Tool Support for Seamless System Development based on AUTOSAR Timing Extensions

Oliver Scheickl, Christoph Ainhauser
BMW Car IT GmbH

Peter Gliwa
Gliwa GmbH

Abstract

Many software-based functions in modern cars have strict timing constraints. Due to the ever growing number of such functions and the trend towards higher integration, the development of automotive systems is becoming more complex. The industry has realised that model-based software engineering approaches are required to cope with this increasing complexity and, among others, to ensure the fulfilment of the timing constraints. Therefore, different kinds of tools are required for the various engineering steps during the whole timing-aware development process.

For AUTOSAR systems, the tooling for timing-aware development is not yet fully satisfying. First, most of the existing tools focus only on a specific step of the engineering process. Second, if at all, today's interaction of tools is often realised as two-sided island solutions based on proprietary import and export interfaces. Tool interaction however is required to allow specific tools for single engineering steps to be integrated into a seamless tool support. To our best knowledge, the existing approaches are neither based on a common infrastructure nor on a standardised timing model. Finally, appropriate tools for the specification of AUTOSAR timing requirements are missing.

This paper proposes a tool support for seamless timing specification, analysis and verification that overcomes the current challenges. The solution integrates a) Artime, a textual editor to specify timing requirements, b) Gliwa's timing suite ***T1***, a target timing measurement tool to gather timing guarantees and c) Artip, a graphical timing verification tool, which compares requirements and guarantees and visualises the results. Our approach integrates all tools necessary for a seamless model-based system development based on the standardised timing model offered by the AUTOSAR Timing Extensions. We implemented Artime, Artip and the integration of Gliwa's timing tool suite ***T1*** as plug-ins for Artop [6].

We applied this approach in a series development project at BMW Group. In the project we showed how our proposed tool support for seamless system development improves the application of the AUTOSAR Timing Extensions in practice. The tool support, its embedding in the engineering process, and the results of the application are presented in this paper.

Keywords: AUTOSAR, Timing Extensions, Timing Model, Timing Analysis, Tool Support, Practical Use, Artop, Artime, T1

1 Introduction

Modern car functions, realized by electronics and software, have driven many innovations in the automotive industry. Meanwhile, complex functions are present in many small-sized premium cars as well and define the new standard of user experience and driver assistance for the near future. The functions are typically provided by interactive distributed real-time systems. The development of these vehicle electrical systems is a complex task (see for example [2, 7]).

Some of the car functions must be provided within given time bounds to function properly. Timing constraints are typically driven by physics (e.g. repetition rate of control loops) or derived from end user requirements (e.g. immediate reaction to user input). As many parts of modern vehicle electrical systems are software-based, correct timing behaviour of the software is vital to the fulfilment of timing constraints.

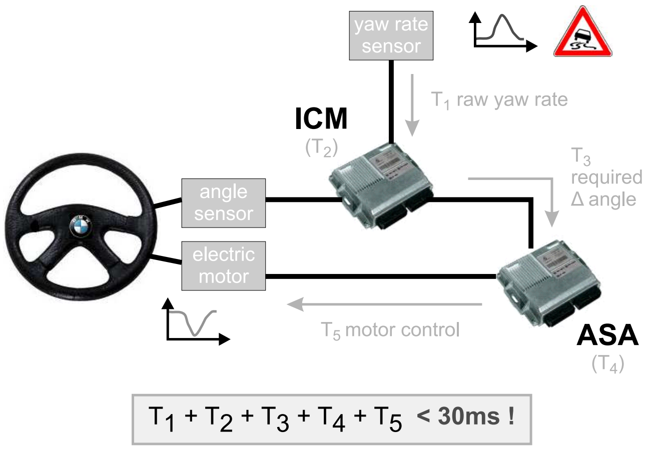


Figure 1: Example for timing constraints derived from physics

For a better understanding, we provide an example from the chassis domain in a car. The example function is an active steering, which is capable of overlaying the drivers steering action with an additional or subtractive steering angle. This allows changing the steering ratio gradually according to the vehicle speed and also implements a number of stability enhancing functions. Figure 1 illustrates the data path and the separate time segments which add up to the total end-to-end time. With the vehicle dynamics model of the car and the active steering function on his mind, the functional developer defines a minimum reaction time for the complete chain, here $30ms$.

The example includes sensors (yaw rate, angle), buses (CAN, Flexray), ECUs[1] (ICM[2], ASA[3]) and actuators (motor driver).

The angle sensor and its CAN connection to the ICM is also affected by the timing constraints. For the sake of simplicity it is not considered in this paper.

T_1 and T_5 in the example are mostly hardware dependent and typically produce a constant delay in the path. T_3 reflects the communication overhead produced by the Flexray bus – depending on the Flexray configuration. T_2 and T_4 represent the time spent "within" an ECU for receiving, processing and transmitting data. This paper focuses on such ECU internal timing. It is influenced by a number of things: scheduling strategy, execution times, blocking times, operating system configuration, communication strategies, communication buffer sizes and synchronisation effects. Flexray communication typically introduces synchronisation effects since the ECU internal schedule with its periodical tasks based on its ECU internal clock source has to be synchronised with the independent Flexray timing. The use case described later in Sec. 5 for example includes timing constraints related to the Flexray synchronisation.

It is important to a) be aware of the system's timing constraints, b) clearly document and specify the constraints and c) verify that these constraints are met in the production software. Documentation and verification of timing constraints become especially important for the distributed development process typical of the automotive industry. For a single ECU, suppliers and car manufacturers contribute parts of the software as black-boxes. More and more, these black-boxes are AUTOSAR software components with well-defined interfaces. When integrated together in one ECU, these components influence the timing on a system level with all its preemptions and concurrent run-time requirements [8, 7].

Since AUTOSAR Release 4.0 (Dec. 2009), timing behaviour can be described as so-called *Timing Extensions* for software components, compositions, basic software and for the entire system [1]. The resulting specifications can be exchanged between development teams either as timing requirements or as timing guarantees. We outline our timing development methodology in Sec. 2.

During the practical application of the AUTOSAR Timing Extensions we realized that a successful roll-out in production projects is largely determined by the availability of related tools. Therefore, we implemented and integrated appropriate tools, as depicted in Fig. 2. The goal of our approach is to enable *systematic* and *highly automated* verification of timing constraints of an AUTOSAR system. To achieve this, the following building blocks are required:

- The origin of timing requirements for subsystems are timing constraints of functions. In Sec. 2 we outline **Timex**, a model to capture such function timing constraints. In this paper, however, we concentrate on the already derived timing requirements within the scope of AUTOSAR.

- A formal timing model is the basis as well as the common exchange format of all tools. The standardized **AUTOSAR Timing Extensions** provide this timing model.

- Specification tools are needed to make the model usable for developers and system designers. Therefore we developed a textual timing editor called **Artime** to specify timing requirements. Artime is a plug-in for Artop [6].

- Measurement based timing analysis tools are used to gather timing behaviour information from a given embedded system at run-time. The results are different kinds of timing properties which are then interpreted as guarantees. In our tool chain, we use the timing analysis tool ***T1*** [4].

- We developed a simple verification tool called **Artip** that compares timing requirements and timing guarantees. The system designer and integrator can use it to visualize the fulfilment of the systems timing requirements by the according guarantees.

- For all the tools listed above, a common integration platform is required for a successful and smooth application. We employ **Artop** as this tool platform. Artime and Artip are realized as Artop plug-ins. We implemented an additional, tool-specific Artop plug-in, called **ArT1**, to connect the Artop environment to the separate ***T1*** timing suite.

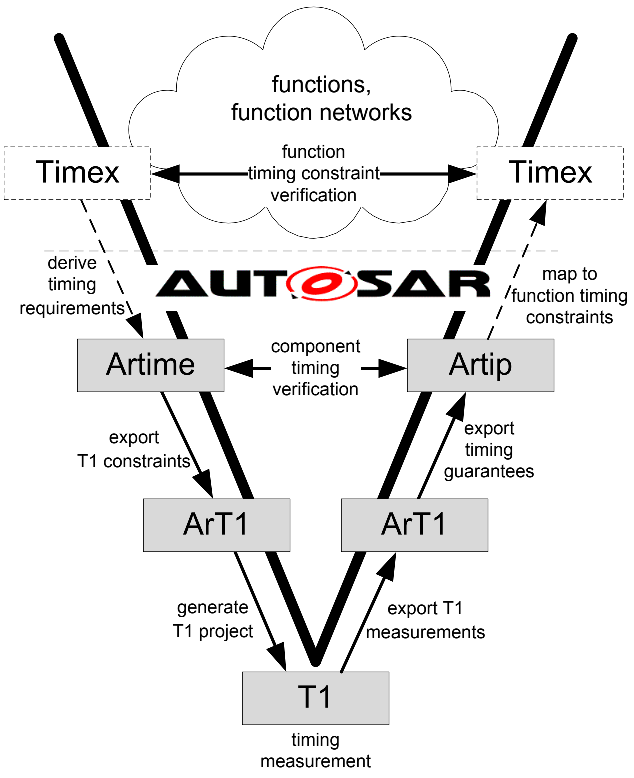


Figure 2: The tools at their position in the V model.

[1]**E**mbedded **C**ontrol **U**nit
[2]**I**ntegrated **C**hassis **M**odule
[3]**A**ktuator **S**teuergerät **A**ktivlenkung = actuator control unit active steering

Our approach focuses solely on the abstraction levels covered by AUTOSAR. In contrast to TIMMO-2-USE [10], the higher levels of abstraction (e.g. the Vehicle Level, the Analysis Level and the Design Level as mentioned in [3]) are not in this scope. In addition, we do not consider any functional requirements, which are not related to timing since many established tools and methodologies already exist here.

Current requirement specifications in the ECU development typically formulate timing requirements in a rather unspecific manner in natural language. Further, development experience in mass production projects has shown that the CPU load (a typical rather abstract timing requirement) can function as a very easy to handle "summary value" but is not sufficient. The AUTOSAR standard [1] and the research community [7] provide techniques to specify timing requirements precisely and a sophisticated distributed development methodology.

Our proposed tool support for seamless system development shows how the AUTOSAR Timing Extensions standard can be rolled out in series car manufacturing and brings theoretical real-time system development concepts to practice.

2 Timing Constraints in Distributed Development

Traditionally, and supported by the AUTOSAR development partnership, the automotive industry is working in a distributed development process. Subsystems are typically developed by suppliers according to the car manufacturer's specification of the subsystem's desired functionality. The car manufacturer has the role of a system designer and system integrator in that distributed development process. The suppliers usually only exchange information with the car manufacturer and not among each other, although they collaboratively develop the same automotive system. The car manufacturer, who has a view on the entire system, knows all timing constraints of the system and must ensure their fulfilment after all subsystems have been integrated into the overall system.

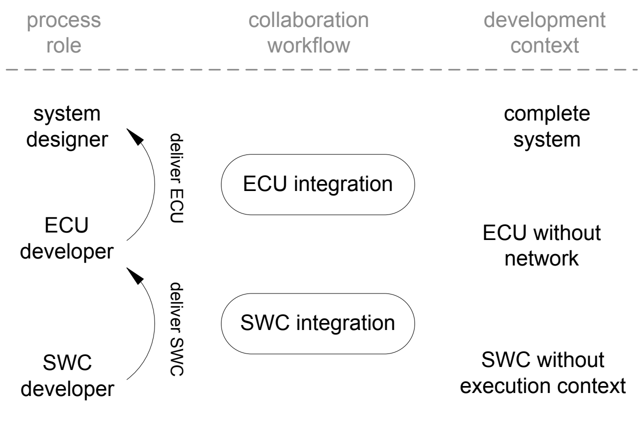


Figure 3: Roles, collaboration workflows and development contexts in distributed development of automotive real-time systems.

Figure 3 summarises the roles and collaboration scenarios of distributed development of automotive systems. A *SWC*[4] *developer* supplies a SWC. This role is rather new to the automotive industry and mainly driven by AUTOSAR. Software components are integrated into an ECU by an *ECU developer*, who in turn delivers his ECU to the *system designer*. If an ECU developer integrates a third party SWC in his ECU, we call this collaboration workflow *SWC integration*. If a system designer integrates a third party ECU in his system, we call this collaboration workflow *ECU integration*.

Each single supplier develops and bears for his subsystem and its timing requirements. However, the subsystem implementations also influence the timing behaviour of other subsystems. Thereby they influence the timing behaviour of the entire system. Thus, every subsystem takes part in fulfilling the overall timing constraints of the system. As the suppliers of the subsystems do not collaborate directly with each other, but only with the system designer, the system designer must control the timing behaviour of the subsystems by coordinated subsystem timing requirements. These **subsystem timing requirements** must be derived from the given **timing constraints of the system**.

In [7] we propose a timing model and a methodology for distributed development of automotive real-time systems. The work drives the following main ideas: The origin of timing constraints are the functions of the system. Therefore these constraints are the basis for our approach. We call them *function-triggered* timing constraints. These timing constraints are independent of the realization of the functions using hardware and software. Many car functions have such constraints. They typically stem from physics and car safety (e.g. sample rates in the chassis domain) or from customer requirements (e.g. tolerated end-to-end latencies for function activation). The function-triggered timing constraints must be identified and specified by the system designer using an appropriate timing model. We developed such a model, called TIMEX. Thereafter, the overall system is decomposed into subsystems. Subsystems are SWCs, ECUs or communication busses. SWCs and ECUs are typical subsystems provided by suppliers. The busses are typically developed (i.e. configured or defined) by the system designer. All subsystems are developed by several different suppliers (or also the system designer itself) and integrated into one system, see Fig. 3. This means that so-called timing requirements for subsystems must be derived from the function-triggered timing constraints of the system functions. Note the difference in the meaning of a timing constraint (for functions, independent from the realization of the functions) and timing requirements (for subsystems, derived from the constraints) in our nomenclature. The resulting implementation of a subsystem shall be accompanied by *timing guarantees* of its timing behaviour to indicate how the timing requirements are fulfilled. The subsystem structure as well as the timing requirements and guarantees of the subsystems are also modeled using TIMEX.

[4]In AUTOSAR, a **S**oft**w**are **C**omponent is a collection of software functions which implement a certain functionality

The derivation of subsystem timing requirements is performed in a way such that they can be verified independently from each other (no more timing dependencies between subsystems). The goal is to ensure the system's correct timing behaviour just by comparing each subsystem timing requirements with its timing guarantees. The system timing constraints shall be fulfilled if all requirements are fulfilled by their guarantee. Therefore several rules are applied for the derivation process, which ensure this subsystem decoupling. Basically, we define mathematical relations between functionally associated requirements, such that in a worst case still the function's timing constraints are fulfilled. An example therefore is that the sum of the maximum latency requirements along a data path must be less than or equal to the maximum latency of the entire path, which represents the function (see the example in Sec. 1).

In an iterative development process it can happen that a requirement is not fulfilled by its guarantee, i.e. by the implementation. In that case the timing correctness of the system cannot be ensured. However, this does not necessarily mean the function development failed. Often such non-fulfilments stem from wrong timing configurations (priorities, offsets or periods). By supervising the timing requirements and providing traces visualising the run time situation around the violation, the timing suite ***T1*** focuses on solving such timing problems.

The system integrator has an overview of all requirements and guarantees of the subsystems. In our work in [7] we propose a constraint logic programming approach to model the problem of finding an appropriate new set of timing requirements, based on the current set of guarantees. The approach is based on the idea that unused spare time can, in some cases, be redistributed to resolve unfulfilled timing requirements. The new timing requirements are considered by the subsystem suppliers by new appropriate timing configurations, if possible. In [7] we define a number of rules based on predicate logic that control in which cases requirements can be changed, or adapted to the new situation.

The timing requirements that are specified with Artime in this work are already derived from function-triggered timing constraints. The derivation process is outside the scope of this paper.

3 AUTOSAR Timing Extensions

Since Release 4.0, the AUTOSAR Timing Extensions [1] are part of the AUTOSAR standard. It represents a timing model as formalisation basis of relevant timing dependencies and according timing requirements and guarantees in an AUTOSAR system.

The name of the model is driven by the fact that it extends the *AUTOSAR Software Component Template* and *System Template* and allows to selectively enrich the specification with timing information where needed.

The Timing Extensions concept is based on the fundamental elements *event*, *event chain* and *timing requirements and guarantees*. In the following, these elements are described in more detail.

3.1 Timing views

According to the AUTOSAR methodology, the development of an AUTOSAR system undergoes different phases. The AUTOSAR Timing Extensions support this methodology and provide five different timing views to an AUTOSAR system: VfbTiming, SwcTiming, SystemTiming, BswModuleTiming and EcuTiming.

SwcTiming allows for the definition of a timing extension for a concrete software component. Thus, all the timing requirements specified in this view must be fulfilled by the target, independent in which system context the respective software component is applied.

SystemTiming is used to define timing extensions for a specific system. In this case, system relevant properties like the topology, software component deployment and signal mapping are known. Thus, timing requirements specified in this view are specific for this system.

For a more detailed description of all the different timing views, please refer to [1].

Considering the example described in Sec. 1, different timing views can be applied during the different development phases. First, for the the complete function a VfbTiming is defined, containing the top-level timing requirements like for example the end-to-end reaction time requirement of $30ms$ as mentioned in the example. This view is typically specified by the *system designer* as mentioned in Sec. 2. Afterwards, a SystemTiming is derived from the VfbTiming and includes a system specific refinement of the original requirement at Vfb[5] level. In the example, the latency requirement is divided into the segments T1 to T5 by taking the topology and deployment decisions into account. This is still the task of the *system designer*. In the next development step, an EcuTiming is extracted for each involved ECU and exchanged between *system designer* and *ECU developer* during the collaboration workflow *ECU integration* as mentioned in Sec. 2.

3.2 Events and Event Chains

Events are used to describe system behaviour which can be observed during runtime of the system (see Fig. 4). The activation of an AUTOSAR RunnableEntity or the reception of an AUTOSAR VariableDataPrototype at the receiver port of a SWC are examples of such observable system behaviours. It is important to note that the specification itself is independent of the actual occurrences of these observable events. The event describes the required behaviour, the occurrences of that behaviour can then be observed at runtime.

Events can be correlated with *event chains*. An event chain specifies a causal relationship between the associated stimulus event and response event. In addition, chains can be hierarchically decomposed into segments (see Fig. 4).

The AUTOSAR Timing Extensions provide a predefined set of event types and restrict their usage for the different views mentioned above. Figure 5 shows two examples of predefined event types. An event of type *TDEventVariableDataProtoype* is used to describe the point

[5]Virtual Function Bus

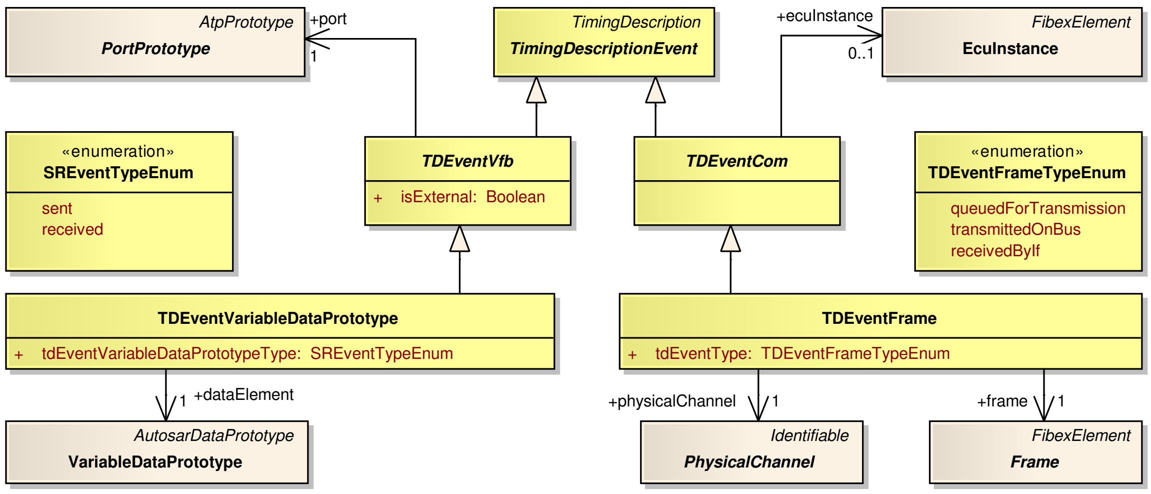


Figure 5: Exemplary event types [1]

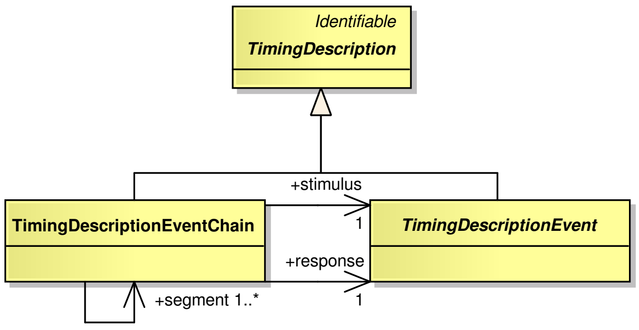


Figure 4: The concept of events and event chains [1]

in time when the referenced VariableDataProtoype has been sent or received by the associated software component. An event of type *TDEventFrame* is used to describe the point in time when the referenced frame has been queued for transmission or received by the associated ECU instance.

The full list of event types and a more detailed description of the event chain concept is available in the official AUTOSAR specification [1].

Considering the example described in Sec. 1, an event chain must be defined from the sensor data acquisition to the actuator access. Thus, the stimulus event targets to the VariableDataPrototype of the sensor software component which is used to get the sensor data from the ECU hardware abstraction layer. The response event targets to the VariableDataProtoype of the actuator software component which is used to communicate the target value to the ECU hardware abstraction layer. Furthermore, the whole event chain is refined by sub chains. The sub chains represent T1 to T5 as shown in Fig. 1.

3.3 Timing Requirements and Guarantees

Timing requirements and guarantees are the central elements of the AUTOSAR Timing Extensions. They are used to restrict or respectively assure the timing behaviour of the referenced system context. Depending on the type, timing requirements and guarantees use events, event chains or concrete RunnableEntities to specify the scoped system context. The AUTOSAR standard thus supports the methodology for distributed development of automotive real-time systems described in Sec. 2.

In the use case presented in this paper (see Sec. 5), we focus on the requirement types *latency*, *execution order* and *execution time*.

A *latency* requirement bounds the time duration between the occurrences of the stimulus and the response events of the associated event chain. With an *execution order* requirement, the requested control flow regarding the activation of RunnableEntities can be restricted. *Execution time* requirements specify the allowed lower and upper time bounds for the execution of RunnableEntities and can be either of type *net* (do not consider interruption and external calls during execution) or *gross* (do not consider interruption for the determination of the execution time).

The full list of timing requirements and guarantees is described in the official AUTOSAR specification [1].

Considering the example described in Sec. 1, a latency requirement of $30ms$ must be defined for the event chain from the sensor to the actuator (see exemplary event chain in Sec. 3.2). The requirement is of type *Reaction*, because it bounds the time delay from the perspective of the stimulus event: every time the stimulus event occurs, a reaction is expected to occur within $30ms$. In a next step, when more details about the system context are known (e.g. topology, deployment), the end-to-end latency requirement is refined into the several segments T1 to T5. As described in Sec. 2, the refinement must ensure that the sum of the requirements for the segments fulfills the higher level constraint for the whole end-to-end chain.

3.4 Application Experience

As described above, our approach concentrates on a specific feature subset of the AUTOSAR Timing Extensions.

We assume that this subset is sufficient for most of the envisioned real world applications. The additional features are designed for specific environment conditions.

The original AUTOSAR Timing Extensions are derived from a pure top-down development process as envisioned by the AUTOSAR methodology. However, real world applications show that timing gets typically more relevant at later phases of the development process. In this case, timing requirements are not subsequently derived during system development, but identified and defined in the implementation and integration phase in a bottom-up manner. Thus, the AUTOSAR Timing Extensions must also provide the features requested in such constellations. During the realization of our approach we detected respective feature gaps (e.g. the definition of the *execution time* requirement). These gaps will be fixed in a future release of AUTOSAR.

4 Tools for Specification and Verification of Timing Constraints

The AUTOSAR Timing Extensions are designed to be the exchange format for timing tools in the AUTOSAR environment. As Fig. 6 shows, timing tools along the automotive E/E development process use the Timing Extensions to a) import the required input information (e.g. ArT1 imports the specified timing requirements) and b) export the offered output information (e.g. Artime exports the specified timing requirements).

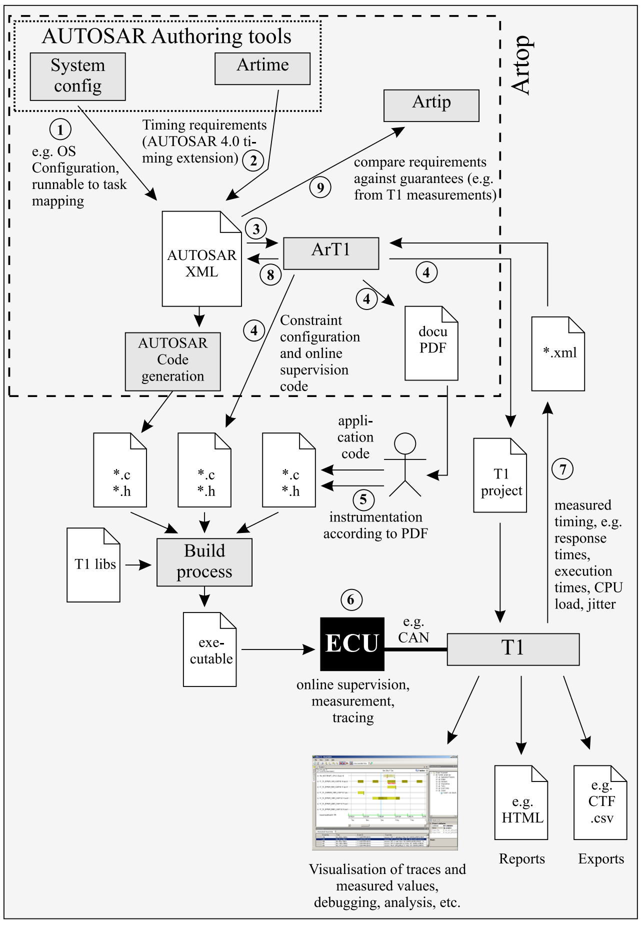

AUTOSAR Authoring tools
System config
Artime
Artop
Artip
1
e.g. OS Configuration, runnable to task mapping
Timing requirements (AUTOSAR 4.0 timing extension) 2
9
compare requirements against guarantees (e.g. from T1 measurements)
AUTOSAR XML
3
ArT1
8
4
4
docu PDF
4
Constraint configuration and online supervision code
AUTOSAR Code generation
*.xml
*.c *.h
*.c *.h
*.c *.h
application code
5
instrumentation according to PDF
T1 project
7
measured timing, e.g. response times, execution times, CPU load, jitter
T1 libs
Build process
6
exe-cutable
ECU
e.g. CAN
T1
online supervision, measurement, tracing
Visualisation of traces and measured values, debugging, analysis, etc.
e.g. HTML
Reports
e.g. CTF .csv
Exports

Figure 6: Tools and data-flow

For the several engineering steps along the development process, different features and kinds of timing tools are required. This paper proposes one possible tool support which covers all the required features during the whole development of an automotive ECU. In an early phase of the development, tool support is required for the specification of timing requirements (Artime, see Sec. 4.1). In a later phase, a timing analysis tool is required to gather information about the timing behaviour of the system (***T1***, see Sec. 4.2). In the end, verification is required in order to compare timing requirements and guarantees and to inform the developer about possible mismatches (Artip, see Sec. 4.3).

As our approach shows, different tools together compose a continuous tool support as an integrated tool chain. We use Artop, the AUTOSAR tool platform [6] as common basis for all the tool chain elements. On the one hand, the single tools can benefit from Artop, since it provides basic tooling features like the AUTOSAR meta model, model import/export features or workspace management. On the other hand, since Artop is based on the Eclipse Platform[6], all the different tools can be composed to an integrated development environment (IDE) to form an integrated tool chain. Artime and Artip are implemented as Artop plug-ins and are fully integrated. ***T1*** is a distinct tool platform but, using the plug-in ArT1 as shown in Fig. 6, ***T1*** can also be integrated into Artop and accomplish the required integrated tool chain.

4.1 Artime - Timing Specification Tool

The AUTOSAR Timing Language (Artime) is a textual language for the formal specification of timing requirements and guarantees using a well-defined textual syntax. The Artime formalism is compatible to the AUTOSAR Timing Extensions described in Sec. 3.

In 2011, Artime has been contributed to Artop as example language of the ARText framework[7]. ARText users profit from the easy-to-learn textual representation of an AUTOSAR model, since it improves the readability and understandability of the model. In addition, such a textual notation facilitates model management like version or change control.

Artime provides several features for the textual specification of timing requirements and guarantees. *Syntax highlighting* and *text completion* by providing proposals for keywords or model references are very useful features during this engineering step. In addition, the Artime *scoping* mechanism supports the developer in creating a valid and consistent specification by proposing context-aware model references.

Artime allows to specify most of the requirement types of the AUTOSAR Timing Extensions (see Sec. 3) and has already been used in series projects at BMW Group.

Timing specifications made using Artime are fully compatible with the formalism defined in the AUTOSAR Timing Extensions standard. This feature is used by Artime to make on-the-fly transformations from

[6]http://www.eclipse.org/platform/overview.php
[7]http://www.artop.org/artext/

Artime models to AUTOSAR standard models. The developer is then able to export his timing specification to the official AUTOSAR exchange format (XML) and vice versa. This enables even a mixed specification tool environment, consisting of Artime and other arbitrary specification tools that are also compatible to the AUTOSAR timing extensions.

In the following, some specification snippets are shown to illustrate the key language elements of Artime. Composed together, the code snippets formulate a complete and syntactically correct Artime example to specify a latency requirement for a RunnableEntity that is only valid in the scope of the ECU *ICM*.

Import - Import statements make the specification easier to read, since model elements can be referenced without qualifying them with the fully qualified name.

```
package icm.timingspec
import components.integration.IntComHdl.*
```

View - Select the desired Timing Extensions view that shall be created (e.g. EcuTiming) and define the respective scope (here ICM).

```
timing ICMTiming ecu System.ICM {
```

Events - Easily and quickly define timing events for given AUTOSAR model elements.

```
  runnable RunnableCom =
    IntComHdl_Behavior.IntComHdl_InputQM_66ms
    context compositions
            .System::blackbox
            .functionLayer.comHandler

  event IntCom_InputQM_Activated
   = RunnableCom::ACTIVATED

  event IntCom_InputQM_Terminated
   = RunnableCom::TERMINATED
```

Event Chains - Define event chains by relating events.

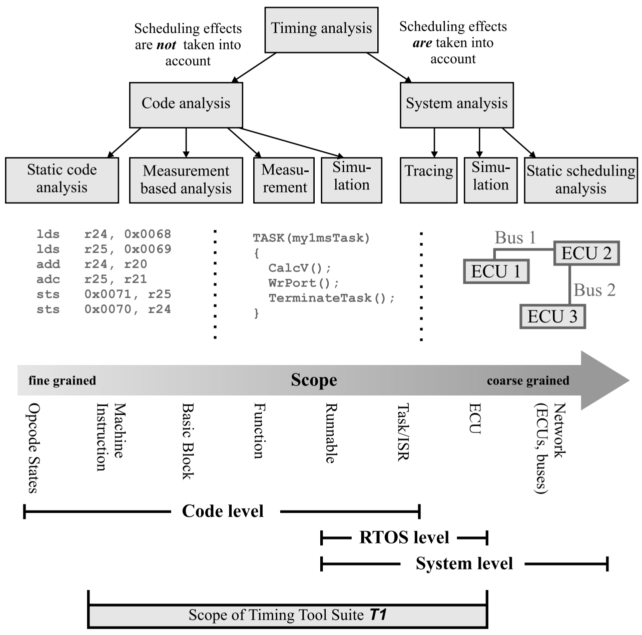
```
  eventchain IntCom_InputQM_Chain
    stimulus IntCom_InputQM_Activated
    response IntCom_InputQM_Terminated
```

Timing Constraints - Constrain the timing behaviour of AUTOSAR systems or subsystems.

```
  requirement latency IntCom_InputQM_Latency {
    scope IntCom_InputQM_Chain
    min 0 usec
    max 800 usec
  }
}
```

4.2 T1 - Timing Analysis Tool

Timing analysis tools as depicted in Fig. 7 analyse timing properties at two different levels: the code level and the system level [4], [9], [5]. At the code level, they deal with computation time, which is unaffected by preemption and depends only on the amount and complexity of code. At the system level, they deal with real time, which is affected by ECUs and buses. End to end timing requirements as described in the example in Sec. 1 are typical for the system level view. When considering a single ECU, two kinds of timing views are possible: the RTOS level and the code level. On the RTOS level, computation time, preemption from higher priority activity and blocking from lower priority activity are relevant. The code level finally explicitly excludes any preemptions/interruptions and considers an isolated function or code-fragment only. Figure 7 clearly indicates that the system-, RTOS and code level views are closely related to the granularity of the scope.



```
lds   r24, 0x0068
lds   r25, 0x0069
add   r24, r20
adc   r25, r21
sts   0x0071, r25
sts   0x0070, r24
```

```
TASK(my1msTask)
{
  CalcV();
  WrPort();
  TerminateTask();
}
```

Figure 7: System, RTOS and code level timing analysis overview

The V-model distinguishes two different phases of development: On the left side of the V, we have the early phase of design and construction of the system. On the right side of the V, we have the late phase of testing, verifying and validating the constructed system or system artifacts (see also Fig. 2).

Early phase code level analysis is performed by simulating the CPU executing the real software or by computing the worst-case execution time (WCET) of software by modelling the processor and software. At the late phase, computation time can be measured but preemption must either be disabled or taken into account when regarding the code level.

RTOS level analysis is performed in an early development phase by simulating both code level and preemption effects or by computing the worst-case response time (WCRT) using static scheduling analysers that model the operating system and the interaction of tasks and interrupts. In the late phase, response times can be directly measured.

Gliwa's ***T1*** is a timing suite based on measurement and tracing. The scheduling related events task activation, task/interrupt start, task termination and interrupt end are instrumented by ***T1*** as part of the build process. The instrumentation overhead is processor and project dependent but e.g. for the Infineon TriCore as less as 178ns per event at 180 MHz processor speed. This results in a CPU load required for tracing of less than 0.5% for typical 32 bit automotive application. For a screenshot of a ***T1*** trace, see Fig. 10.

T1 is typically used in a late phase of the V-model for verification purposes and "at the bottom" of the V-model for timing debugging. However, with the ***T1.delay*** component, early phase topics can be addressed: ***T1.delay*** injects additional processor utilisation and thus allows the user to simulate the load of future development. ***T1.delay*** can also measure the available headroom of each task/interrupt for a given software.

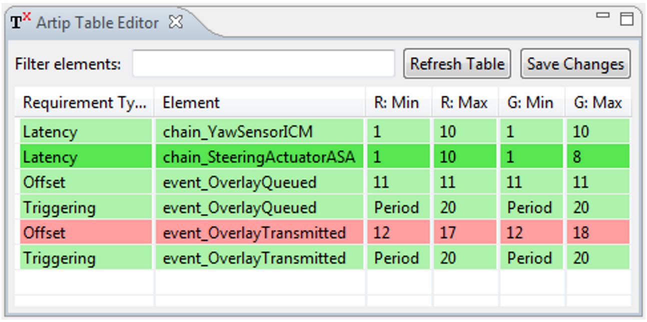
T1 includes full awareness of preemption, allowing it to perform analysis at both the code level and the RTOS level, see also bottom of Fig. 7.

Tracing is the only technique that allows observation and debugging of actual timing defects. Many timing problems are easily solved once they are understood. Visualisation, as provided by ***T1***, is a highly efficient way to understand timing behaviour, exploiting the enormous capacity of the human brain to take in visual information. Thus, ***T1*** has been successfully used in more than 70 mass production automotive projects.

4.3 Artip - Timing Verification Tool

The ARtop TIming comPare (Artip) tool provides capabilities to compare timing requirements and guarantees within an AUTOSAR model, as defined by the AUTOSAR Timing Extensions [1].

Its primary goal is to compare *timing requirements* against *timing guarantees*. If for instance a latency timing requirement for a certain event chain is not fulfilled by the respective timing guarantee, Artip visualises this fact and informs the developer about the wrong behaviour. This simple feature is very important during system development, since it emphasises possible timing defects and alerts the developer visually.



Filter elements:

Requirement Ty...	Element	R: Min	R: Max	G: Min	G: Max
Latency	chain_YawSensorICM	1	10	1	10
Latency	chain_SteeringActuatorASA	1	10	1	8
Offset	event_OverlayQueued	11	11	11	11
Triggering	event_OverlayQueued	Period	20	Period	20
Offset	event_OverlayTransmitted	12	17	12	18
Triggering	event_OverlayTransmitted	Period	20	Period	20

Figure 8: The example of this paper modeled using the AUTOSAR Timing Extensions and displayed by Artip

Figure 8 shows a requirement and guarantee snapshot of the example of Fig. 1 as it is displayed by the Artip table editor. Each requirement is displayed in a separate table row. Light green rows indicate exactly fulfilled requirements. Dark green rows indicate requirements that are "over-fulfilled" by their guarantee and offer spare time that can eventually be redistributed (see Sec. 2). Red rows indicate not fulfilled requirements in the current development state of the system.

Artip is meant to be more than just a requirement-guarantee comparator tool. According to our development plans, Artip shall provide extension points for other Artop plug-ins. This way, other tools can access the results of the comparison algorithms and integrate these results in their own algorithms or user interfaces. Furthermore, Artip will be designed to be extendable with additional compare features. For example, we plan to add in the future a mechanism to observe the change of guarantee values over time. Also additional other options to display comparison results are conceivable, such as graphical display methods or a textual representation similar to Artime.

5 Use Case

In this section, we show the application of our development methodology described in Sec. 2 and the proposed tool support for seamless timing development in a series development project at BMW Group which started development in 2011. In this project, BMW acts both as *software component developer* and as *system designer*, and the tier-1 supplier acts as *ECU developer*. See Sec. 2 for the definition of the different roles. Thus, the collaboration workflow of the project is *SWC integration*.

Compared to the theoretical considerations of Sec. 2 however, such a practical scenario with BMW collaborating in a double role, additional possibilities for timing requirements appear for the *SWC integration* collaboration workflow in this special case. In general, the ECU developer has the task to ensure that timing requirements of the integrated software component are fulfilled. In the use case project here, these SWC timing requirements are relative to the system's communication bus schedule. This is possible since BMW designs requirements for its own SWC in the role of SWC developer *and* defines the bus schedule in the role of system designer.

The ECU representing the use case is a component in the chassis domain and appropriate timing of the deployed software is fundamental for correct functional behaviour. BMW defines the system architecture – i.e. the network architecture with its buses and ECUs – but also develops certain SWCs. These SWCs are made available to the supplier who integrates these SWCs and creates a complete software, which then is flashed onto the ECU. At various points in the operating system schedule so called "black boxes" are present, which hold the runnables related to the BMW SWCs. Other projects use the term "container" instead of black box.

For the first time in a mass production project, BMW Group used Artime as part of the requirements document to specify the timing requirements. Already in the supplier evaluation phase, Artime specifications were available so that the competing suppliers were introduced into the future process of ECU timing verification. The timing requirements of the first versions of the software were checked and assured manually by the suppliers. This means, the Artime files were interpreted by a developer and then checked using existing timing measurement techniques.

The AUTOSAR Timing Extensions and Artime allowed a very precise description of timing requirements with clearly defined syntax and semantics. This approach eliminates misinterpretation of timing requirements with its textual description as part of the requirements documentation.

One of the timing requirements used in this project targets at software execution relative to Flexray communication. Flexray is a bus, which comes with its own clock source and requires strict timing behaviour because it allows transmission at dedicated time-slots according to the Flexray configuration only. These dedicated time-slots have to be respected by the software, in which at least partly the timing is based on the ECU's own clock source. As a result, all ECUs with a Flexray bus synchronise their software in one way or another with the Flexray cycle.

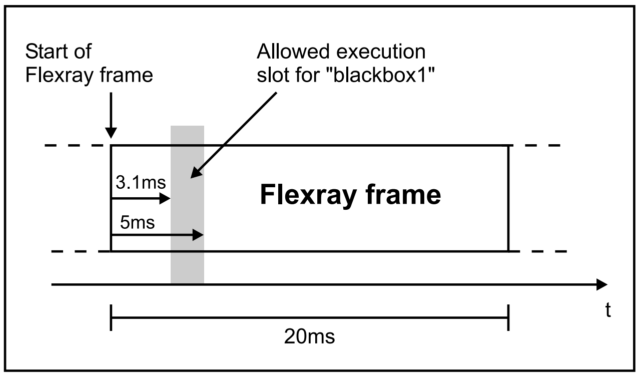


Figure 9: Example for a timing requirement relative to the Flexray schedule

Figure 9 visualises the Flexray-related timing requirement of the project. The code in the black box needs to get executed no earlier than 3.1ms and no later than 5ms after the Flexray cycle started. The corresponding Artime code to formalize the requirement is shown below.

```
runnable IntComHdl  =  components
                       .integration
                       .IntComHdl
                       .IntComHdl_Behavior
                       .IntComHdl_InputQM_66ms
                       context
                       compositions
                       .System::blackbox
                       .functionLayer.comHandler

event blackbox1Activated =IntComHdl::ACTIVATED
event blackbox1Terminated=IntComHdl::TERMINATED

/* 20ms */
requirement offset read20msStaticFrameStart {
    source flexRayStart
    target blackbox1Activated
    min 3100 usec
    max 5000 usec
}

requirement offset read20msStaticFrameStart {
    source flexRayStart
    target blackbox1Terminated
    min 3100 usec
    max 5000 usec
}
```

In the following the data- and information-flow between the OEM and the supplier will be explained. Numbers in circles refer to the arrows in Fig. 6.

The Artime file is sent to the supplier together with the general requirements document. The supplier can now open the Artime file with the Artime plug-in for Artop, which is capable of storing the timing requirements in the AUTOSAR XML format using the AUTOSAR Timing Extensions ②. The operating system has been previously configured ①. The supplier then uses the ArT1 plug-in to extract the RTOS configuration, the task-to-runnable-mapping and the timing requirements ③. This information is used by ArT1 in order to achieve two things ④. Firstly, the ***T1*** target code configuration with all the ***T1*** timing constraints to supervise is written to a C module `T1_config.c`. Secondly, user instructions are generated how to instrument all AUTOSAR events not related to scheduling ⑤. Scheduling events like task activation, start and termination are statically instrumented by ***T1*** already, so there is no need for manual instrumentation. Any other event like transmission of data xyz via RTE services requires the user to add a function call to `T1_TraceEvent(...)`. With a future improvement, this manual step will be replaced by hooking on to the generated RTE code, so that the instrumentation is fully automated. However, until this improvement is available the generated documentation gives clear instructions, which do not require special knowledge about ***T1*** or the RTE.

The supplier now builds the executable, downloads it to the ECU and uses ***T1*** to measure/trace and *supervise* the timing constraints ⑥. ***T1.cont*** is configured in a way that any violation will a) invoke a user (i.e. supplier) defined callback which enters an error into the error-buffer and b) optionally stop tracing with a predefined hold-off. This allows to debug the cause of the timing problem – the downloaded trace shows the run-time-situation around the problem. ***T1.scope*** visualises timing traces in a way which very much resembles an oscilloscope. Above a time axis, the task- and interrupt-states are indicated by different colours. Bright yellow for the *ready* state, dark yellow for the *running* state, green for the *waiting* state and no colour indicates the *terminated* state. Figure 10 shows the supervision of the timing requirement described above in ***T1.scope*** [8].

At this point, the link from the left side of the V-model/the requirements to the right side/the verification is complete. The following steps describe how the results can be fed back. At the time writing, this feedback path from the right side of the V-model to the left side has not been implemented yet. Thus, the following paragraph gives an outlook of the ArT1/ArTip features to come.

The measured timing properties like response times, core execution times, CPU utilisation, jitter, period etc. are exported into an XML file ⑦. ArT1 reads this file and allows the user to store measured timing properties as AUTOSAR timing *guarantees* in the AUTOSAR XML format ⑧. These can then easily be compared to the original *requirements* using ArTip ⑨, see also Sec. 4.3.

This use case demonstrates the application of our seamless tool support in the distributed development of an automotive ECU. The project did benefit a lot from this solution, since error-prone manual steps and unclear specification formats were replaced by interacting tools based on the AUTOSAR Timing Extensions as common timing model.

[8]Since the integration of the ***T1*** ↔ Artime coupling into the ECU project described has not been completed, Fig. 10 shows a constructed trace

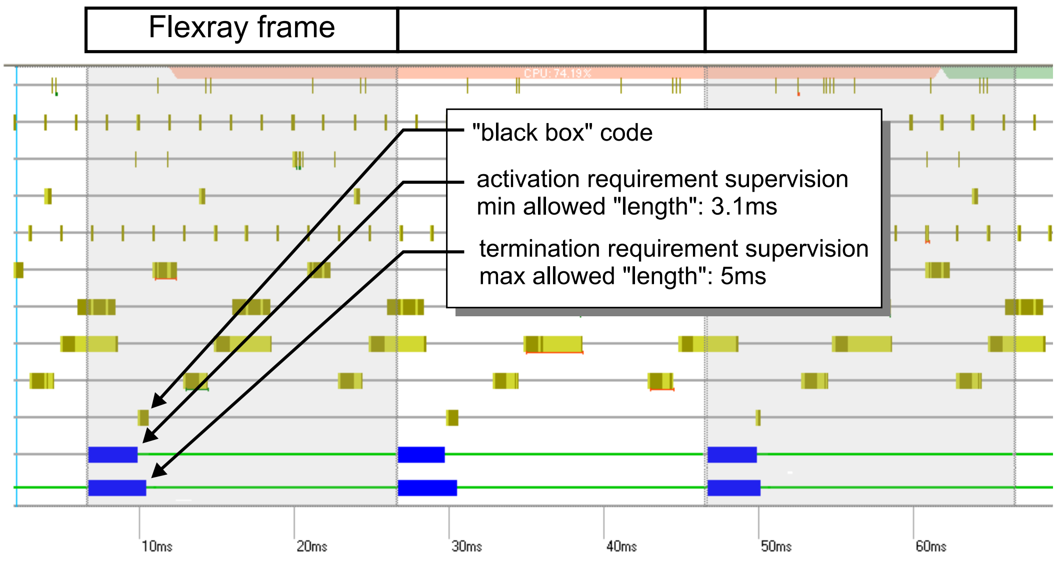

Flexray frame
CPU: 74.13%
"black box" code
activation requirement supervision min allowed "length": 3.1ms
termination requirement supervision max allowed "length": 5ms
10ms
20ms
30ms
40ms
50ms
60ms

Figure 10: T1 trace visualising the Flexray frames and timing requirements supervision

6 Conclusion

The approach introduced in this paper joins timing tools designed for different phases of the development process by enabling their interaction based on Artop [6] and the AUTOSAR Timing Extensions [1].

The implementation has been proven by a demonstrator and is currently being used in a mass production project. The tool support, especially Artime and its interaction with the timing measurement at the supplier of the ECU using ***T1***, leverages the practical application of the AUTOSAR Timing Extensions industry standard. We expect more predictable and verifiable timing of embedded software, improved collaboration between BMW Group and its suppliers and, at the same time, reduction of development costs by using such a seamless tool support and the versatile timing specification capabilities of the AUTOSAR Timing Extensions.

References

[1] AUTOSAR Development Partnership. Specification of Timing Extensions, Version 1.0.0, Release 4.0.1.

[2] M. Broy, I. H. Krüger, A. Pretschner, and C. Salzmann. Engineering Automotive Software. In *Proceedings of the IEEE*, volume 95, pages 356 – 373, Feb. 2007.

[3] EAST-ADL Association. EAST-ADL - An Architecture Description Language. http://www.east-adl.info/.

[4] P. Gliwa and A. Mayer. Application Performance Measurement on Automotive Microcontrollers. In *Proceedings of VDI congress "Elektronik im Kraftfahrzeug" (Baden-Baden)*, 2011.

[5] M. Jersak, P. Gliwa, and K. Richter. Planung und Absicherung der Echtzeitfähigkeit von Software und vernetzten Steuergeräten. In *Simulation und Test in der Funktions- und Softwareentwicklung für die Automobilelektronik III*, 2010.

[6] M. Rudorfer, C. Knüchel, S. Voget, S. Eberle, and A. Loyer. Artop - an Ecosystem Approach for Collaborative AUTOSAR Tool Development. In *Proceedings of ERTS2*, 2010.

[7] O. Scheickl. *Timing Constraints in Distributed Development of Automotive Real-time Systems.* PhD thesis, Technische Universität München, 2011.

[8] O. Scheickl, C. Ainhauser, and M. Rudorfer. Distributed Development of Automotive Real-time Systems based on Function-triggered Timing Constraints. In *Proceedings of ERTS2*, 2010.

[9] The ALL-TIMES Consortium. D4.4.2 Report on main project results. Technical Report Ver. 1.2, 2010.

[10] TIMMO-2-USE Consortium. TIMMO-2-USE - Mastering Timing Tools, Algorithms and Languages. http://timmo-2-use.org/pressreleases/31032011.pdf.