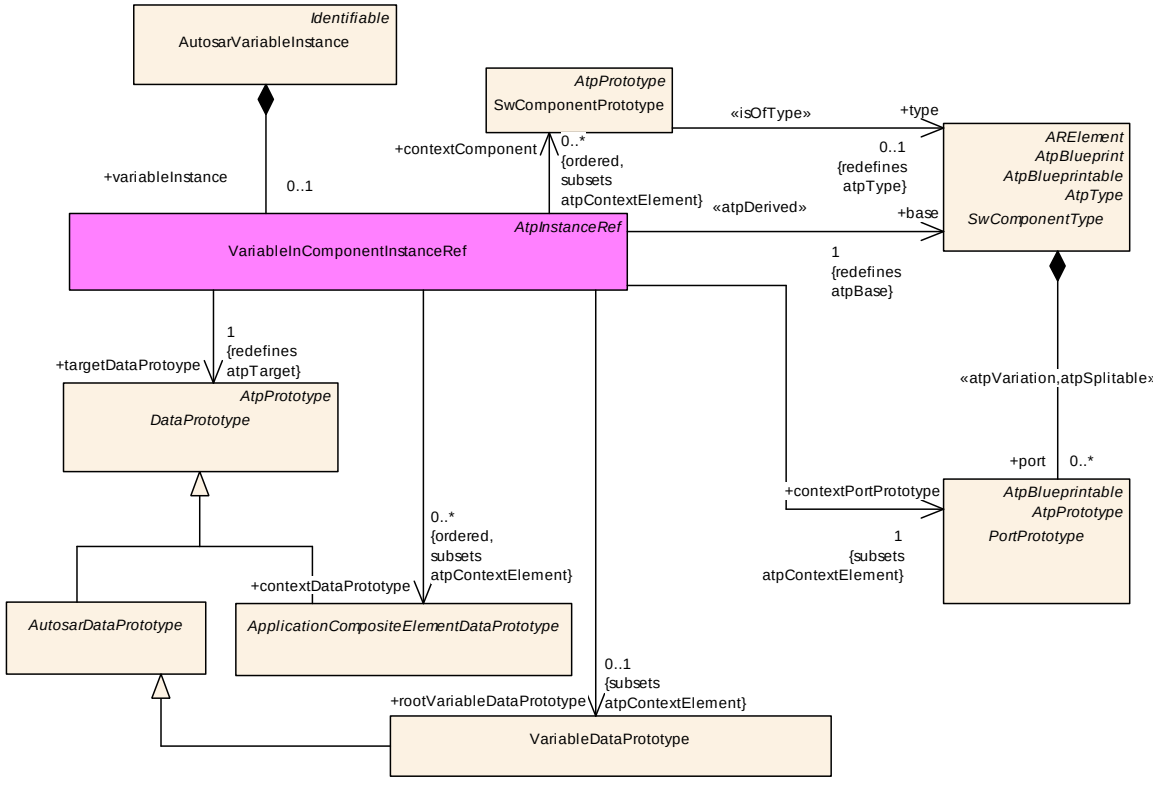


Figure 3.24: The required context information to reference a variable instance within AUTOSAR.

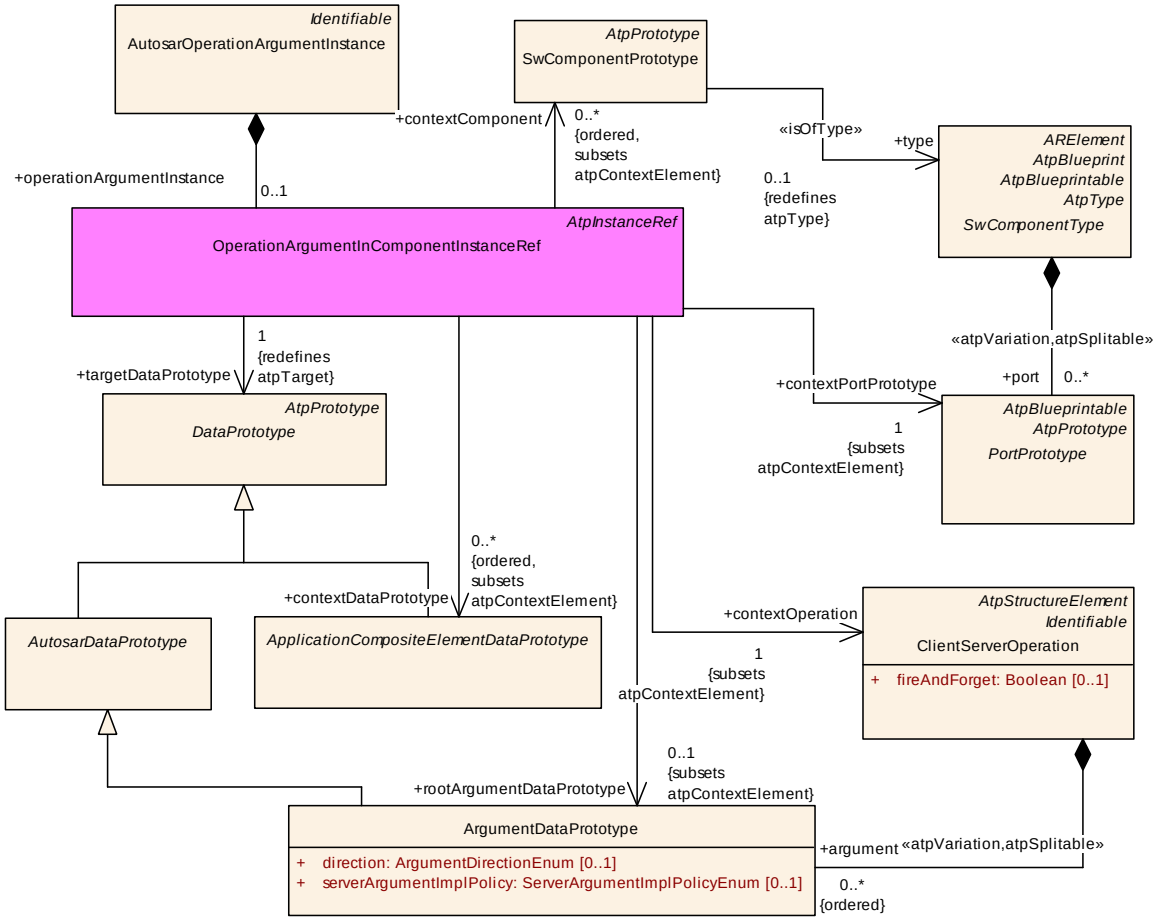


Figure 3.25: The required context information to reference an operation argument instance within AUTOSAR.

3.5.2.6 Occurrence Expression Language Syntax

The occurrence expression language is based on the syntax of the formula language defined in the Generic Structure Template [7]. It extends the language by additional functions and additional references to model elements. In the following, the implications of the extensions to the syntax are presented based on the grammar definition.

Note: The grammar defined for the formula language is not part of the listing below. It presents only the timing specific extensions of the formula language and the enhanced functions and references.

3.5.2.6.1 Interpreting an Occurrence Expression

Based on the specification mechanism described in the previous sections it is possible to use the occurrence expression formula to refine the timing specification to the intended precision. This section describes how such an occurrence expression has to be interpreted. The duty of the interpreter is to determine the occurrences of the Tim-

ingDescriptionEvent for which the occurrence expression is defined. This is done in two ways, depending on whether the occurrence expression is used as a content filter or as a complex event.

3.5.2.6.1.1 Interpreting a Content Filter

In this case, the occurrence expression is defined for an atomic event. Only the unary timing function *TIMEX_value(<reference to argument or variable>)* is allowed to be used for the content filter. On each occurrence of the atomic event the interpreter checks whether the content filter defined by the expression is fulfilled. This is done by evaluating the function *TIMEX_value* based on its operand type:

AutosarVariableInstance the value of the referenced variable is evaluated at the point in time the atomic event occurs.

AutosarOperationArgumentInstance the value of the referenced argument is evaluated at the point in time the atomic event occurs.

[constr_4592]{DRAFT} **Restricted usage of AutosarVariableInstance for Content Filter** ⌈If a content filter is defined for an atomic event then references to AutosarVariableInstances are only allowed if the atomic event is of type TDEventVariableDataPrototype. Only if such an atomic event occurs, the value of the variables can be evaluated. Thus, also the scope of the atomic event shall be the same as the AutosarVariableInstance, meaning that they shall point to the same VariableDataPrototype.⌋*()*

[constr_4572]{DRAFT} **Restricted usage of AutosarOperationArgumentInstance for Content Filter** ⌈If a content filter is defined for an atomic event then references to AutosarOperationArgumentInstances are only allowed if the atomic event is of type TDEventOperation. Only if such an atomic event occurs, the value of the operation arguments can be evaluated. Thus, also the scope of the atomic event shall be the same as the AutosarOperationArgumentInstance, meaning that they shall point to the same ClientServerOperation. Finally, references to an AutosarOperationArgumentInstance with argument direction "out" are only allowed, if the atomic event of type TDEventOperation refers either to the point in time when the operation call response has been sent (TD-EVENT-OPERATION-TYPE=OPERATION-CALL-RESPONSE-SENT) or to the point in time when the operation call response has been received (TD-EVENT-OPERATION-TYPE=OPERATION-CALL-RESPONSE-RECEIVED).⌋*()*

3.5.2.6.1.2 Interpreting a Complex Event

In this case, the occurrence expression is defined for a complex event. All features of the occurrence expression language can be used for this expression type. At a specific

point in time *t*, the interpreter evaluates the expression to determine if the complex event has occurred.

Considering the occurrence expression defined for the example given in Section 3.5.2.5.1, the interpreter "implements" a function *EC(t)* which returns TRUE, if the complex event EC occurs at time *t*:

```
EC(t) =
( TIMEX_occurs( t, /Example/Expression/E1 )
   || TIMEX_occurs( t, /Example/Expression/E2 ) )
&& TIMEX_value( t, /Example/Expression/EC/DE3 ) > 3
&& abs( TIMEX_timeSinceLastOccurrence( t, /Example/Expression/E1 ) -
   TIMEX_timeSinceLastOccurrence( t, /Example/Expression/E2 ) ) <= 0.0005
```

Since the expression satisfies [constr_4570], it shall only be evaluated at occurrence times of *E1* or *E2*, because only then the complex event *EC* can occur and the expression can return TRUE.

As shown in the sketched trace in Figure 3.26 the timing description events called *E1* and *E2* occur at different times. On the left hand side of this figure the two events occur within a time interval of 0.0005 seconds. The point in time the given occurrence expression is evaluated is the point in time the event *E2* occurs. The result of the occurrence expression at this point in time, $t_{evaluate}$ respectively t_{E2}, is TRUE. On the right hand side of this figure the two events do not occur within a time interval of 0.0005 seconds. The point in time the given occurrence expression is evaluated is the point in time the event *E1* occurs. The result of the occurrence expression at this point in time, $t_{evaluate}$ respectively t_{E1}, is FALSE.

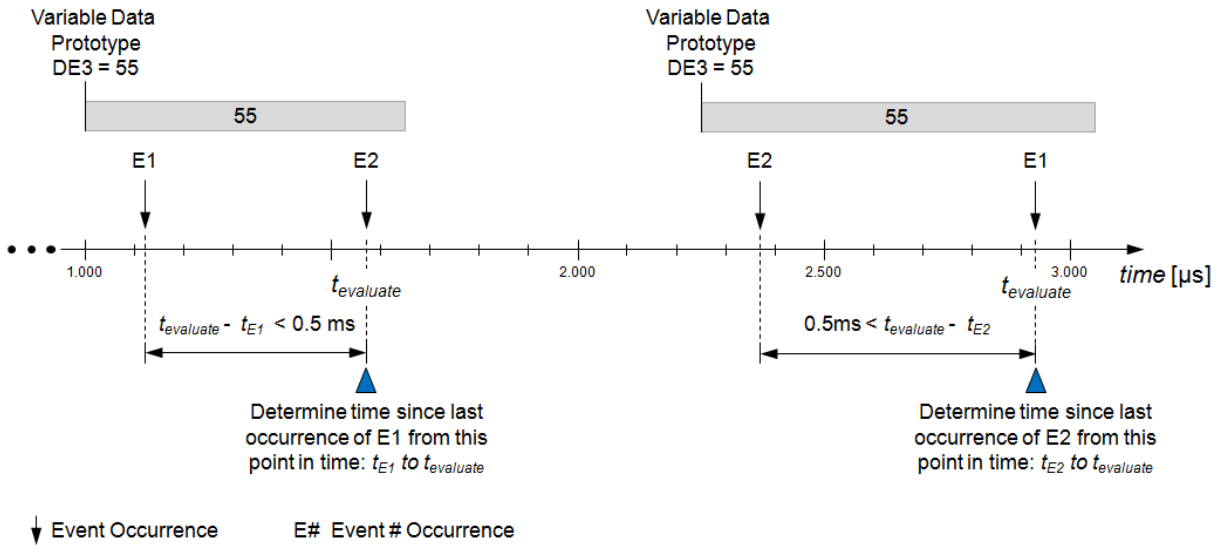


Figure 3.26: Trace showing various occurrences of the timing description events *E1* and *E2*, as well as the value of the variable *DE3*.

Based on the several functions provided by the occurrence expression language, the interpreter requires the following information from the system:

- the value of a referenced AutosarOperationArgumentInstance at time *t*.
- the value of a referenced AutosarVariableInstance at time *t*.
- the occurrences of a referenced TimingDescriptionEvent at time *t* and before.

There are different ways to gather the required information:

- Model analysis and simulation: In a deterministic system environment, occurrences of TimingDescriptionEvents can be determined offline, for example the point in time a frame will be transmitted in the static segment of a FlexRay network.
- Target trace: The required information can be gathered from a running system by recording the points in time a TimingDescriptionEvent has occurred.

If the interpreter has the required information as input, the different functions provided by the occurrence expression language can be interpreted as follows:

- TIMEX_value(t, <reference to an AutosarVariableInstance>) returns the variable value at time *t*.
- TIMEX_value(t, <reference to an AutosarOperationArgumentInstance>) returns the operation argument value at time *t*.
- TIMEX_occurs(t, <reference to a TimingDescriptionEvent>) returns TRUE (or 1) if the referenced event has occurred at time *t*, else it returns FALSE (or 0).
- TIMEX_hasOccurred(t, <reference to a TimingDescriptionEvent>) returns TRUE (or 1) if the referenced event has occurred *at least once* before or at time *t*.
- TIMEX_timeSinceLastOccurrence(t, <reference to a TimingDescriptionEvent>) returns the time difference between *t* and the point in time of the last occurrence of the referenced event. The unit of time is seconds.
- TIMEX_angleSinceLastOccurrence(t, <reference to a TimingDescriptionEvent>) returns the angle difference between *t* and the point in time of the last occurrence of the referenced event. The unit of angle is degree.
- TIMEX_modeActive(t, <reference to a TimingModeInstance>) returns TRUE (or 1) if the referenced mode is active at time *t*, else it returns FALSE (or 0).

3.5.2.7 Time Base Referencing for Timing Description Events

Please refer to [8] chapter "Time Base Referencing for Timing Description Events".

3.6 TimingConstraint

Timing constraints can be applied either on:

- `TimingDescriptionEvent`: classifies a single event or a group of events with a temporal restriction, for example a period, a latency or a time interval considered as synchronous. Also the direction has to be considered, which means in the semantics of the constraint it matters whether an event source (forward semantics) or an event sink (backward semantics) is considered.
- `TimingDescriptionEventChain`: a condition or property for this event chain is set. As the event chain has a semantic of a directed acyclic graph, the direction is obvious, but it matters whether a single event chain or a group of event chains are constrained.

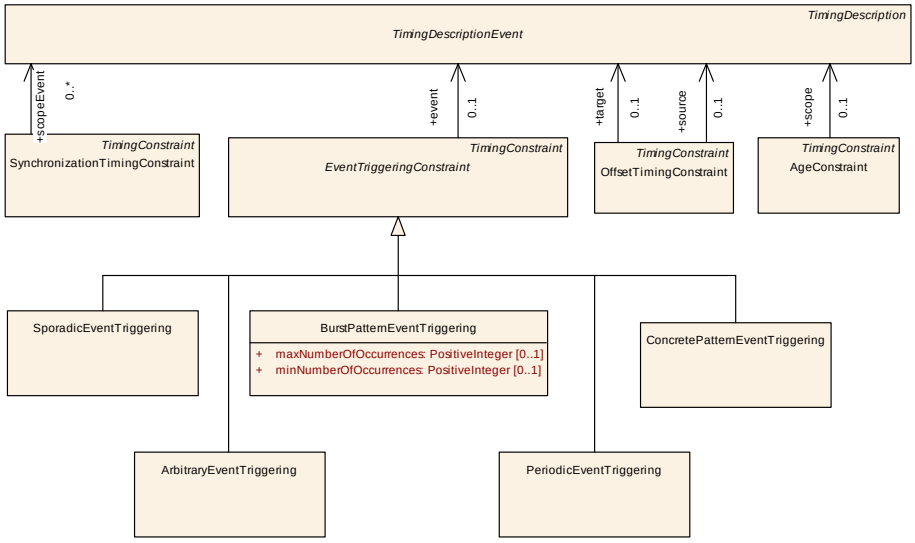


Figure 3.27: `TimingConstraint` vs `TimingDescriptionEvent`

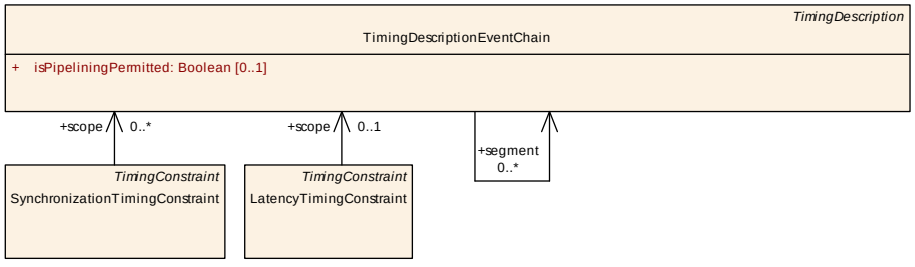


Figure 3.28: `TimingConstraint` vs `TimingDescriptionEventChain`

Mentioned in context of a requirement specification, Timing Constraints can be used as functional requirements and therefore can be tested. For usage in context of a performance specification, Timing Constraints can be used as system properties or timing guarantees.

The following table gives an overview over scope and usage of the different types of Timing Constraints described in the following chapters:

Table 3.31: Constraints

3.6.1 EventTriggeringConstraint

[TPS_TIMEX_00071]{DRAFT} **`EventTriggeringConstraint` specifies occurrence behavior respectively model** ⌈The element `EventTriggeringConstraint` is used to specify the particular occurrences of a given timing description event.⌋*(RS_TIMEX_00001, RS_TIMEX_00002, RS_TIMEX_00006, RS_TIMEX_00008)*

AUTOSAR offers five basic types of event triggering as depicted in Figure 3.29.

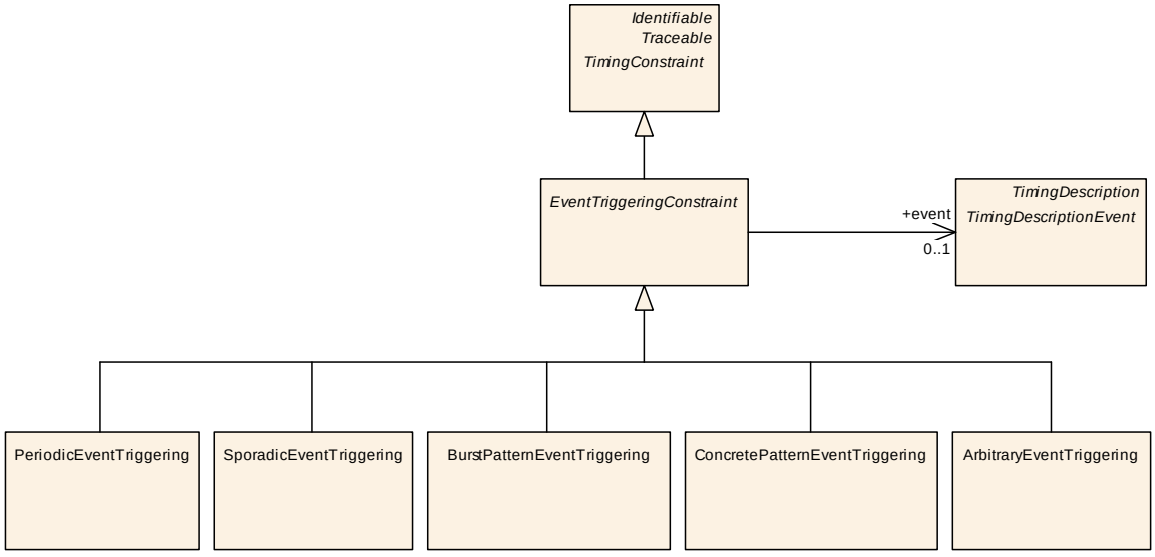


Figure 3.29: The different types of event triggerings

Table 3.32: EventTriggeringConstraint

3.6.1.1 PeriodicEventTriggering

[TPS_TIMEX_00076]{DRAFT} **`PeriodicEventTriggering` specifies periodic occurrences of events** ⌈The element `PeriodicEventTriggering` is used to specify the characteristics of a timing description event which occurs periodically.⌋*(RS_TIMEX_00001, RS_TIMEX_00002, RS_TIMEX_00006, RS_TIMEX_00008)*

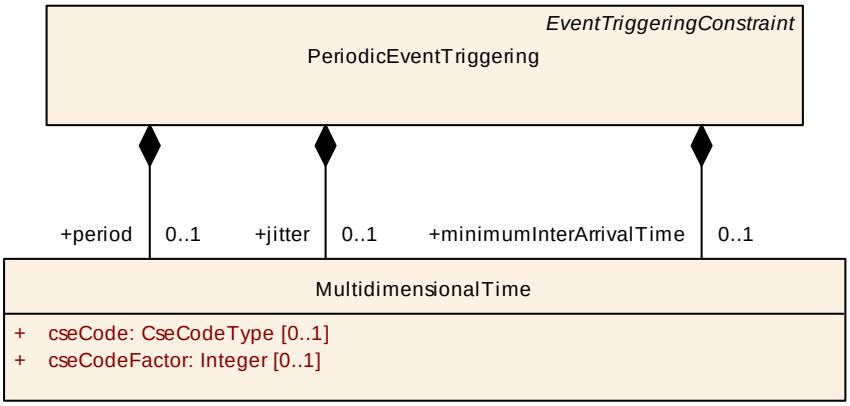


Figure 3.30: PeriodicEventTriggering

Table 3.33: PeriodicEventTriggering

The Periodic Event Triggering is characterized by the following parameters:

- Period
- Jitter
- Minimum Inter-Arrival Time

The listed parameters are required ones and are described in the following.

Period This parameter `period` specifies the periodic distance between subsequent occurrences of the event.

Jitter This parameter `jitter` specifies the maximum deviation from the period.

Minimum Inter-Arrival Time This parameter `minimumInterArrivalTime` specifies the minimum distance between subsequent occurrences of the event. Note, that if the value of the parameter `minimumInterArrivalTime` is less than the value of the parameter `period` minus the value of the parameter `jitter`, then the parameter `minimumInterArrivalTime` has no effect on the properties of the periodic event triggering constraints.

[constr_4589]{DRAFT} **Maximum value of the parameter `minimumInterArrivalTime`** ⌈The value of the parameter `minimumInterArrivalTime` shall be less than or equal the value of the parameter `period`.⌋*()*

Let t_n be the point-in-time of the n-th occurrence of the event. A Periodic Event Triggering Constraint is satisfied if, and only if at least one reference point-in-time $t_{reference}$ exists such that for every occurrence of the event at t_n the following holds true: $t_{reference} + (n-1)period \leq t_n \leq t_{reference} + (n-1)period + jitter$ and for all of those event occurrences the minimum distance shall be less than or equal to `minimumInterArrivalTime`.

$$\exists t_{reference} \mid \forall n: \quad t_{reference} + (n-1)period \leq t_n \leq t_{reference} + (n-1)period + jitter$$
$$AND \quad \forall n: \quad t_{n+1} - t_n \leq minimumInterArrivalTime$$

Figure 3.31 illustrates the parameters of the `PeriodicEventTriggering`. The upper part of this figure shows the case that the value of `jitter` is less than the value of the parameter `period`; whereas the lower part of this figure shows the case that the value of `jitter` is greater than or equal the value of the parameter `period`.

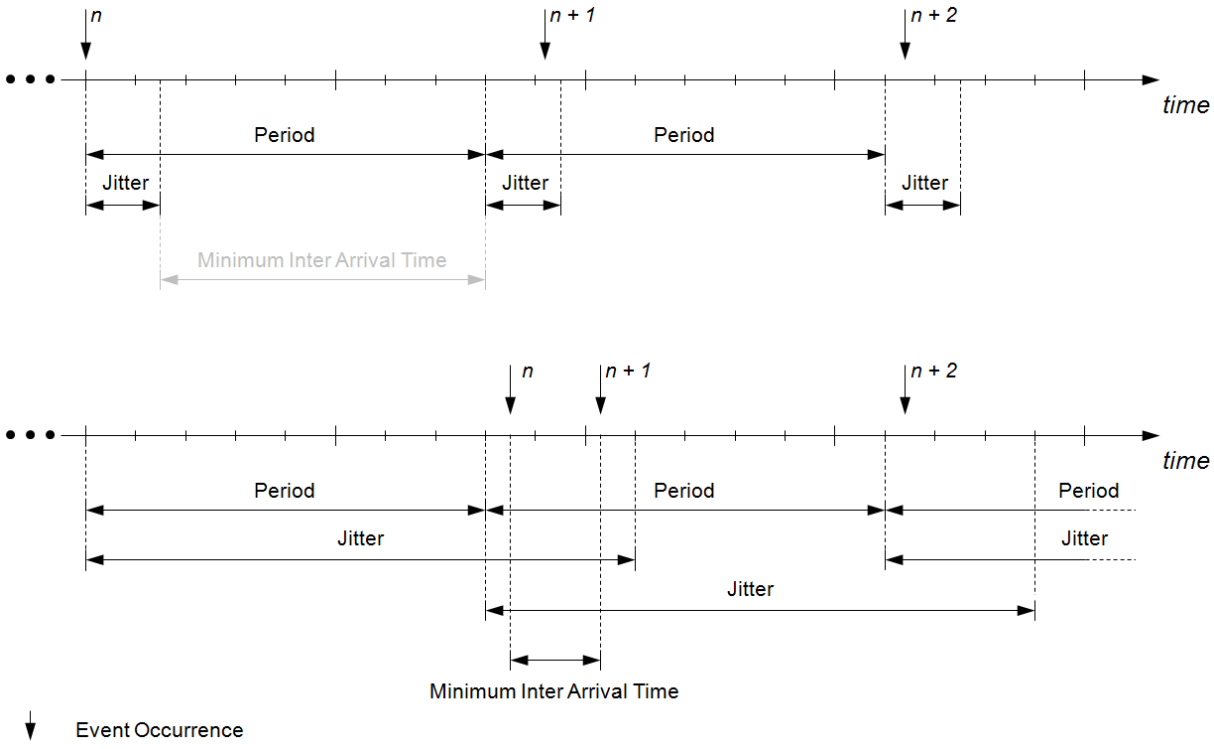


Figure 3.31: Parameters characterizing the Periodic Event Triggering

3.6.1.1.1 Examples

A Periodic Event Triggering Constraint is specified with the following parameters: `period` is six milliseconds (6ms) and `jitter` is two milliseconds (2ms). In other words, one imposes a timing constraint on an event to occur every six milliseconds and specifies that a deviation of two milliseconds is tolerable. In addition, it is assumed that the `minimumInterArrivalTime` is one millisecond (1ms) and therefore has no impact on the timing of the event's occurrences. This timing constraint is shown in Figure 3.32. The repeating gray-colored rectangles in this figure indicate the time intervals during which the event may occur; in other words it marks the subsequent time intervals the event is expected to occur.

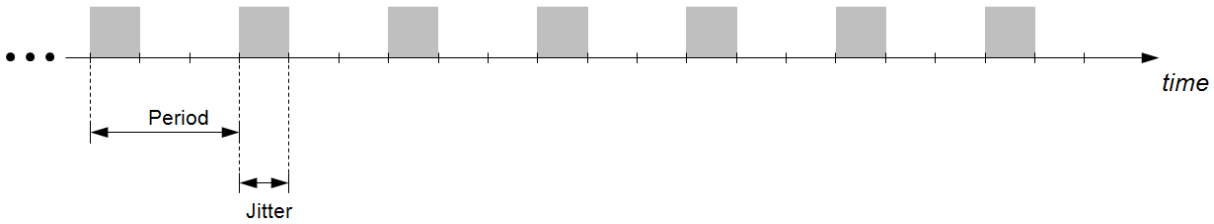


Figure 3.32: Example of a Periodic Event Triggering Constraint

The following figures show various event occurrences recorded during the observation of a system subject to analysis. The time interval for the observation is given by $t_{end-observation} - t_{start-observation}$. In the given example the system is observed for a period of 33.6 milliseconds.

The subsequent event occurrences shown in Figure 3.33 satisfy the given periodic event triggering constraint, because all occurrences of the event observed during the observation time interval happen in their corresponding time interval given by `period` and `jitter`.

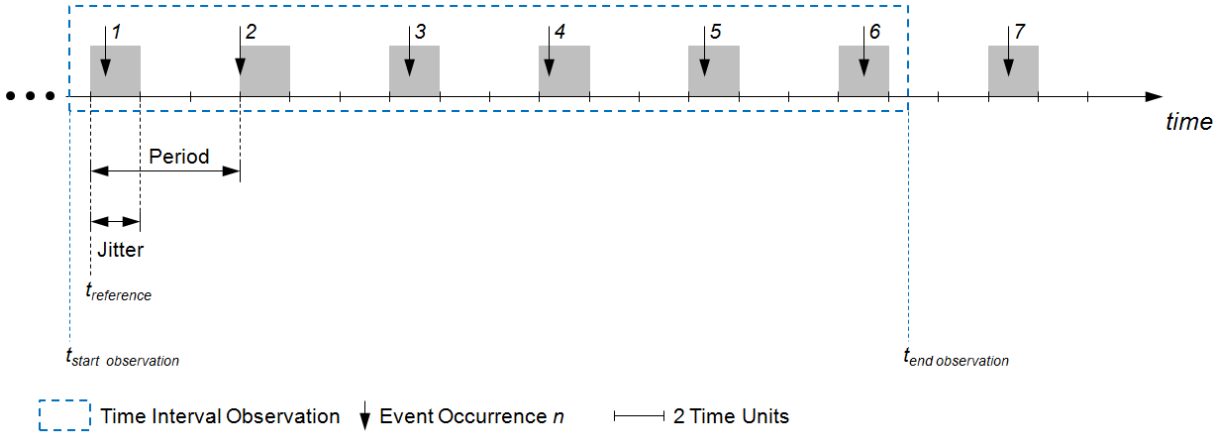


Figure 3.33: Event occurrences satisfying the given Period Event Triggering Constraint shown in the example at the beginning of this subsection.

The subsequent event occurrences shown in Figure 3.34 satisfy the given periodic event triggering constraint, because all occurrences of the event observed during the observation time interval happen in their corresponding time interval given by `period` and `jitter`. In contrast to the example shown in Figure 3.33 the reference point-in-time is another one.

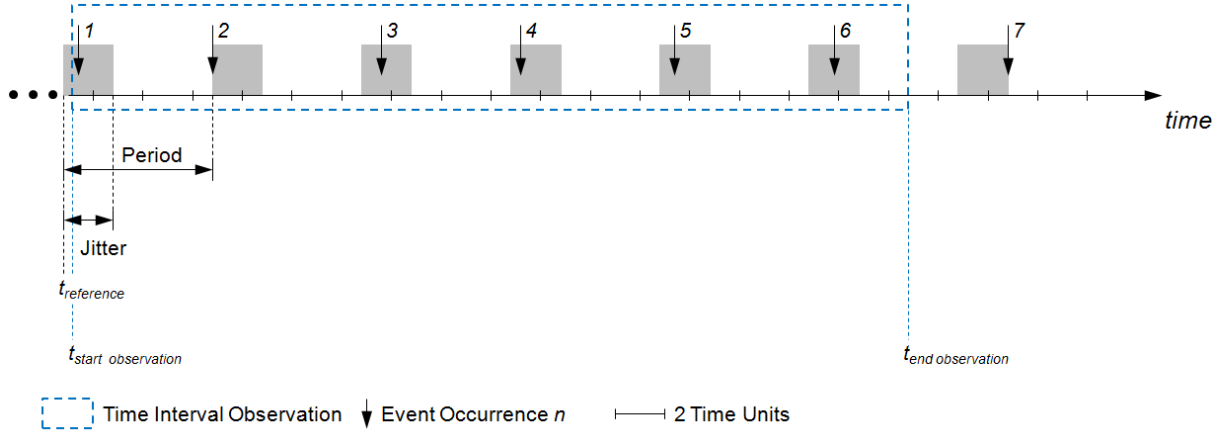


Figure 3.34: Event occurrences satisfying the given Period Event Triggering Constraint shown in the example at the beginning of this subsection, but with another reference point-in-time $t_{reference}$.

The subsequent event occurrences shown in Figure 3.35 violate the given periodic event triggering constraint, because the fifth occurrence of the event does not happen in its corresponding time interval given by `period` and `jitter`. In other words, there does not exist a reference point-in-time that ensures that all occurrences of the event observed during the observation time interval happen in their corresponding time interval given by `period` and `jitter`. And this results in a violation of the parameters `period` and `jitter`.

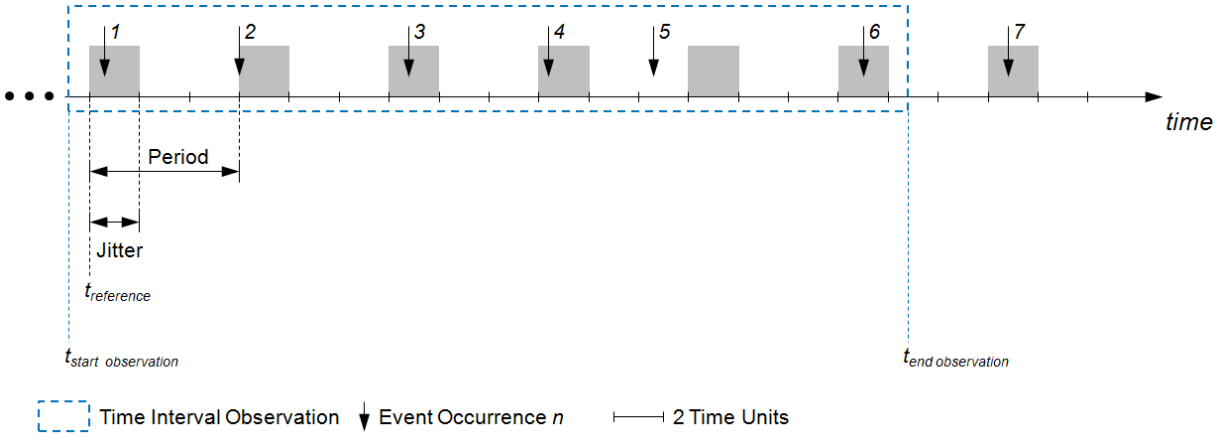


Figure 3.35: Event occurrences violating the given Period Event Triggering Constraint shown in the example at the beginning of this subsection.

The subsequent event occurrences shown in Figure 3.36 violate the given periodic event triggering constraint, because the fourth occurrence of the event does not happen in its corresponding time interval given by `period` and `jitter`. In other words, the fourth occurrence of the event happens in the time interval the fifth occurrence of the event happens and therefore violates the specified `jitter`.

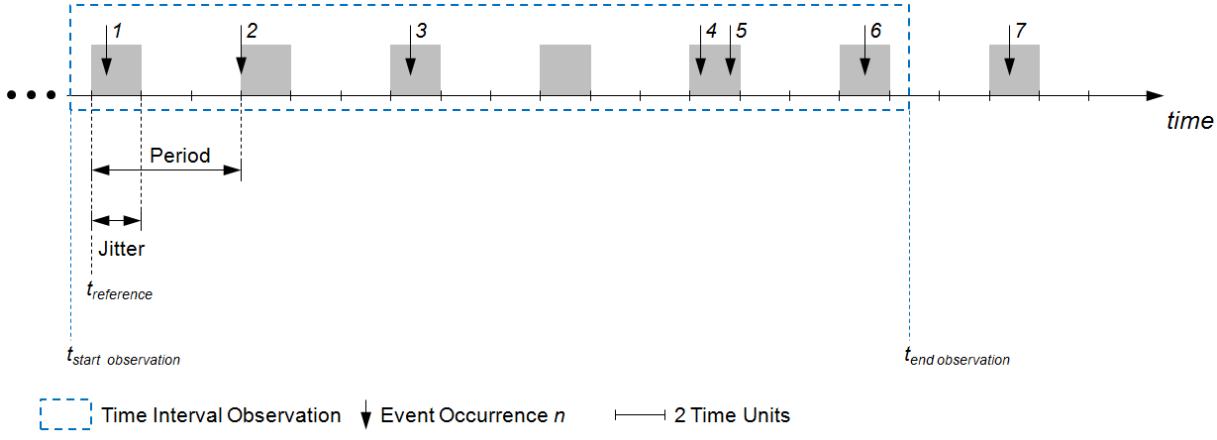


Figure 3.36: Event occurrences satisfying the given Period Event Triggering Constraint shown in the example at the beginning of this subsection.

3.6.1.2 SporadicEventTriggering

[TPS_TIMEX_00077]{DRAFT} **`SporadicEventTriggering` specifies sporadic occurrences of events** ⌈The element `SporadicEventTriggering` is used to specify the characteristics of a timing description event which occurs sporadically.⌋ *(RS_TIMEX_00001, RS_TIMEX_00002, RS_TIMEX_00006, RS_TIMEX_00008)*

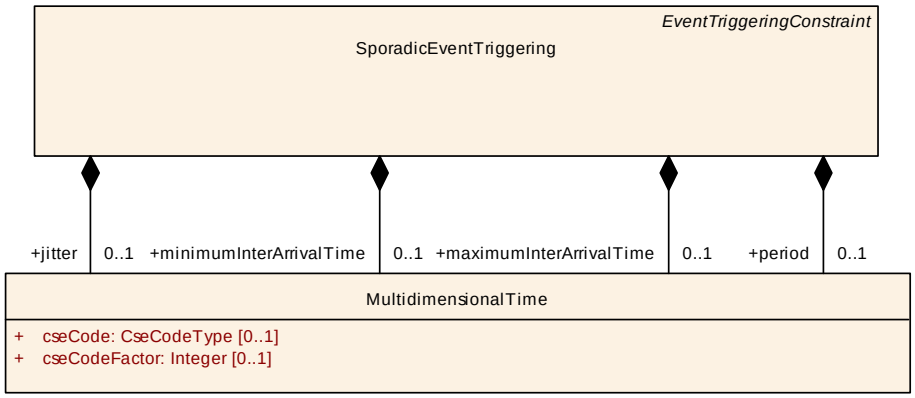


Figure 3.37: SporadicEventTriggering

Table 3.34: SporadicEventTriggering

This is a generalization of the periodic event triggering described in subsection 3.6.1.1. The difference is that the event can, but not necessarily shall occur. For this reason, there is one additional parameter required for the specification of the `SporadicEventTriggering`, namely the `maximumInterArrivalTime`, which specifies the largest possible time distance between two event occurrences.

The Sporadic Event Triggering is characterized by the following parameters:

- Minimum Inter-Arrival Time

- Maximum Inter-Arrival Time
- Period
- Jitter

The first two parameters are required ones and the last two parameters are optional. These parameters are described in the following and Figure 3.38 illustrates the parameters of the `SporadicEventTriggering`.

Minimum Inter-Arrival Time This parameter `minimumInterArrivalTime` specifies the minimum distance between subsequent occurrences of the event.

Maximum Inter-Arrival Time This parameter `maximumInterArrivalTime` specifies the maximum distance between subsequent occurrences of the event.

Period This optional parameter `period` specifies the periodic distance between subsequent occurrences of the event.

Jitter This optional parameter `jitter` specifies the maximum deviation from the period.

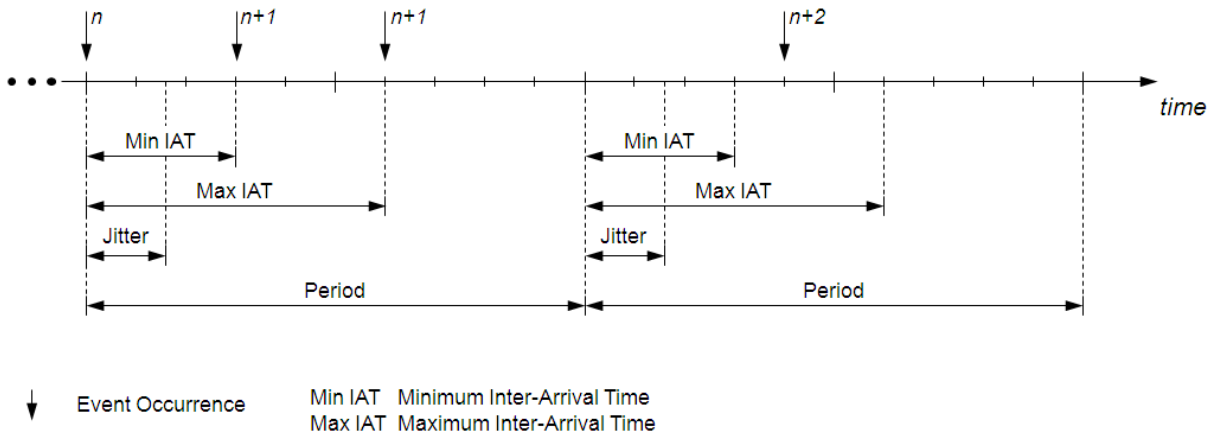


Figure 3.38: Parameters characterizing the Sporadic Event Triggering

3.6.1.3 ConcretePatternEventTriggering

[TPS_TIMEX_00078]{DRAFT} **`ConcretePatternEventTriggering` specifies concrete pattern of occurrences of events** ⌈The element `ConcretePatternEventTriggering` is used to specify the characteristics of a timing description event which occurs as a concrete pattern.⌋*(RS_TIMEX_00001, RS_TIMEX_00002, RS_TIMEX_00006, RS_TIMEX_00008)*

This describes events which occur following a known pattern.

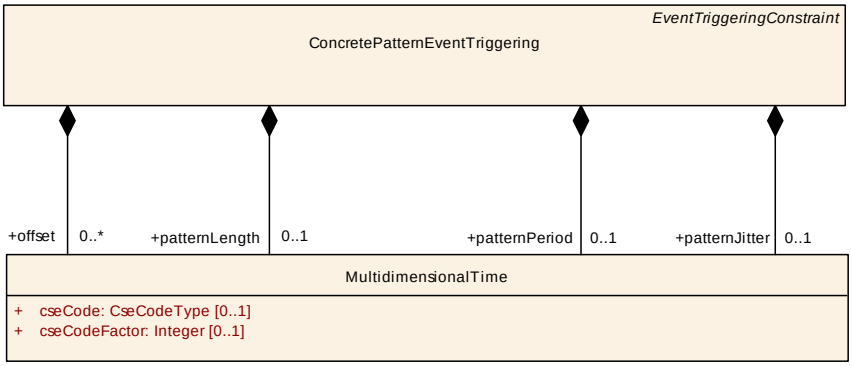


Figure 3.39: ConcretePatternEventTriggering

Table 3.35: ConcretePatternEventTriggering

The Concrete Pattern Event Triggering is characterized by the following parameters:

- Pattern Length
- Offset
- Pattern Period
- Pattern Jitter

The first two parameters are required ones, whereas the two last parameters are optional. The parameters are described in the following and are illustrated in Figure 3.40 and Figure 3.41.

Pattern Length This parameter `patternLength` specifies the time interval the pattern occurs in.

Offset This parameter `offset` specifies a list of point-in-times in the time interval given by the parameter `patternLength` at which the event occurs.

Pattern Period This optional parameter `patternPeriod` specifies the time distance between the beginnings of subsequent repetitions of the given burst pattern.

Pattern Jitter This optional parameter `patternJitter` specifies the maximum deviation of the time interval's starting point from the beginning of the given period. This parameter is only applicable in conjunction with the parameter `patternPeriod`.

The constraints listed below apply to the `ConcretePatternEventTriggering` and shall be considered when using this event triggering constraint.

[constr_4585]{DRAFT} **Specifying `patternLength`** ⌈The `patternLength` shall be specified such that the following holds: $0 \leq max(\texttt{offset}) \leq \texttt{patternLength}$.⌋*()*

[constr_4590]{DRAFT} **Specifying `patternLength`, `patternJitter` and `patternPeriod`** ⌈The pattern length, pattern jitter and pattern period shall be specified such that the following holds: $\texttt{patternLength} + \texttt{patternJitter} < \texttt{patternPeriod}$.⌋*()*

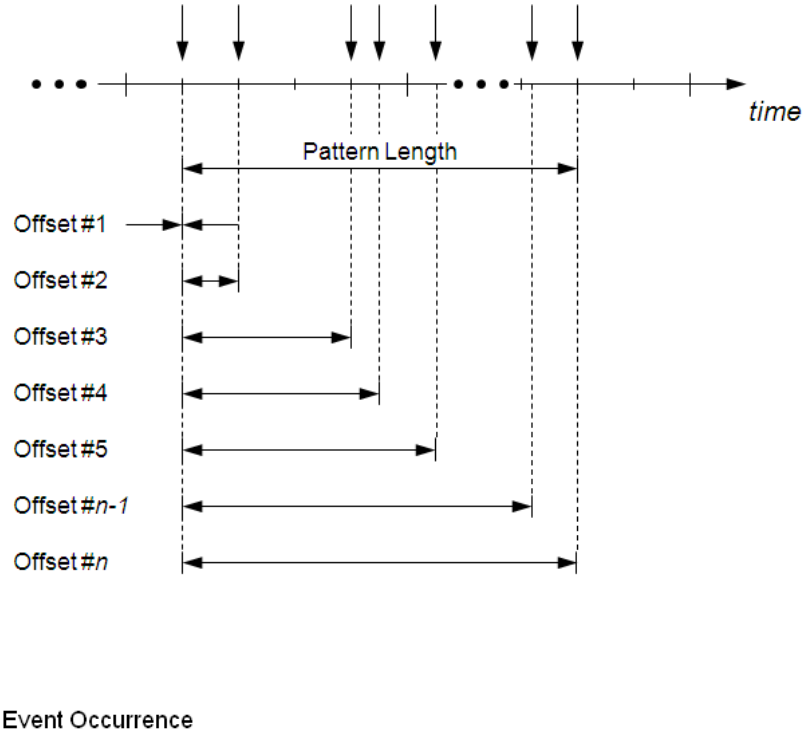


Figure 3.40: Parameters characterizing the Concrete Pattern Event Triggering

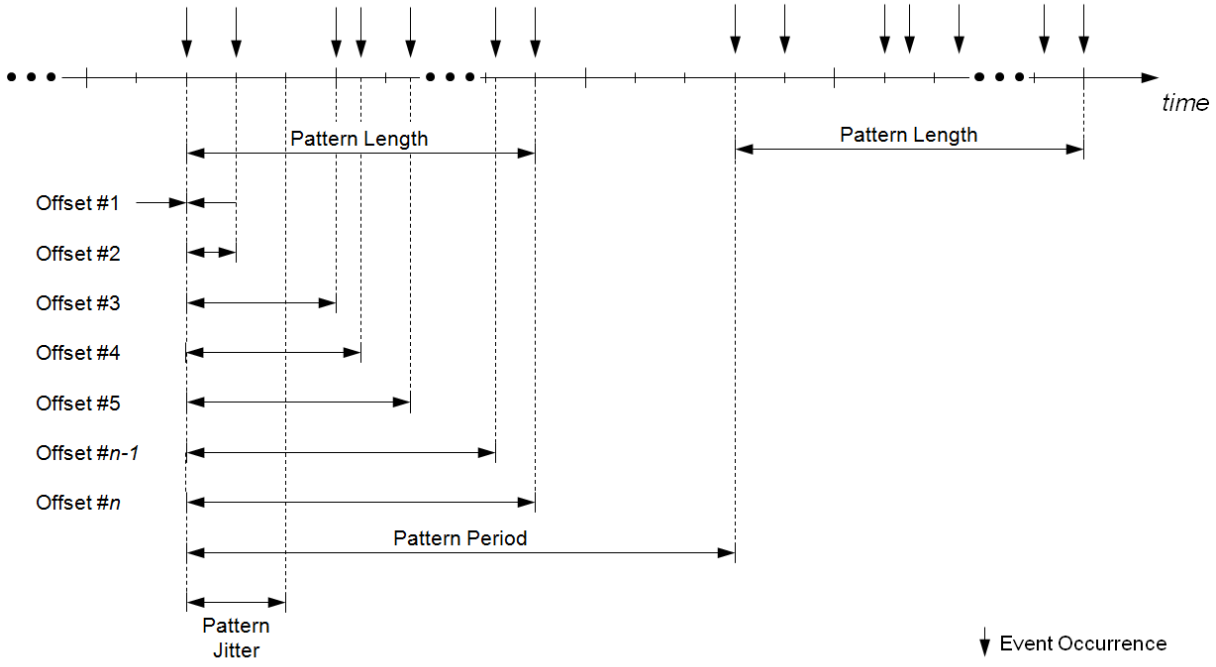


Figure 3.41: Parameters characterizing the Concrete Pattern Event Triggering when periodically being repeated

3.6.1.4 BurstPatternEventTriggering

[TPS_TIMEX_00079]{DRAFT} **`BurstPatternEventTriggering` specifies burst of occurrences of events** ⌈The element `BurstPatternEventTriggering` is used to specify the characteristics of a timing description event which occurs as a burst.⌋ *(RS_TIMEX_00001, RS_TIMEX_00002, RS_TIMEX_00006, RS_TIMEX_00008)*

The purpose of the `BurstPatternEventTriggering` is to describe a burst of occurrences of one and the same event. The Burst Pattern Event Triggering is characterized by the following parameters:

- Pattern Length
- Minimum Inter Arrival Time
- Maximum Number of Occurrences
- Minimum Number of Occurrences
- Pattern Period
- Pattern Jitter

The first three parameters are required ones, whereas the last three parameters are optional.

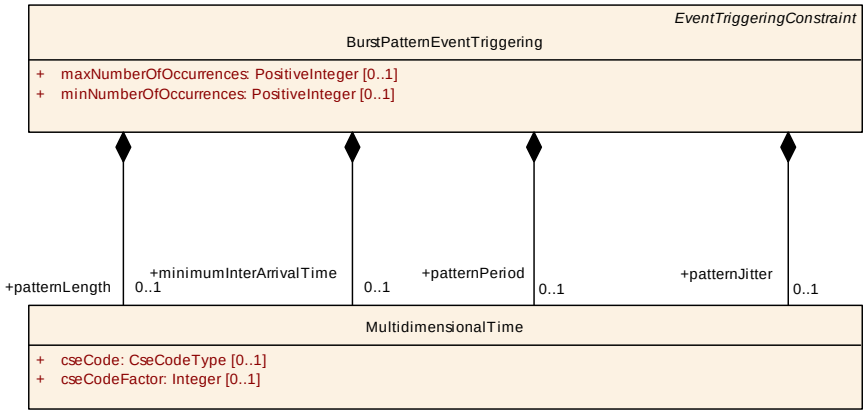


Figure 3.42: BurstPatternEventTriggering

Table 3.36: BurstPatternEventTriggering

The parameters are described in the following and are illustrated in Figure 3.43 and Figure 3.44.

Pattern Length This parameter `patternLength` specifies the duration of the time interval within which the event repeatedly occurs. The event occurs at arbitrary points in time within the given time interval.

Minimum Inter-Arrival Time This parameter `minimumInterArrivalTime` specifies the minimum distance between subsequent occurrences of the event within the given time interval.

Maximum Number of Occurrences This parameter `maxNumberOfOccurrences` specifies the maximum number of times the event can occur within the time interval. In other words, the event may never occur or any number of times between one (1) and the specified maximum number of occurrences. If the parameter `minNumberOfOccurrences` is specified then the event occurs at least the number of times specified by `minNumberOfOccurrences` and at maximum by `maxNumberOfOccurrences`.

Minimum Number of Occurrences This optional parameter `minNumberOfOccurrences` specifies the minimum number of times the event occurs within the given time interval. In other words, this parameter specifies the minimum number of times the event occurs in the given time interval. The value zero (0) for this parameter is permitted.

Pattern Period This optional parameter `patternPeriod` specifies the time distance between the beginnings of subsequent repetitions of the given burst pattern.

Pattern Jitter This optional parameter `patternJitter` specifies the maximum deviation of the time interval's starting point from the beginning of the given period. This parameter is only applicable in conjunction with the parameter `patternPeriod`.

The constraints listed below apply to the `BurstPatternEventTriggering` and shall be considered when using this event triggering constraint.

[constr_4574]{DRAFT} **Specifying minimum and maximum number of occurrences** ⌈The minimum and maximum number of occurrences shall be specified such that the following holds: $0 \leq$ `minNumberOfOccurrences` $\leq$ `maxNumberOfOccurrences`.⌋*()*

[constr_4575]{DRAFT} **Specifying minimum inter-arrival time and pattern length** ⌈The minimum inter-arrival time and pattern length shall be specified such that the following holds: $0 <$ `minimumInterArrivalTime` $\leq$ `patternLength`.⌋*()*

[constr_4576]{DRAFT} **Specifying pattern length, pattern jitter and patter period** ⌈The pattern length, pattern jitter and pattern period shall be specified such that the following holds: `patternLength` $+$ `patternJitter` $<$ `patternPeriod`.⌋*()*

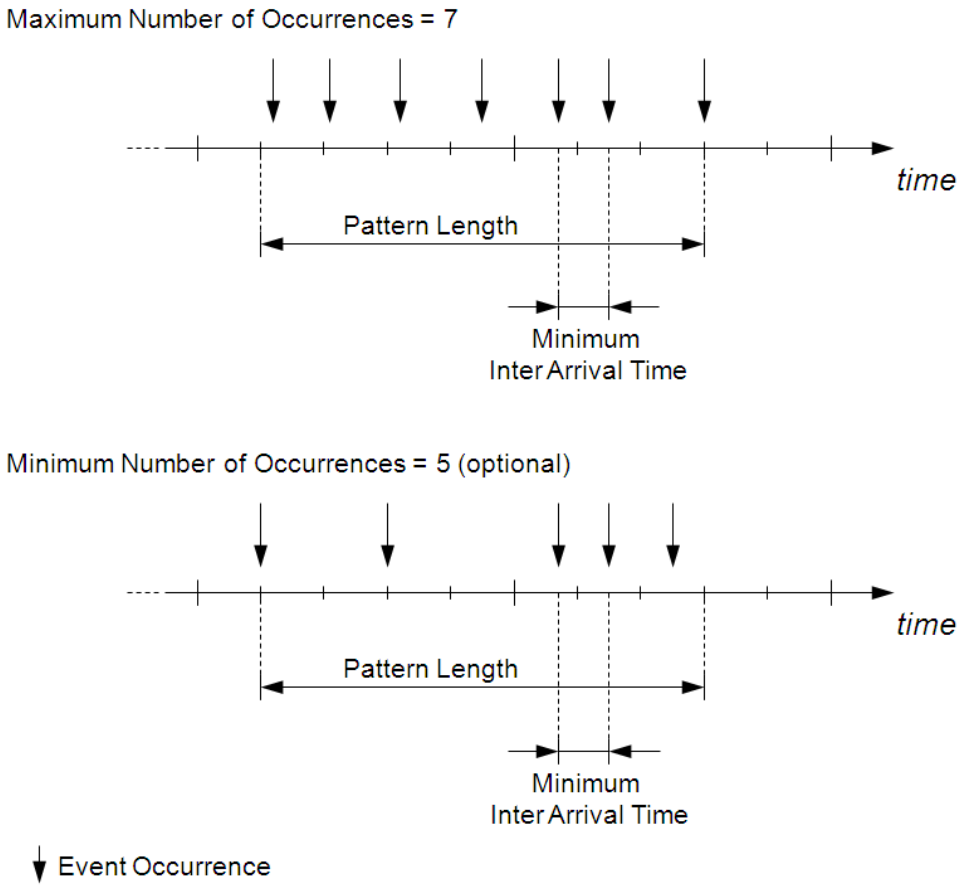


Figure 3.43: Parameters characterizing the Burst Pattern Event Triggering

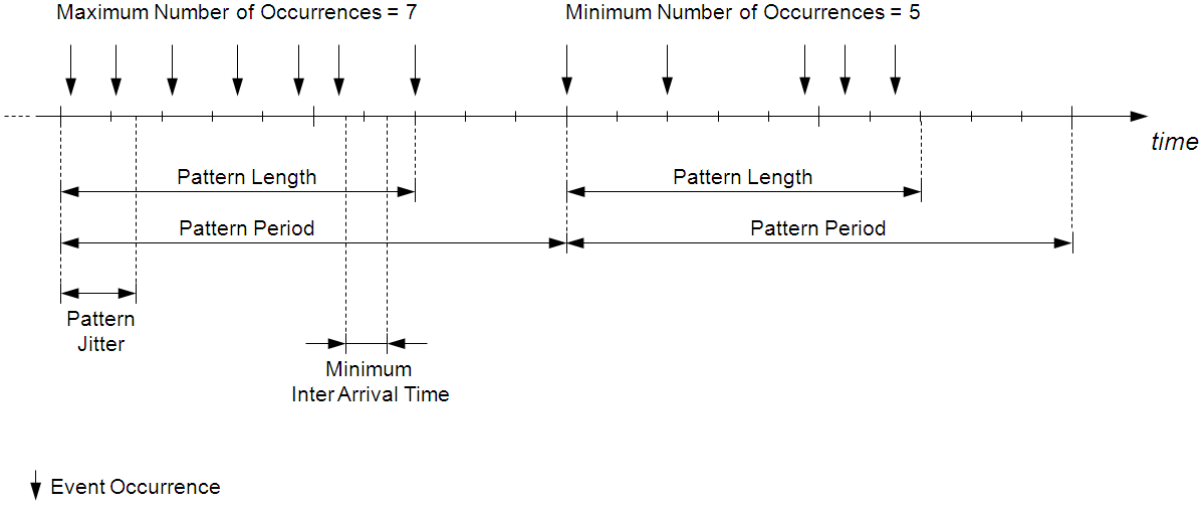


Figure 3.44: Parameters characterizing the Burst Pattern Event Triggering when periodically being repeated

3.6.1.5 ArbitraryEventTriggering

[TPS_TIMEX_00080]{DRAFT} **`ArbitraryEventTriggering` specifies arbitrary occurrences of an event** ⌈The element `ArbitraryEventTriggering` is used to specify the characteristics of a timing description event which occurs arbitrarily.⌋ *(RS_TIMEX_00001, RS_TIMEX_00002, RS_TIMEX_00006, RS_TIMEX_00008)*

This describes the occasional occurrence of a timing event.

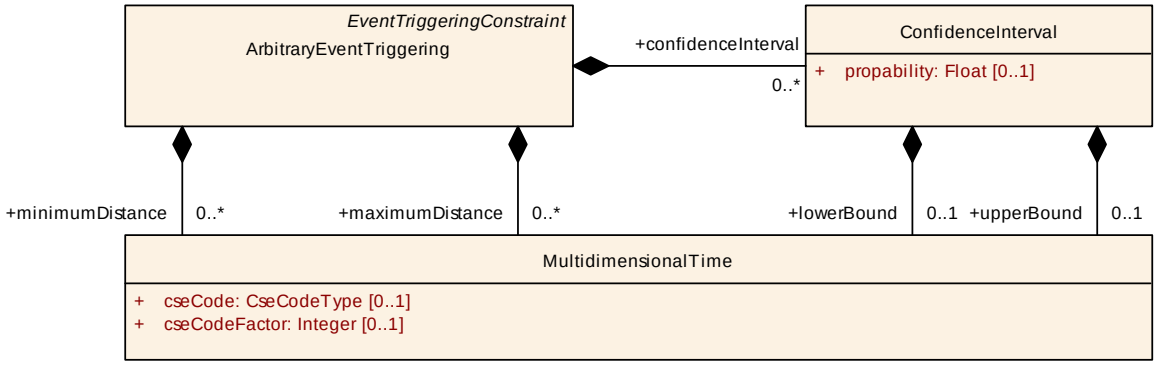


Figure 3.45: ArbitraryEventTriggering

Table 3.37: ArbitraryEventTriggering

Table 3.38: ConfidenceInterval

In contrast to the `ConcretePatternEventTriggering`, this event triggering is not as strict to the occurrence of an event, but generally describes event occurrences.

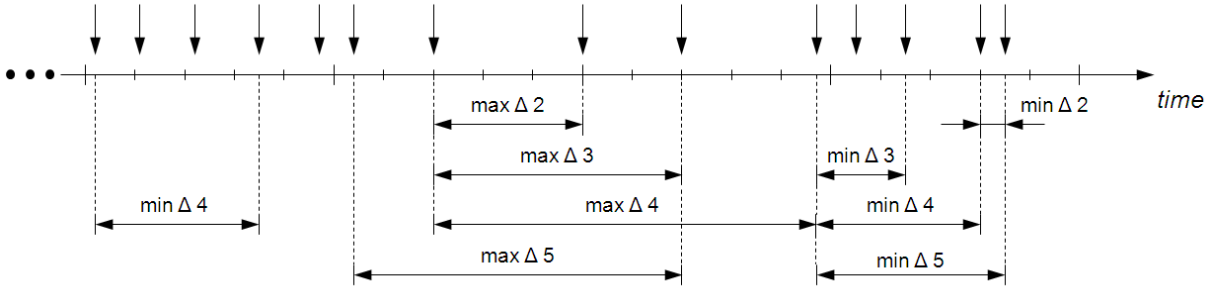
The Arbitrary Event Triggering is characterized by the following parameters:

- Minimum Distance
- Maximum Distance

These parameters are required ones and are described in the following. Figure 3.46 illustrates the parameters of the `ArbitraryEventTriggering`.

Minimum Distance The parameter `minimumDistance` specifies the minimum distance between n subsequent event occurrences, and $n = 2, 3, 4, ...$

Maximum Distance The parameter `maximumDistance` specifies the maximum distance between n subsequent event occurrences, and $n = 2, 3, 4, ...$



min Δn Least minimum inter-arrival time between *n* subsequent occurrences of the event E and n = {2, 3, 4, ...}

max Δn Major maximum inter-arrival time between *n* subsequent occurrences of the event E and n = {2, 3, 4, ...}

Event Occurrence

Figure 3.46: Parameters characterizing the Arbitrary Event Triggering

3.6.2 LatencyTimingConstraint

[TPS_TIMEX_00072]{DRAFT} **`LatencyTimingConstraint` specifies latency constraints** ⌈The element `LatencyTimingConstraint`[1] is used to specify the amount of time that elapses between the occurrence of any two timing description events.⌋*(RS_TIMEX_00001, RS_TIMEX_00002, RS_TIMEX_00012)*

For example, this can be the time it takes for a packet of data on a bus network to get from one designated point to another, or the time it takes for a function/task to be executed on a processor.

In the timing specification a `LatencyTimingConstraint` is associated with one `TimingDescriptionEventChain`, and specifies the minimum and/or maximum time duration between the occurrence of the stimulus and the occurrence of the corresponding response of that chain. However, in multi-rate networks, data can get lost or get duplicated because of potential different producer and consumer periods. Data loss occurs, if the consumer's period is greater than the producer's period (undersampling). Accordingly, data duplication occurs, if the consumer's period is smaller than the producer's period (oversampling). This is depicted in figure 3.47.

[1] A synonym for delay

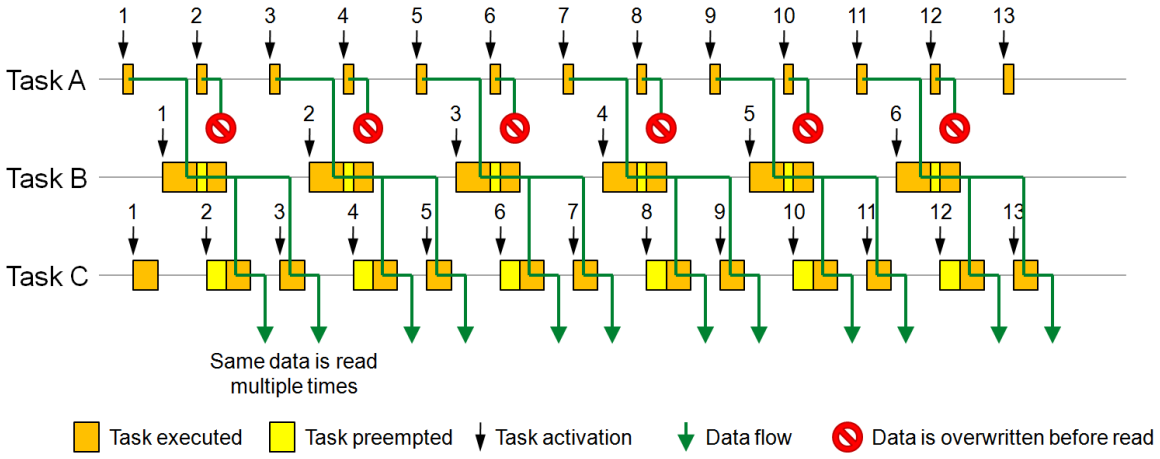


Figure 3.47: Loss and duplication of data due to under- and oversampling.

Considering under- and oversampling, two end-to-end latency semantics are of interest for automotive systems and can thus be expressed with the AUTOSAR timing extensions. These are the *age* of a certain response and the *reaction* to a certain stimulus.

The *data age timing constraint* 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. In this case the constrained time of importance is the delay from the latest stimulus to a given response.

The *reaction time constraint* 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 cases the constrained time of importance is the delay from a given stimulus to the first corresponding response.

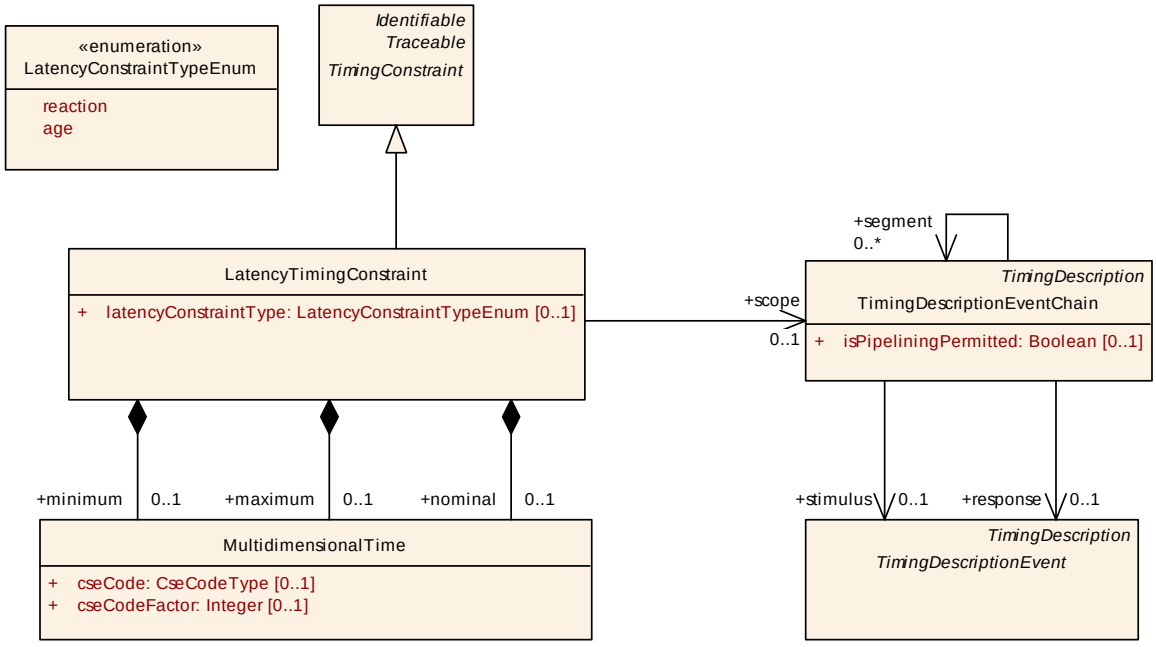


Figure 3.48: Latency constraint

Table 3.39: LatencyTimingConstraint

Table 3.40: LatencyConstraintTypeEnum

The attributes `minimum`, `maximum`, and `nominal` of a `LatencyTimingConstraint` can be used to define a lower and upper bound, as well as a nominal value for the latency of the event chain in the scope.

The application of latency constraints leads to some interesting observations:

- In systems without over- and under-sampling, *age* and *reaction* are the same. But timing constraints are implementation-independent. Thus, at specification time when the implementation is not necessarily known, the correct latency constraint semantics has to be specified.
- The minimum reaction and the minimum age latency of an event chain are always equal.

3.6.3 AgeConstraint

Sometimes it is necessary to specify the age of data, when it arrives at a component on its required port with `SenderReceiverInterface`. If the sender of the data is known, a `TimingDescriptionEventChain` can be defined from the sender to the receiver port and a `LatencyTimingConstraint` with *age* semantic represents the specification of the data age. However, the actual sender of the data may be unknown. In this case the definition of a `TimingDescriptionEventChain` is not possible.

[TPS_TIMEX_00073]{DRAFT} **`AgeConstraint` to specify age constraints** ⌈The element `AgeConstraint` is used to specify a minimum and maximum age that is tolerated when a variable data prototypes is received.⌋ *(RS_TIMEX_00001)*

Instead of an event chain, the scope of an age constraint is a `TDEventVariableDataPrototype`. Every time the scoped event occurs, the `VariableDataPrototype` shall have the specified data age.

At a later stage during the development, when the refined software architecture exposes the relation between the actual sender of the data and the receiver, an event chain between the sending and receiving point in time shall be defined and associ-

ated with a LatencyTimingConstraint (see 3.6.2) in order to refine the previous defined age constraint.

Typically, the age constraint restricts the time interval between the physical creation of the original sensor data by the corresponding sensor hardware and the availability of the data in the communication buffer (of the RTE) of the receiving SWC.

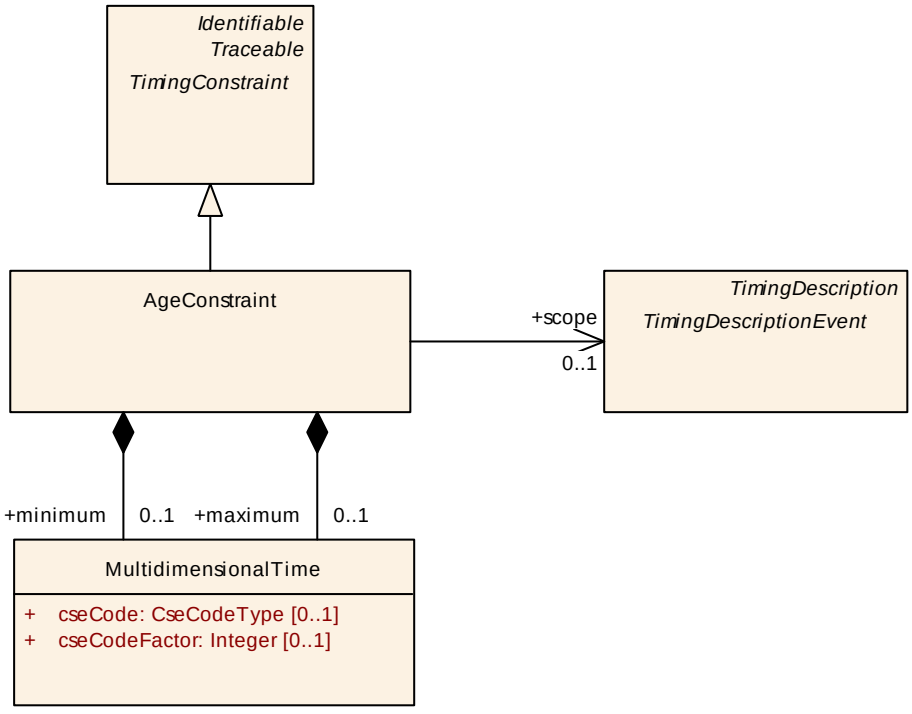


Figure 3.49: Age constraint

An AgeConstraint can define a minimum and maximum age for the VariableDataPrototype referenced by the TDEventVariableDataPrototype scope.

[constr_4573]{DRAFT} **Restricted usage of AgeConstraint** ⌈An AgeConstraint shall only be defined for events of type TimingDescriptionEvent associated with the receipt and reading of data.⌋*()*

Table 3.41: AgeConstraint

3.6.4 SynchronizationTimingConstraint

The objective of synchronization in a distributed environment is to establish and maintain a consistent time base for the interaction between different subsystems, in order to obtain correct runtime order and avoid unexpected race conditions. While mechanisms to establish synchronization need to be provided at the implementation level, the necessity for synchronization needs to be expressed at design level. For this purpose, synchronization constraints are used.

[TPS_TIMEX_00074]{DRAFT} **`SynchronizationTimingConstraint` specifies synchronicity constraints** ⌈The element `SynchronizationTimingConstraint` is used to specify a synchronization constraint among the occurrences of two or more timing description events.⌋ *(RS_TIMEX_00001, RS_TIMEX_00002, RS_TIMEX_00007, RS_TIMEX_00008, RS_TIMEX_00017)*

A `SynchronizationTimingConstraint` is imposed either on events (3.6.4.2) or on event chains (3.6.4.1).

Table 3.42: SynchronizationTimingConstraint

Table 3.43: EventOccurrenceKindEnum

Table 3.44: SynchronizationTypeEnum

[constr_4588]{DRAFT} **`SynchronizationTimingConstraint` shall either reference events or event chains** ⌈The `SynchronizationTimingConstraint` shall either reference timing description events or timing description event chains, but not both at the same time.⌋*()*

3.6.4.1 SynchronizationTimingConstraint on Event Chains

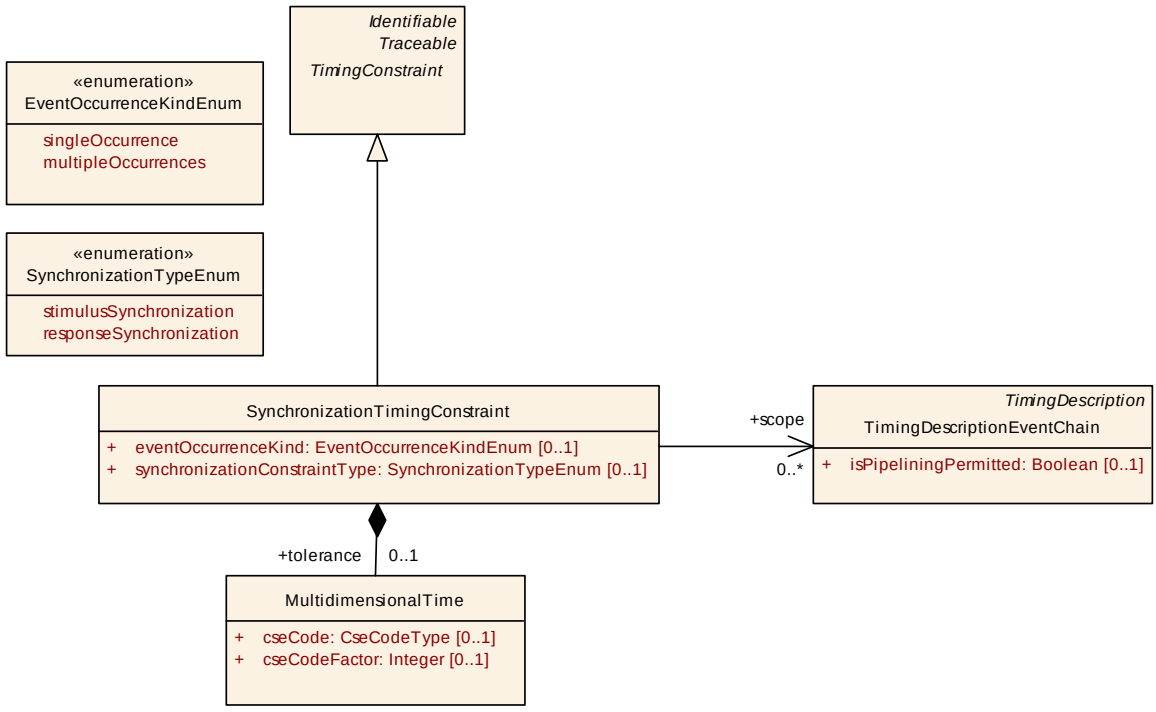


Figure 3.50: Synchronization Timing Constraint on Event Chains

The purpose of the `SynchronizationTimingConstraint` is to impose a synchronization constraint among either the stimulus or response event occurrences of two or more event chains. In the former case (stimulus synchronization) the referenced event chains shall have the same response event (join), or in the latter case (response synchronization) they shall have the same stimulus event (fork).

The `SynchronizationTimingConstraint` is characterized by the following parameters:

- Tolerance
- Event Occurrence Kind
- Synchronization Constraint Type

The parameters are described in the following and are illustrated in Figure 3.51 and Figure 3.52.

Tolerance The parameter `tolerance` specifies the time interval within which the referenced events shall occur synchronously. The events may occur in any order within this time interval. The time interval starts at the point-in-time when one of the referenced events occurs.

Event Occurrence Kind The optional parameter `eventOccurrenceKind` specifies whether the referenced events shall occur only once (single occurrence) or may occur multiple times (multiple occurrences) in the given time interval.

Synchronization Constraint Type The parameter `synchronizationConstraintType` specifies whether the `SynchronizationTimingConstraint` is imposed on the stimulus or response events of the referenced event chains.

[constr_4580]{DRAFT} **`SynchronizationTimingConstraint` shall reference at least two event chains** ⌈In the case, that the `SynchronizationTimingConstraint` is imposed on event chains then at least two (2) timing description event chains shall be referenced.⌋*()*

[constr_4587]{DRAFT} **Specifying attribute `synchronizationConstraintType`** ⌈The attribute `synchronizationConstraintType` shall be specified if the `SynchronizationTimingConstraint` is imposed on event chains.⌋*()*

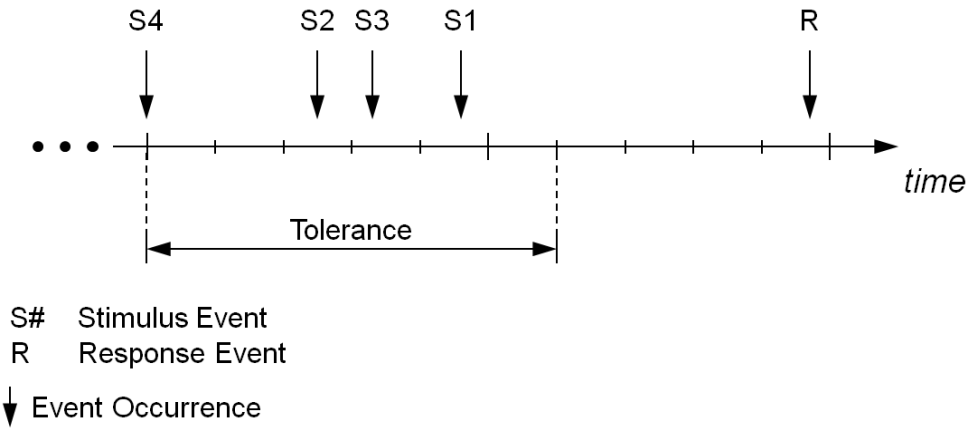


Figure 3.51: Parameters characterizing the Synchronization Timing Constraint imposed on the stimulus events of event chains.

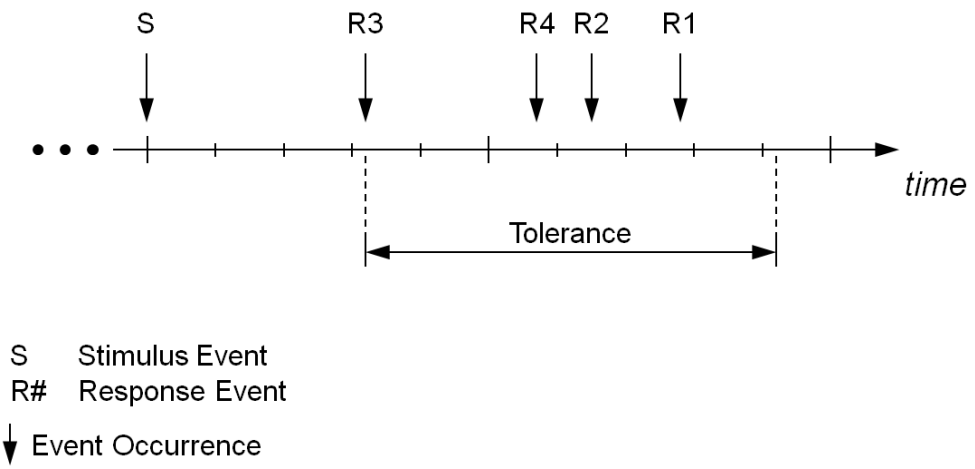


Figure 3.52: Parameters characterizing the Synchronization Timing Constraint imposed on the response events of event chains.

An example for synchronizing on *stimuli* of event chains would be an adaptive cruise control that expects data from different sensors, which shall be sampled (quasi) simultaneously with respect to a predefined tolerance.

An example for synchronizing on *responses* of event chains would be the blinking of different indicator lights, which shall occur (quasi) simultaneously with respect to a predefined tolerance.

3.6.4.2 SynchronizationTimingConstraint on Events

As mentioned above, the purpose of the `SynchronizationTimingConstraint` is to impose a synchronization constraint among either the stimulus or response event occurrences of two or more event chains. However, in some cases the complete event chains are not entirely known, or not available in the scope of the model, at the point in time the timing constraint shall be specified. For this purpose, the AUTOSAR Timing Extensions allow the specification of synchronization constraints on events. In this case, the events referenced by the constraint are related implicitly, because they have a common stimulus (in case of constraint type `responseSynchronization` or a common response (in case of constraint type `stimulusSynchronization` not known yet, or not available in the scope of the model.

At a later stage during the development, when the refined software architecture exposes the complete event chains (e.g. because the common stimulus gets known), the respective event chains shall be specified and associated with a `SynchronizationTimingConstraint` on event chains (see 3.6.4.1) in order to refine the previously defined `SynchronizationTimingConstraint` on events.

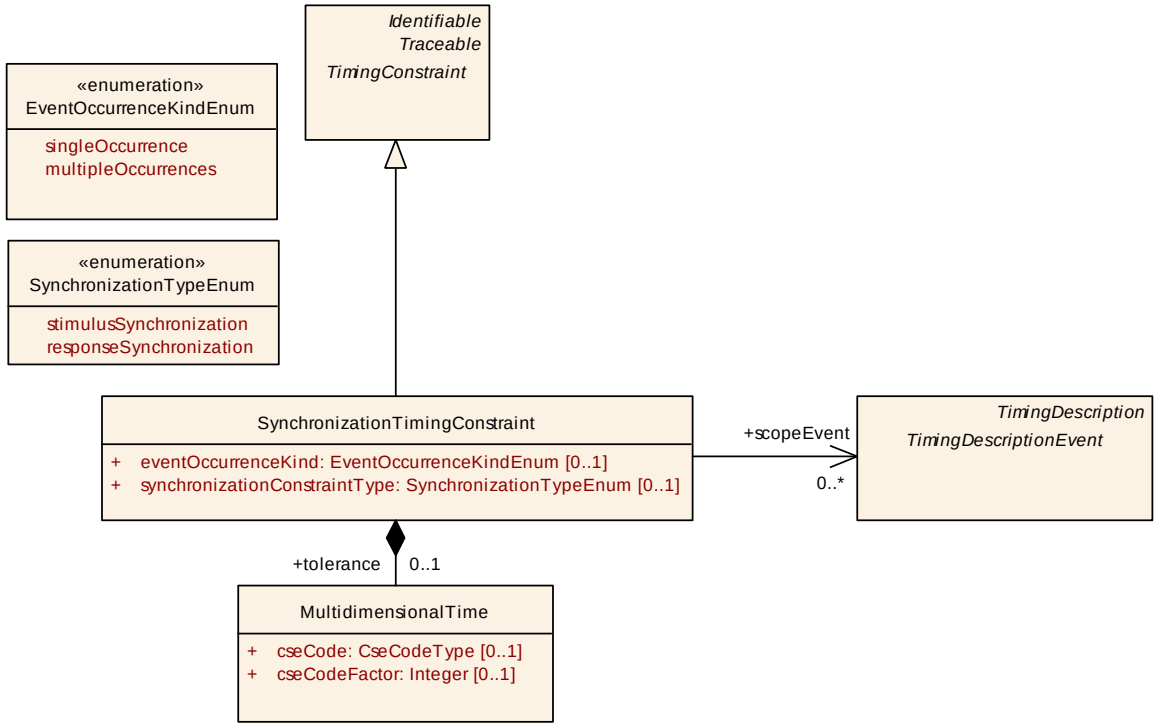


Figure 3.53: Synchronization Timing Constraint on Events

The purpose of the SynchronizationTimingConstraint is to impose a synchronization constraint among the occurrences of two or more events. The SynchronizationTimingConstraint is characterized by the following parameters:

- Tolerance
- Event Occurrence Kind
- Synchronization Constraint Type

The parameters are described in the following and are illustrated in Figure 3.54.

Tolerance The parameter tolerance specifies the time interval within which the referenced events shall occur synchronously. The events may occur in any order within this time interval. The time interval starts at the point-in-time when one of the referenced events occurs.

Event Occurrence Kind The parameter eventOccurrenceKind specifies whether the referenced events shall occur only once (single occurrence) or may occur multiple times (multiple occurrences) in the given time interval.

Synchronization Constraint Type The parameter synchronizationConstraintType specifies whether the associated events of the SynchronizationTimingConstraint have a common stimulus or response.

[constr_4579]{DRAFT} **SynchronizationTimingConstraint shall reference at least two events** ⌈In the case, that the SynchronizationTimingConstraint is imposed on events then at least two (2) timing description events shall be referenced.⌋ *()*

[constr_4586]{DRAFT} **Specifying attribute synchronizationConstraintType** ⌈The attribute synchronizationConstraintType shall be specified if the SynchronizationTimingConstraint is imposed on events.⌋*()*

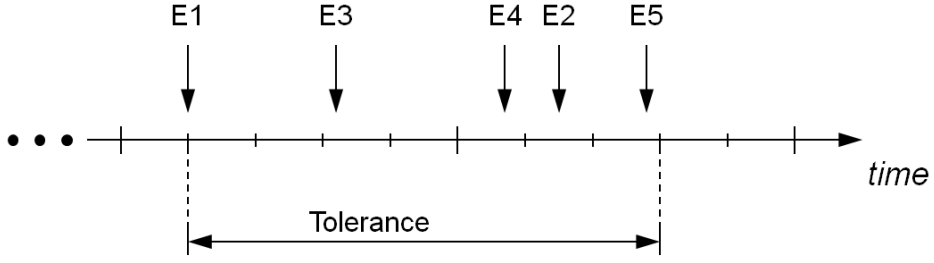


Figure 3.54: Parameter characterizing the Synchronization Constraint

3.6.5 OffsetTimingConstraint

[TPS_TIMEX_00081]{DRAFT} **`OffsetTimingConstraint` specifies offset between occurrences of events** ⌈The element `OffsetTimingConstraint` is used to specify an offset between the occurrences of two timing description events.⌋*(RS_TIMEX_00001, RS_TIMEX_00002, RS_TIMEX_00008)*

An `OffsetTimingConstraint` bounds the time offset between the occurrence of two timing events, without requiring a direct functional dependency between the source and the target.

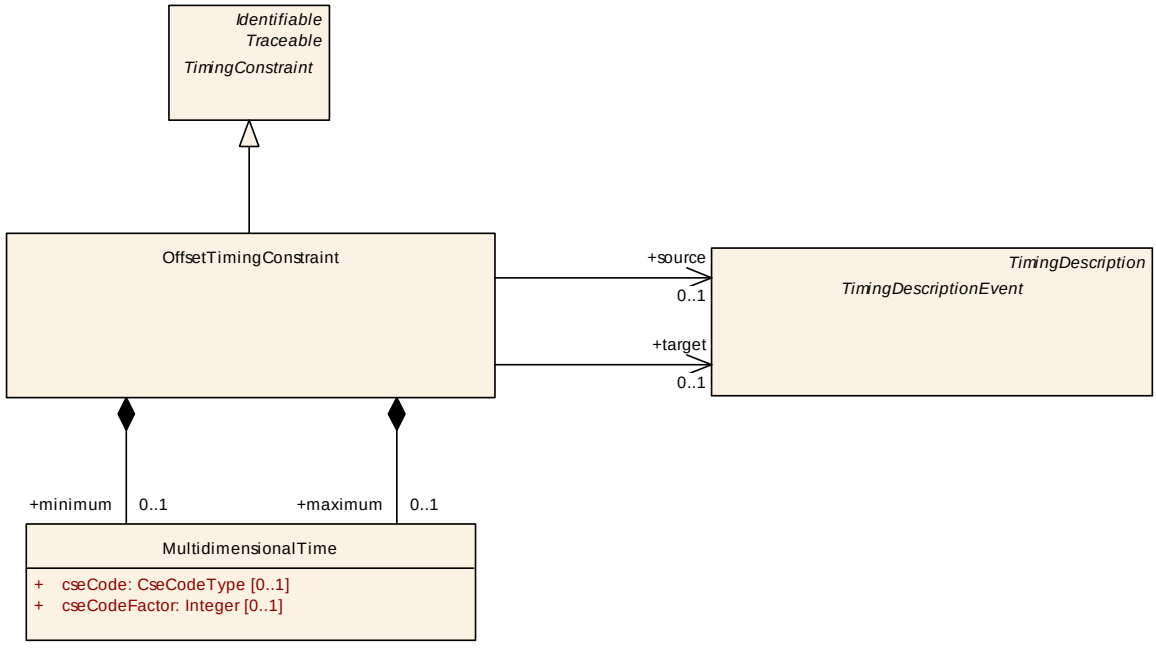


Figure 3.55: Offset Timing Constraint

Table 3.45: OffsetTimingConstraint

3.6.6 Traceability of Constraints

[TPS_TIMEX_00089]{DRAFT} **`TimingConstraint` is a `Traceable`** ⌈The element `TimingConstraint` and all of its specializations, commonly called timing constraints, are traceable.⌋ *(RS_TIMEX_00010)*

The support for traceability [4] enables one to specify relationships between timing constraints and corresponding AUTOSAR elements that satisfy those timing requirements.

3.7 Logical Execution Time

`Logical Execution Time` (`LET`) is currently restricted to `CP`.

3.8 System Level Logical Execution Time

Please refer to [8] chapter "System Level Logical Execution Time".

3.9 Blueprinting

[TPS_TIMEX_00091]{DRAFT} **Blueprinting `VfbTiming`** ⌈`VfbTiming` can be blueprinted.⌋ *(RS_TIMEX_00016)*

The primary purpose of blueprinting `VfbTiming` is to annotate Application Interfaces and attach timing constraints, like age- and periodic event triggering constraints, to events of type `TDEventVfb` which reference port prototype blueprints. The concept of Blueprints and its details are described in [4].

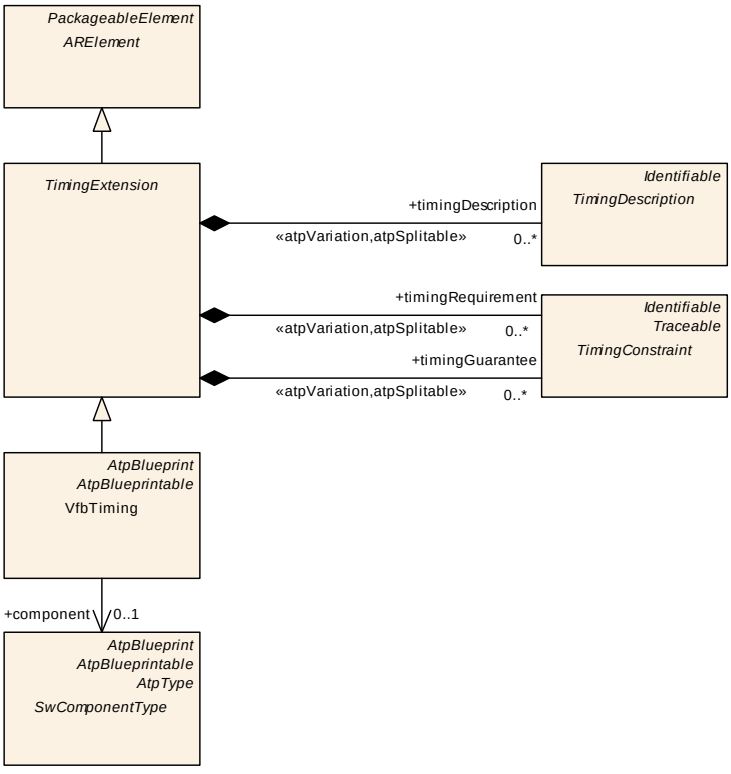


Figure 3.56: VFB Timing Blueprint

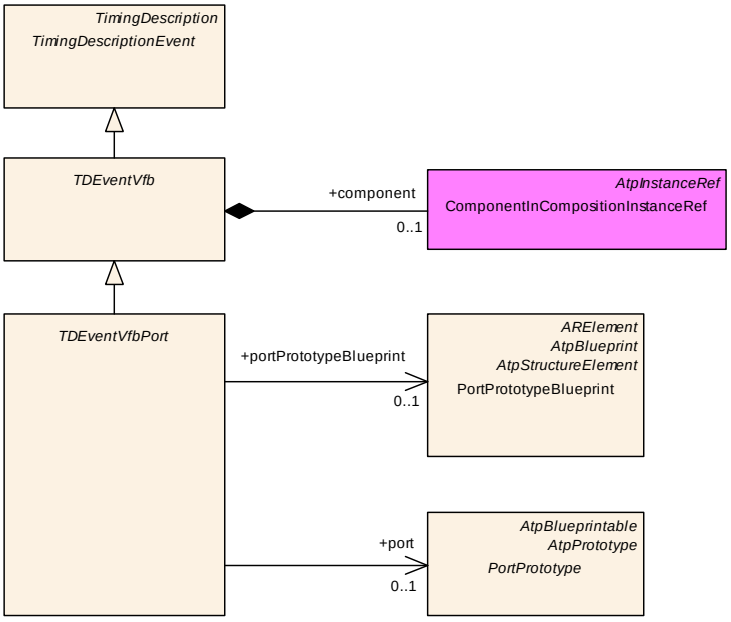


Figure 3.57: `TDEventVfb` Blueprint

[constr_4577]{DRAFT} **`TDEventVfb` shall reference `PortPrototypeBlueprint` only in Blueprints** ⌈An event type `TDEventVfb` only shall reference `PortPrototypeBlueprint` in blueprints.⌋ *()*

[constr_4578]{DRAFT} **Only `VfbTiming` shall be a Blueprint** ⌈Only the `VfbTiming` is blueprintable.⌋ *()*

3.10 Methodology

The AUTOSAR methodology (see [9] for a general introduction) provides several well-defined process steps, and furthermore artifacts that are provided or needed by these steps.

For each of these views a special focus of timing specification can be applied, depending on the availability of necessary information, the role a certain artifact is playing and the development phase, which is associated with the view.

[TPS_TIMEX_00075]{DRAFT} **Optional use of timing extensions** ⌈The elements `TimingExtension`, `TimingDescription`, and `TimingConstraint` of the timing extensions are derived from the element `ARElement`. This enables one to deliver timing extensions in a separate document. In addition, there are no external references from any template that point to timing extensions elements.⌋ *(RS_TIMEX_00003)*