ITEA 2

INFORMATION TECHNOLOGY FOR EUROPEAN ADVANCEMENT

ITEA 2 - 09033

TIMMO-2-USE

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

Report type	Deliverable D5
Report name	Methodology description V1
Report status	Consortium Confidential
Version number	Version 1.0
Date of preparation	2011-06-15

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

Wendel Ramisch, INCHRON GmbH

Document History

Version	Date	Description
0.1	2011-05-04	Initial Draft
1	2011-07-08	First released version

Table of contents

1 Introduction

In this document the intermediate 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 0) 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 4 are mapped to this generic methodology. One important distinctive characteristic of the GMP is the integration of top-down and bottom-up development aspects into one single methodology.

The currently covered use-cases that are described in Section 4 are the following:

- Integrate Re-useable Component
- Specify timing budgets
- Specify synchronization timing constraints
- Develop Control Application

All of these methodology instances are additionally modeled in SPEM (Software Process Engineering Metamodel) using the EPF (Eclipse Process Framework).

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)

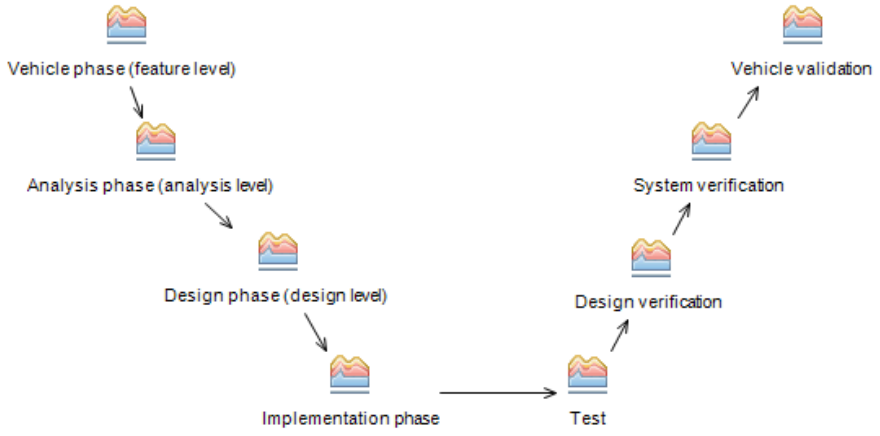

Figure 1- The different Phases of the TIMMO Methodology

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.

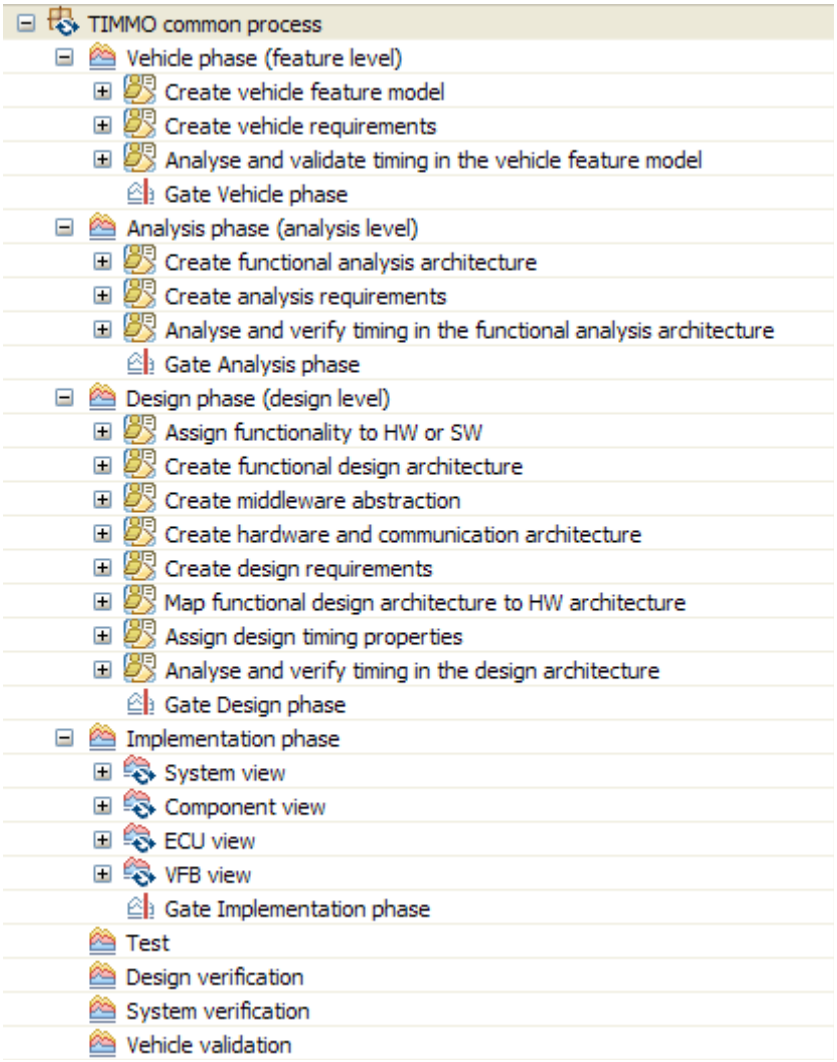

Figure 2 - Different phases and tasks of the TIMMO methodology

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. Due to 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 EAST-ADL 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:

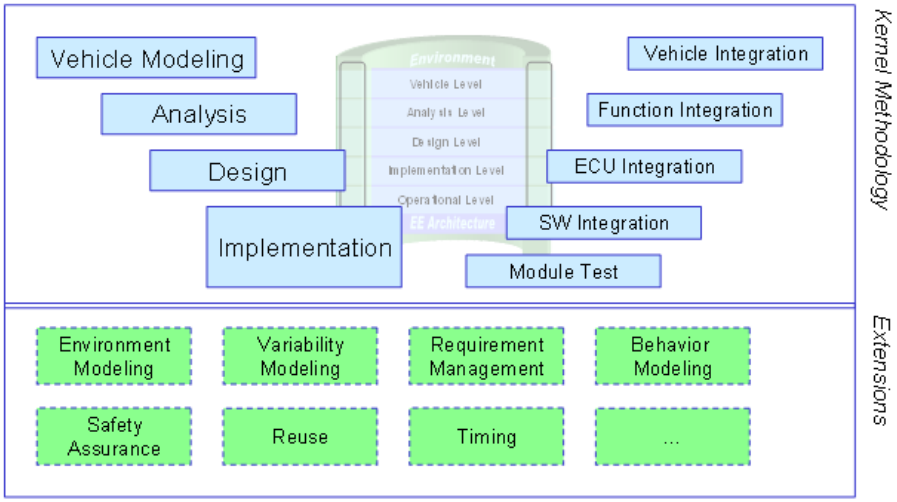

Figure 3 - The structure of the EAST-ADL methodology

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

This chapter describes the TIMMO-2-USE Generic Method Pattern that is the basis for all steps to be taken during the course of a phase that deals with creating, altering, processing, and utilizing timing information.

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 *and* its value.

Introduction

As shown in Figure 4, the TIMMO-2-USE Generic Method Pattern consists of the five tasks[1] called “Create Solution”, “Find Timing Properties”, “Analyze”, “Verify and Validate”, and “Specify Timing Requirements”.

By and large, these tasks are carried out on every level of abstraction defined by the EAST-ADL respectively during every phase of the corresponding EAST-ADL methodology. As shown in Figure 5 there are two exceptions: The first exception is that on the Vehicle Level respectively at the beginning of the Vehicle Phase, a formal work product “Timing Requirements” is not available.

And the second exception is that on the Operational Level respectively at the end of the Operational Phase the task “Specify Timing Requirements” is not carried out.

[1] The term “task” is used for a number of subsequent steps to be taken to process the timing related input work products and create the corresponding timing related output work products.

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 abstractions.

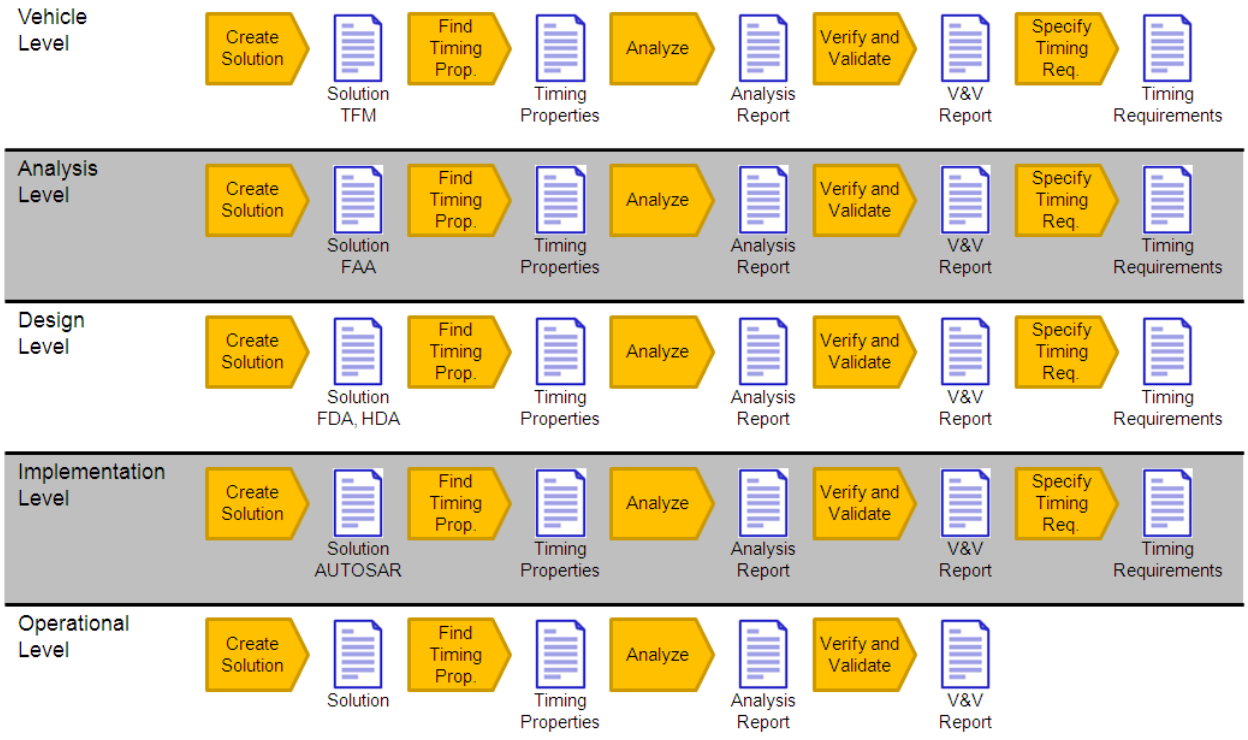

Figure 5 - Instantiation of TIMMO-2-USE Generic Method Pattern

This instantiation is shown in Figure 5. On every level of abstraction respectively in every phase the corresponding tasks are conducted – except the “Specify Timing Requirements” on the Operational Level. At the end of the Vehicle-, Analysis-, Design-, and Implementation Phase the work product “Timing Requirements” evolving from the particular is phase is passed as basis for subsequent activities in the following phase – except the Operational phase.

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

Create Solution

Based on the given requirements[2], 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, Functional Analysis Architecture FAA, Functional Design Architecture FDA, Hardware Design Architecture HDA, and Environment Model EM.

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

Find Timing Properties

Once the solution has been created, the timing properties of this solution are specified and the values of these timing properties are determined and assessed. The methods applied to determine – find – the particular values are manifold: [timing] expert estimation, simulation, analysis, educated guess, knowledge from previous projects respectively iterations, etc. The most appropriate and suitable method should be selected for this purpose.

The objective of this task is to find timing properties that are inherent in the solution and its requirements. The methods to perform this do not involve taking any design decisions. The properties found in this task shall rather form the basis for such design decision in subsequent tasks.

Note that the purpose of this task is *not* to define new types of timing properties, but to decide which of the timing properties, like latency,

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

response time, execution time, sampling rates, etc. are used to describe the dynamic behavior of the solution.

If several solutions are available, then each of those solutions is annotated with timing information. And with regard to the dynamic – temporal – behavior of the solutions there may be different critical paths leading to different sets of timing properties and their values.

Analyze

Based on the solution and its timing properties the specific those timing properties are assessed with regard to the target system. The timing properties found so far are taken as basis for creating more elaborate and comprehensive timing properties. These timing properties are likely, but not necessarily, involved in some form of design decisions. In order to perform the assessment the scope of the analysis is broadened in order to include a larger part of the system and all timing properties associated to that part of this system.

The methods applied to assess the values of the timing properties are manifold and the most appropriate and suitable method should be selected for this purpose. Such a method could be as simple as an addition of values, or it could be more complex, like applying a calculus on the given numbers. In addition, the methods being used for analyses may vary depending on the phase: On higher levels of abstractions other methods are used than on lower levels of abstraction. For example, scheduling analysis is used on implementation level, but not on abstraction levels like Vehicle Level.

At this point, several solutions (alternatives) may be available and in this case the purpose of the task “Analyze” 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 and Validate

Eventually, the timing properties are [“officially”] verified and validated.

By and large the two primary questions are answered:

1. Has the specification of the timing properties’ values be done right?
2. Have the right [values of the] timing properties be specified? Are the given timing requirements be satisfied by the specified timing properties?

During the course of this task the values of the timing properties are compared against the values of the given timing constraints received from the higher level of abstraction respectively previous phase. 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. Essentially, this is the task which “compares the numbers of timing properties with given timing constraints”.

When several solutions (alternatives) are available the purpose of the task “Verify and Validate” is to verify and validate the timing properties of every solution. 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 and Validate” the results of the verification and validation are checked. And a decision must be taken either to continue or to repeat the phase. At this point one can decide whether and if so how to repeat the phase. 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 Timing Requirements

Once the quality gate has been passed all or some of the obtained timing properties 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 important for respectively the basis for design decision to be taken in subsequent steps.

These timing requirements are the basis for any design work being conducted on the next level of abstraction respectively next phase.

3.1 Example

This paragraph introduces a very simple example that is used to explain how the Generic Method Pattern is applied respectively utilized.

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. This solution is shown in the upper part of Figure 6. The solution is a function/component with one required and one provided port. The function/component receives a signal from the environment via its required port and emits a signal to the environment via its provided port.
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. The 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. The event is playing the role of the response. A latency timing constraint (TC) is imposed on this event chain (EC) and its value is 125 ms including a variation – jitter – of 30 ms resulting in a time range of 110 ms to 140 ms.

Component	Latency Timing Constraint	Minimum	Maximum
Function	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” that is supposed to satisfy the given functional and non-functional requirements, specifically the timing requirements. This solution is shown in the lower part of Figure 6. It consists of two functional devices («FD») and two functions/components («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 subject to be developed; and the purpose of the functional device called “Actuator” is to “control/impact” the environment. Two functions/components («AF») called “F1” and “F2” processing the data received from the environment via the functional device “Sensor” and control/impact the environment via the functional device called “Actuator”. The functional device “Actuator” provides additional data to the function/component called “F1”.

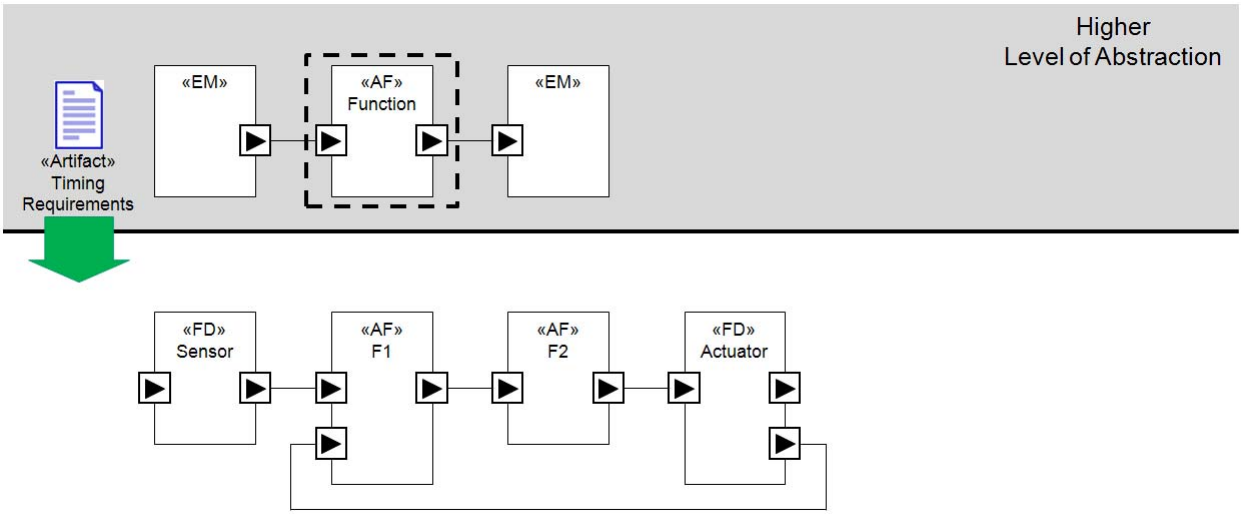

Figure 6 - A simple example to demonstrate the use of the TIMMO-2-USE Generic Method Pattern

Example – Find Timing Properties

During the course of the task “Find Timing Properties” the solution is annotated with events, event chains, and timing constraints as shown in the lower part of Figure 7 – depicted by the red colored event chain drawn above the functions/components called “Sensor”, “F1”, “F2”, and “Actuator”. On this level of abstraction the given event chain including its latency timing constraint is broken down into four subsequent event chains, playing the role of event chain segments, and latency timing constraints are imposed on those four 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 functional device called “Sensor”, because the solution provides data for example periodically.

In this example, an event chain referring to the second provided port of functional device called “Actuator” and the second required port of the 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.

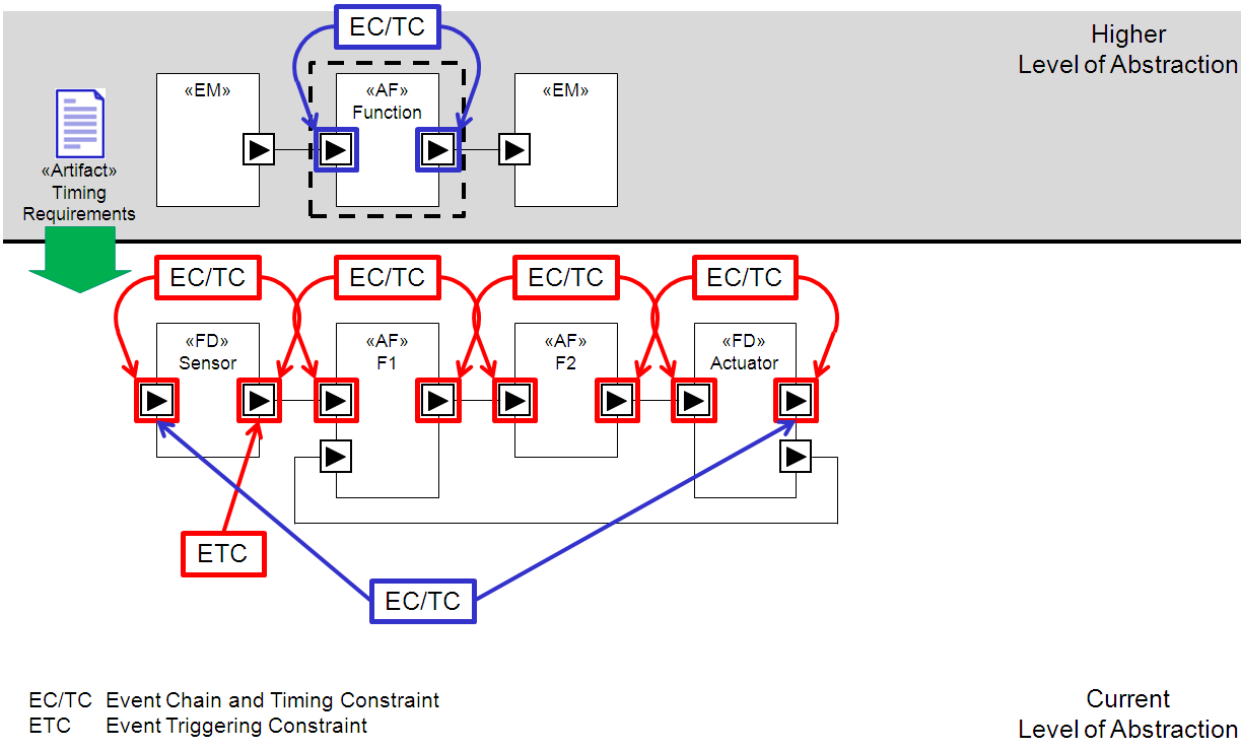

Figure 7 - The simple example to demonstrate the use of the TIMMO-2-USE Generic Method Pattern annotated by timing information

Furthermore, an event chain can be specified referring to an event that is observed at the required port of the functional device called “Sensor” and an event that is observed at the provided port of the functional device called “Actuator”. And a timing constraint is imposed on this event chain. This timing constraint – the property and the value – may be the same as the given one.

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 functional device 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 functional device 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
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
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 functional device 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

In this step – carrying out the task “Analyze” – 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 “Find Timing Properties” 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
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
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 and Validate

The obtained values of the timing properties are now compared against the given timing constraint specified at the higher level of abstraction. For this purpose, an event chain is specified that references the event observable at the required port of the functional device called ”Sensor”, playing the role “Stimulus”, and that references the event observable at the provided port of the functional device 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 7. 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 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.

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 “Find Timing Properties”.

Figure 8 shows a simplified view of the methodology with regard to this approach. The task “Transform Timing Properties” on the lower level of abstraction transforms the timing properties’ values of a solution created on this level of abstraction into values of timing properties that can be used at the higher level of abstraction. The transformed timing properties, including their values, are an optional input work product for the task “Find Timing Properties” conducted on the higher 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.

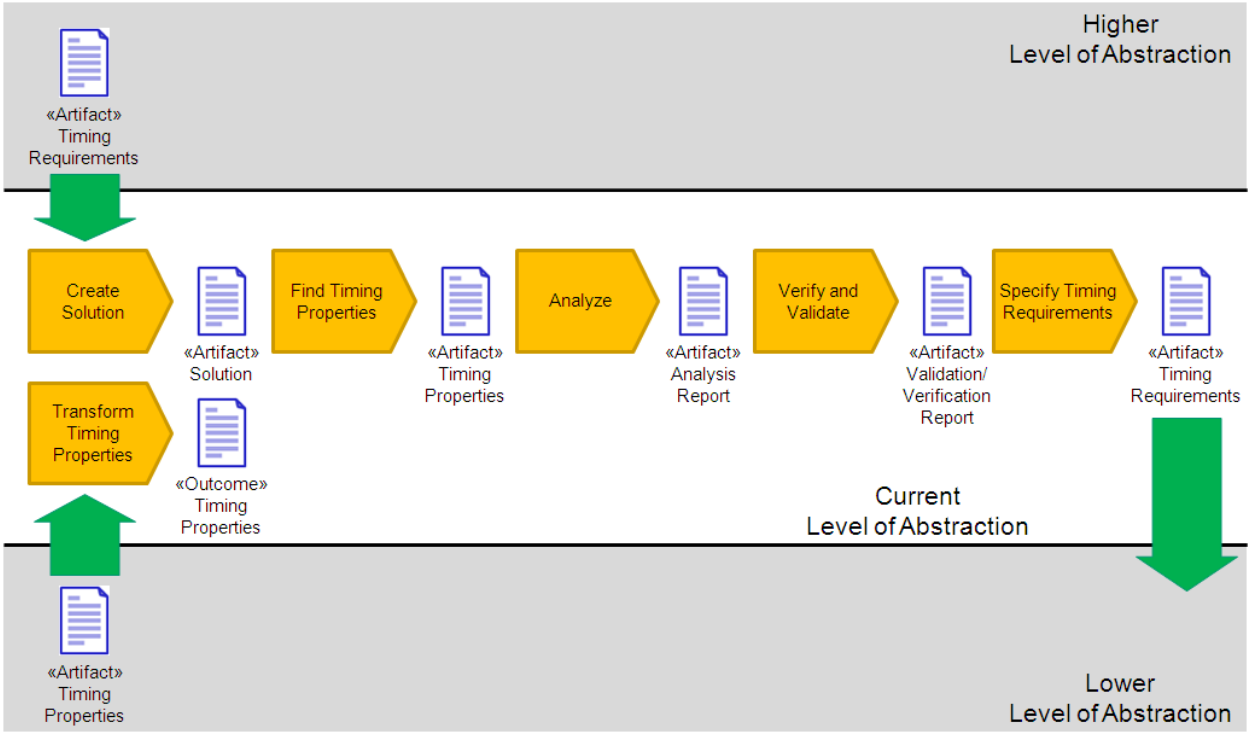

Figure 8 - Abstracting Timing Properties

4 Application of the Generic Pattern to Use Cases

In this chapter several specific use cases defined in Deliverable D1 are addressed. All of these methodology instances are based on the Generic Methodology Pattern introduced in Section 3.

4.1 Integrate Re-useable 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. This use case also applies when new platform generations are being developed. In that case new functionalities are integrated step-wise during the development phase.

Usually, new functionalities will not be introduced independent of the existing system's functionalities but will be integrated into the existing system's ECU(s) and communication topology.

Changing a system's architecture necessarily changes its behavior with respect to timing.

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

Overview

The integration may cover a single ECU, or even several ECUs including their communication paths. In this use case, only one ECU will be taken into account, and the focus will be on the Design phase. In phase 2 of the TIMMO-2-USE project, this use case will be elaborated describing the integration of a more sophisticated functionality finally spanning more than one ECU.

Prior to the detailed integration process, an ECU (or several alternative ECU candidates) has to be chosen which the design function in question will be integrated on. There might be several aspects to be considered when choosing an ECU, like the functional domain which it belongs to (e.g. body controller), physical location (e.g. near front wheels), availability of input signals (e.g. sensor signals, buses), or availability of processor capacity (idle time). However, the ECU selection process is not part of this methodology.

The investigations on the use case "Integrate Re-usable Component" assume that the software system executed on the target ECU, which the design function is to be integrated into, was developed according to the Generic Methodology Pattern (see Section 3). Therefore, we assume that it is described in an EAST-ADL model on design level including timing information. 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, e.g. WCET and activation periods of functions.

The use case considers two integration scenarios:

A) adding a legacy design function

B) developing and adding a new design function

In scenario A, it is assumed that also for the legacy design function an EAST-ADL model exists which contains TADL2 compliant timing information.

For scenario B, there are two approaches how to start integration:

1) Developing the new design function stand-alone without taking into account interactions with the target system. In this case a separate EAST-ADL 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.

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

The approach B1 starts identically to developing a new functionality from scratch, that is, include treatment of timing information according to the generic methodology 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, i.e. integrating a legacy design function.

In this use case the scenarios A and B1 will be taken into account, so this use case will deal with **integration of one EAST-ADL model into another, both models already containing timing information**.

The further considerations basically reflect the *Design* phase. In 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. Thus models can be investigated stand-alone. In contrast to this, in the *Design* phase components are declared as to be realized in hardware or software which results 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 9 illustrates the integration process, and how it maps to the generic methodology presented in section 3.

In the following paragraphs, the existing system which the design function shall be 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).

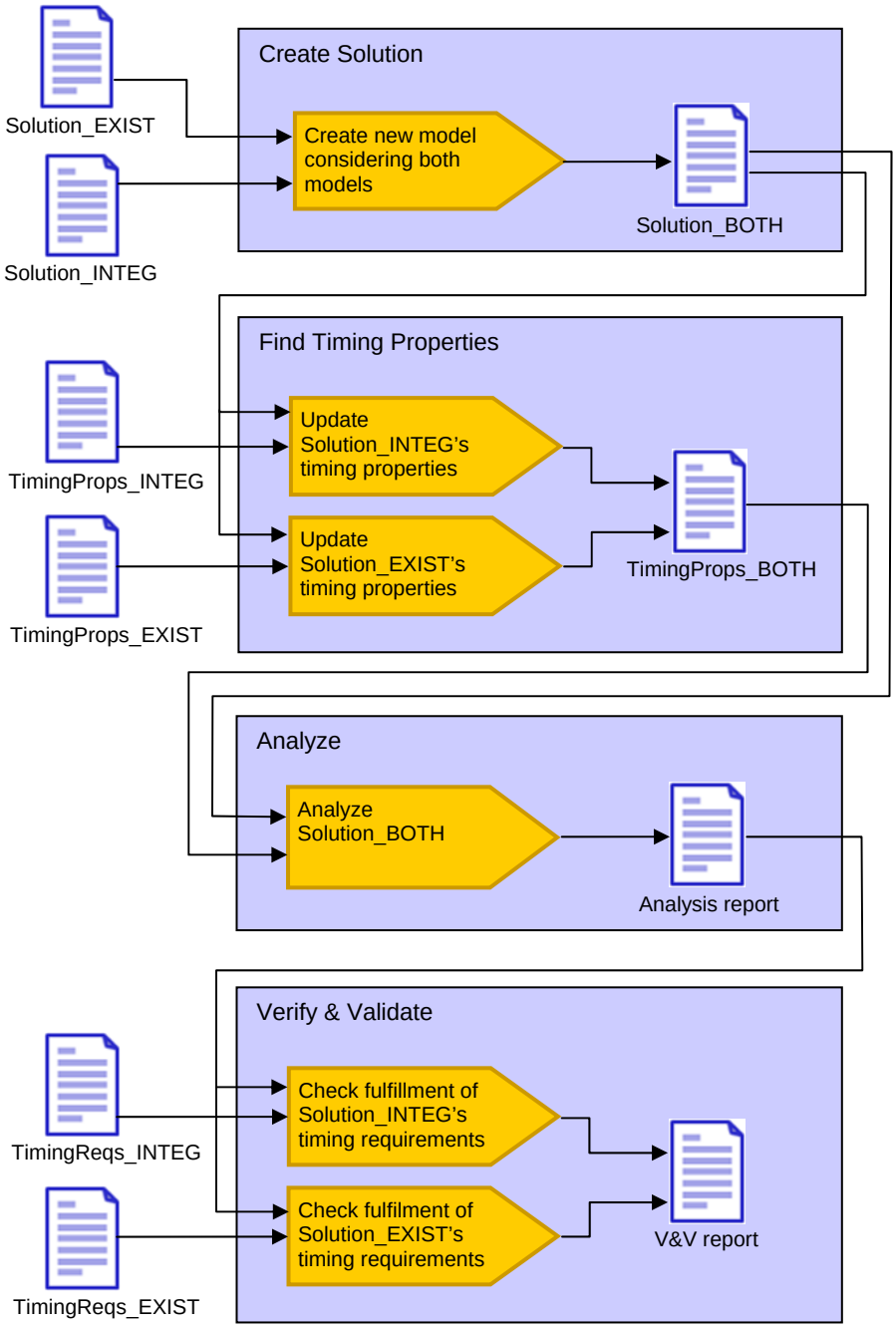

Figure 9 - Generic methodology applied on integration

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, reuse of input (and output, if applicable) ports is advised, e.g. in case both solutions use the same sensor signal.

It is assumed that the resulting model Solution_BOTH will contain the same events like the previous models Solution_EXIST and

Solution_INTEG, so that all timing requirements applied on the models, specified in the level of abstraction above, persist in the resulting model.

Find Timing Properties

The scope of the task *Find Timing Properties* is twofold and therefore described in two subtasks.

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 execution times the following two 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 value of a timing property, the new value is computed by applying an extrapolation formula. The most simple 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.

The methodology will not give advice on how to update the necessary timing properties; this is subject to the 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.).

Analyze

In the task *Analyze*, 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 10).

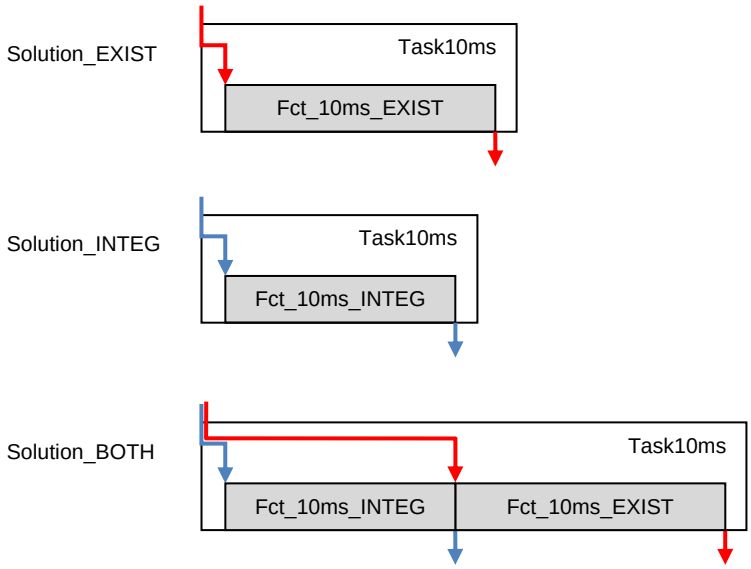

Figure 10 - Timing behavior before and after integration

Figure 10 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 compared to before the integration. Also the response time of function Fct_10ms_EXIST will increase in the depicted scenario.

Verify & Validate

The task *Verify & Validate* compares the analysis results (timing behavior of Solution_BOTH) with the requirements. It consists of two parts:

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

Specify Timing Requirements

The scope of the task *Specify Timing Requirements* is to transform the timing properties of Solution_BOTH into timing requirements for the next (lower) level of abstraction. 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

On the *Implementation* Level similar tasks have to be performed as described here 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.

4.2 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. Such time limits, hereafter called end-to-end delays, are specified based on a user's perception with respect to a certain functionality.

In a design, the data and control flow paths between a stimulus and a response generally go through several components. These paths from stimulus to response 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.

Time budgeting is thus about how to divide 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 an explicit or implicit user requirement or expectation. 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 11 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.

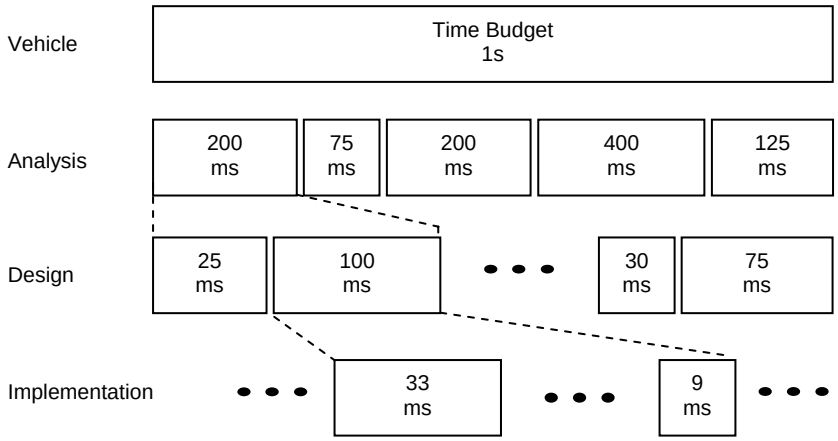

Figure 11 - The principles of time budgeting

Mapping to generic methodology

Figure 12 presents the time budgeting process, and how it maps to the generic methodology presented in section 3. The *Find timing properties* and the *Analyze timing properties* tasks have been split into two subtasks each in order to illustrate the activities to be performed in these tasks in more detail. Moreover, the tasks *Verify timing properties* and *Specify timing requirements* 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.

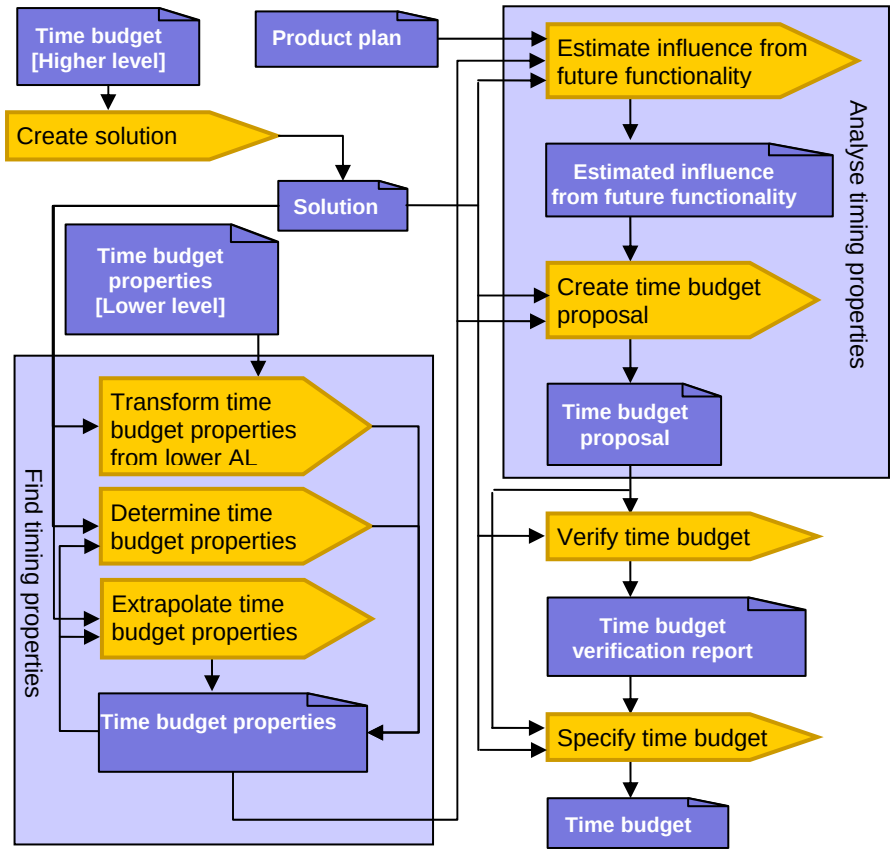

Figure 12 - Generic time budgeting methodology

Time budget properties

A time budget 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
- Blocking time
- Interference time
- Invocation delay
  - Event-triggered: Release delay
  - Time-triggered: Task period

Slack vs. margin

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 WCET 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.

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 input 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 take 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.

Find timing properties

The task *Find timing properties* identifies time budget properties that are a direct implication of the solution and its timing requirements. These properties can be obtained using the following strategies:

1. Transformed from a lower abstraction level
2. Determined from the solution
3. Determined from an extrapolated solution at lower abstraction level

Each of these strategies is represented by a separate task. All tasks contribute to the same output work product *Time budget properties*. The following paragraphs describe the tasks in more detail.

The purpose of the task *Transform time budget properties from lower abstraction levels* is to reuse 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 task *Determine time budget properties* 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 level may transform and apply to its models in the *Transform time budget properties* task of that abstraction level.

The task *Extrapolate time budget properties* 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. 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 in the task *Determine time budget properties*. 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.

Analyze timing properties

In *Analyze timing properties*, the time budget properties are further processed to obtain a time budget proposal. This is done in two consecutive tasks:

1. Estimate influence from future functionality
2. Create time budget proposal

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 *Estimate influence from future functionality* compares 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 makes 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. Unknown functionality that will be included into the system in future generations of the system may also be considered in this task.

A final time budget proposal is formed in the task *Create time budget proposal* based on the identified time budget properties and the estimated influence from future functionality. It should be noted that this input information only serves as a guideline for the budgeting process. It is, for instance, sometimes desired to add a margin to a known WCET, 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.

The set of delay constraints in the identified time budget properties cover, in general, only a part of the end-to-end event chain. A major challenge in this task is to assign time budgets to segments for which no time budget properties have been found. Since no information is available, this has to be done based on behavioral models of the concerned components with the help of the developer’s previous experience.

Verify time budget

The task *Verify 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 time budget

The task *Specify 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

The tasks in *Find timing properties* produce results that more or less reflect properties that are inherent in the solution and input requirements. The only way that a developer can influence these properties is by either changing the solution or the requirements. On the other hand, the tasks in *Analyze timing properties* appeal to a big extent to the subjective judgment and experience of the developer. Symbolic time expressions can be a powerful tool to navigate through this freedom.

The concept will be illustrated on the example shown in Figure 13. 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.

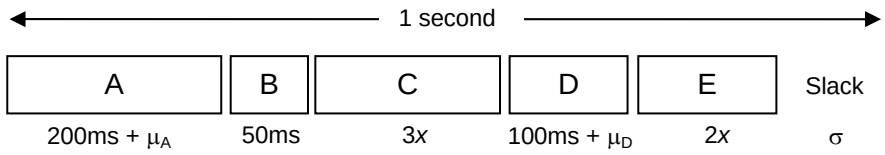

Figure 13 - Time budgeting example using symbolic time expressions

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 13, 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.

Instantiation on abstraction levels

The process outlined in Figure 12 is in general applicable on all abstraction levels. However, some of the tasks in *Find timing properties* and *Analyze timing properties* do not exist for all abstraction levels, or are less important. Figure 14 illustrates this relationship.

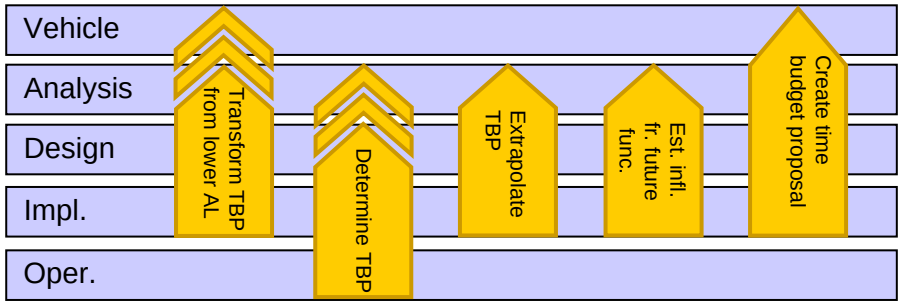

Figure 14 - The degree of presence of Find and Analyze timing properties tasks at different abstraction levels

The task *Transform time budget properties* inherently requires that there exists a lower abstraction level from which properties can be transformed. For this reason, it is not present at the operational level. Since the vehicle level primarily focuses on the user's needs and perception, the task is not very present at that level either, although it sometimes makes sense to peak at results from early implementation.

The task *Determine time budget properties* requires that there exists at least a structural model, and preferably also a behavioral model, to analyze. This does not exist at vehicle level. Although such models do exist on analysis level, they often do not contain sufficiently detailed information that it is possible to directly determine any time budget properties. Such cases are better suited for the task

Extrapolate time budget properties. However, that task both requires structural, and preferably also behavioral, models to start the extrapolation from, as well as lower abstraction levels as target for the extrapolation. These conditions do not hold for the vehicle and operational levels.

Since the vehicle level only should reflect the user's needs and perception, possible influence from future functions should not affect the vehicle level's model. Moreover, the operational level does not give rise to further time budget requirements. For these reasons, the task *Estimate influence from future functionality* does not occur on this abstraction level.

As time budgets are not specified on the operational level, the task *Create time budget proposal* is not present at that level.

4.3 Specify synchronization timing constraints

Problem statement

A vehicle offers many different features such as braking, steering etc, to the driver. 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. These constraints will then have to be decomposed into smaller pieces. The purpose of the decomposition is however primarily to simplify the fulfillment of the constraint in the design rather than supporting different stakeholders of the development process. Thus, this use case deals with the formulation of synchronization constraints and how they are decomposed into manageable pieces during the design.

Overview

As already mentioned, synchronization constraints can be imposed on both input and output. Input synchronization means that for a given response, there is a set of stimuli which should have occurred within a certain time window prior to the response. Similarly, output synchronization means that for a given stimulus, there is a set of responses which should occur within a certain time window after the stimulus. In addition, the constraint construct uses the notion of an upper and lower bound on how near in time the window will have to appear to the given response/stimulus. Thus, the braking example illustrates the output synchronization where the pressing of the brake pedal corresponds to the stimuli and the brake actuations are the responses. The time window would in this case represent the

maximum allowed skew in time among the actuations at the different wheels, e.g., 20 ms. The upper bound would represent a delay constraint on the time from the brake pedal is pressed until the actuation is completed, e.g., 200 ms.

From an end-user (i.e. driver) perspective (vehicle level), only output synchronization constraints are imposed as requirements on the system. The reason is that it is not possible to place requirements on how the environment affects the system. It is only possible to constrain how the system affects the environment. Input synchronization constraints are introduced on the analysis level when the vehicle features are translated into functions with sensors and actuators. The main challenge to address when formulating a synchronization constraint is the identification of the events which represent the stimuli/responses as well as finding concrete figures for the parameters *lower*, *upper* and *width* (of the time window).

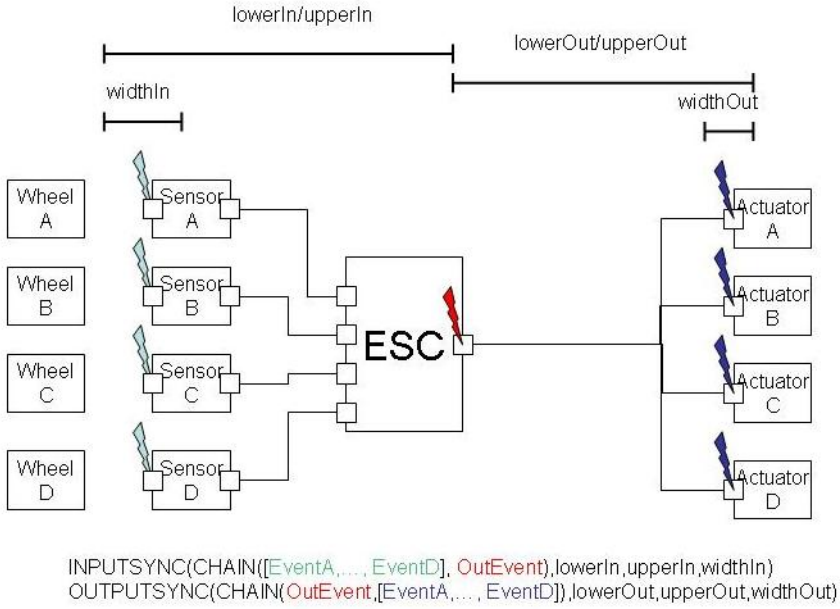

Figure 15 - Illustration of synchronization concepts

An illustration of the different concepts can be seen in Figure 15. The figure shows an example of an ESC (Electronic Stability Control) system modeled in EAST-ADL (analysis level). In the figure, the red arrow represents the stimulus for the output synchronization and the blue arrows are the corresponding responses that need to be synchronized. In this example, this means that the control command issued by the ESC function must be simultaneously acted upon by the wheel actuators to ensure vehicle stability. (A maximum time deviation of *widthOut* is allowed.) The stimuli events for the input synchronization are illustrated using the light blue arrows and in this case the red arrow represents the response. In the example this means that the sensor values representing the wheel data needs to simultaneously sampled in order for the ESC function to compute an accurate control command. (Again, a sampling time difference of *widthIn* is tolerable.)

Mapping to generic methodology

Figure 16 presents the generic process for formulating and decomposing synchronization constraints including the mapping to

the generic methodology. Since the generic methodology applies to all EAST-ADL abstraction levels, some of the tasks may not be possible or needed for a particular level. Typically the methodology tasks involved in formulating the synchronization constraint are performed at vehicle or analysis level, while the tasks concerning decomposition of the synchronization constraint are applied at design and implementation level. It is however still possible to formulate new synchronization constraints also at the lower levels if needed by the selected solution. The instantiation of the tasks at the different abstraction levels is illustrated in Figure 17. The tasks will be described in more detail in the following paragraphs.

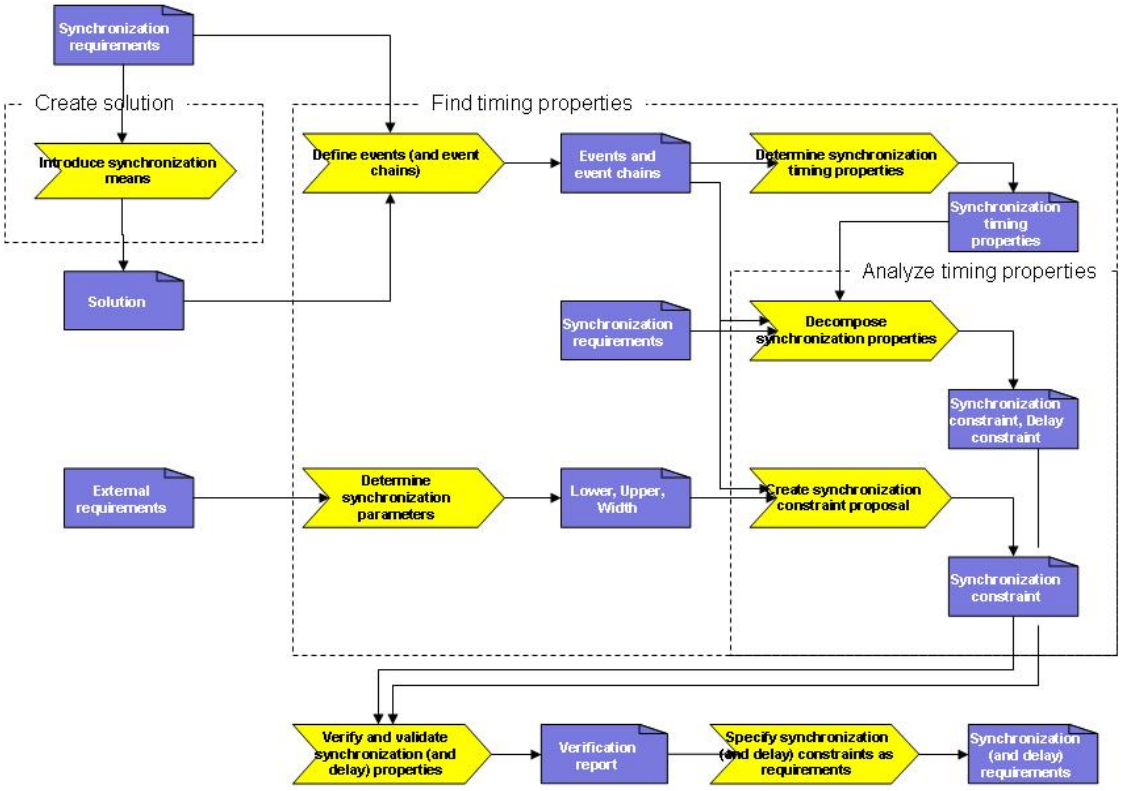

Figure 16 - Generic methodology for formulating and decomposing synchronization constraints

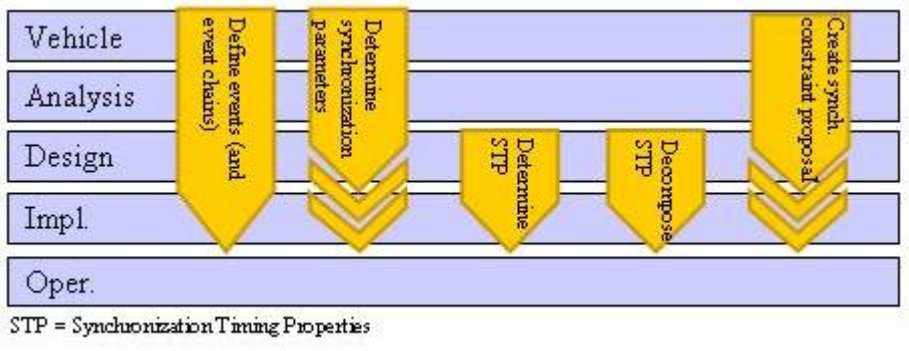

Figure 17 - The degree of presence of Find and Analyze timing properties tasks at different abstraction levels

Introduce synchronization means

The task *Create solution* is not part of the timing methodology since it concerns the regular design activities involved when moving from a higher abstraction level to a lower. It may however be the case, that existing synchronization requirements call for the introduction of certain technical solutions in order to meet these requirements. For example, it may be necessary to use time triggered communication or buffering to ensure the simultaneous arrival of distributed messages.

Define events (and event chains)

Before the synchronization constraint can be formulated (or decomposed), it is necessary to identify and define the events (stimuli/responses) that will be used in the constraint formulation and that corresponds to the meaning of the requirement. On the vehicle level, this typically means the events that are located at the interface to the plant model since this represents the outer boundary of the system. When moving from a higher abstraction level to a lower, it is important to make sure that the mapping of events between the levels is adequately done. For the events their occurrence pattern has to be specified using the *period* parameter. This setting could be made more or less beneficial concerning the synchronization requirement and should be carefully selected.

Determine synchronization parameters

At vehicle level, the synchronization requirement will typically not be expressed in terms of the parameters *lower*, *upper* and *width* but it will rather be implied as a consequence of other requirements dealing with human perception, laws of physics, etc. These requirements will in turn in many cases be fuzzy and non-quantified. Derivation of concrete figures for the mentioned parameters will therefore typically be done based on experience or rules-of-thumb. At analysis level, the behavior of the selected solution algorithms may result in certain figures in order to fulfill the vehicle level requirement.

Create synchronization constraint proposal

This task addresses the formulation of a new synchronization constraint in TADL format. Hence, when the events and synchronization parameters are defined, it is possible to formulate the actual constraint. However, as part of this task it is also important to analyze how easy it will be to actually fulfill the constraint in the final design. Normally there are several choices available for the parameters and a suitable trade-off between desired synchronization effect and design feasibility must be made. In particular, the relation between *width*, *lower*, *upper* and the periodic behavior of the events plays an important role. For example, if *width* is 20 ms and the events occur periodically every 10 ms, the synchronization constraint will always be satisfied. In contrast, if *width* is 10 ms and the event periods 20 ms, the constraint may or may not be satisfied depending on how the events actually occur in reality.

Decompose synchronization properties

In contrast to the previous task, this task addresses the decomposition of an existing synchronization requirement. That is,

when the synchronization requirement exists in TADL format, the task is to decompose the constraint into a number of delay constraints and a “shorter” synchronization constraint. For example, if lower=0 ms, upper=100 ms and width=10 ms, involving stimuli A, B, C and response D, this could be divided into a synchronization constraint with width=10 ms, lower=0 ms, upper=50 ms involving A’, B’, C’ and D plus three age constraints A’ to A’, B’ to B’ and C’ to C’ with lower=0, upper=50 ms. To arrive at the figure 50 ms as a suitable separation point, it is necessary to analyze the underlying timing properties that are associated with the events. An example for when the decomposition is feasible is shown in Figure 18 and Figure 19. The purpose of the decomposition is to simplify the constraint handling by minimizing the set of entities that each constraint affects. The idea is that the decomposition maintains consistency. That is, if the “local” constraints are satisfied, so is also the “global” constraint. From a constraint point of view consistency is rather straight forward to obtain. The difficulty lies in ensuring that the constraints also are met by the solution.

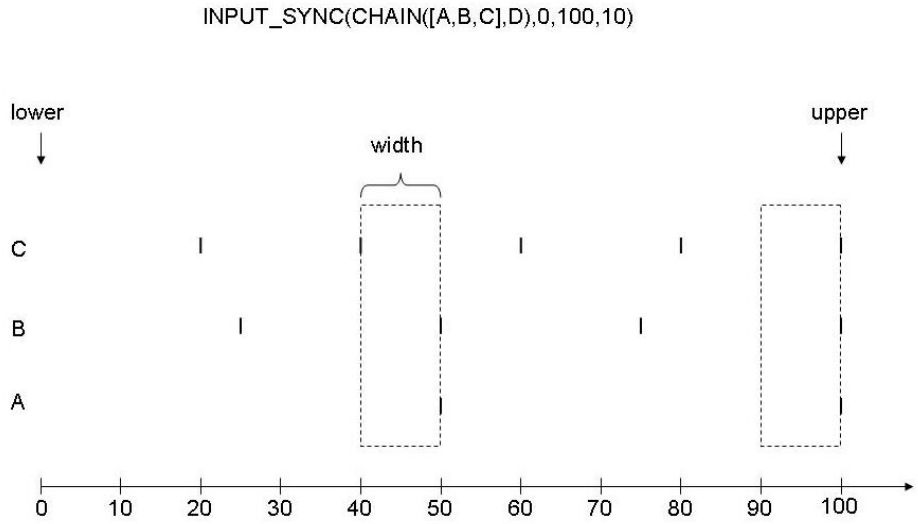

Figure 18 - Original synchronization constraint

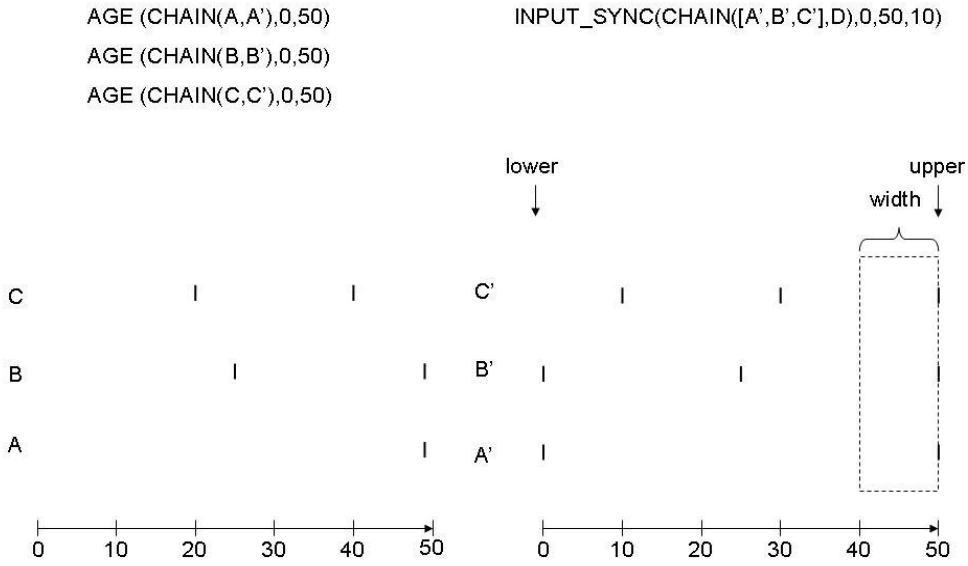

Figure 19 - Decomposed synchronization constraint

Determine synchronization timing properties

To be able to perform any analysis of the timing properties related to synchronization constraints, the timing properties affecting the validity of the constraint must first be determined. This means any information that can be obtained regarding the events and their timely propagation through the system. This includes WCET, communication delays, jitter, response times and offsets. These properties will in most cases have to be estimated since concrete figures typically are known only at the implementation level. In principle, the same techniques as in the time-budgeting use case can be applied to perform the estimation. Of course, if the system is designed based on already existing components, the figures would be available with a higher accuracy although they would probably still need adaptations due to the new system setting.

Verify and validate synchronization (and delay) properties

When the constraints have been formulated (in TADL), the analysis part of the tasks *Create synchronization constrain proposal* and *Decompose synchronization properties* should also have ensured that the constraints indeed are satisfied by the design constructs within the abstraction level. It is the responsibility of this task to actually make this check and report the result. What is also performed here is the validation of the constraints compared to the synchronization requirement from the higher abstraction level. That is, to make sure that the formulated constraints indeed express the intended meaning. This validation is perhaps most important at vehicle and analysis level where the originating requirement is not formalized in TADL but needs to be interpreted.

Specify synchronization (and delay) properties as requirements

This final step involves adding the modeling constructs necessary to indicate that the formulated constraints indeed are to be considered as requirements on the lower abstraction level.

4.4 Develop Control Application

The main goal of this use case is to enable co-engineering between control engineers and real-time engineers. This might be called “resource constrained control”. In the following the necessity of this co-engineering is explained and a methodology implementing it is presented.

Problem Statement

Control design is usually performed assuming idealized timing assumptions, including sampling without jitter and negligible delay from controller input (sensing) to output (actuation), etc. Of course, each ECU does not only execute a single control application but performs several functionalities in parallel. Each of them is developed under the above stated assumptions.

In the case of a single core ECU, these control applications compete for processor time. Since only one controller application can be executed at the same time, a real-time kernel arbitrates the access to the processor according to a scheduling policy. This of course leads to increased and varying response times for the control applications due to preemptions, blocking effects, etc. Obviously, the idealized assumptions stated above do not hold under such circumstances.

In Figure 20 the effects of scheduling that lead to varying input-output delays are demonstrated.

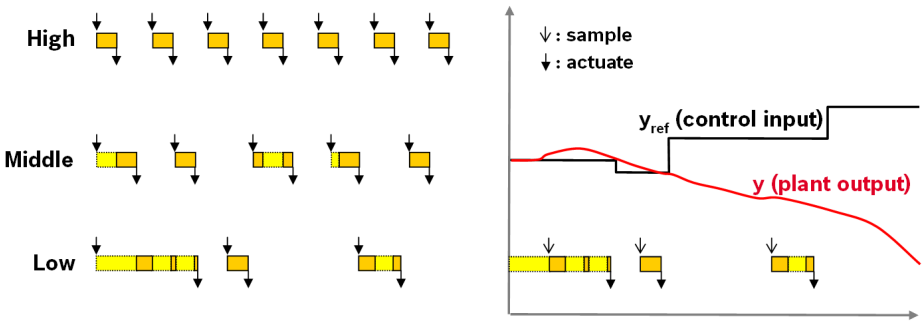

Figure 20 - Effect of scheduling on control performance

On the left-hand side, an activation graph of 3 tasks executing control applications is shown. As can be observed, the lowest priority task executes very irregularly due to preemptions of the two higher priority tasks.

On the right-hand side the impact on control quality is shown. The input-output delay between sampling and control varies drastically due to the preemptions. The executed control application does not succeed in adjusting the plant output (y) to the desired control input values (y_{ref}).

There are several possible solutions in such situations. Of course, the priority of the task executing the critical control application could be increased. But then, most certainly, another control application would fail. Another possibility is to decrease the sampling periods of the control applications. Then, however, the overall system load would

increase and the deadlines of the control applications (which are usually equal to the periods of the tasks executing them) would be much harder to satisfy by the real-time kernel. Additionally, the number of control applications that can be accommodated by a single ECU would decrease using such an approach. This is not acceptable, since this leads to bad hardware utilization and, thus, increased cost per functionality.

Consequently, the question that is addressed by this use case is the following: How can we systematically integrate several control applications onto the same ECU while ensuring good control quality, adherence to real-time constraints and system extensibility for future features?

Co-Design Approach

The main idea of this use case is to perform co-design between control engineering and real-time engineering. The starting point for this approach is the following inherent conflict of goals:

1. For control engineering, short sampling intervals are ideal, since they approximate best the idealized assumptions stated above. This leads to better control quality.
2. For the real-time behavior of the global system, long sampling intervals are ideal, since the same computations for the control applications are repeated less often. This leads to less processor load, translating into less resource sharing and making it easier for the real-time kernel to satisfy all deadlines. Additionally, more slack for additional functionality is available on the ECU.

The possible maximum and minimum sampling rates can be determined analytically as follows. The minimum sampling rate for a control application depends on the time constants of the plant that is controlled. A rule of thumb is that the sampling rate of the controller should be 10 times smaller than the smallest time constant of the plant. But even higher sampling rates may be required if the Eigenvalues of the plant are placed badly, or if the plant is nonlinear. A control engineer may use an analysis model in MATLAB/Simulink to investigate the necessary minimum the sampling rate.

The maximum sampling rate for a control application, here called scheduling threshold, is bounded either

- by the computational capacity of the ECU executing the control application, meaning that it shall not be overloaded, or
- by real-time constraints of other tasks, that are violated due to the increased interference resulting from the higher sampling rate of the control application.

This maximum sampling rate can, for instance, be determined using scheduling analysis techniques. Please note that it might not make sense for all kinds of control applications to fully exploit the scheduling threshold, e.g. when the controlled system (aka plant) is changing its state relatively slowly like, for instance, in cabin temperature control. More precisely, in cases where the scheduling threshold permits a shorter sampling rate than the time constant of the plant, i.e. the response time of the plant to a control input, the time constant must be taken as lower bound for the sought-after ideal sampling rate rather than the scheduling threshold.

Figure 21 shows the minimum and the maximum sampling rates that span the search space, called co-design area, for the trade-off exploration between good control performance and good real-time behavior.

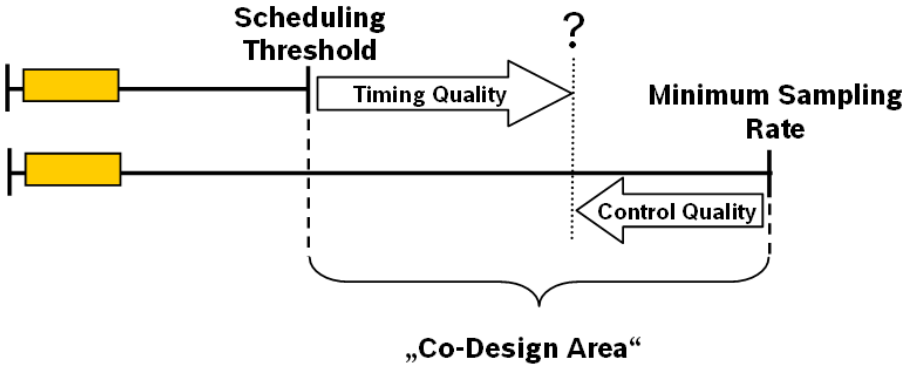

Figure 21 - Co-design area between control engineering and real-time engineering

Resource Constrained Control Application Development

In order to systematically explore the above explained design space a new development paradigm for control applications is needed.

In the current state-of-practice, the controller design and implementation is done independently of timing and integration considerations. The system integrator takes the resulting software implementations, and must cope with the overall system's timing behavior. Practically, he has no degrees of freedom, since sampling rates, interdependencies between processes, etc. are inherently incorporated in the controller designs, and are, thus, immutable.

This use case stresses that the overall timing behavior of a system, comprising of several control applications, shall be considered as explicit design goal.

To do so, a feedback loop between control design and system integration, taking into account the overall system's timing behavior, is necessary. Only then it is possible to choose controller designs offering at the same time

1. good control performance, and
2. good overall system timing behavior offering slack for future functionalities.

Mapping to Generic Methodology Pattern

The co-design approach between control engineering and real-time engineering targeted by this use case is performed during functional design which is located at Design level.

Since some controller sampling rates may already be chosen at Analysis level, *Controller Timing Requirements* determined at Analysis level are taken as input for this use case. Thereby, the sampling rates are typically not fixed. It is usually rather the case that ranges for possible sampling rates are specified. The exact choice is then taken at Design level based on the overall timing behavior of the integrated system.

Figure 22 shows the detailed methodology instance. In the following, the different activities are explained.

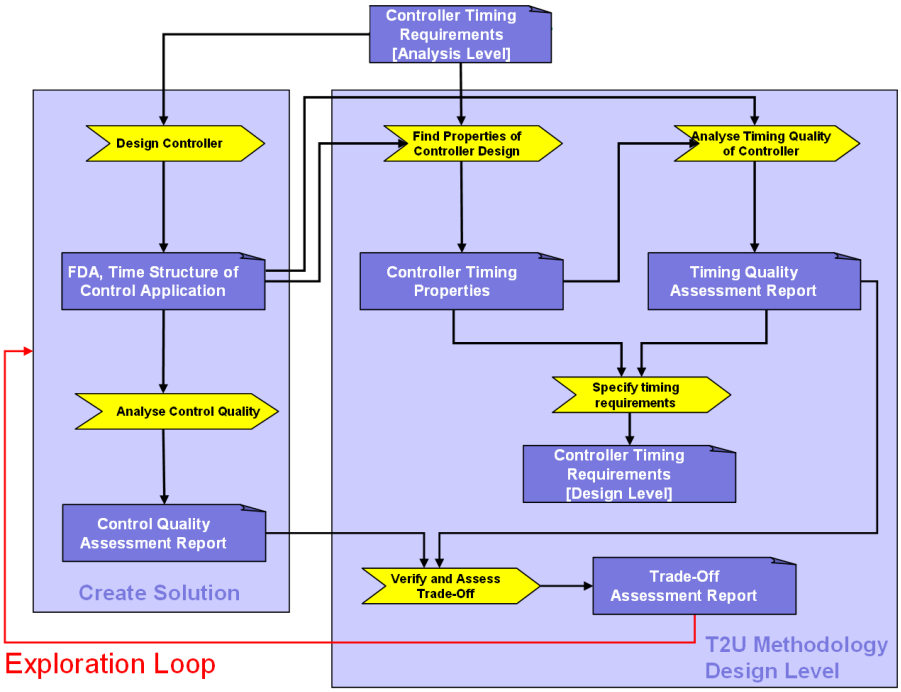

Figure 22 - Application of TIMMO-2-USE Generic Methodology Pattern

Design Controller

The task *Design Controller* consists of splitting the different calculations that are necessary for performing the control task into

1. N different execution units (i.e. processes) $eu_{1, ..., } eu_N$ with
2. different repetition patterns (i.e. periods) $p_{1,..., } p_N$.

This activity is called *control design through time structuring*. For the trade-off analysis between control quality and timing quality, the time structures of all considered control applications represent the search space that shall be explored.

In the following the simplest case of control design through time structuring, assuming linear systems as shown in Figure 23, is explained.

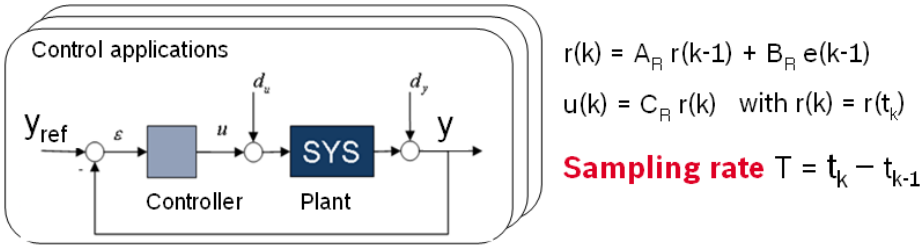

Figure 23 - Example Linear Control

We assume that N control applications shall be developed and integrated onto the same ECU.

For the simplest case of time structuring we assume that all necessary calculations, i.e. the internal controller state r(k) and the controller output u(k), for each of the N control applications are performed in single processes that are repeated with fixed sampling rates.

In the subsequent steps of the methodology, an exploration loop is implemented that explores the trade-off formulated above:

What are the optimal sampling rates for the N given control applications such that their individual control quality is sufficient AND such that the overall timing behavior of the system is satisfactory?

Analyze Control Quality

This activity consists in evaluating the control quality of the created solutions for the control application. Control quality metrics can either quantify the transient performance or the steady state performance. A possible measure for transient performance is *Peak Overshot*[3].

A possible steady-state control quality metric can be defined using the integral cost function of the tracking error *e,* denoting the difference between the desired plant behavior y_{ref} and the actual plant behavior y, at a given point in time *t*:

$$quality = \int_0^\infty e(t)^2 dt \quad ,where \quad e = y_{ref} - y$$

Find Properties of Controller Design

This task consists in determining all timing properties that are necessary to analyze the dynamic timing behavior of the overall system implementing the control applications. The required timing properties depend on the time structuring approach chosen during the task *Design Controller*.

The minimum set of required timing properties for each considered control applications consists of:

1. The estimated (worst-case) execution times for each of the execution units
2. The sampling rates for each of the execution units
3. Mapping of the execution units into the runtime systems (task mapping)
4. Precedence relations and data flow.

Analyze Timing Quality of Controller

During this task, the overall timing behavior of the ECU executing the implemented control applications is evaluated. Of course, all timing constraints of the system must be satisfied. However, for the trade-off analysis between control quality and timing quality performed during the task *Verify and Assess Trade-off*, we want to apply more

[3] Compare R. C. Dorf and R. H. Bishop. Modern Control Systems. Addison Wesley, 1995.

sophisticated timing metrics expressing the extensibility of the overall system. A simple possible metric for extensibility is the load situation on the ECU. More expressive metrics are based on sensitivity analysis[4].

Specify Timing Requirements

This task consists in defining timing requirements for the implementation phase. These mainly consist in upper bounds for execution times and finally chosen sampling rates translating into task mapping constraints. Additionally, precedence constraints may be necessary. Please note that this task is only performed if the task *Verify and Assess Trade-off* has been successfully completed. Only then it makes sense to proceed to the implementation level.

Verify and Assess Trade-off

In the first step of this task it is verified whether all real-time constraints on the ECU executing the considered control applications are satisfied. If this is not the case, the current solutions must be modified, and the methodology iterates back to the task *Design Controller.*

In the second step, the actual trade-off analysis between the control quality metrics calculated during the task *Analyze Control Quality* and the timing quality metrics calculated during the task *Analyze Timing Quality of Controller* is performed.

If the control quality or the timing behavior is not satisfactory, the methodology iterates back to the task *Design Controller*. More precisely, another solution (i.e. time structure for one or several control applications) that offers a more suitable trade-off is searched.

[4] 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.

5 Conclusion & Outlook for the second year

During the first year work package 4 created a concept that puts the results of WP2 (Timing Augmented Description Language) and WP3 (Algorithms & Tools) into the context of the software development process in the automotive industry.

Therefore, work package 4 developed a Generic Timing Methodology (GMP) that is now used as baseline for the technical work within TIMMO-2-USE.

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.

Based on the GMP, work package 4 developed methodology instances that are specialized for the use-cases described by D1. Currently covered use-cases include:

- Integrate a Software Component into an existing System
- Develop Control Applications
- Specify Time Budgets in Collaborative Development Settings
- Specify Synchronization Constraints

In the second year of TIMMO-2-USE the GMP and the specific methodology instances for the use cases will be refined and extended according to the findings during validation in WP5. In parallel additional specific methodology instances for currently not addressed use cases will be developed.

Additionally, possible tool support for treating timing aspects in the automotive software development process will be highlighted. In particular, the application of WCET analysis techniques and simulation will be in focus.

6 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/

7 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.