ITEA 2

INFORMATION TECHNOLOGY FOR EUROPEAN ADVANCEMENT

ITEA 2 - 09033

TIMMO-2-USE

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

Report type	Deliverable D13
Report name	Methodology description V2
Report status	Public
Version number	Version 1.0
Date of preparation	2012-07-31

TIMMO-2-USE Partners

AbsInt Angewandte Informatik GmbH

Arcticus Systems AB

Chalmers University of Technology

Continental Automotive GmbH

Delphi France SAS

dSpace GmbH

INCHRON GmbH

Institute National de Recherche en Informatique et Automatique INRIA

Mälardalen University

Rapita Systems Ltd, UK

RealTime-at-Work

Robert Bosch GmbH

Symtavision GmbH

Technische Universität Braunschweig

University of Paderborn

Volvo Technology AB

Project Coordinator

Dr. Daniel Karlsson

Volvo Technology AB

Dept 6270, M2.7

405 08 Göteborg

Sweden

Tel.: +46 31 322 9949

Email: Daniel.B.Karlsson@volvo.com

Authors

Cecilia Ekelin, Volvo Technology AB

Arne Hamann, Robert Bosch GmbH

Daniel Karlsson, Volvo Technology AB

Ulrich Kiffmeier, dSpace GmbH

Stefan Kuntz, Continental Automotive GmbH

Oscar Ljungkrantz, Volvo Technology AB

Mesut Özhan, INCHRON GmbH

Table of contents

List of Figures

1 Introduction

In this document the final results of the TIMMO-2-USE methodology are described in detail.

The main goal of the TIMMO-2-USE methodology is to address practical use-cases that require special consideration of timing aspects. Related "timing augmented" methodologies, like the TIMMO and ATESST2 methodologies (see Section 2) do not offer such detail and mainly describe the application of timing analysis and simulation techniques for validation purposes. These aspects are also covered in the TIMMO-2-USE methodology, but additionally it is described how design decisions can be taken based on timing information. In other words, the TIMMO-2-USE methodology introduces a constructive feedback between automotive software system design and real-time systems engineering.

The basis of the TIMMO-2-USE methodology is the *Generic Methodology Pattern (GMP)* described in Section 3. All practical use cases that are described in Section 5 are mapped to this generic methodology. The covered use-cases that are described in Section 5 are the following:

- Integrate reusable component
- Specify timing budget
- Specify synchronization timing constraints
- Revise erroneous timing information
- Exchange models

One important distinctive characteristic of the GMP is the integration of top-down and bottom-up development aspects into one single methodology. Therefore, it is crucial to being able to transform timing information 1) from higher to lower abstraction levels, and 2) from lower to higher abstraction levels. A discussion on how these transformations can be done is given in Section **Error! Reference source not found.**.

The GMP general structure was designed with the aim of being applicable to other aspects of software system development apart from timing. To demonstrate this, an extension of the GMP is presented in Section 3.3 that renders it compatible to ISO 26262 compliant safety methodologies.

In order to integrate the results of the technical work packages within TIMMO-2-USE, the concepts of "TADL guides" (Section 4.1) and "Tool mentors" (Section 4.2) were developed for the TIMMO-2-USE methodology.

TADL guides explain the usage of TADL2 (Timing Augmented Description Language 2) concept during the different methodology tasks, whereas Tool mentors link to timing related tools and algorithms that are relevant for the completion of a task at hand.

2 Starting Point

In previous projects, software system development methodologies were developed taking into account timing aspects. In the following sections, the most prominent projects and the developed methodologies are shortly presented and related to the TIMMO-2-USE methodology.

2.1 The TIMMO Methodology

In the ITEA2 predecessor project TIMMO (TIMing MOdel), a system development methodology was defined explicitly taking into account the real-time behavior of the developed system, an aspect that is ignored in many comparable methodologies.

The TIMMO methodology describes the application of the Timing Augmented Description Language (TADL), that was also developed in the TIMMO project and that is extended in WP2 of TIMMO-2-USE, in the context of the automotive software system development process. Based on the information captured by TADL, the TIMMO methodology highlights the possibilities of applying timing analyses to help the designer taking design decisions and verifying the system's adherence to timing constraints. This guideline on how timing analyses can be applied during the development process of automotive software systems represents the main novelty of the TIMMO methodology.

The TIMMO methodology is based on EAST-ADL at the higher levels of abstraction and on AUTOSAR at implementation level (compare Figure 1).

- Vehicle Phase (EAST-ADL)
- Analysis Phase (EAST-ADL)
- Design Phase (EAST-ADL)
- Implementation Phase (AUTOSAR)

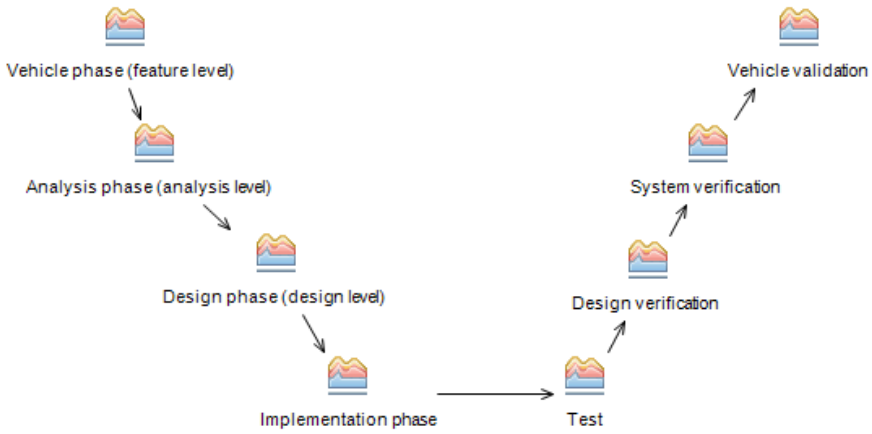

Figure 1- The different Phases of the TIMMO Methodology

The development steps (*tasks*) that are performed in the different phases of the TIMMO methodology are shown in Figure 2. Please note that the TIMMO methodology allows design iterations at each phase. Each task or sequence of tasks involved in creating the solution in the current phase can be repeated based on the knowledge gained in the timing analysis tasks ("Analyze timing …"). For this reason, each phase ends with a milestone acting as gateway for checking the real-time behavior of the created solution before continuing system development in the subsequent phase.

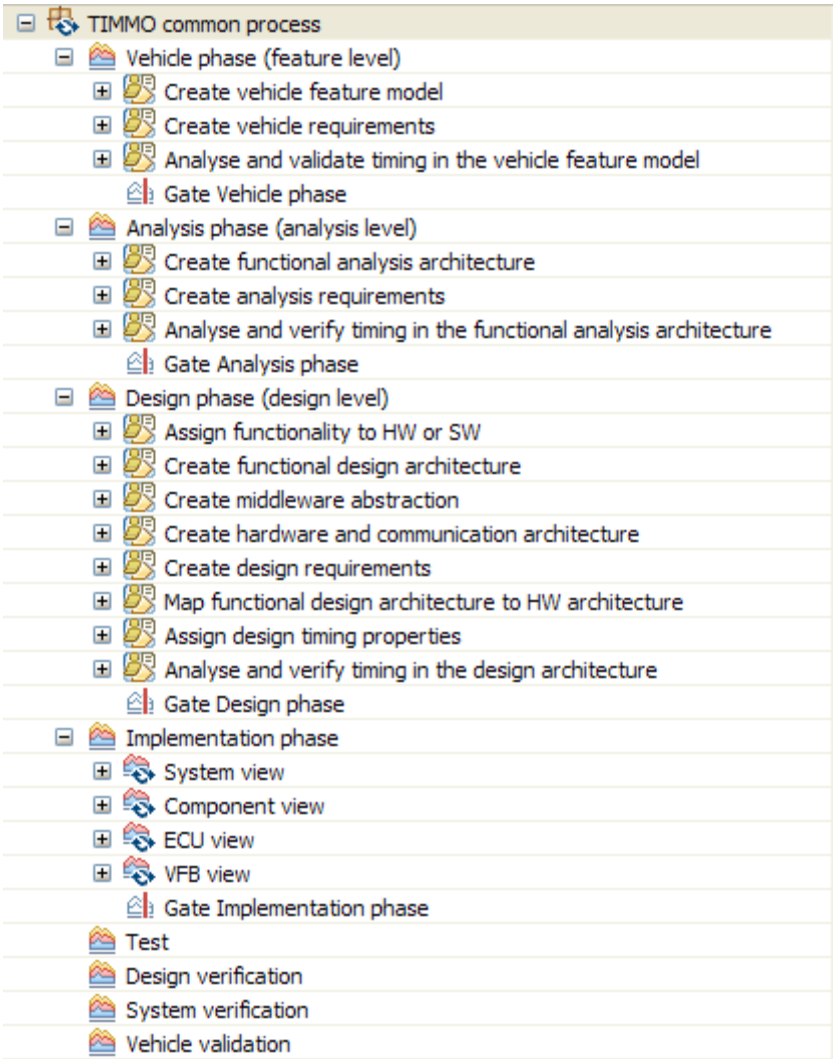

Figure 2 - Different phases and tasks of the TIMMO methodology

Timing Analyses

In the following, the timing analyses that can be performed during the different phases to support the developers in taking design decisions and helping her to ensure the correct real-time behavior are briefly sketched.

Vehicle phase

Timing analysis during the vehicle phase focuses on two aspects. First, the logical validation of the timing requirements is performed. This consists in a first (in most cases subjective) evaluation of the general satisfiability of the timing requirements through timing experts.

The second aspect consists in performing consistency checks of the timing requirements.

Analysis phase

During the analysis phase the timing behavior of initial versions of the functional models are checked against the timing requirements formulated at vehicle phase. Additionally, robustness checks are performed to early detect critical paths in the functional architecture that need special focus in the subsequent phases.

Design phase

During the design phase the first implementation decisions are taken, including the mapping of functionalities to computational resources and utilized communication media. Based on these decisions also many timing properties of the systems are fixed or can be estimated. Therefore, the timing models that can be derived at design phase are much more detailed compared to the previous phases. This enables more detailed timing analyses assessing the approximate dynamic behavior of the software system under development.

At design phase so-called *Response Time Analyses Techniques* can be applied for the first time. They are performed to verify the system's adherence to end-to-end timing requirements. Response time analysis can be performed for a wide range of scopes, spanning from single tasks to complex cause-effect chains involving several ECUs.

Implementation phase

In the implementation phase all details for accurate timing analyses are available. However, while in the previous phases the results of timing analysis can be used to take design decisions, the focus during the implementation shifts to pure validation, i.e. it is checked in detail if all imposed timing requirements from the previous phases are satisfied.

AUTOSAR defines four different views on the developed software system:

- Virtual Function Bus (VFB) View
- System View
- Component View
- Electronic Control Unit (ECU) View

Each of these views focuses on different aspects, and thus different timing analysis techniques are applied. For instance, on system view the validation of global end-to-end delays, e.g. maximum reaction constraints, spanning several ECUs are of interest. In the case of the ECU view, the focus lies on response time analysis on task level and deadlock analysis for shared resources.

Relation to the TIMMO-2-USE methodology

The TIMMO methodology is one of the corner stones for the TIMMO-2-USE methodology. The main differences compared to the TIMMO-2-USE methodology are twofold:

- The TIMMO methodology has a pure top-down view on the development process of automotive software systems. In contrast, the TIMMO-2-USE methodology explicitly considers also bottom-up aspects that play an important role for many use cases.
- The TIMMO methodology's main use case lies on the application of timing analyses during the development process. The TIMMO-2-USE methodology covers many more practical use cases that require the consideration of timing aspects. Examples include the specification of time budgets, the integration of new functionalities into an existing system, the development of control applications, etc.

2.2 The ATESST Methodology

The purpose of the EAST-ADL Methodology, developed in the ATESST2 project, is to give guidance on the use of the EAST-ADL language for the construction, validation and reuse of a well-connected set of development models for automotive embedded software.

Given the complexity of the development activities in automotive embedded software development, it is mandatory to structure the methodology so as to enable a relatively fast and easy access to the EAST-ADL language for a small kernel of essential development activities which can then be seamlessly extended to a comprehensive treatment of the language including more specialized development activities which may not necessarily be used in any development project. Hence the methodology is structured into two major components, as illustrated in

Figure 3:

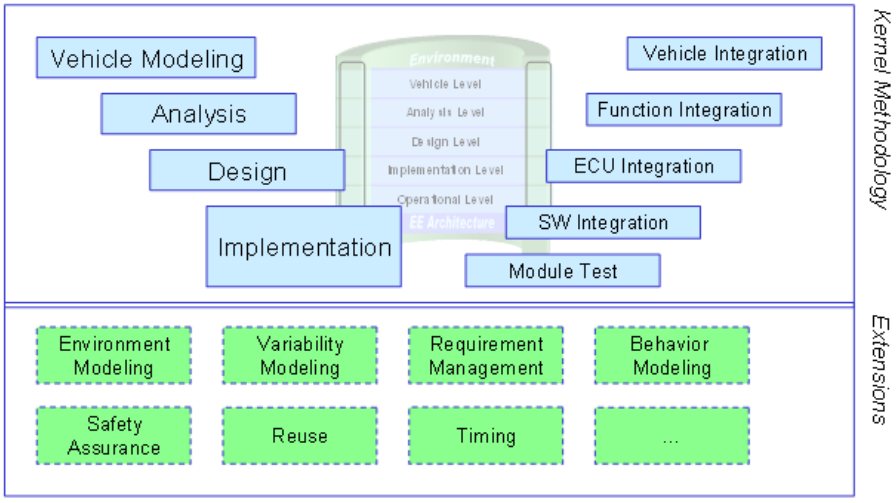

Figure 3 - The structure of the EAST-ADL methodology

The main component, the kernel methodology part, comprises a top-down description of the central constructive phases of automotive embedded software development.

The left side of the kernel methodology directly reflects the abstraction levels adopted by EAST-ADL. These phases describe the tasks and

activities that need to be performed on the respective abstraction level in order to efficiently use the language in automotive embedded system development. The implementation phase, however, contains a reference to the AUTOSAR methodology. It therefore only describes how to transit from the design phase to implementation in AUTOSAR.

On the right side, integration and verification and validation is found. The focus in the EAST-ADL methodology is in these phases on the V&V aspects.

The kernel methodology is extended into a comprehensive methodology for automotive development projects by adding three additional and orthogonal activities to each of these phases:

- Specification of V&V cases to be executed and evaluated during the corresponding integration phase. V&V cases are most typically test cases, but can also include reviews etc.
- Verification of the model on a given abstraction level to the requirements of the model at the abstraction level directly above.
- V&V activities on the model artifacts of a given level itself, i.e. peer reviews, consistency checks, check of modeling guidelines etc.

The second main component of the EAST-ADL methodology consists of a set of complementary loosely-coupled extensions to the kernel methodology. Each of these extensions may be used as an add-on to the kernel activities. The following extensions are currently included:

- **Environment Modeling:** Modeling of the (typically analog or discrete-analog) environment of the system to be developed.
- **Requirements and V&V:** Detailed handling of complex requirements and V&V artifacts.
- **Safety Assurance:** Development of Safety-critical systems
- **Timing:** Detailed handling of timing requirements and properties.
- **Variability Modeling:** Detailed handling of variability modeling.
- **Behavior modeling:** Detailed handling of behavioral modeling

The main idea is that the user of the methodology can compose any set of extensions with the kernel. In order to illustrate the intended correlation and interaction between the extensions, the EAST-ADL methodology presents four different configurations (where a configuration is a set of extensions plus the kernel) of increasing complexity:

- **Core:** Only basic structural models in the kernel methodology.
- **Quality:** Requirements and V&V extensions are added to Core.
- **Quality+:** Variability, timing, behavior and reuse added to Quality.
- **Safety:** Safety added to Quality+.

The timing extension

All timing aspects, including analysis, are captured in the timing extension. The timing extension contains a simplified and collapsed version of the TIMMO methodology, and has a clear focus on specification of timing constraints in the vehicle, analysis and design phases. The reason is that the analyses indicated in the vehicle and analysis phases of the TIMMO methodology are of relatively informal nature. Detailed timing analysis is not available until a hardware architecture is defined in the design phase. The implementation phase of the EAST-ADL methodology does not contain any timing since AUTOSAR v3.1, to which the methodology interfaces, does not support timing.

The timing extension of the EAST-ADL methodology contains the following tasks:

- **Capture Vehicle Timing:** End-to-end timing constraints as well as other timing constraints relevant for Vehicle Features are defined.
- **Capture Internal Analysis Timing:** A budget of delay timing constraints making up end-to-end timing as well as other timing constraints constraining elements inside the FunctionalAnalysisArchitecture are defined.
- **Capture External Analysis Timing:** End-to-end timing constraints as well as other timing constraints on external input and outputs are defined
- **Assess Timing Feasibility:** Consistency of timing constraints and feasibility of meeting timing constraint under a chosen DesignArchitecture is assessed.
- **Capture External Design Timing:** End-to-end timing constraints as well as other timing constraints on external input and outputs are defined.
- **Capture Internal Design Timing:** A budget of delay timing constraints making up end-to-end timing as well as other timing constraints constraining elements inside the FunctionalDesignArchitecture are defined.

Relation to the TIMMO-2-USE methodology

The EAST-ADL methodology addresses all aspects of the automotive EE development process, whereas the TIMMO-2-USE methodology focuses on a certain set of use cases related to timing that are mapped to a Generic Methodology Pattern (GMP), see Section 3. The GMP summarizes all tasks in all extensions (except timing) of the EAST-ADL methodology in one task: Create solution. The tasks in the timing extension correspond to the other tasks in the GMP. However, such mapping is not straight-forward and will result in a many-to-many relation.

3 Generic Methodology Pattern (GMP)

This chapter describes the TIMMO-2-USE Generic Method Pattern GMP. This method pattern is the basis for all steps to be taken during the course of a phase and level of abstraction respectively.

Important Assumptions

The following assumptions shall be kept in mind when reading the following paragraphs:

1. All tasks can be repeated an arbitrary number of times.
2. A sequence of tasks can be repeated an arbitrary number of times.
3. A role or roles performing a task have access to all artifacts that are a) available at the beginning of a phase, and b) created by tasks during the course of the phase. For all details about the work product dependencies refer to the EPF model [4].
4. The term “Timing Property” is used in such a way that it refers to the timing property type *and* its value.

Introduction

As shown in Figure 4, the TIMMO-2-USE Generic Method Pattern consists of the six tasks called “Create Solution”, “Attach Timing Requirements to Solution”, “Create Timing Model”, “Analyze Timing Model”, “Verify Solution against Timing Requirements”, and “Specify and Validate Timing Requirements”. In essence, these tasks have the following purposes:

- “Create Solution” describes the definition of the architecture without any timing information.
- “Attach Timing Requirements to Solution” describes the formulation of timing requirements in terms of the current architecture.
- “Create Timing Model” describes the definition of a formalized model for the calculation of specific timing characteristics based on properties of the current architecture.
- “Analyze Timing Model” describes the actual execution and evaluation of all necessary “calculations” according to the timing model.
- “Verify Solution against Timing Requirements” describes the comparison of the obtained analysis results with the specified timing requirements.
- “Specify and Validate Timing Requirements” describes the identification of mandatory timing characteristics and their promotion to timing requirements for the next development phase.

Please note that for the GMP to be compatible with safety methodologies according to ISO 26262, another task "Refine, Introduce & Validate Requirements" was introduced. This task, however, is treated separately in Section 3.3. Here, only those tasks of the GMP that are relevant for timing considerations are discussed.

By and large, these tasks are carried out at every level of abstraction of the EAST-ADL. Since the EAST-ADL, as well as TIMMO-2-USE, defines a phase for every level of abstraction these tasks are carried out for every level of abstraction: Vehicle, Analysis, Design, Implementation and Operational Level. As shown in Figure 5 there are two exceptions: The first exception is that at the beginning of the Vehicle Phase, a formal work product "Timing Requirements" is not available. The second exception is that at the end of the Operational Phase the task "Specify and Validate Timing Requirements" is not carried out.

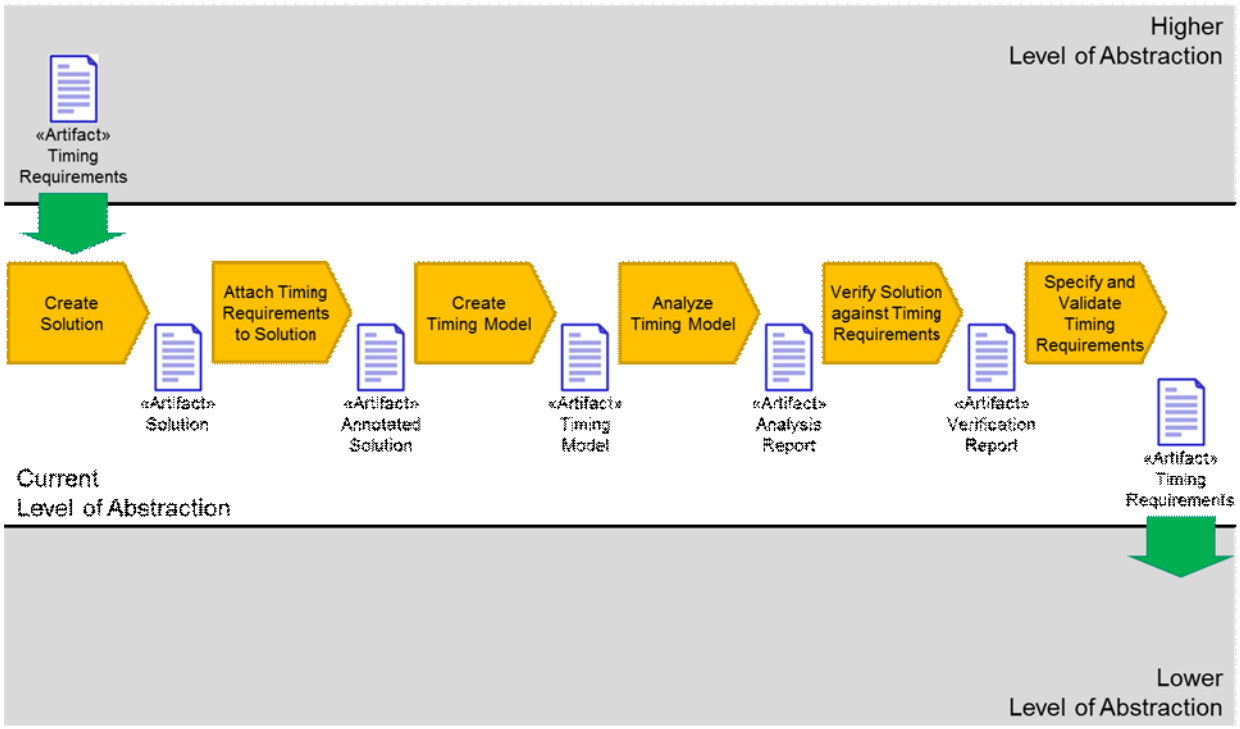

Figure 4 - TIMMO-2-USE Generic Method Pattern

Instantiation

As already indicated in the previous paragraph the TIMMO-2-USE Generic Method Pattern can be applied on all levels of abstraction defined by the EAST-ADL.

Figure 5: Instantiation of TIMMO-2-USE Generic Method Pattern

This instantiation is shown in Figure 5. In every phase of the methodology the corresponding tasks are conducted – except the "Specify and Validate Timing Requirements" in the Operational Phase. At the end of the Vehicle-, Analysis-, Design-, and Implementation Phase the work product "Timing Requirements" is passed to the following phase as basis for subsequent activities in that phase – except at the end of the Operational phase.

In the following, all tasks and their purpose are described in more detail. The tasks are described in the order as they appear in Figure 4 (from left to right).

Create Solution

Based on the given requirements[1], including timing requirements, that originate from the higher level of abstraction respectively previous phase, a solution is created or an already existing solution is revised. While creating/revising the solution the given timing requirements must be considered. In other words the given timing requirements, like any other non-timing requirement, guide the creation of the solution. The resulting solution is captured in appropriate models. In case of EAST-ADL these models are the Technical Feature Model TFM on the Vehicle Level, Functional Analysis Architecture FAA on the Analysis Level, Functional Design Architecture FDA and Hardware Design Architecture HDA on the Design Level, and Environment Model EM

[1] A solution created on the higher level of abstraction respectively in the previous phase, is considered as requirement – a set of requirements – as well.

which is present on all levels of abstraction. Out of these, the first three models primarily capture timing requirements and properties related to the system's application. The Hardware Design Architecture provides parameters for execution and hardware delays. The Environment Model provides characteristics and constraints imposed by the surrounding systems.

Several solutions (alternatives) can evolve from the task "Create Solution" and each of those solutions shall have the potential to satisfy the given requirements. However, each solution may result from specific design decisions that have been taken during the course of this task.

Attach Timing Requirements to Solution

Based on the created solution the timing requirements are formulated in a way that is suitable for further processing on the current level of abstraction. Timing requirements that are carried over from a previous phase must be transformed and attached to the solution architecture accordingly, such that they are "compatible" with the timing model (and the timing properties) on the current level of abstraction.

In a nutshell: Timing requirements are expressed using events, event chains, and timing constraints that are imposed on these events and event chains. Events refer to locations, usually ports, in a solution model at which the occurrences of the events are observed; while event chains specify a causal relationship between events and their temporal occurrences. During every phase, a solution model is created based on the requirements and on the solution model created in the previous phase. An event specified in the previous phase referring to an observable location in the corresponding solution model possibly has to be transformed or mapped into an event referring to an observable location in the solution model created during the current phase. This transformation has to be performed for all events and event chains, and especially the *values* of the timing requirements imposed on event chains.

This task must be performed for every alternative solution that evolve from the task "Create Solution".

Create Timing Model

Once the solution has been created and the timing requirements have been attached in a way that suits the current level of abstraction, a timing model for this solution is created.

The timing model defines how – based on the timing properties of the solution – specific timing analysis methods can be applied, in order to predict / calculate the dynamic behavior of the solution and the timing characteristics (e.g. the WCRT of a control function) emerging from it.

The timing properties required by the various timing analysis methods need to be determined and assessed. The methods applied to determine the particular values are manifold: expert knowledge and estimation, knowledge from previous projects or iterations within the

current project, formalized analysis, simulation, etc. In addition, the methods being used may vary depending on the phase: On higher levels of abstraction other methods are used than on lower levels. For example, scheduling analysis is used on implementation level, but not on vehicle level.

The most appropriate and suitable method should be selected for this purpose.

Note that the purpose of this task is not to define new types of timing analysis methods or timing properties, but to decide how these can be practically used to describe the dynamic behavior of the solution.

This task must be performed for alternative solutions that evolve from the task “Create Solution”. And with regard to the dynamic behavior of the solutions there may be different timing models leading to different sets of timing properties and their values.

Analyze Timing Model

Depending on the specifics of the timing model and the different timing analysis methods which are applied, all necessary calculations are executed and their results – looking at the whole picture and the target system – are evaluated.

It may happen that several alternative solutions are available, and in this case the purpose of the task “Analyze Timing Model” is to identify and quantify the strengths of every solution with regard to the dynamic – temporal – behavior. One can select the most appropriate and/or promising solutions in order to proceed with the development.

Verify Solution against Timing Requirements

In order to answer the question – does the solution satisfy the given timing requirements – the values of the timing properties obtained by the analysis are compared against the values of the requirements attached to the solution.

The primary purpose of this task is to decide whether to continue conducting the subsequent tasks in the development process, or to repeat any or a sequence of previous tasks. In other words at this point it is decided “whether the numbers are good enough for progressing”, or whether those numbers have to be revised (iteration). It could also happen that the solution subject to timing analysis must be revised, or even worse, a new solution must be searched.

If several alternative solutions are available then the purpose of the task “Verify Solution against Timing Requirements” is to verify the timing properties of every solution. Eventually, one has to select the most appropriate solution – one solution – in order to proceed with the development.

Milestone: Quality Gate

At a quality gate, which is not shown in the given figures, immediately following the task “Verify Solution against Timing Requirements” the results of the verification are checked, and a decision is taken to either continue or repeat the phase. Of course, if the quality gate is negative the necessary actions depend on the kind of defect detected. For example, sometimes it would only be necessary to repeat a specific or a number of tasks, rather than all tasks in the phase.

Specify and Validate Timing Requirements

Once the quality gate has been passed all or some of the obtained timing properties and transformed timing requirements are converted into corresponding timing requirements.

The result of the task is *not* that *all* timing properties that were found in the previous tasks are converted into timing requirements, but only those of them which are fundamental and important for design decision to be taken in subsequent steps. One criterion for identifying timing properties as timing requirements is that they were critical for the verification performed.

These timing requirements are the basis for any design work being conducted during the next phase.

3.1 Example

This section introduces a very simple example that is used to explain how the Generic Method Pattern is applied respectively utilized. In particular, it describes how the Generic Method Pattern is applied on the Design Level.

Example – Introduction

At the beginning of a phase the solution and the corresponding timing requirements are available from the previous phase respectively higher level of abstraction – the Analysis Level. This solution is shown in the upper part of Figure 6. The solution consists of two functional devices («FD») and one function/component («AF»). One of the functional devices, the one on the left-hand side in the figure, represents the sensor and the other functional device, the one on the right-hand side in the figure, represents the actuator. The purpose of the functional device named “Sensor” is to provide data from the environment to the E/E system to be developed; while the purpose of the functional device called “Actuator” is to “control/impact” the environment. The [analysis] function/component («AF») called “Function” processes the data received from the environment via the functional device “Sensor” and controls/impacts the environment via the functional device called “Actuator”.
In the artifact “Timing Requirements” attached to the solution one event chain is specified. This event chain and the timing constraint are depicted by the blue colored event chain drawn above the function/component called “Function” in Figure 7. The event chain references an event and its occurrence can be observed at the required port of the functional device called “Sensor”. This event is playing the role of the stimulus. The event chain references a second event and its occurrence can be observed at the provided port of the functional device called “Actuator”. Thisvent is playing the role of the response.
A ReactionConstraint (TC) is imposed on this event chain (EC) and its value is 125 ms including a variation – jitter – of 30 ms yielding in a time range of 110 ms to 140 ms.

Scope	ReactionConstraint	Minimum	Maximum
Sensor - Function - Actuator	125 ms, -15 ms, +15 ms	110 ms	140 ms

Example – Create Solution

On the current level of abstraction – in the current phase – a solution is created by performing the task “Create Solution”. The created solution is supposed to satisfy the given functional and non-functional requirements, specifically the timing requirements.

The solution is shown in the lower part of Figure 6. It consists of two Hardware Functions («HF»), two Basic Software Functions («BSF»), and two Logical Device Managers («LDM»): one called “Sensor” and the other called “Actuator”. Additionally, two [design] functions/components («DF») called “F1” and “F2” are part of the system architecture. The combination on the left hand side in the figure corresponds to the sensor, and the combination on the right hand side corresponds to the actuator. The two [design] functions/components («DF») called “F1” and “F2” processing the data received from the environment via the HF, BSF and LDM, and control/impact the environment via the LDM, BSF, and HF. The Logical Device Manager “Actuator” provides additional data to the function/component called “F1”.

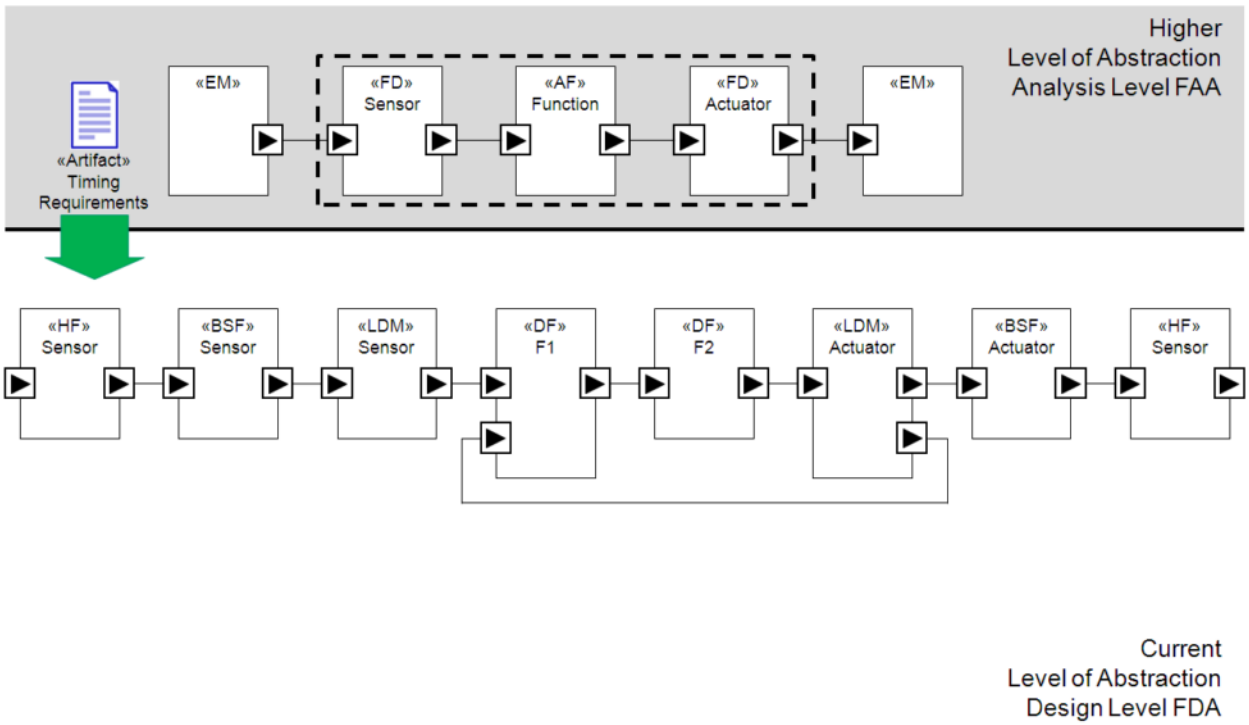

Figure 6 - A simple example to demonstrate the use of the TIMMO-2-USE Generic Method Pattern

Example – Attach Timing Requirements to Solution

The timing requirements originated from the previous phase, the Analysis Phase, are transformed into timing requirements that correspond with the solution created in the current phase, the Design Phase. In the example the given timing requirement, the event chain and timing constraint in the upper part of Figure 7, are transformed into an event chain with corresponding timing requirement that is imposed on the current phase’s solution.

The result of the transformation is

Scope	Latency Timing Constraint	Minimum	Maximum
HF “Sensor” … HF	125 ms, -15 ms, +15 ms	110 ms	140 ms

“Actuator”			

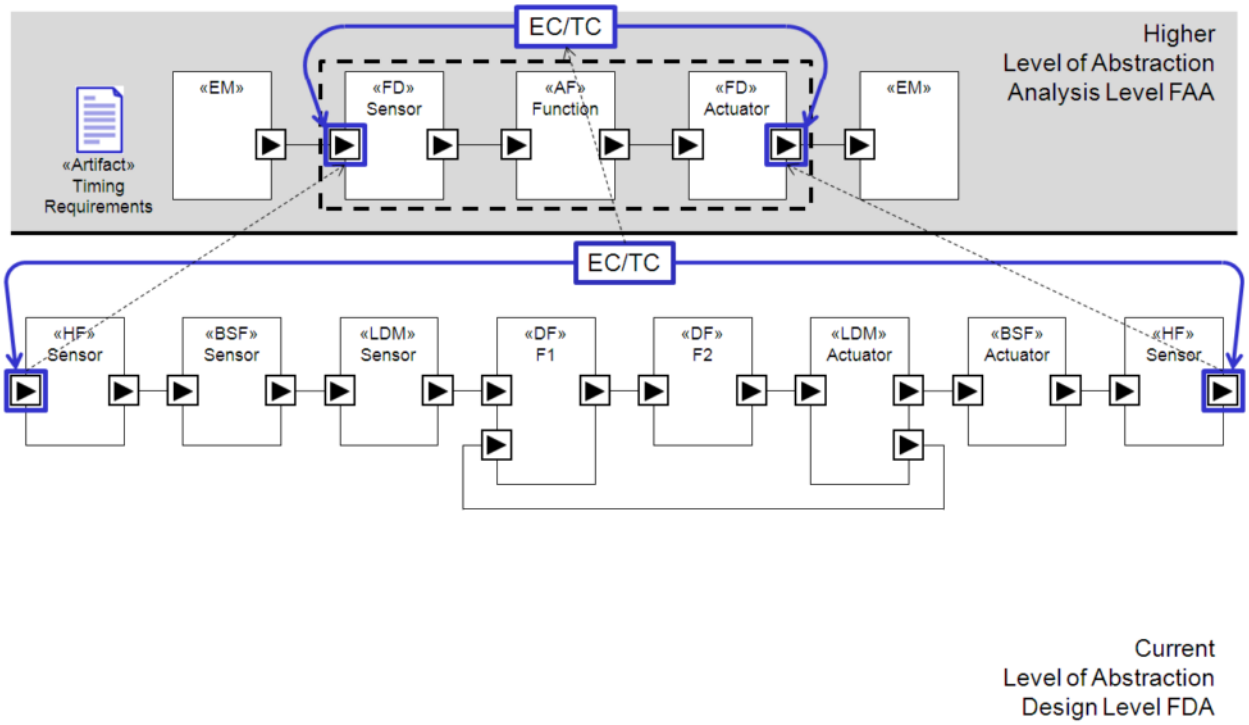

Figure 7 - A simple example to demonstrate the use of the TIMMO-2-USE Generic Method Pattern and transforming timing requirements between levels of abstraction.

Observation: On the first view it seems obvious that the event chain/timing constraint specified on the higher level of abstraction (Analysis Level) is transformed in a one-to-one manner to an event chain/timing constraint on the current level of abstraction (Design Level). And a valid question is whether this timing requirement shall be transformed at all.

Example – Create Timing Model

During the course of the task “Create Timing Model” the solution is annotated with events, event chains, and timing constraints as shown in the lower part of Figure 8 – depicted by the red colored event chain drawn above every element of the solution. On this level of abstraction the given event chain including its latency timing constraint is broken down into seven subsequent event chains, playing the role of event chain segments, and latency timing constraints are imposed on those seven event chains respectively event chain segments. In addition a *periodic* event triggering constraint is imposed on the event that is observed at the provided port of the basic software function called “Sensor”, because the solution provides data for example periodically.

In this example, an event chain referring to the second provided port of logical device manager called “Actuator” and the second required port of the [design] function/component called “F1” is not specified, because this path is considered unimportant with regard to timing. Note that in other cases this path could possibly have a significant impact on the dynamic behavior of the system, e.g. in a control application, and then must be considered accordingly.

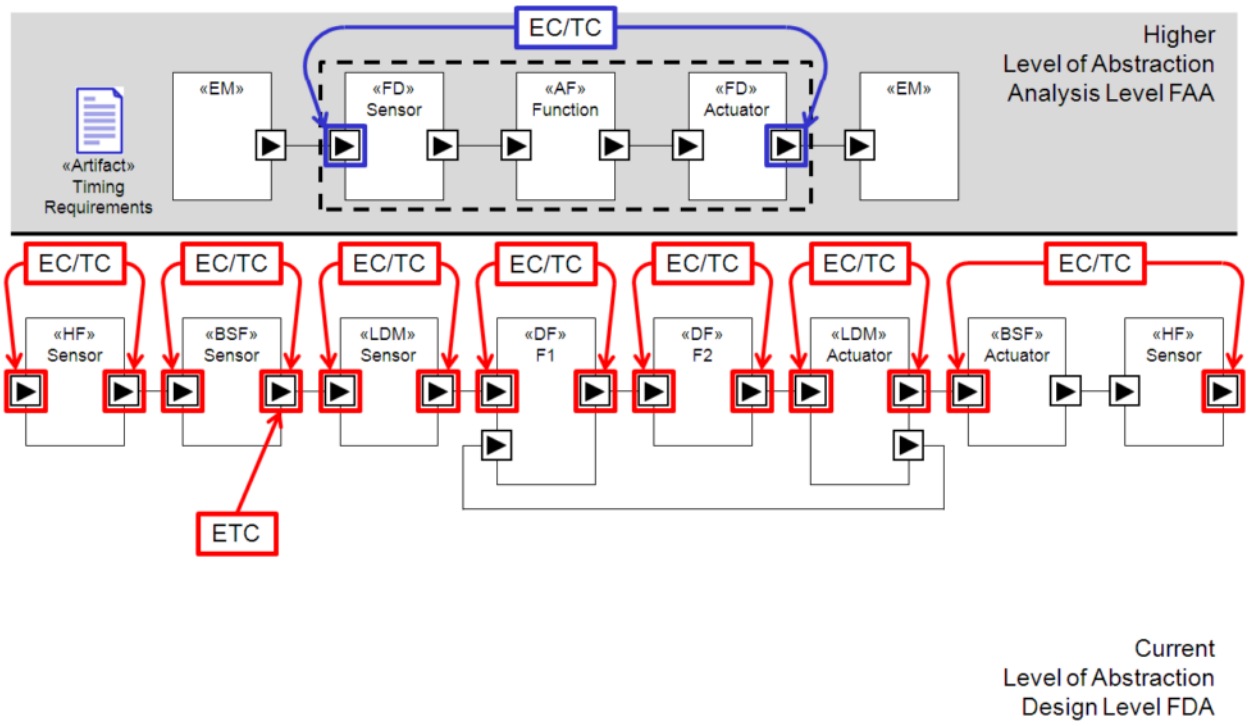

Figure 8 - The simple example to demonstrate the use of the TIMMO-2-USE Generic Method Pattern annotated by timing information

Furthermore, an event chain is specified referring to an event that is observed at the required port of the hardware function called “Sensor”, and to an event that is observed at the provided port of the hardware function called “Actuator”. On that event chain a timing constraint is imposed. This timing constraint – the property and the value – may be the same as the given one [timing requirement].

The values of all those timing properties are determined, too, and for good reasons one could specify the following latency timing constraints:

1. A latency timing constraint imposed on the combination HF, BSF, and LDM called “Sensor” of 30 ms including a variation of -2 ms and +5 ms resulting in a time range of 28 ms to 35 ms.
2. A latency timing constraint imposed on the function/component called “F1” of 20 ms including a variation of -1 ms and +2 ms resulting in a time range of 19 ms to 22 ms.
3. A latency timing constraint imposed on the function/component called “F2” of 45 ms including a variation of -5 ms and +3 ms resulting in a time range of 40 ms to 48 ms.
4. A latency timing constraint imposed on the combination LDM, BSF, and HF called “Actuator” of 25 ms including a variation of -2 ms and +10 ms resulting in a time range of 23 ms to 35 ms.

The following table summarizes the values of all determined latency timing constraints.

Component	Latency Timing Constraint	Minimum	Maximum
HF, BSF, LDM “Sensor”	30 ms, -2 ms, +5 ms	28 ms	35 ms
F1	20 ms, -1 ms, +2 ms	19 ms	22 ms
F2	45 ms, -5 ms, +3 ms	40 ms	48 ms
LDM, BSF, HF “Actuator”	25 ms, -2 ms, +10 ms	23 ms	35 ms
	Totals:	110 ms	140 ms

Additionally, the value of the periodic event triggering constraint that is imposed on the event observable at the provided port of the basic software function called “Sensor” is 10 ms including a variation – jitter – of 2 ms resulting in a time range of 8 ms to 12 ms.

Example – Analyze Timing Model

In this step – carrying out the task “Analyze Timing Model” – the values of the timing properties specified are scrutinized.

In the example, executable models that are available for every component are used to perform simulations in order to analyze the timing behavior of the given solution. During the simulations it turns out that the function/component “F1” tends to have a slightly larger response time than specified during the task “Find Timing Properties” – typically 5 ms – which leads to a variation of +8 ms.

Further analyses show that the assumptions made during the task “Create Timing Model” with regard to the dynamic behavior of the inter-connect between “Actuator” and “F1” were not correct. It turns out that the variation of the response time is not as large as presumed before. Continuing simulations lead to the fact that the latency timing constraints can be adjusted accordingly; in this case the variation is not more than +2 ms.

Table 1 summarizes the values of all determined latency timing constraints.

Component	Latency Timing Constraint	Minimum	Maximum
HF, BSF, LDM “Sensor”	30 ms, -2 ms, +5 ms	28 ms	35 ms
F1	20 ms, -1 ms, +8 ms	19 ms	28 ms
F2	45 ms, -5 ms, +3 ms	40 ms	48 ms
LDM, BSF, HF “Actuator”	25 ms, -2 ms, +2 ms	23 ms	27 ms
	Totals:	110 ms	138 ms

Table 1: New values of the latency timing constraints after performing timing analyses on the given solution

Example – Verify Solution against Timing Requirements

The obtained values of the timing properties are now compared against the given timing constraint transformed from the higher level of abstraction.

Alternative #1: For this purpose, an event chain is specified that references the event observable at the required port of the hardware function called ”Sensor”, playing the role “Stimulus”, and that references the event observable at the provided port of the hardware function called “Actuator”, playing the role “Response”.

Alternative #2: For this purpose, an event chain is specified that references the event observable at the required port of the basic software function called ”Sensor”, playing the role “Stimulus”, and that references the event observable at the provided port of the basic software function called “Actuator”, playing the role “Response”.

This event chain and the timing constraint imposed on it are depicted by the blue colored event chain shown in the bottom part of Figure 9. A latency timing constraint is imposed on this event chain and the value of this latency timing constraint is as follows:

Latency Timing Constraint	Minimum	Maximum
120 ms, -10 ms, +18 ms	110 ms	138 ms

A comparison of this timing property of the solution with the given one mentioned in the introduction of the example shows that the solution satisfies the given timing constraint respectively latency timing constraint: 110 to 138 ms versus 110 to 140 ms.

Example – Specify and Validate Timing Requirements

As a formal step the determined timing property – latency timing constraint – and its value – 110 ms to 138 ms – are declared as timing requirement/constraint which shall be considered in the next phase, in particular when carrying out the task “Create Solution” in the following phase. Note, that the timing properties, associated with every functional device and function/component, are not converted into timing requirements.

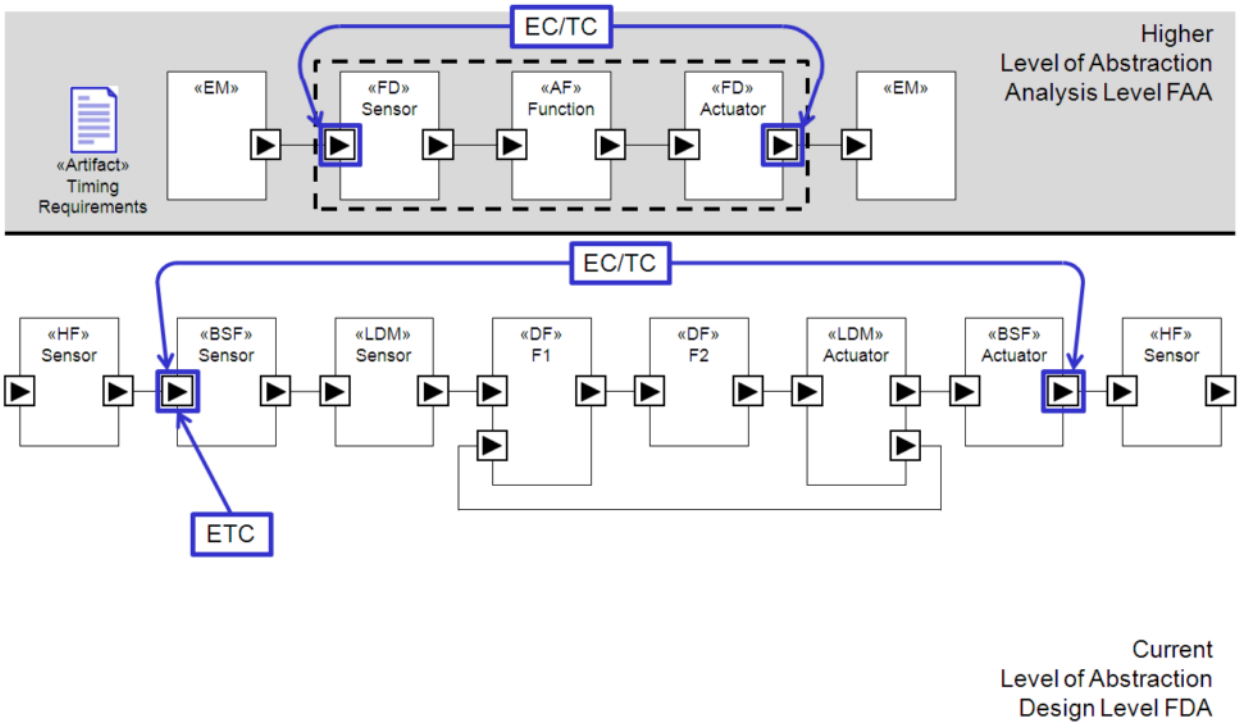

Figure 9 - A simple example to demonstrate the use of the TIMMO-2-USE Generic Method Pattern and specifying timing requirements for the next level of abstraction.

3.2 Abstracting Timing Properties

This sub-section describes the idea of "Abstracting Timing Properties" on a lower level of abstraction in order to use them on a higher level of abstraction. The results of this abstraction are used as additional (optional) input work product for the task "Create Timing Model".

Figure 10 shows a simplified view of the methodology with regard to this approach. The task "Transform Timing Properties" on the current level of abstraction transforms the timing properties' values of a solution created on the lower level of abstraction into values of timing properties that are used at the current level of abstraction. The transformed timing properties, including their values, are an optional input work product for the task "Attach Timing Requirements to Solution" conducted on the current level of abstraction. The idea behind this is that values of timing properties that are obtained during later phases of the development process can be used on higher levels of abstraction respectively in earlier phases of the development process. This is an important capability in order to support iterative development processes.

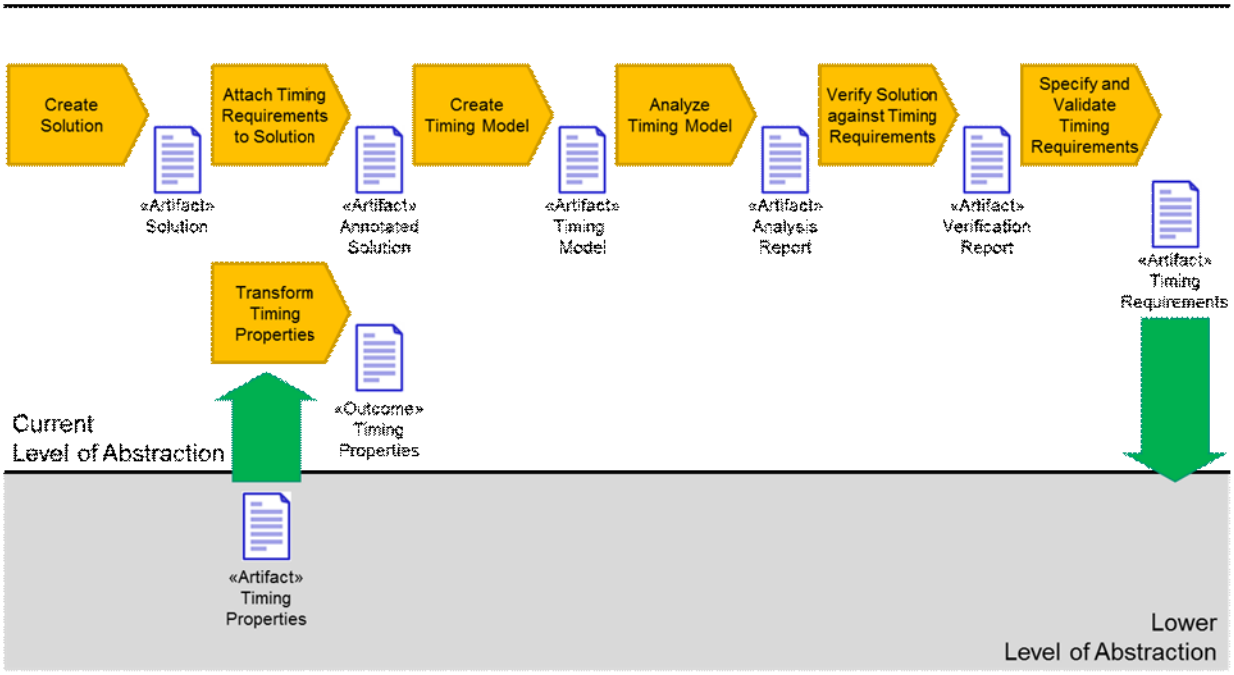

Figure 10 - Abstracting Timing Properties

3.3 Extending the GMP with Safety Aspects

The GMP general structure was designed with the aim of being applicable to other aspects of the E/E development as well, not only timing. To demonstrate and validate this, it has been applied to functional safety according to ISO 26262 in cooperation with the TimeSafe project.

When investigating ISO 26262, the need to refine requirements at the current abstraction level before the actual solution is created and modeled, was manifest. In particular, ISO 26262 explicitly requires the technical safety requirements (design level) to be specified before the system design is created and the hardware and software safety requirements (implementation level) to be specified before the hardware and software design is created. For this reason the first task "Refine, Introduce and Validate Requirements" was introduced. The seven tasks of the GMP applied to safety are shown in Figure 11 and the first task is briefly described as follows.

Figure 11 - Generic Method Pattern applied to functional safety

Refine, Introduce & Validate Requirements

In this task, the *requirements* from previous/upper abstraction level are refined and complemented to this abstraction level. This means that some *assumptions on the design* are made on this abstraction level. Typically the requirements introduced or refined in this work task are later refined and/or formalized when the design has been made. For functional safety, the functional, technical, hardware or software safety requirements are introduced in this work task and allocated to the design and detailed in later work tasks, thus creating the functional/technical safety concept or being part of the hardware/software design. The refined and introduced requirements are also *validated*. This means that the requirements are checked to be consistent and corresponding to the actual expectations (intended requirements) of the stakeholders.

4 Integration of the results of WP2 and WP3 into the methodology

In order to integrate the results of the other technical work packages into the TIMMO-2-USE methodology, the concepts of "TADL guides" and "Tool mentors" were developed in joint effort with work packages 2 and 3.

TADL guides explain the usage of TADL2 concept during the different methodology tasks, whereas Tool mentors link to tools and algorithms that are relevant for the completion of the task at hand.

4.1 TADL2 guides

A TADL guide describes which language elements can be used in a specific task of the TIMMO-2-USE methodology to describe timing information. TADL guides, therefore, represent a link between the technical results of the work packages 2 (Language) and 4 (Methodology).

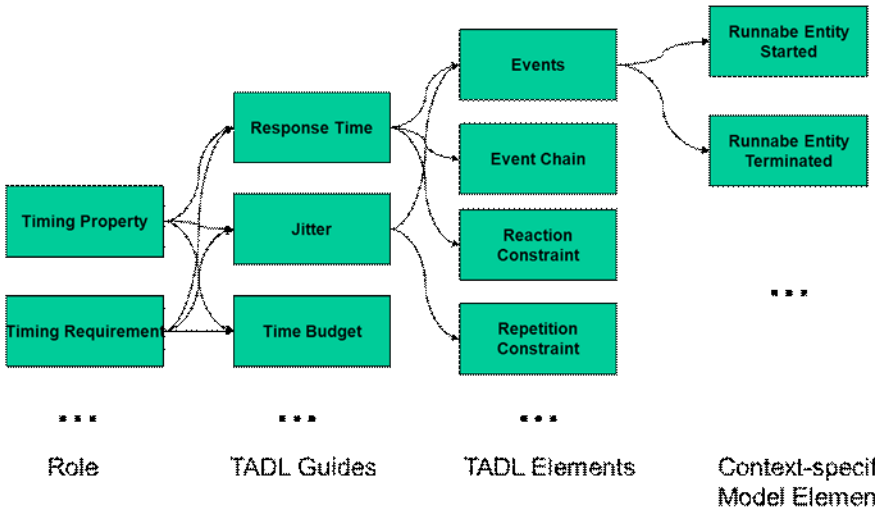

Figure 12 - Structure of TADL guides

Figure 12 shows the overall structure of the TADL guides. Basically, each TADL guide can be interpreted either as timing property or as timing requirement depending on the role it is referenced in. Then, there is the actual description of the TADL2 language concept. Examples for TADL guides that were created are:

- Execution time (Worst-case, Best-case, Simulated, Measured)
- End-to-end Latency
- Sampling Rates
- Time Budget
- Response Time

- Communication Delay
- Slack
- Repetition pattern
- Synchronization

Each TADL guide references the TADL elements, i.e. classes of the TADL2 meta-model, that are necessary to use the TADL guide. Examples are events, event chains, etc.

In order to maximize the usefulness of the TADL guide, the guide for the same TADL2 concept is instantiated for each abstraction level it is applicable for. Additionally, each TADL guide is completed with context specific model elements, like the list of relevant EAST-ADL and AUTOSAR events, and a context specific example.

The concrete TADL guides are not included in this deliverable. They can be found in the EPF version of the TIMMO-2-USE methodology.

4.2 Tool mentors

A Tool mentor describes which algorithm or tool can be used in a specific task of the TIMMO-2-USE methodology to solve a specific timing-related problem at hand. Tool mentors, therefore, represent a link between the technical results of the work packages 3 (Tools & Algorithms) and 4 (Methodology). Tool mentors give *precise* hints on the possibilities to apply tools and algorithms to solve specific problems depending on the context. Therefore, for each considered tool or algorithm, different Tool mentors for each relevant combination of abstraction level and methodology task were created.

All tool mentors for the tools and algorithms developed in the TIMMO-2-USE project were created using the following template:

1. *Abstraction level*: Abstraction level the Tool mentor can be applied to. If a tool/algorithm can be applied on various abstraction levels a separate Tool mentor was created for each abstraction level
2. *Use Cases*: Main use cases of TIMMO-2-USE for that the Tool mentor can be applied.
3. *Covered aspect*: Kind of timing information that is delivered by the tool/algorithm, like for instance worst-case execution time (WCET)
4. *Algorithm*: Detail on the underlying formalism and technique
5. *Inputs*: Details on the required input data like, for instance, source code, binary code, etc.
6. *Particular constraints on inputs*
7. *Preparation of input*: Explanation of ways on “How to get the data ready for applying the algorithm”.
8. *Invocation of the algorithm*: Hints on how to use the tool

9. *Outputs:* Kind and quality of results delivered by the tool / algorithm
10. *Visualization of results*: Information on how the output can be assessed, e.g. textual report, graphical visualization, etc.

The concrete Tool mentors are not included in this deliverable. They can be found in the EPF version of the TIMMO-2-USE methodology or in D12.

5 Application of the Generic Method Pattern to Use Cases

In this section the application of the GMP to the covered main use cases, identified within the TIMMO-2-USE project, is conducted. For each main use case a different instance of the GMP was created, giving details on all timing related activities. The different use cases are additionally modelled with SPEM (Software Process Engineering Metamodel) using EPF (Eclipse Process Framework). This version can be found under [4].

5.1 Integrate reusable component

Problem statement

In the context of the automotive industry, an OEM offers a range of vehicles marketed in different classes which provide different extents of functionalities related to safety, comfort, or similar criteria. Caused by marketing tendencies and proceedings in technology, vehicles are being enriched by new functionalities either newly invented or taken over from higher class vehicles. In that case new functionalities are integrated step-wise into an existing system during the development phase. Similar integration problems arise when a new platform generation is being developed and functions are moved from one ECU to another.

Usually, new functionalities cannot be introduced independently of the existing system's functionalities due to interference with the existing system's ECU resources and communication network.

Changing a system's architecture necessarily changes its behavior with respect to timing. For instance, end-to-end latencies might increase due to additional preemptions of tasks or arbitration of bus messages.

This use case addresses the challenges which arise during the process of integration.

Overview

In the following, the focus will be on the Design phase but similar considerations apply to the Implementation phase, i.e the workflow tasks are basically the same.

The integration may cover one or more ECUs including their communication paths. We assume that one or several target ECUs onto which the design function in question shall be integrated has been selected. The actual ECU selection process is not covered by this use case. However, there are several aspects that must be considered when choosing the target ECU(s), like for instance the targeted functional domain (e.g. body controller), the physical location (e.g. near front wheels), or the availability of input signals (e.g. sensor signals, buses), etc.

The investigations on the use case "Integrate Re-usable Component" assume that the software system executed on the target ECU(s),

which the design function is to be integrated into, was developed according to the Generic Methodology Pattern (see Section 3). This means, that the model contains all components necessary for fulfilling the system's functionality, and all timing properties of the components are known and described.

The use case considers two integration scenarios:

A) Adding a legacy design function:

   Here it is assumed that the legacy design contains TADL2 compliant timing information.

B) Developing and adding a new design function:

   For scenario B, there are two approaches are distinguished:

   B1) Developing the new design function stand-alone without taking into account interactions with the target system. In this case a separate model is created for the new design function including timing information. In a second step this new model is merged into the existing one as in scenario A.

   B2) Developing the new design function directly into the existing model explicitly taking into account interactions with the target system.

Approach B1 includes the development of a new functionality from scratch. It handles timing information according to the GMP resulting in a stand-alone solution. This stand-alone solution would then have to be integrated into the existing solution which is identical to scenario A.

In the remainder of this section, the scenarios A and B1 are discussed. The focus lies, thus, on the **integration of one EAST-ADL model into another, both models already containing timing information**.

The further discussions refer to the *Design* phase. They can also be applied to the *Implementation* phase. During the *Vehicle* and *Analysis* phases, models consist of pure functional components where end-to-end delays are composed by chaining budget segments of the components, and resources are considered to be infinite. Consequently, integration effects cannot be investigated during those phases. In contrast, during the *Design* phase components are declared to be realized in hardware or software resulting in a Hardware Design Architecture (HDA) and a Functional Design Architecture (FDA). On this level, and on the lower *Implementation* level, the integration aspect can be investigated, i.e. the interference of components due to competition for common resources.

Mapping to Generic Methodology Pattern

Figure 13 illustrates the integration process, and how it maps to the GMP presented in Section 3.

In the following paragraphs, the existing system which the design function is integrated into is referenced with the suffix _EXIST (e.g. Solution_EXIST) while the design function to be integrated is referenced with the suffix _INTEG (e.g. Solution_INTEG). The final

system including both solutions is referenced with the suffix _BOTH (e.g. Solution_BOTH).

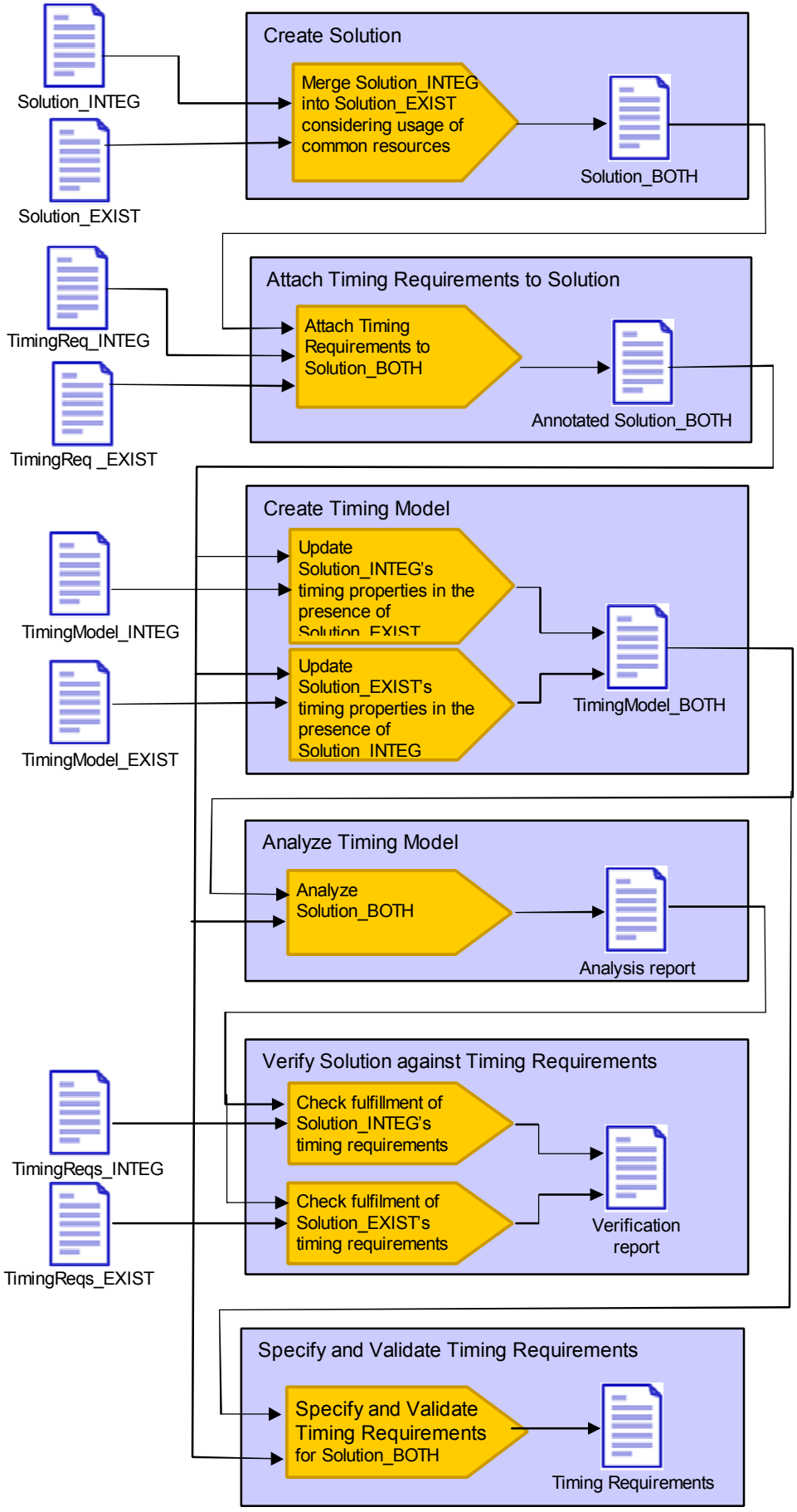

Figure 13 - Generic methodology applied on integration of a reusable component.

Create Solution

When performing the task *Create Solution*, the components of Solution_EXIST and Solution_INTEG have to be brought together to become Solution_BOTH. From the functional perspective, the solutions still may co-exist in the resulting model as long as no functional synergy is detected. This also implies that both solution topologies including inter-component communication may remain unchanged. However, it will often be a design goal to search for synergies in order to provide cost-efficient solutions. Therefore, reuse of input/output ports and network messages is advised, e.g. in case both solutions use the same sensor signal, or a required signal is already available on another ECU.

Attach Timing Requirements to Solution

It is assumed that the resulting Solution_BOTH will contain the same events like the previous Solution_EXIST and Solution_INTEG, so that all timing requirements applied on the previous solutions persist in Solution_BOTH. If functional synergy is exploited in Solution_BOTH, then some requirements will refer to common event chain segments indicating timing dependencies.

Create Timing Model

At first, a timing model is created for Solution_BOTH as described in the GMP. Large parts of this timing model may be adopted from the previous solutions without modification. As described above, some parts of the previous timing models may overlap, if functional synergy is exploited, i.e. two events from the previous solutions are merged into one and the resulting event chains have common segments. If for example, a sensor signal is reused as a common resource in Solution_BOTH, then the segments of a timing event chain have to be adapted, but the requirement for the end-to-end latency is the same.

The following two subtasks are described in more detail:

1) Update _INTEG's timing properties,
2) Update _EXIST's timing properties.

The focus in subtask 1 is on updating the timing properties of Solution_INTEG, like WCET of functions. This is necessary, since usually the target system already accommodating Solution_EXIST is different from the system which Solution_INTEG was developed on. There might be different ways of updating the timing properties. For instance, for worst-case execution times the following approaches are possible:

- Transforming Solution_INTEG's timing properties from the old to the new hardware/software design architecture.

  One possible method here is *extrapolation*, i.e. given an old timing property value the new value is computed by applying an extrapolation formula. The simplest case is linear extrapolation. For example, if the processor clock rate changes, then the new WCET may be estimated as $WCET_{new} = WCET_{old} * Clock_{old} /$

$Clock_{new}$, where Clock is the number of processor cycles per second. For this simple formula it is assumed that the number of processor cycles for reading and writing memory remains the same. Note, that extrapolation is a kind of estimation, so it may be necessary to add a safety margin to the new WCET and to classify it accordingly. One advantage is that extrapolation can be supported by tools.

- Measuring execution times of Solution_INTEG's components on the new target – this follows a bottom-up approach and requires the availability of the target processor and the possibility of easily porting Solution_INTEG on the target processor before integration.
- Complete re-computation of the WCET by static analysis of the new function in the environment of Solution_BOTH. This also requires detailed knowledge about the implementation on the new target.

The methodology does ot give advice on how to update the necessary timing properties; this is subject to specific characteristics of a particular project.

Subtask 2 deals with updating the timing properties of Solution_EXIST. These timing properties might change in the presence of the integrated design function. Examples of timing properties subject to change are:

- WCET (e.g. due to caching effects, pipelining, etc.)
- Scheduling parameters (e.g. priorities, periods, runnable order, etc.).
- Arbitration of network messages.

Analyze Timing Model

In the task *Analyze Timing Model*, the Solution_BOTH model is analyzed by means of, for instance, simulation and/or static analysis. This will result in timing property values and metrics relevant for judging the timing behavior of Solution_BOTH.

In particular, it is necessary to also re-analyze the timing behavior of components originating from Solution_EXIST, because after integration some of their timing property values may have changed. For instance, response times (WCRT) may increase due to inter-component interference from added components (see Figure 14).

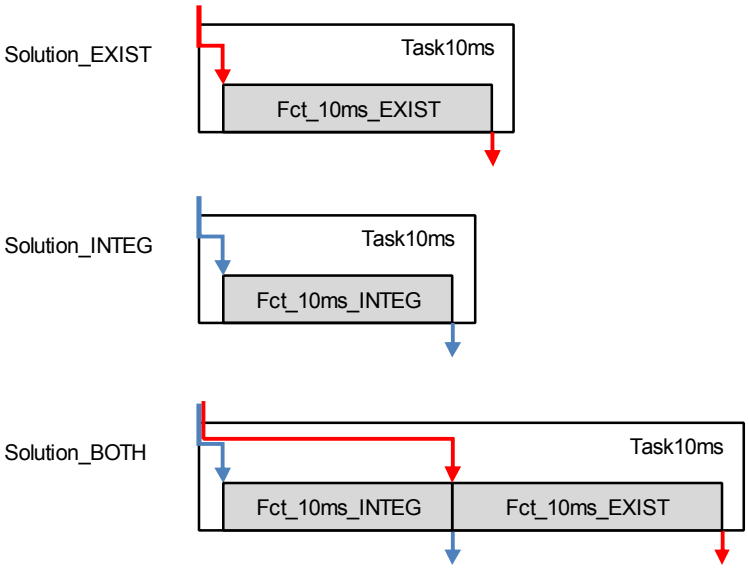

Figure 14 - Timing behavior before and after integration

Figure 14 illustrates possible effects due to integration. Both Solution_EXIST and Solution_INTEG have functions which are activated with 10ms period. Fct_10ms_INTEG has been mapped into the same 10ms task which contains Fct_10ms_EXIST. Certain considerations led to the design decision that Fct_10ms_INTEG shall be placed at the beginning of the task. Of course, this leads to an increased response time of the 10ms task (and all lower-priority tasks) compared to before the integration. Also the response time of function Fct_10ms_EXIST will increase in the depicted scenario.

Verify Solution against Timing Requirements

The task *Verify Solution against Timing Properties* compares the analysis results for Solution_BOTH with the requirements.

Besides verifying the timing requirements of the integrated Solution_INTEG also the timing requirements of the original system Solution_EXIST, which is now a part of Solution_BOTH, has to be re-verified.

Specify and Validate Timing Requirements

The scope of the task *Specify and Validate Timing Requirements* is to identify timing requirements for the next lower level of abstraction. These timing requirements refer to event chain segments in the timing model of Solution_BOTH. The time budgets for each segment must not exceed the specified total budget for the overall event chain. They have to be transformed into requirements for the lower abstraction layer in the next step.

The activities to be done in the task *Specify Timing Requirements* are not specific to this use case. Therefore, this task is not described here in more detail.

Remark

If the reusable component (Solution_EXIST) was already applied in other types of vehicles as described in the beginning, then it is likely that not only a *Design*-level solution exists, but also an *Implementation*-level solution, which has to be ported to the new platform.

On the *Implementation* Level similar tasks have to be performed as described for the *Design* Level, i.e. an existing AUTOSAR Solution_EXIST has to be integrated into an AUTOSAR Solution_INTEG. Additional complexity results from the re-use of common software components, e.g. for basic software services. Again, the timing properties of both solution parts will persist, but their values may change and must be verified or validated against the original requirements.

5.2 Specify timing budgets

Problem statement

A driver generally has certain expectations on the reactivity of the vehicle he is driving. For example, it would not be acceptable to wait for 5 seconds for the doors to unlock after he has pressed the key. A more acceptable time limit would be 1 second. The main characteristic of this example is that the time limit refers to a stimulus from and to a response to the environment of the system. Such time limits, hereafter called end-to-end delays (or requirements), are specified based on a user's perception with respect to certain functionality.

In a design, the data and control flow paths between a stimulus and a response generally go through several components. Each path delimited by a stimulus and a response that relate to the environment are called end-to-end event chains. The components in the end-to-end event chain are to be implemented by different suppliers or in-house development teams. It therefore has to be clear for each such supplier or team exactly how big portion of the total end-to-end delay is available for the component that they implement.

A similar situation occurs when a control algorithm needs to impose a maximum age on its sensor input data. In that case, an event chain is defined from sensor to the input of the control function. This type of time budgeting and the one mentioned previously are handled in a similar way methodologically, but requires a different use of TADL notation due to the need of a slightly different semantics (reaction vs. age). The following description will not make a clear distinction between these two types of time budgeting.

Following from the above, time budgeting is about dividing an overall end-to-end delay into smaller segments, in order to specify how big portion a component (or subcomponent) in the path between stimulus and response may take.

Overview

An end-to-end delay generally originates from either an explicit or implicit user requirement or expectation, or from control performance needs. Other sources of end-to-end delays are legislation, standards or legacy. The methodology described here focuses on how to distribute such an end-to-end latency over the components and subcomponents in the end-to-end event chain.

At the same time with this top-down segmentation of the end-to-end delay, another part of the development project starts with defining hardware, software platforms and other low level details. Legacy functions are also already being introduced. All this means that there is already early in the development process detailed information about the final solution that could be useful when assigning time budgets. Thus, it is beneficial to also introduce a bottom-up flow of timing information for the purpose of time budgeting. This will reduce the number of design iterations. A major issue is how to handle this mix of

bottom-up and top-down information. Figure 15 illustrates the main idea of time budgeting.

For example, on vehicle level, a requirement may postulate that “The doors shall be unlocked not later than 1 second after a valid transponder key has been recognized”. This requirement specifies the end-to-end delay that is to be segmented over the end-to-end event chain on the various abstraction levels.

Since the operational level is the lowest abstraction level, time budgeting is not performed at this level. It only serves to feed the bottom-up flow with measured execution data, and to verify that no task execution times in the final implementation exceed the time budgets specified on implementation level.

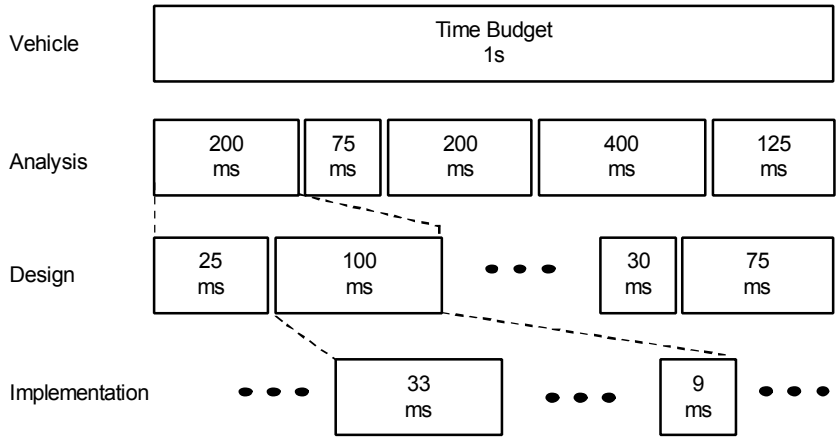

Figure 15 - The principles of time budgeting

The time budgeting process contains, in addition to the above, clear elements of negotiation between OEMs and suppliers, as well as exchanging timing models between the different parties. For this reason, it is heavily encouraged to combine the time budgeting process with the processes proposed in the use cases Negotiate time budgets and Exchange models.

Definitions

Before describing the time budgeting methodology, we need to clarify the concepts of time budget influencing property, slack and margin.

A *time budget influencing property* is a property that has the potential to influence the response time of a certain end-to-end event chain, and thereby also the required time budget. The following properties with this potential have been identified:

- Worst-case execution time (WCET)
- Response time
- Communication delay
- Interference time
- Task period

Slack is a portion of an end-to-end delay that is not allocated to any budget segment. Thus, there is only one slack per end-to-end delay. Slack is generally not communicated to suppliers, but rather serves as a reserve for interference from other not yet implemented functionality.

Margin is a part of a budget segment that is excess to the response time of the corresponding component. There is thus at most one margin per segment. Since margin is part of a budget segment, it is (at least implicitly) communicated to suppliers.

Mapping to generic methodology

Figure 16 presents the time budgeting process, and how it maps to the GMP presented in Section 3. The *Create timing model* and the *Analyze timing model* tasks have been split into several subtasks each in order to illustrate the activities to be performed in these tasks in more detail. Moreover, the other GMP tasks have been renamed to better reflect their purposes in the context of this use case. The following paragraphs will describe the figure in more detail.

Refine, Introduce and Validate time budgets

This task translates the time budgets from the immediately higher abstraction level into a textual equivalent on the current abstraction level with an imagined or anticipated solution (or at least main characteristics thereof) in mind. Such textual budget requirement are usually of the form “The Controller shall not take longer time than 50ms from receiving an input to producing an output”. The Controller is in this case a known entity from the higher abstraction level, and it is assumed that it will be refined into a distinct set of functions/components at this abstraction level.

Create solution

The solution is created as specified in the generic methodology. It should however be emphasized that this solution shall be created while taking the textual time budget requirements into account. This means, for instance, that if the time budget over a series of components is very tight, it may not be appropriate to allocate the components on different ECUs scattered across the vehicle, so that a large portion of the available budget is wasted on communication. Measures must be taken to maximize the probability that the solution meets the time budget requirements. In order to make sound decisions about the distribution of components based on time budgets also information about the amount of interference is needed. This information can, for instance, be derived bottom-up from existing parts of the solution.

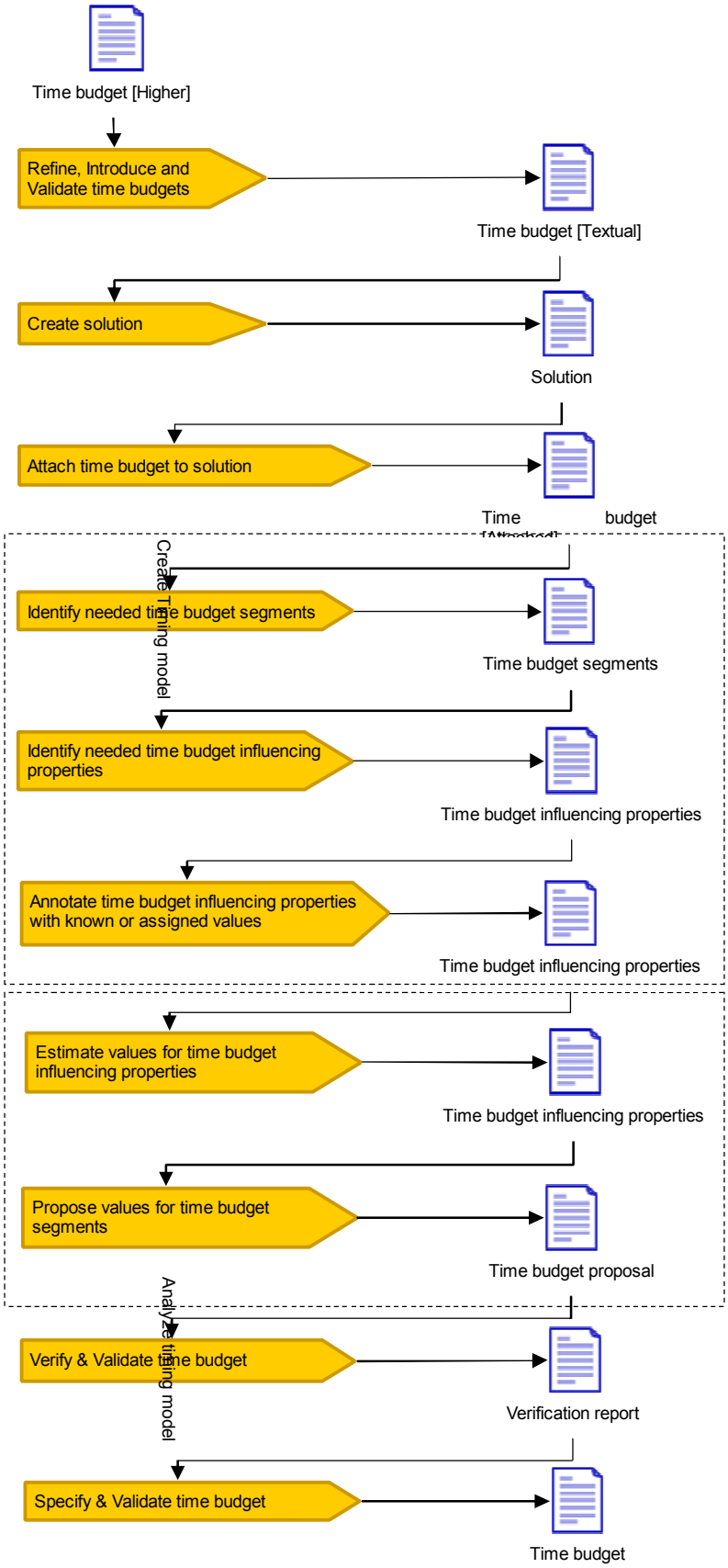

Figure 16 - The time budgeting methodology

Attach time budget to solution

In this task, the textual time budgets are formalized based on the solution. In other words, the textual requirements are translated into TADL2 elements, which are attached and linked to the solution. This task hence models (formalizes) the boundaries in which the subsequent time budgeting process must stay.

Create timing model

The GMP task Create timing model is divided into three tasks with respect to time budgeting:

- Identify needed time budget segments
- Identify needed time budget influencing properties
- Annotate time budget influencing properties with known or assigned values

The task *Identify needed time budget segments* refines the time budgets from the higher abstraction level with respect to the more detailed solution structure that has been developed at this abstraction level, and with respect to the distribution of components to suppliers if applicable. Note that the task only identifies which refinements (segments) would be necessary for passing on as budget requirements to the next abstraction level. It does not make any estimations on exact numbers, i.e. exactly how long budget should be assigned to a certain segment.

An extreme case would be to create a segment for each function/component that is in the scope of a certain time budget. However, in the case that several suppliers are involved in implementing two or more consecutive functions/components, such an approach might be too restrictive and impose unnecessary constraints. It is sufficient to provide one overall time budget for all consecutive components in the time budget event chain. If such an approach is not compliant with respect to another time budget requirement, the approach cannot be applied. Figure 17 illustrates this approach with an example.

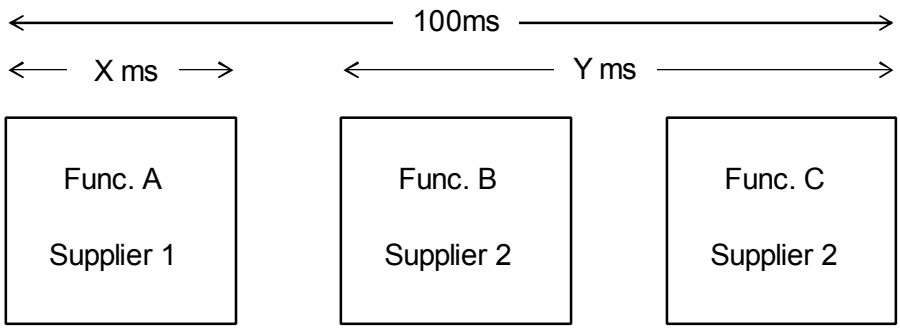

Figure 17 - Example of a budget segment identification strategy

The example in Figure 17 contains three functions. Function A is to be implemented by supplier 1, and functions B and C by supplier 2. The chain of these three functions has a time budget of 100ms. This overall

budget will by this task be divided into two segments, one for each supplier. The time bounds in these segments are still unknown, as indicated by the variables X and Y (see Symbolic time expressions).

The task *Identify needed time budget influencing properties* is the first step towards acquiring the necessary information for completing the identified time budget segments with concrete values. The task tries to answer the question of what background information is needed and relevant for assigning concrete values to a certain segment. However, the task does not fill these properties with concrete values.

In the task *Annotate time budget influencing properties with known or assigned values*, the time budget influencing properties identified previously are completed with time values when possible. This activity falls into one of three categories:

1. The information can be retrieved from a lower abstraction level
2. The property is already known from another context
3. The property is something that we impose on the system

The first category reuses information that has already been derived for the parts of the solution that has already been developed bottom-up at a lower abstraction level. The lower-level properties cannot directly be copied to the current abstraction level, since the solution structure looks different and has less detail. The events on the lower abstraction level therefore have to be mapped to events on the current abstraction level. Once this is done, the delay constraint itself can be copied and contain the same information as it did on the lower abstraction level, with the difference that it is associated with the current-level events.

The second category tries to find information from an external source, for example another project where the same function has been used.

Other timing properties of the system do not follow from higher abstraction levels or can be found by a particular analysis method, but they are rather design decisions. In the third category, the developer may assign timing properties, based on his experience, in such a way that he believes that the system will perform timingly. Previously known properties, belonging to categories 1 and 2, must be considered. The assigned time budget influencing properties will be verified in a separate task and possibly be revised in a later iteration.

Analyze timing model

The GMP task *Analyze timing model* is divided into two tasks with respect to time budgeting:

- Estimate values for time budget influencing properties
- Propose values for time budget segments

The task *Estimate values for time budget influencing properties* fills the remaining blanks (or rather unbound variables in symbolic expressions) with concrete values. There are at least two different strategies on how to do this:

1. Analyze the solution, its requirements and time budget influencing properties
2. Extrapolate values based on rapid prototyping/sneak-peak on lower abstraction levels.

The first strategy analyses the solution and its requirements for time budget properties that are a direct implication of the solution and the requirements at the current abstraction level. Typical techniques for obtaining such properties are formal analysis and simulation. At operational level, the task performs measurements on a physical running system, which a higher abstraction level may transform and apply to its models in the task *Annotate time budget influencing properties with known or assigned values* of that abstraction level.

The second strategy addresses a problem that occurs in particular at high abstraction levels. The information needed for finding the sought time budget properties is not present at that level, and it is not found among the transformed properties. It is thus not feasible to apply strategy 1. In such cases, it might be necessary to conduct rapid prototyping to quickly obtain a temporary extrapolation of the system models that will be developed in later development phases at lower abstraction levels. The analysis is then performed on these lower-level temporary models in the same way as suggested in strategy 1. The result is then transformed back to the model at the original abstraction level and the temporary models are discarded. Naturally, such an approach will not give 100% accurate results, but will still give a hint on which values are reasonable. In order to make this strategy feasible and efficient, it is important that all steps, including the extrapolation, are automatic.

The task *Propose values for time budget segments* fills in the values of the time budget segments identified previously. This has to be done considering the acquired time budget influencing properties, in particular the response times. In general, it is desirable to add a slight margin to the budget segment compared to the estimated and analyzed response times in order to provide for more relaxed implementation. However, it could even be the case that the resulting time budget for a certain component is smaller than a WCET property over the same component that was transformed from a lower abstraction level. In such cases, the lower-level solution needs to be reworked to comply with the (new) time budget.

When making a time budget, not only the current solution (regardless of abstraction level) needs to be considered, but also the influence of future functionality. Future functionality refers to both functionality that is planned but not yet implemented, and to still unknown functionality that potentially is to be included in future generations of the system. The task *Propose values for time budget segments* therefore also needs to compare the solution and its time budget properties with the product plan to identify which functionality is still to be added to the system, and also make an assessment of the influence of unknown functionality. Based on this information, the developer needs to assess how much the still missing functionality affects the end-to-end event chain currently under investigation. This will eventually lead to introducing slack in the final time budget. Typical properties that are affected are communication delay (increased congestion) and task execution periods (increased competition for computation power), which both lead to a longer end-to-end delay.

Verify & validate time budget

The task *Verify & validate time budget* compares the time budget proposal with the initial requirements. The main criterion to be checked is that the sum of the segments, including slack, does not exceed the end-to-end delay requirement.

Specify & validate time budget

The task *Specify & validate time budget* makes a final revision of the time budget proposal and documents this as a requirement for the next phase. It should be noted that slack is **not** part of the requirements that are handed over to the next phase/abstraction level, whereas margins are included as part of the budget segment and thus **is** part of the requirement.

Application of symbolic time expressions for time budgeting

In some tasks of the time budgeting process, most notably related to *Create timing model* and *Analyze timing model*, timing properties that are inherent in the solution and input requirements are obtained, whereas others appeal to a big extent to the subjective judgment and experience of the developer. Symbolic time expressions can be a powerful tool to capture the relation between properties and help navigating through the design space.

The concept will be illustrated on the example shown in Figure 18. The figure shows an end-to-end delay of 1 second, which shall be distributed over five components and communication links (A-E). The delays of components A, B and D are assumed to be either transformed, determined or extrapolated WCETs with values 200ms, 50ms and 100ms respectively. Each component has further been assigned a margin, μ_X, where X is the name of the component. Margins of 10ms and 20ms have been added to the WCETs of components A and D respectively, to create some additional space in the resulting budget segments. This was, in this example, not found necessary for the other components. These values cannot be further elaborated unless the solution or input requirements are changed. A slack, σ, has moreover been introduced. For the sake of the example, the slack is assigned 100ms.

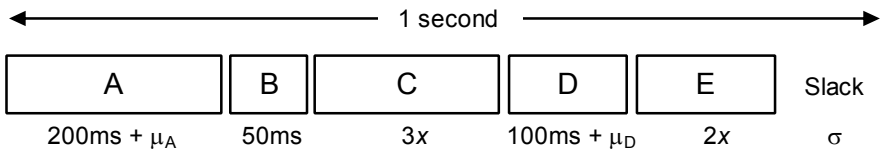

Figure 18 - Time budgeting example using symbolic time expressions

The only remaining unknowns are the WCETs of components C and E. These values are to be filled in based on the developer's experience. The main idea behind the approach suggested here is to evenly distribute the remainder of the end-to-end delay on the components with unknown delay based on a weight. The weight shall reflect the relative need for a long time budget. By inspecting the behavioral models and other descriptions of the components, the developer will get a feeling for how long time the component would need to perform its

task. In the example of Figure 18, component C is expected to need 50% longer execution time than component E. This leads to the following equations:

$200 + 10 + 50 + 3x + 100 + 20 + 2x + 100 = 1000$

$x = 104$

This gives a budget of 312ms for component C and 208ms for component E.

This approach can also be extended to include the slack and margins. As a second example, we could assign $0.1x$ and $0.2x$ as the margins of components A and D respectively, and x as slack. This leads to the following equations:

$200 + 0.1x + 50 + 3x + 100 + 0.2x + 2x + x = 1000$

$x = 103$

Thus, $\mu_A = 10.3$, $\mu_D = 20.6$, $\sigma = 103$, and the budgets of components C and E are assigned to 309ms and 206ms respectively.

The main advantage of using the symbolic time expression capability of TADL instead of a pure equation solver is that the developer's underlying thoughts and intentions are saved in the model, and thus can be elaborated by tools.

5.3 Specify synchronization timing constraints

Problem statement

A vehicle offers many different features to the driver, such as braking, steering etc. Today, these features are typically implemented using both mechanical and electronic components. The fact that the electronic system of the vehicle is integrated with different mechanical solutions implies that the vehicle electronic system inherently contains a certain degree of parallelism. That is, the system needs to monitor and control several simultaneous sources of input and output. Quite often it is also the case that the input or output needs to be synchronized in order to provide a notion of simultaneity. For example, when braking, it is crucial that the brake forces that are applied at each wheel also are applied at the same time. A correct behavior is governed by the introduction of synchronization constraints during the vehicle design. Thus, this use case deals with the formulation of synchronization constraints and how they are refined during the design.

This use case only addresses the problem of synchronization of events, regardless the order in which they occur.

Overview

According to the timing constraint logic of TADL, the synchronization constrain is specified as follows: given a set of events and given an occurrence of any event in this set, then all the other events of the set must occur at least once within a certain temporal window. Such a temporal window is called *tolerance*.

Since the synchronization constraint deals with time delay between event occurrences, at a first glance it may look similar to the use case “Specify Time Budgets”. However, the main difference relies on the concept of *generic sets of events* versus the concept of event chain used by the “Specify Time Budget” use case. An event chain is a *particular* set of *two* events composed by a stimulus and a response. A *generic* set of events is composed by *any number* of events, for which their nature of stimulus/response is not necessarily characterized. This implies that while the two events of the event chain are ordered with respect to time (the response occurs *after* the stimulus), a generic set of events can be composed by an arbitrary number of events for which is not defined an order relationship. Moreover, the events of a generic set of events may not have a characterization in terms of stimulus/response.

To give more insights about this constraint, we go through some examples. For instance, consider the adaptive cruise control system in a passenger car. An adaptive cruise control system takes the decision of the torque that must be applied to each wheel based on the information on the current vehicle speed and on the current distance to the vehicle in front. To compute the correct torque value, the controller requires the received information at least to represent the environment at the same time instant. This means that the events “pick the current vehicle speed” and “pick the distance to the front vehicle” must occur

at the same time. It is then required synchronization between these two. To compute the correct torque value, it is also required that the information picked by the sensors arrives to the controller within a certain time delay. Such an additional requirement can be accommodated by using the “Specify Time Budget” use case considering for example the events “pick the current vehicle speed” and “the information is available at the controller input” as an event chain (note that such events represent a stimulus and a response). Finally, to design a correct braking system, we also want the four wheels to stop at the same time when a braking action is performed. Hence, the set composed by the four events “stop wheel” related to each wheel must be also synchronized.

From this example several interesting properties arise that allows us to give a comprehensive understanding of the synchronization constraint in a more general framework. In the example, we need first a synchronization constraint over the set of the two events “pick the current vehicle speed” and “pick the distance to the front vehicle”: despite this set has dimension two, it does not represent an event chain being the events two stimuli. Second, we need a synchronization constraint over the set of the four events “stop wheel” associated to the four wheels of the car: such a set of events has dimension four, and all the events can be classified as responses. Finally, for both the aforementioned sets of events is not important the order in which the elements of each set occur. In summary, according to the TADL description of this use case, synchronizing the events corresponds to impose a tolerance on each set of event.

The full function of the adaptive cruise control at the presented level of abstraction is realized by combining two synchronization constraints together with an end-to-end delay constraint, that represent the maximum time allowed between the brake pedal pressure occurrence and the car stop occurrence. However, in both the defined generic sets of events no ordering is required, while in the end-to-end delay constraint an ordering is implicitly defined through the stimulus and the response of the event chain. We remark that is not difficult to think about functions that may require a certain order of several events occurrences in addition to some synchronization constraints. In such cases, the synchronization constraint must be used in combination with some ordering constraint.

According to the EAST-ADL development phases, such generic sets of events may become richer, or new generic sets of events that must be synchronized may appear. For instance, consider the braking system of a passenger car as example, and consider the following feature provided at the Vehicle Level stated as: ”stop the car within 500ms after a brake pedal pressure occurrence has been detected”. In this statement it is easy to locate two events that are one stimulus and one response (“brake pedal pressure occurrence” and “stop the car”), and an end-to-end time delay constraint (“500 ms”) that can be accommodated with the “Specify Time Budget Constraint” use case.

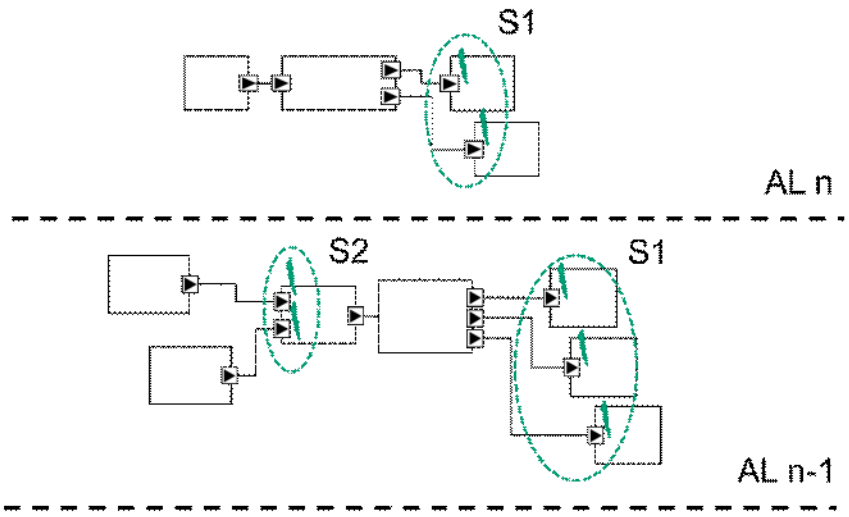

Figure 19 - Refinement and identification of new sets of events. At the abstraction level *n* the set S1 is composed by two events. After a refinement of the solution at the abstraction level *n-1*, the set S1 is composed by three events, and a new set of events (S2) that must be synchronized is identified.

use-case at the Analysis level. Such a feature can be stated as:"*Stop simultaneously the four wheels within 500 ms after a brake pedal pressure occurrence has been detected*". The difference of the statements at the Vehicle Level and at the Analysis Level relies on the number of events that must be taken into account. In the first case, as the events are defined, it is possible to locate a stimulus and a response that occur in series, while in the second statement the events "The wheel X has stopped", where X is front-left, front-right, rear-left and rear-right, are four responses that we want to occur at the same time, disregarding the order in which they occur. Hence, it is possible to specify a synchronization constraint over such a set, by setting the tolerance equal to zero as specified by the requirement "simultaneously". The example shows how moving from the Vehicle Level to the Analysis level a new set of events that require synchronization has been detected. Nevertheless, there can also be cases in which a set of events for which a synchronization constraint has been defined, may increase its dimension when moving along the abstraction levels. In this case it is enough to include the new events in the generic set and apply the tolerance on the new set.

Mapping to generic methodology pattern

Figure 2 presents the generic process for formulating and decomposing synchronization constraints including the mapping to the generic methodology. We remark that going through the levels of abstraction, it is still possible to formulate new synchronization constraints also at the lower levels if needed by the selected solution. Moreover, existing synchronization constraints can also be refined. The tasks will be described in more detail in the following paragraphs.

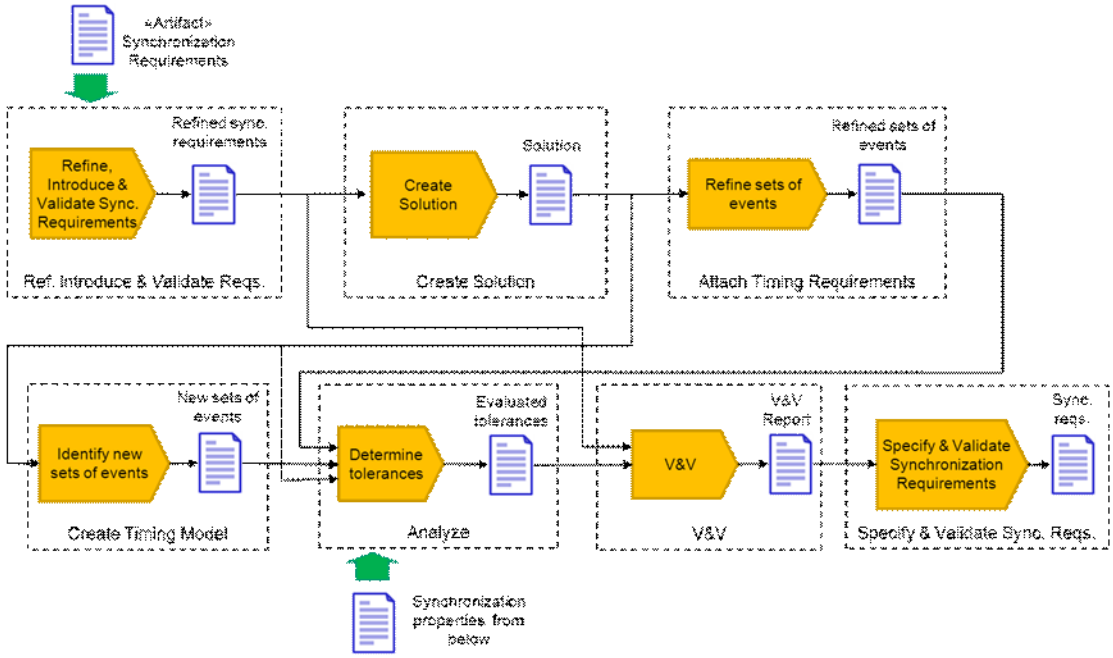

Figure 20 - Mapping of the specify synchronization constraints into the Generic Methodology Pattern

Refine, introduce and validate synchronization requirements

This task transforms the timing requirements imported from the higher abstraction level to be understood at the current abstraction level. Once a set of timing requirements is obtained, verified and validated at an upper abstraction level, this task refines such time requirements to be attached to the current abstraction level solution. Moreover, this task should perform a validation activity to make sure that the formulated requirements indeed express the intended meaning.

In the case of synchronization, this task refines the synchronization requirements imported from the upper abstraction level so that they can be attached at the current abstraction level solution.

Create solution

As for any function we wish to implement, the Create solution task generates a solution that satisfies all the non-timing related requirements, such as functional requirements, safety requirements, etc. A solution is picked in the space of all possible solutions that implement the desired function.

Once such a solution has been designed, its time properties must be derived and time requirements must be attached.

Refine sets of events

This task maps the refined synchronization requirements to the current solution. For instance, it refines the existing timing model by adding or

subtracting events to the existing generic sets of events for which a synchronization constraint is required. The refinement of such sets of events depends on the particular solution created by the task Create solution. The required tolerance imported from the higher abstraction layer and initially attached to a given set of events, is now attached to the correspondent refined set of events.

Identify new sets of events

When a solution is created it can happen that new events are defined and they need synchronization to correctly implement a required function. Given the current solution, this task identifies if there are new sets of events that must be synchronized. Furthermore, this task imports the timing properties offered by the lower abstraction level, merging them together with the timing requirements in a unique artifact.

Determine tolerances

Given the timing model obtained from the current solution, this task determines its timing properties. For instance, it quantifies the tolerances of both the existing and refined sets of events and the new sets of events devised by the current solution. Whenever possible, such quantification is obtained both by analysis and by simulations. At the end of this task, the timing properties of the devised solution are qualified, quantified, and are merged together with the timing properties offered by the lower abstraction level and with the timing requirements imported from the higher abstraction level. Hence, the timing properties of the current solution, the timing properties exported from the lower abstraction level and the timing requirements coming from the upper level are ready to be verified.

Verify and validate

This task verifies if the time properties extrapolated from the current solutions together with the time properties exported from the lower abstraction level meets the requirement provided by the task Refine, introduce and validate timing requirements. Notice that the comparison between the current solution time properties and the timing requirements derived from the upper level follows a top-down approach, while the comparison between the current solution properties and the exported timing properties from the lower abstraction level follows a bottom-up approach.

Specify and validate synchronization requirements

This final step involves adding the modeling constructs necessary to indicate that the formulated constraints indeed are to be considered as requirements for the lower abstraction level. The value of the tolerances devised during the task Determine tolerances of the new sets of events identified by the task Identify new sets of events become tolerance requirement of these new sets of events.

5.4 Revise erroneous timing information

Problem statement

When developing time-critical systems, it is almost inevitable that some unintended behavior passes through the development process. It is then of great importance that the verification techniques can catch the unintended behavior so that the designer may have a good chance to revise it.

This use case provides a generic pattern on how a violation of a timing requirement is detected and corrected. The pattern should then be combined with another use case, related to the type of requirement violation, describing a straight-forward error-free development process that provides the detailed hints on how to accurately develop a system with respect to the problem addressed by that use case (and the requirement violation in question).

The generic pattern is moreover exemplified with a set of decision trees related to the time budgeting problem. The example demonstrates, for each abstraction level, how to decompose a specific type of timing requirement violation and how to proceed to identify the cause of the violation, and also what measure could be taken to revise the system so that the requirement finally holds.

Generic pattern

As opposed to most other use cases, the starting point of this use case is not to start constructing or refining a system. The starting point is rather when the system has already been constructed, and we want to find out if its timing behavior conforms to its requirements. The use case is activated when verification has revealed a requirement violation, which needs to be corrected. The methodology of this use case thus starts with the GMP task Verify timing requirements and continues with an iteration (which according to the assumption of the GMP is implicit) where the solution is modified to comply with the requirement.

In some cases, the requirement violation cannot be resolved at the same abstraction level as where the violation was found. It might be needed to acquire more detailed design information from a lower abstraction level, or to give feedback to a higher abstraction level that there might be a risk of a violated requirement.

Figure 21 illustrates the process.

Identify timing requirement violation

This task is in principle equivalent with the GMP task Verify solution against timing requirements. The task compares the solution together with the timing model and analysis results with the attached timing requirements.

The main difference is that this task expects that there is a mismatch between solution and timing model on the one hand, and the timing requirements on the other – i.e the timing requirements are violated.

The output of the task is a *Timing requirement violation*. This artifact identifies the violated timing requirement and any diagnostic information that could help in locating the cause of the violation.

Find cause for timing requirement violation

Once a timing requirement violation has been identified, the task *Find cause for timing requirement violation* investigates potential diagnostic information trying to find out one or several possible causes of the problem. The task of finding causes is to a large extent dependent on the experience of the developer.

For common types of requirement violations, an organization could build a set of decision trees, which elaborates on possible causes and corrective actions for those types of requirement violations. In section 5.4.1, such an approach is illustrated on an example of the requirement violation exceeded time budget.

Resolve violation at the current abstraction level

This task executes corrective actions that are derived based on the identified cause, in order to resolve the requirement violation. Usually, these actions refer to modifying information that was produced by a task in the normal development process. It is therefore advisable to find and consult that task (again). The charts in the example of section 5.4.1 also provide a hint on corrective actions related to a certain cause of a certain requirement violation.

After having performed the corrective action, it is also important to check what other parts of the system that may have been affected.

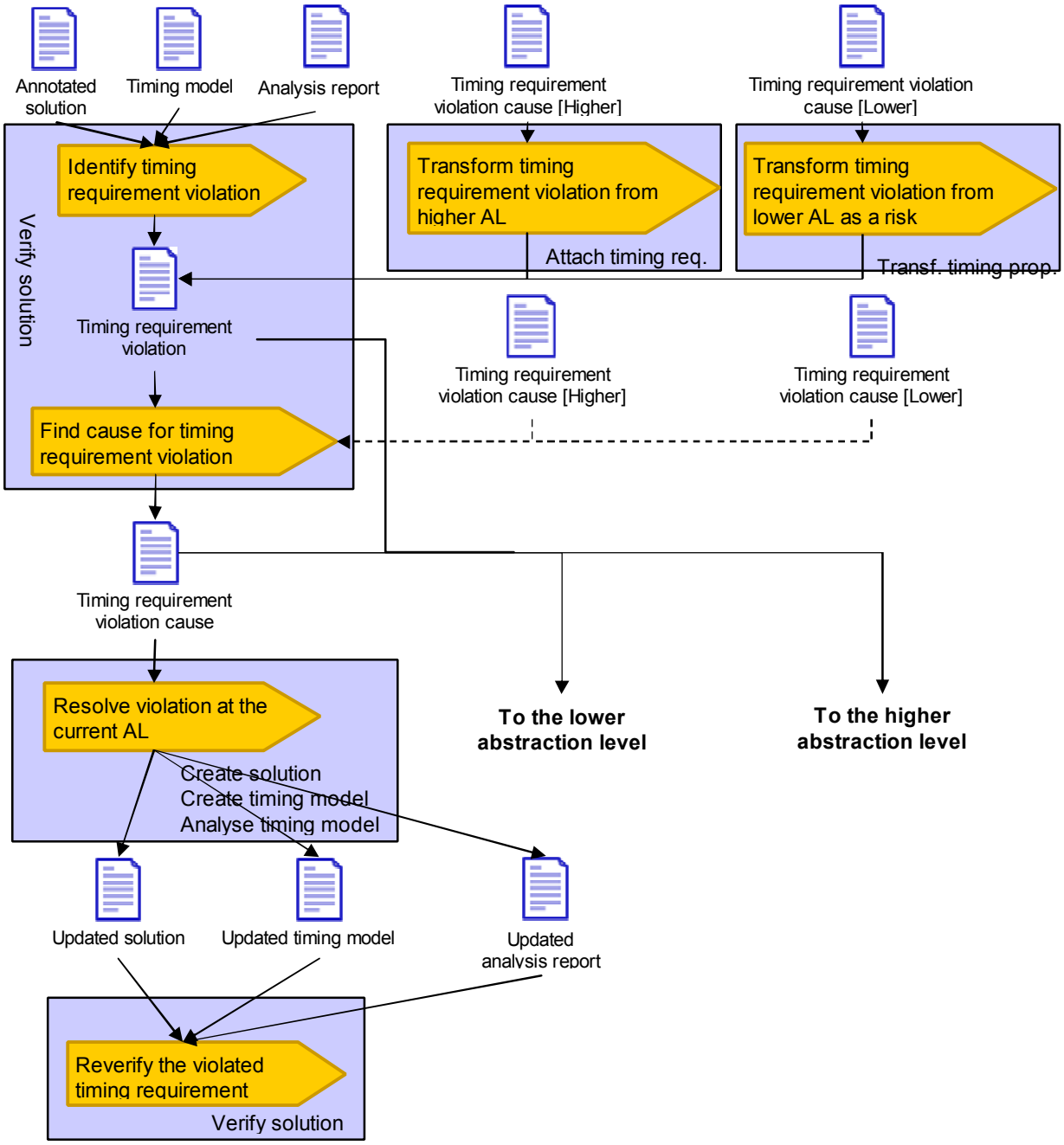

Figure 21 - Process for the use case Revise erroneous timing information

Note that changes may influence all lower abstraction levels, and many components and functions on each of those abstraction levels.

Reverify the violated timing requirement

After revising the system, it is necessary to re-verify it with respect to the previously violated requirement. If the requirement is still not satisfied, the whole process needs to be done again. Otherwise, the process ends successfully.

Transform timing requirement violation from higher abstraction level

If it was not possible to resolve the requirement violation at a certain abstraction level, it is advisable to examine other neighbouring abstraction levels. In case there is ground for believing that there is a need for obtaining more detailed and refined information, it is necessary to involve the lower abstraction level.

This task identifies the requirement at the lower level that corresponds to the violated requirement at the original abstraction level.

After having retrieved the more detailed information at the lower level, it needs to be transferred back to the original abstraction level as a property in the timing model of that abstraction level.

Transform timing requirement violation from lower abstraction level as a risk

If it was not possible to resolve the requirement violation at a certain abstraction level, it is advisable to examine other neighbouring abstraction levels. In case there is ground for believing that there is a need for modifying or relaxing requirements, it is necessary to involve the higher abstraction level. Since the requirement in question was already verified at the higher abstraction level before being given to the current level, it is not appropriate to say that it has been violated. Instead, we say that there is a risk that the requirement will be violated and that it needs further investigation.

This task identifies the requirement at the higher level that corresponds to the violated requirement at the original abstraction level.

After having modified the higher-level requirement, it needs to be transferred back to the original abstraction level as an updated requirement.

5.4.1 Example: Exceeded time budget

This section provides an overview on how the search for causes and corrective actions could be organized for the requirement violation *Exceeded time budget*. It should be pointed out that the charts are in no way exhaustive, and should only be regarded as inspiration.

The example is organized in several charts, one chart per abstraction level or AUTOSAR view. The top-level node captures the requirement violation. Below the top node there is a hierarchy of possible causes for the requirement violation. The leaf nodes capture either a corrective action or a redirection to another abstraction level (i.e. another chart).

The charts have the form of a decision tree, which in the end concludes one distinct corrective action. It is however important to understand that each design and requirement violation is unique and that it therefore is difficult, not to say impossible, to provide an exhaustive recipe on how to revise the system. The decision trees

presented here are hence incomplete in the sense that there may exist more causes and corrective actions than those listed in the charts. In addition, it is necessary to get an understanding of all branches in the cause hierarchy, or in the worst case even leaf causes (causes at the lowest level), before deciding on which actions to take. It may even be the case that several corrective actions need to be taken to extinguish a requirement violation.

Figure 22 provides a legend for the different nodes in the charts.

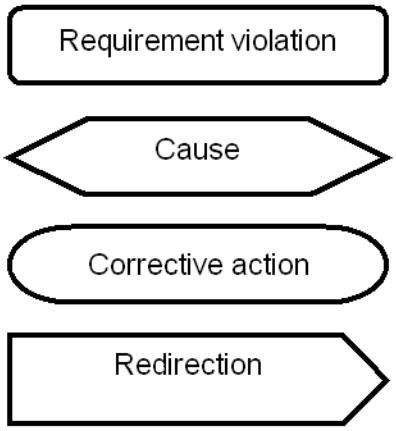

Figure 22 - Legend for revision charts

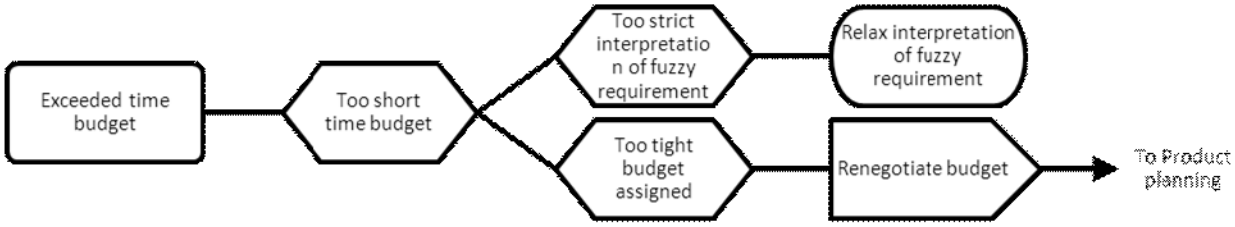

Figure 23 - Revise exceeded time budget at vehicle level

Figure 24 - Revise exceeded time budget at analysis level

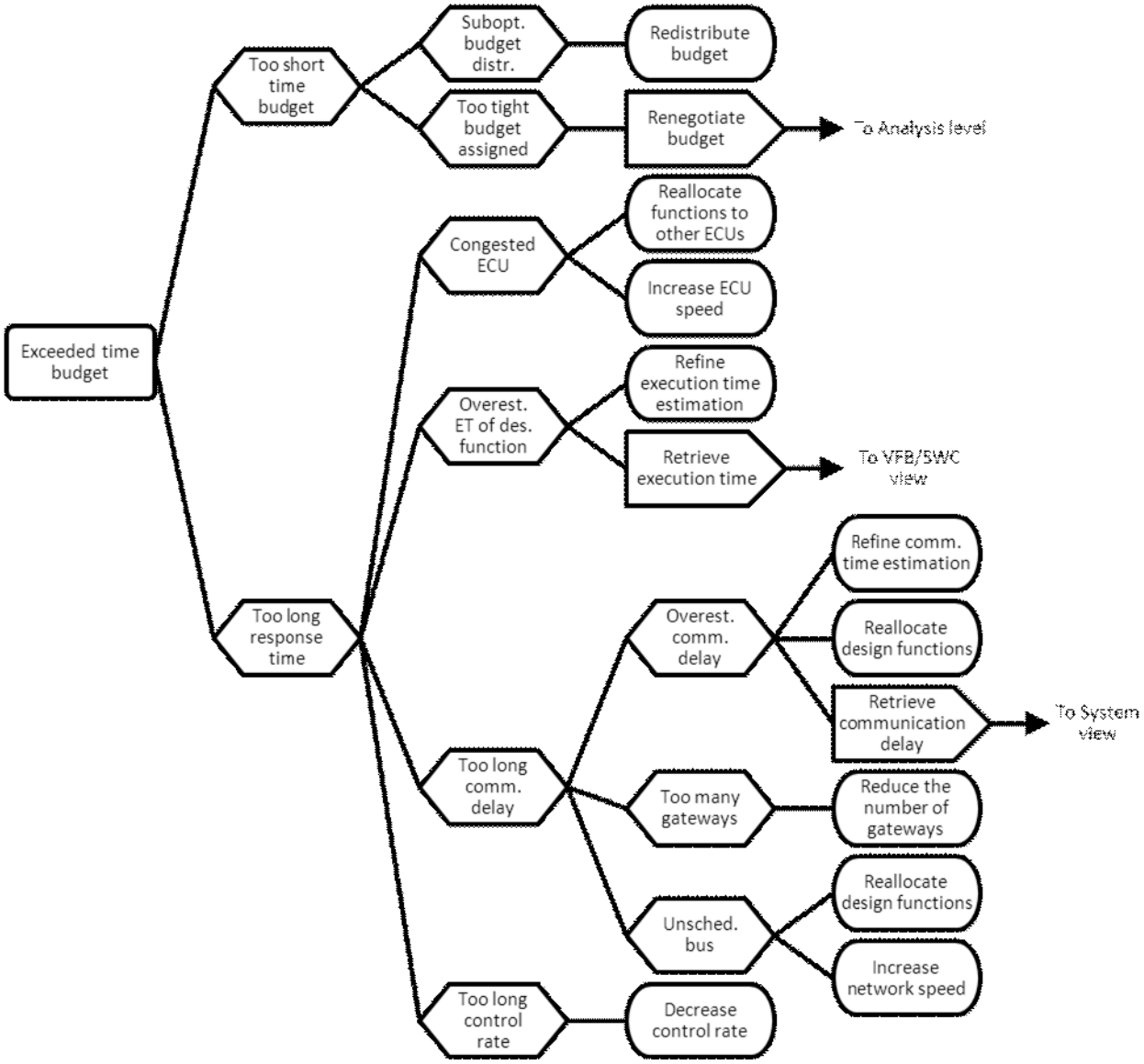

Figure 25 - Revise exceeded time budget at design level

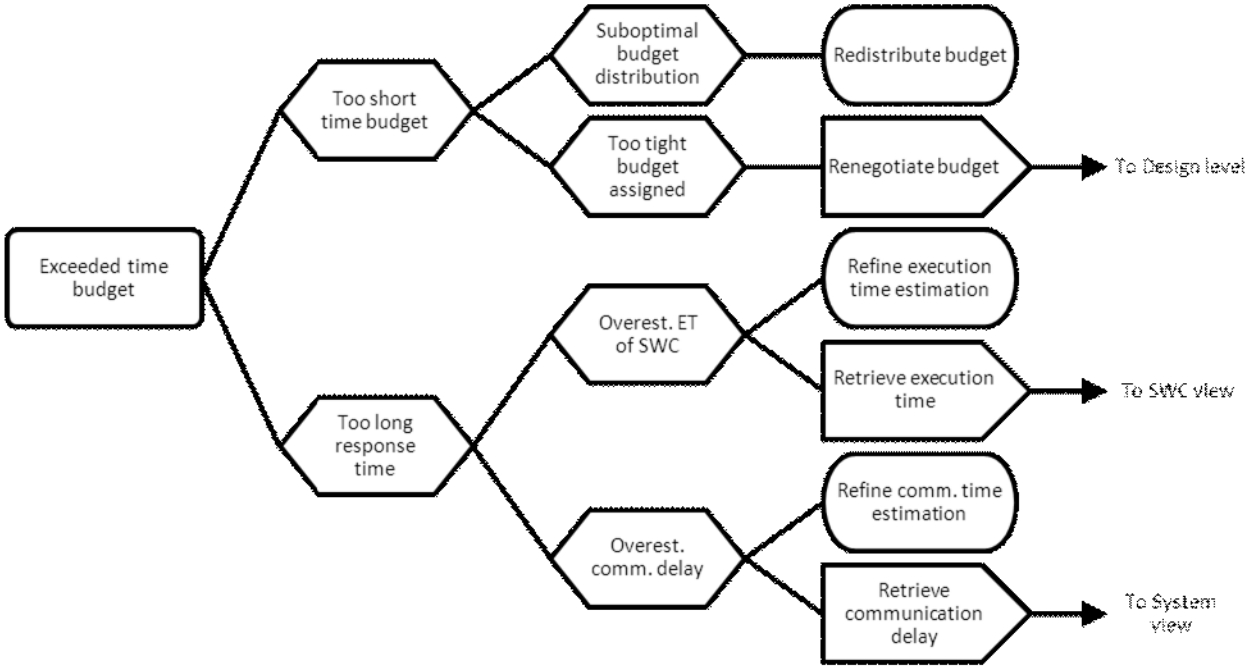

Figure 26 - Revise exceeded time budget in the VFB view

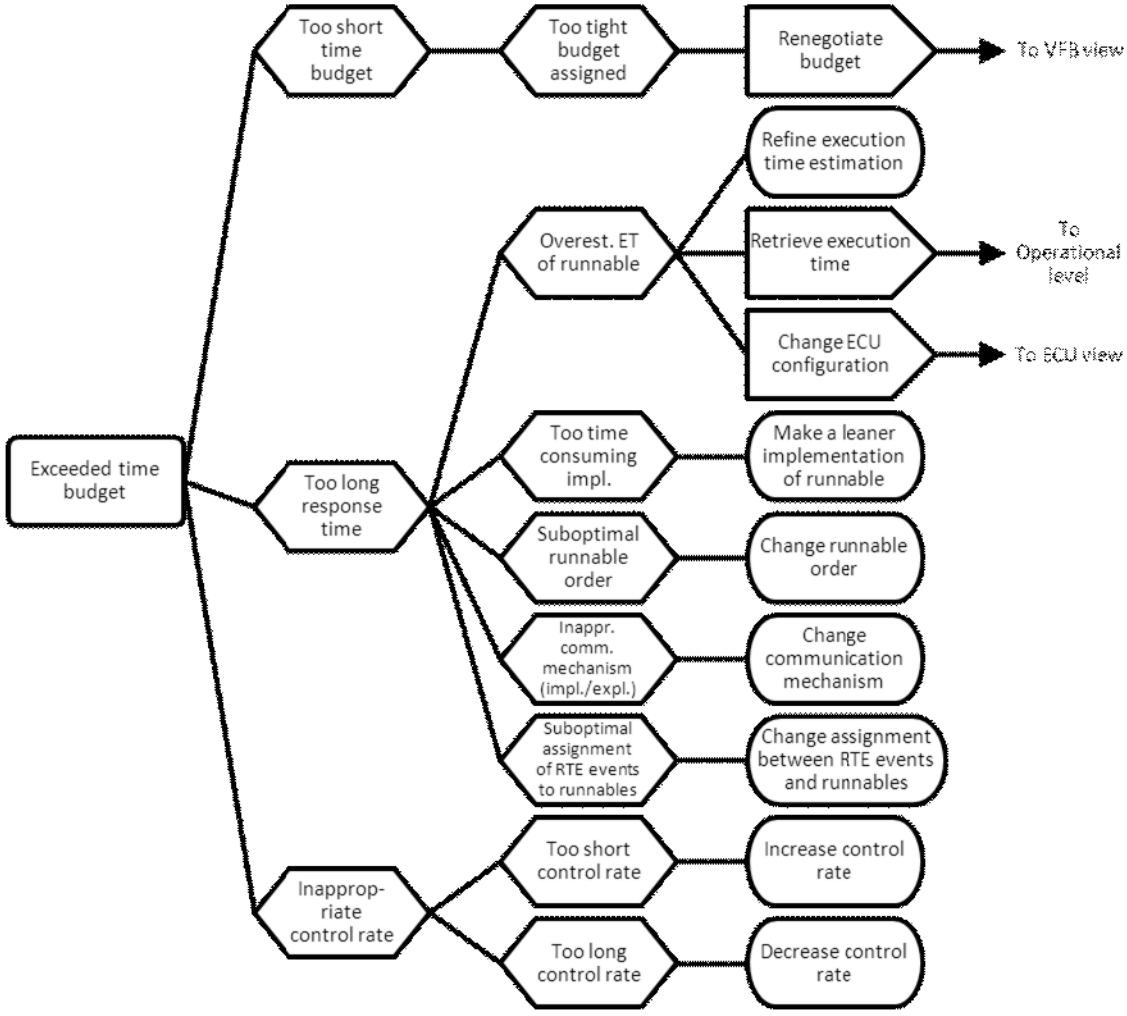

Figure 27 - Revise exceeded time budget in the SWC view

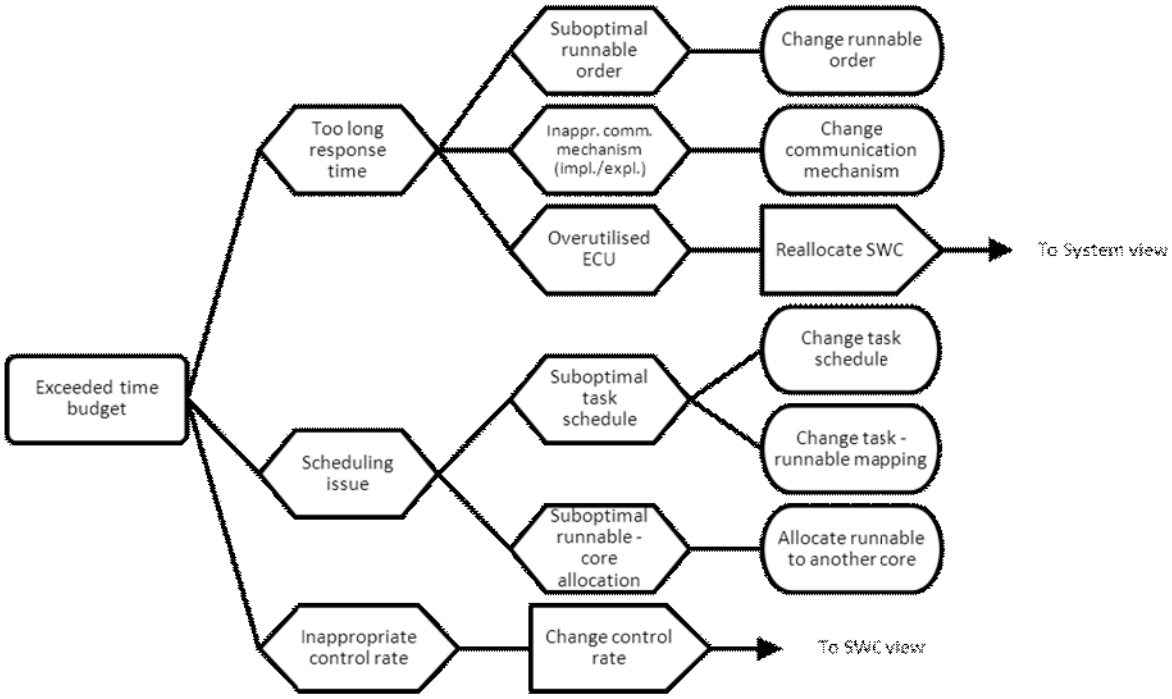

Figure 28 - Revise exceeded time budget in ECU view

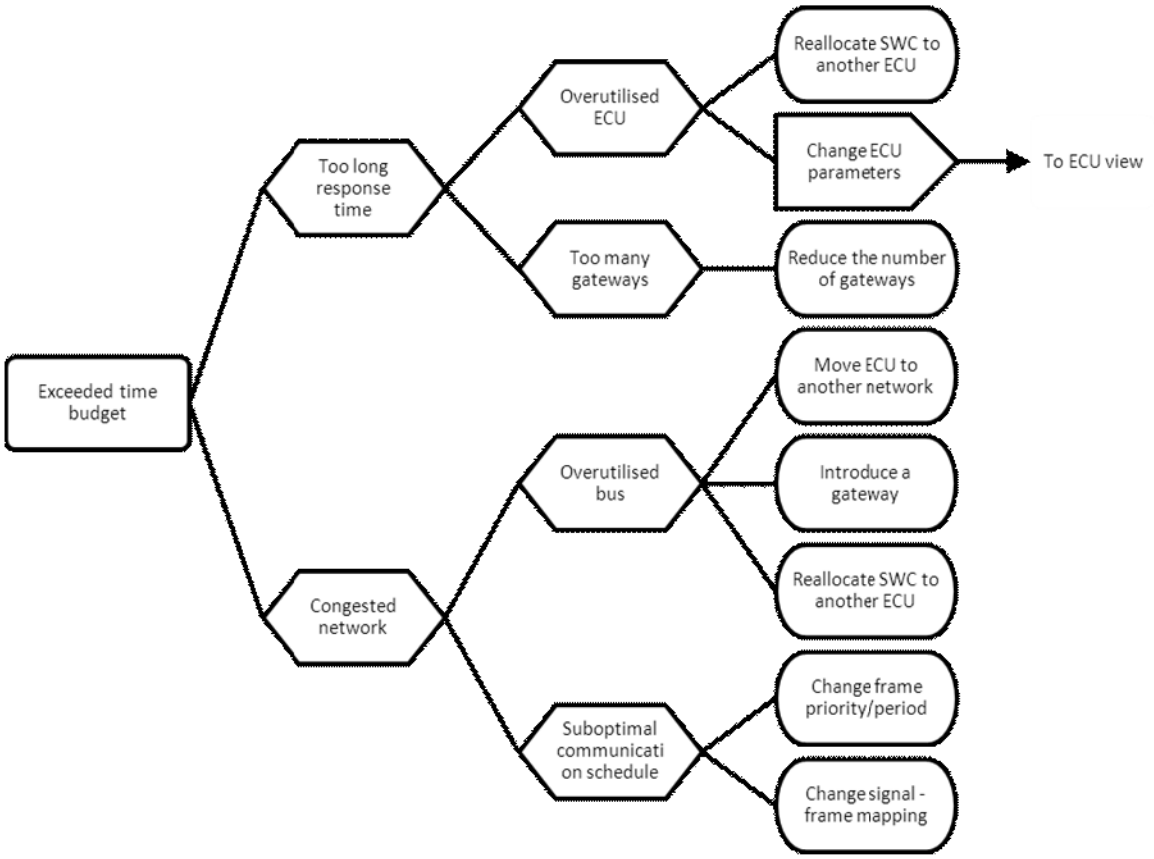

Figure 29 - Revise exceeded time budget in System view

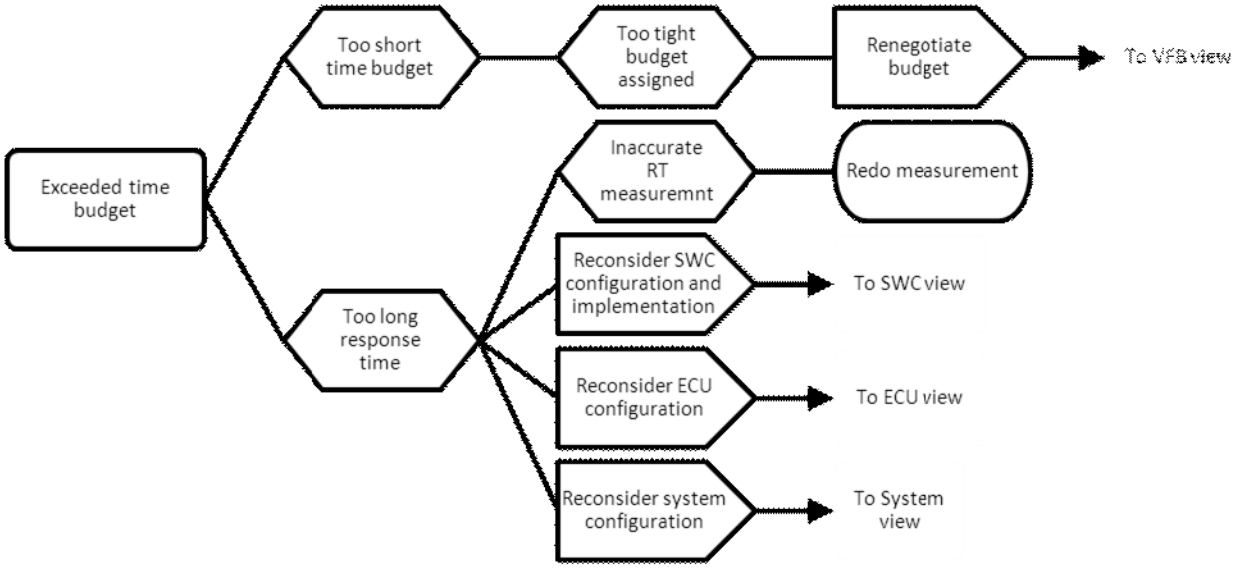

Figure 30 - Revise exceeded time budget in Operational level

5.5 Exchange models

Problem statement

The scope of this use case is a single ECU that is under the responsibility of the OEM.

Modern automotive systems are developed in the context of a complex supply chain. The OEM contracts the development of many components to so-called Tier-1 suppliers, which in turn entrust the development of some sub-components to their contractors.

Obviously, following such design style none of the involved parties is in possession of all system details that are necessary to reason about the global timing behavior. Unfortunately, there is no simple loophole to this situation, since the individual parties aim to protect their intellectual property and hesitate to share implementation details. This results in the fact that most timing related integration problems can only be discovered at integration time, i.e. when the implementation is done, which of cause leads to lengthy and costly development iterations.

The aim of this use case is to define a process between an OEM and his suppliers that allows to reason about the system timing behavior already at early design stages while protecting the individual intellectual properties. The process is based on the exchange of so-called timing models and is split into two parts:

1. The supplier side describing the supplier's development process and describing how the timing model for the developed functionality can be derived.
2. The OEM side describing the OEM's development process and describing how the timing models delivered by the suppliers can be used to validate the overall system timing behavior.

Please note that the use case focuses on the development of control applications which are the source of a large part of timing constraints in automotive systems.

5.5.1 Supplier Side

Figure 31 shows the supplier side of the use case. Its general structure follows the GMP that is described in Section 3.

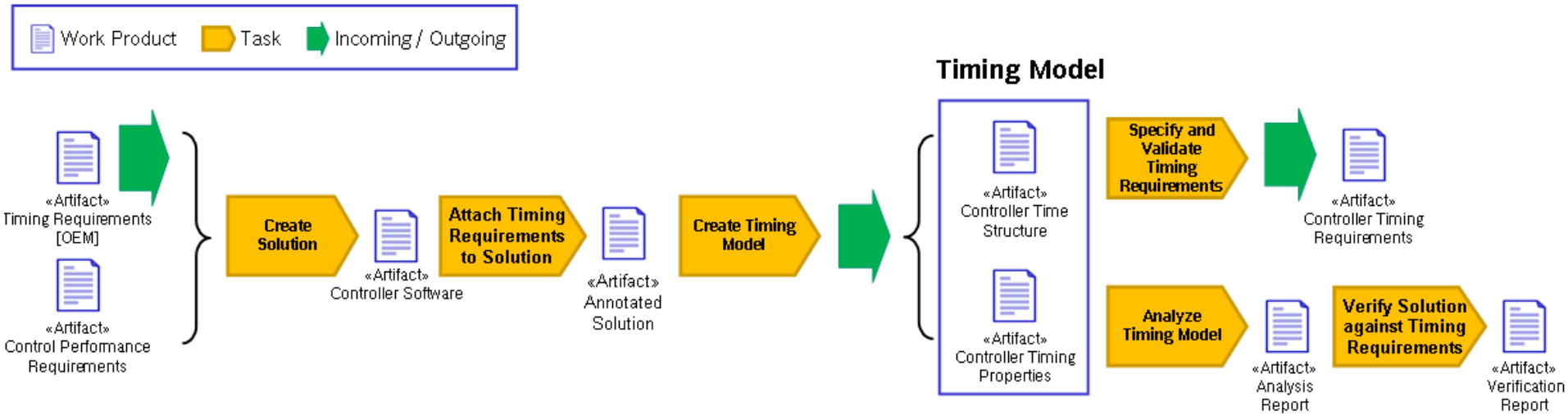

Figure 31 - Supplier Side of the Exchange Models Use Case

The task of the supplier is to create a solution taking into account timing requirements of the OEM and additional functional control performance requirements. After attaching the OEM's timing requirements to the created solution, the timing model can be extracted and communicated to the OEM for the purpose of timing validation of the overall system. The timing model contains the following artifacts:

1. *Controller timing structure*: executable entities (e.g. tasks and runnables) and their execution patterns.
2. *Controller timing properties*: execution times of the executable entities, etc.

Additionally, the supplier can derive *Controller timing requirements* from the chosen solution that are necessary to ensure the control performance of the functionality (e.g. minimum sampling periods, maximum delay, etc.). These requirements complement the OEMs timing requirements and must be fulfilled in the final solution, i.e. they must be validated by the OEM after system integration.

Obviously, before delivering the solution to the OEM, the supplier must check whether or not all timing requirements imposed by the OEM are met. For this purpose analyzes the timing model is analyzed and the solution's timing properties are compared against the OEM's timing requirements.

The different involved tasks are described in the following sections.

Create Solution

This task consists of the actual functional control design. This complex engineering task is not in the scope of TIMMO-2-USE and will, therefore, not be described in detail here.

However, during the functional design the control engineer has to take into account functional control performance requirements which are usually motivated by the desired user experience, safety consideration, and system mechanics. The engineer's primary goal is to find a control approach that satisfies these control performance requirements.

Additionally, the control engineer needs to take into account timing requirements that are imposed by the OEM, and that are motivated by global system timing considerations that are out of the supplier's scope (system must not be overloaded, reservation of slack for future functionalities, etc.). Such timing requirements are usually communicated using *timing budgets* (see Section 5.2) representing, for instance, execution time budgets that might not be exceeded. Satisfying the imposed timing requirements is usually no simple task for the control engineer. For instance, it might be necessary to choose a less sophisticated control approach to fit the algorithms in the assigned time budget, or code optimization might be required, which is a tedious task that usually reduces the reusability of the code.

The outcome of this task is the source code for the required functionality. Please note that the actual source code is usually not delivered to the OEM and remains intellectual property.

Attach Timing Requirements to Solution

This task corresponds to the original task of the GMP. The supplier annotates the timing requirements coming from the OEM to the created solution using its structural model as reference.

Create Timing Model

During this task a timing model for the controller implementation is created. Please note that the timing model represents an abstraction of the concrete implementation that allows to reason about its timing behavior. The outcome of this task contains the following artifacts:

1. Controller time structure: runtime structure of the implemented control software in terms of operating system entities on the target platform (for instance tasks and runnables in OSEK based systems) including activation patterns in terms of arrival curves. This information can be expressed using the underlying ADL for the structural model, namely EAST-ADL or AUTOSAR, and TADL2.
2. Controller timing properties: estimated / measured / analyzed execution times of the runtime entities for the target hardware platform.

Specify and Validate (Controller) Timing Requirements

During this task the control engineer derives timing requirements from the required control performance requirements. These consist usually in maximum response times, jitter constraints, minimum sampling rates, and maximum delay constraints (i.e. end-to-end response times). These *Controller Timing Requirements* are communicated along with the timing model to the OEM for ECU integration purposes. More precisely, the OEM must ensure/validate the adherence to these timing requirements to ensure the correct functional integration of the delivered control application.

Analyze Timing Model

This task consists in analyzing the timing model of the developed control application in order to derive timing properties that allow validating whether or not the timing requirements imposed by the OEM are satisfied. Possible timing requirements that can be validated during this task include maximum execution time budgets and maximum execution rates (i.e. periods).

Verify Solution against Timing Requirements

This task corresponds to the original task of the GMP. The timing properties of the developed control application are compared with timing requirements imposed by the OEM. If all timing requirements are satisfied, the solution (along with the timing model) can be delivered to the OEM.

5.5.2 OEM / Integrator Side

Figure 32 shows the OEM side of this use case. Its general structure follows the GMP that is described in Section 3. The integrator gets the Timing Models of the delivered subsystems that are necessary to reason about the integrated system's timing behavior from his suppliers.

Note that reasoning about the system's timing behavior works differently in early phases (i.e. vehicle and analysis phase). There, the OEM estimates or guesses timing properties of planned subsystems to compare and choose between different system approaches. However, in this use case we focus on the design and implementation levels. At that levels the system approach has already been chosen and the OEM steers the system development into the desired direction by imposing timing requirements, like time budgets (see Section 5.2), to his suppliers. However, he is (in his idealized role as integrator) not actively specifying and developing system parts on his own that he has to characterize with respect to their timing characteristics.

Using the timing models of all delivered sub-systems, the OEM can create an integrated timing model of the whole system. Based on this timing model she can validate all timing requirements motivated by the individual functional implementations, and thus check if system integration is successful in terms of real-time behavior. Additionally, the OEM can determine timing quality metrics for the integrated system, like, for instance, slack for future functionality, robustness to (slight) changes, etc. In case timing errors are detected or desired timing quality metrics are not satisfactory, the OEM can adjust the timing requirements for (a subset of) his suppliers, for instance by assigning smaller time budgets and go into another design iteration.

The different involved tasks are described in the following sections.

Create Solution

This task consists in integrating the different artifacts delivered by the suppliers. This includes integrating the functional entities to get an executable system (on the considered level of abstracting), and integrating the structural component models like AUTOSAR or EAST-ADL.

Attach Timing Requirements to Solution

This task corresponds to the original task of the GMP. The OEM annotates the controller timing requirements coming from the suppliers to the created solution using its structural model as reference.

Create Timing Model

This task consists in creating a global timing model of the integrated system by combining the timing models of all delivered sub-systems.

Analyze Timing Model

This task consists in performing different kinds of timing analyses on the integrated system timing model with the aim to validate all timing requirements. This consists of measuring, simulating or analyzing response times, end-to-end latencies with reaction and age semantics, activation jitters, response-time jitters, blocking times, etc. Depending

on the actual system, the whole range of available timing algorithms and tools might be useful for this task.

Verify Solution against Timing Requirements

The purpose of the task exactly corresponds to its description in the GMP (see Section 3), meaning that the actual timing properties determined during the task "Analyze Timing Model" are compared to the functionally motivated timing requirements coming from the suppliers. If all timing requirements are satisfied, system integration is successful in terms of real-time behavior, and the design process can proceed to its subsequent steps. In the reverse case, the system integrator needs to analyze the cause for the problem and trigger another design iteration for (parts of) the system.

Check Timing Quality [optional]

In case that all functionally motivated timing requirements, i.e. the timing requirements communicated by the suppliers of the sub-systems, are satisfied, the OEM might be interested in further evaluating and optimizing the system's real-time behaviour.

In particular, the OEM might be interested to ensure the extensibility of his system for future functionalities and so-called face-lifts. For this purpose it is crucial to reserve slack in the system, since hardware platforms usually remain static over several years (7-8 years).

One possible metrics for evaluating the system's extensibility is the load on the network and the ECUs. More sophisticated metrics are based on sensitivity analysis [2]techniques.

Specify and Validate Timing Requirements

During this task the OEM formulates timing requirements for the suppliers. These timing requirements are motivated by considerations of the overall system's timing behavior rather than by functional requirements (compare to task "Check Timing Quality"). Thereby, the goal of the OEM is to guide the system development in a desired direction (compare to use case "Specify Time Budgets" in Section 5.2). For instance, in case of timing problems the OEM can increase the timing budget of the affected functionality and thus grant a more relaxed timing constraint for the responsible supplier allowing him to create a better solution.

---

[2] Arne Hamann, Razvan Racu, Rolf Ernst: Multi-dimensional Robustness Optimization in Heterogeneous Distributed Embedded Systems. IEEE Real-Time and Embedded Technology and Applications Symposium 2007.

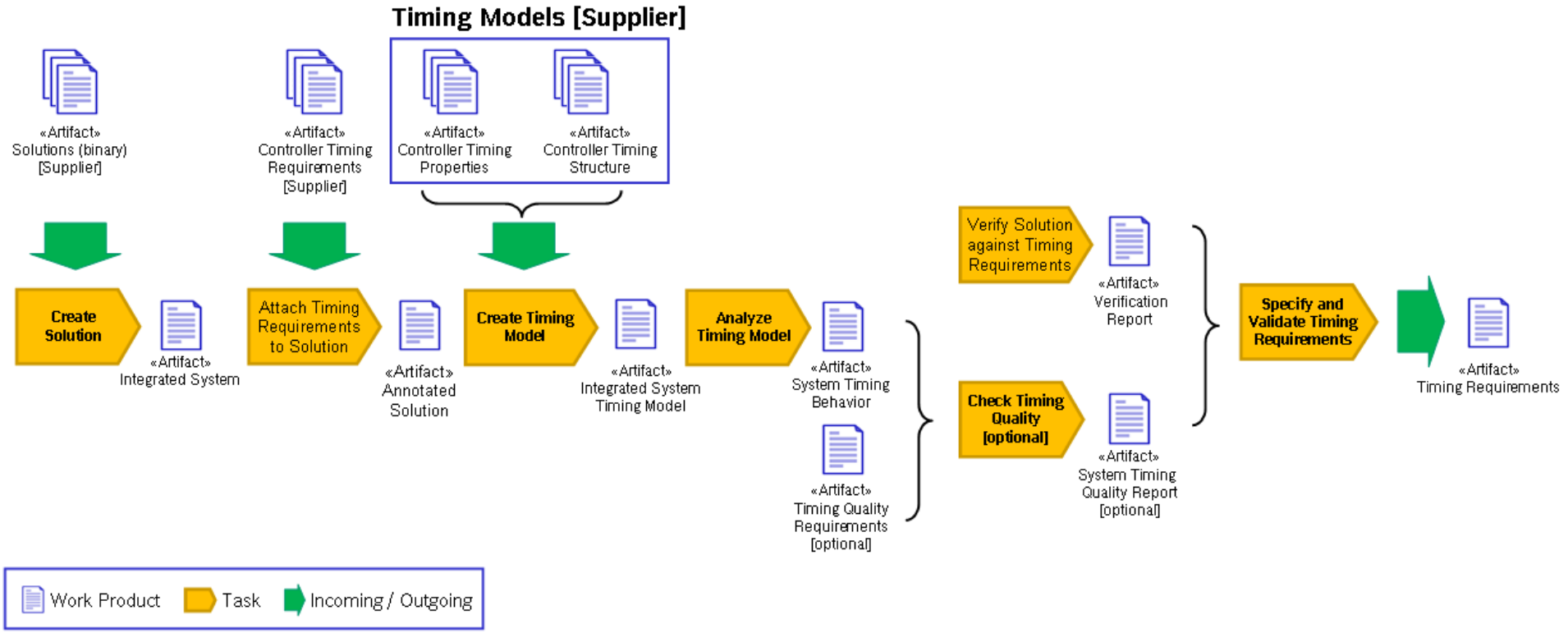

Figure 32 - OEM Side of the Exchange Models Use Case

6 Cross-cutting concerns

6.1 Specify mode dependent timing information

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

TADL2 therefore allows a timing constraint to be dependent on a mode. When a mode is turned on then all its dependent timing constraints become active, and they remain so until the mode is turned off when they are again deactivated.

TADL2 does not provide any means to define modes: it assumes that, for each mode *m*, there is a special event turning *m* on and off. These events provide the interface of TADL2 for modes. A time interval between an event occurrence turning on *m*, and the subsequent event occurrence turning it off, is a "mode window" for *m*: the timing constraints that depend on *m* are active exactly in these windows.

TADL2 does not require that modes are mutually exclusive. However, a timing constraint can only be dependent on one mode. If a constraint is to be active in several modes, then a "super-mode" corresponding to the union of these modes has to be defined outside TADL2. The constraint can then be triggered by an event for that super-mode.

An exact semantics has been defined for mode-dependent timing constraints, which specifies exactly what it means for a timing constraint to be active during a mode. Using these semantics all TADL2 timing constraints can be made mode dependent. Details can be found in deliverable D11.

Example

For a braking system, with an event chain *c* containing a brake pedal actuator event as stimulus and a brake event as response, we may have a mode-dependent reaction constraint where the *maximum* limit for the constraint depends on the velocity *v* as follows:

- *mode1*: *v* in [50,60) km/h => *maximum* = 23.3 ms
- *mode2*: *v* in [60,80) km/h => *maximum* = 17.5 ms
- *mode3*: *v* in [80,90) km/h => *maximum* = 15.6 ms

This can be expressed in TADL2 as three mode-dependent reaction constraints as follows:

```
R1 = reactionConstraint {
       scope = c,
       maximum = 23.3 ms,
```

```
        mode = mode1
    }


R2 = reactionConstraint {
        scope = c,
        maximum = 17.5 ms,
        mode = mode2
    }


R3 = reactionConstraint {
        scope = c,
        maximum = 15.6 ms,
        mode = mode3
    }
```

Note that TADL2 has no means to define the modes themselves: thus, the definitions of *mode1* – *mode3* in terms of the velocity will have to be done outside TADL2.

7 Conclusion

One of the main results achieved in work package 4 is the Generic Timing Methodology (GMP).

The GMP was designed such that it extends established software system development methodologies, such as EAST-ADL and AUTOSAR, with timing aspects. Thereby, the GMP supports both Top-down and Bottom-up development scenarios, and allows applying both in a combined manner. This plays an important role for the daily development routine in the automotive industry.

In various methodology instances the GMP has been refined to give methodology support for many practical use cases. These instances describe in detail how design decisions can be taken based on timing information. In other words, the TIMMO-2-USE methodology introduces a constructive feedback between automotive software system design and real-time systems engineering.

The following use cases were covered during the course of the TIMMO-2-USE project:

- Integrate reusable component
- Specify timing budget
- Specify synchronization timing constraints
- Revise erroneous timing information
- Exchange models

Additionally, the TIMMO-2-USE methodology serves as integration platform of the project results:

- Tool Mentors describe how the different timing related methodology tasks are supported by specialized tools
- TADL guides give hints on how to describe timing information using the Timing Augmented Description Language 2 (TADL2) that was developed in the TIMMO-2-USE Project.

8 EPF Model of the TIMMO-2-USE Methodology

The EPF model of the TIMMO-2-USE methodology can be found under the following web-link:

http://www.timmo-2-use.org/

9 Glossary

Communication delay	A communication delay is the time it takes for a message to be conveyed over a communication medium, such as a CAN bus.
Delay	A delay is the time elapsed between a stimulus event and a response event.
Derived timing property	A derived timing property is a timing property that has been deduced based on either • existing direct timing properties, or • other derived timing properties Finding a derived timing property requires elaborate analysis and estimation.
FDA	Functional Design Architecture
Fuzzy timing requirement	A fuzzy timing requirement is a requirement that is not expressed by a value and a time unit (including units for multi-form time), but is expressed with words that give an intuition of the order of magnitude of the value. Examples of fuzzy timing requirements are: • The end-to-end latency shall be **faster than the reaction time of a human being**. • The light shall be turned on **immediately**. • The brakes at all wheels must actuate **at the same time**. Fuzzy timing requirements are primarily used in the vehicle phase.
HDA	Hardware Design Architecture

Interference	Interference is the total time that a task is interrupted due the execution of other tasks of *higher* priority on the same ECU.
Overall delay	An overall delay refers to a delay with segments. The sum of the delays of the segments must not exceed the overall delay.
Period	The "period" of a task in a time-triggered system is the time it takes between to successive starts of execution of that task.
Solution	A solution is a collective term for all descriptions that can be expressed by EAST-ADL and AUTOSAR, excluding the timing extensions of the respective language. A solution primarily centers (but does not exclude others) around the following concepts at each abstraction level: • Vehicle: Technical feature model • Analysis: Functional analysis architecture • Design. Functional design architecture, Hardware design architecture • Implementation: AUTOSAR Templates • Operational: Physical EE System The interface of timing towards the solution is based on events. Events often refer to ports in the different types of architectures listed above. Ports are therefore a solution element that deserve special attention. Spanning over the highest four abstraction levels, the Environment model also plays an important part.

Synchronisation influencing property	A synchronisation influencing property is a property that has the potential to influence the occurrence of the set of events pointed out by a SynchronizationConstraint. Such a property is therefore critical when it comes to realising a synchronisation property. The following properties with this potential have been identified: • Synchronisation • Response time • Period
Time budget	A time budget is a piece of timing information that captures a delay, which is potentially decomposed into several disjoint segments.
Time budget influencing property	A time budget influencing property is a property that has the potential to influence the response time of a certain end-to-end event chain, and thereby also the required time budget. The following properties with this potential have been identified: • Worst-case execution time (WCET) • Communication delay • Interference time • Task period
Time budget margin	In the time budgeting process, the developer determines the worst-case execution times for each component in the end-to-end event chain. On top of that, the developer may want to provide a slightly more relaxed budget. The difference between the WCET and the budget of a component is called the "margin" of that component. The margin is always communicated to the supplier implementing the function, possibly implicitly in the budget segment.
Time budget segment	A budget segment is a piece of timing

	information that expresses a delay that constitutes a part of a time budget.
Time budget slack	Time budget slack is a portion of an end-to-end delay that is not allocated to any budget segment. Thus, there is only one slack per end-to-end delay. Slack is generally not communicated to suppliers, but rather serves as a reserve for interference from other not yet implemented functionality.
Timing information	A piece of timing information is any information that can be expressed with TADL2.
Timing property	A timing property is a piece of timing information that, with respect to a certain abstraction level, either: 1. Is based on already existing knowledge about the solution at a lower abstraction level 2. Obtained from the solution and the timing requirements that the solution was based on 3. Is assigned by the developer
Timing requirement	A timing requirement at a certain level of abstraction is a piece of timing information that will (together with other timing requirements) serve as a basis for a solution at a lower level of abstraction.
Transformed timing property	A transformed timing property is a timing property that has been translated (or ported) from a timing property at another (lower or higher) abstraction level than the transformed timing property in question.
Worst-case execution time	The worst-case execution time (WCET) is the longest time that a component can execute on a given hardware platform.

10 References

[1] TIMMO Deliverable D7 Methodology Version 2,

http://www.timmo.org/pdf/D7_TIMMO_Methodology_Version_2_v10.pdf.

[2] ATESST 2 Deliverable Methodology.

[3] TIMMO-2-USE Deliverable D1.2 http://www.timmo-2-use.org/. Check for the latest version of this deliverable.

[4] TIMMO-2-USE EPF Model, http://www.timmo-2-use.org/. Check for the latest version of this deliverable.

[5] EAST-ADL Specification, http://www.atesst.org/. Check for the latest version of this deliverable.