ITEA 2

INFORMATION TECHNOLOGY FOR EUROPEAN ADVANCEMENT

ITEA 2 - 09033

TIMMO-2-USE

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

Report type	Deliverable D9.3
Report name	State-of-the-Art Report
Report status	Consortium Confidential
Version number	Version 1.0
Date of preparation	2011-11-29

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 Group Trucks Technology

Advanced Technology & Research

Dept 6260, M2.7

405 08 Göteborg

Sweden

Tel.: +46 31 322 9949

Email: Daniel.B.Karlsson@volvo.com

Authors

Marie-Agnès Peraldi-Frati, INRIA
Ramin Tavakoli Kolagari, Volvo Technology
Kay Klobedanz, University Paderborn
Wolfgang Müller, University Paderborn
Sophie Quinton, TU Braunschweig
Matthias Hanke, TU Braunschweig
Frank Hagl, Continental Automotive GmbH
Stefan Kuntz, Continental Automotive GmbH
Morayo Adedjouma, DELPHI
Reinhold Heckmann, AbsInt
Ulrich Kiffmeier, dSPACE
Wendel Ramisch, INCHRON GmbH
Reinhold Heckmann, AbsInt
Nico Feiertag, SymtaVision
Björn Lisper, Mälardalen University
Ian Broster, Rapita Systems Ltd

Document History

Version	Date	Description
1.0	2011-11-29	First version.

Table of contents

1 Introduction

Purpose

The purpose of this document is to identify the existing models and tools available to solve problems arising in the domain of modeling timing requirements, constraints, and properties at different levels of the design process of automotive embedded systems.

Scope

The scope of this document is to investigate the different models and tools which might be used in the context of the TIMMO-2-USE (T2U) project. Different timing requirements, constraints, and properties must be handled for designing automotive embedded systems. This document aims first at defining a set of timing characteristics generally expressed in such systems. Depending on the level of abstraction (vehicle, analysis, design, implementation, operational) in the design, different notions of time can be handled by models and tools. In the first paragraph we provide a definition of each of these notions of time and we give practical examples of their use. The second paragraph identifies models capable of expressing these timing characteristics. Approaches for timing analysis are presented, including tools supporting them, as well as their potential use in the T2U project. Finally a third paragraph presents the results of projects in connection with the modeling of embedded systems for automotive.

Abbreviations and Acronyms

The table lists all abbreviations and acronyms used in this document.

Abbreviation Acronym	Description
T2U	TIMMO-2-USE

2 Definition and Example of Timing Characteristics

Time is a major concern in Computer Science and Engineering. However, each domain may have its own interpretation and modeling of time. F. Schreiber [1] has described several aspects of time and defines ontology for time in different domains of computers and their applications.

A first form of time is the one used in physical laws, and especially in mechanics. In computer science this time is often referred to as “physical time”, but its nature is above all mathematical.

In digital systems, this ideal time is approximated by circuits, called “clocks”, generating well defined “periodical” signals. This leads to a *discrete model of time*. Unfortunately, a digital system often needs several clocks. This raises the problem of clock synchronization [2].

Distributed systems, because of their spatial extension, experience the same problem to agree on a unique time reading. To address this issue, L. Lamport [3] has introduced the concept of *logical clock*. With logical clocks, partial ordering of events can be obtained without recourse to any physical “real” time. Improvements in logical clocks permit to characterize the causal relationship among events [4]. For performance evaluation or hard real-time property verification, a time model restricted to partial ordering of events is not enough. Synchronization with physical time becomes necessary.

For analysis purposes, the notion of *worst-case* time is essential. Traditional scheduling algorithms and analysis methods (e.g. for processor utilization or response time), provide deterministic timing guarantees (i.e., all task instances meet their deadline) which take into account worst-case timing information. However, these worst-case scenarios may be very rare in practice. Thus, *uncertain* time is introduced for soft real-time systems [28] and even for some hard real-time systems where the application allows for a given failure rate (e.g. the probability of missing a deadline could be as small as the probability of hardware failure).

2.1 Continuous Time

Definition

A continuous time is a varying quantity whose domain is an uncountable set (in general, an interval of the reals) unlike to the discrete time where the domain is countable (natural numbers). The continuous-time template generally results from measurements on a real physical system. The continuity of the time variable means that the signal value can be found at any arbitrary point in time; and this is usually expressed by differential equations.

For modeling, the continuous time can be designed by discrete time by sampling and quantization in such a way that the resulting model, expressed as a signal flowchart, is capable to reproduce the behavior of the continuous-time.

Levels of Abstraction

Automatic control – Analysis and Design level

Examples of Constraints, Properties

The system set its outputs with state=OFF until the command frame appearance.

2.2 Discrete Time

Definition

In digital systems, this continuous time is approximated by circuits, called “clocks”, generating well defined strictly periodical signals. This leads to a discrete model of time. A clock provides access to a discrete time base, and possibly to a dense (continuous) time base, but through a discrete time base. The instants of this time base correspond to "ticks" of the clock. A clock associates time values with instants of the time base. A DiscreteTimeBase represents an ordered discrete set of instants.

Levels of Abstraction

Analysis, Design and Implementation Level

Examples of Constraints, Properties

Duration, period, deadline, etc.

2.3 Logical respectively Multiform Time

Definition

In *logical time* (physical) time passing is represented by event occurrences; for instance a signal generated by an external device. Temporal distances between two occurrences of events linked to a logical clock are not necessarily strictly periodic. However, these events do not have any specific status that distinguishes them from other events. Hence, requirements linked to logical time may refer to statements such as “a task must complete before 10 ms”, and “a car must stop within 50 m”. Both statements express a deadline: “10 ms” for the former, and “50 m” for the latter. This is known as *Multiform Time*.

Levels of Abstraction

Requirement phase, Analysis, Design, and Implementation Level

Examples of Constraints, Properties

Function duration: knock control duration is 20° crank

Period: Every 180° crank knock sensor must be sampled

Deadline: knock control must be computed before 46° crank

The ECU is 20% faster than a standard ECU (e.g. in a certain context, execution times are given assuming a nominal speed of 100 MHz; Our CPU is then 120 MHz)

A car must stop within 50 m.

2.4 Uncertain Time

Definition

There are various ways of representing uncertainty of timing information. One can for example use a probability distribution to represent the execution time of a task, thus expressing how frequent a given execution time will be in practice. Another possibility is to use one single probability to express a *reasonable* case. For example, one can express like this that the execution time of a task is less than a given time with a given probability.

Besides, uncertainty can arise from different sources: from the activation pattern, e.g., in presence of a-periodic tasks, or from the execution time. This may lead to different representations of uncertainty in the same analysis.

Levels of Abstraction

Analysis, Design, and Implementation Level

Examples of Constraints, Properties

Function duration: the execution time of the task is given by the following distribution

Deadline:

- The message must be received within 25 ms in 90% of the cases.
- In any 20 consecutive deadlines the process must always meet at least 18 of them and it must never miss any 2 consecutively.

3 Modeling and Analysis of Timing Information

At the different levels of a design, multiple languages, models and tools can be used to handle timing properties. In the first section, this paragraph describes the capabilities of different models and languages – that could be potentially used in the TIMMO-2-USE project – with respect to time modeling. In the second section several approaches dealing with timing analysis are described, and tools supporting these approaches are mentioned. Special attention is paid to analysis capabilities (simulation or formal proof) of these tools. In the latter section only those analysis approaches and tools are described which are being investigated in the TIMMO-2-USE project. Other tools which are not under consideration in the project are mentioned in the first section associated to the models and languages.

3.1 Timing Modeling and Languages

The models and languages

3.1.1 UML Introduction

Type of Properties

Implicit time

Modeling Concepts

In UML [7], time is seldom part of the behavioral modeling, which is essentially untimed (by default, events are handled in the same order as they arrive in event handlers).

UML describes two kinds of behaviors: the intra-object behavior — the behavior occurring within structural entities — and the inter-object behavior, which deals with how structural entities communicate with each other [8].

The *CommonBehaviors* package defines the relationship between structure and behavior and the general properties of the behavior concept. A subpackage called *SimpleTime* adds metaclasses to represent time and duration, as well as actions to observe the passing of time. This is a very simple time model, not taking account of problems induced by distribution or by clock imperfections. In particular the UML *causality model*, which prescribes the dynamic evaluation mechanisms, does never refer to time (stamps). Instead, the UML specification document explicitly states that "*It is assumed that applications for which such characteristics are relevant will use a more sophisticated model of time provided by an appropriate profile*".

Analysis Capabilities

Not applicable concerning timing aspects.

Associated Tools

UML editors: Enterprise Architect [10], Papyrus [11], IBM Rational Rhapsody [12].

3.1.2 UML SPT Profile

Type of Properties

Discrete time, instant duration, clocks timers

Modeling Concepts

The UML Profile for Schedulability, Performance, and Time (SPT) [9] aimed at filling the lacks of UML in some key areas that are of particular concern to real-time system designers and developers. SPT introduces a quantifiable notion of time and resources. It annotates model elements with quantitative information related to time, information used for timeliness, performance, and schedulability analyses.

SPT only considers metric time, which makes implicit reference to physical time. It provides time-related concepts: concepts of instant and duration, concepts for modeling events in time and time-related stimuli. SPT also addresses modeling of timing mechanisms (clocks, timers), and timing services. But “time” here is only introduced through dedicated stereotype annotations that are not interpreted and given meaning as part of UML semantics. Instead, their purpose is to be understood by external analysis tools to perform schedulability or performance evaluation and after automatic translation from the UML model into a corresponding tool input format. SPT, which relies on UML 1.4, had to be aligned with.UML 2.1.

Analysis Capabilities

Not applicable

Associated Tools

Not applicable

3.1.3 UML SysML Profile

Type of Properties

Time with units, Probability distribution

Modeling Concepts

The OMG Systems Modeling Language [13] (OMG SysML™) is a general-purpose graphical modeling language for specifying, analyzing, designing, and verifying complex systems that may include hardware, software, information, personnel, procedures, and facilities. In particular, the language provides graphical representations with a semantic foundation for modeling system requirements, behavior, structure, and parametrics, which is used to integrate with other engineering analysis models.

Though SysML offers no specific support for Time, it extends UML in several ways: value property with units, and constraint block. A SysML value property defines a value with units, dimensions, and probability distribution.

A SysML constraint block contains equations expressing constraints between value properties. The usages of the constraints in an analysis context are represented in a parametric diagram.

Analysis Capabilities

Not applicable

Associated Tools

SysML editors: Enterprise Architect [10], Papyrus [11], IBM Rational Rhapsody [12].

3.1.4 UML MARTE Profile

Type of Properties

Continuous / discrete / multiform time

Modeling Concepts

Modeling and Analysis of Real-Time and Embedded Systems (MARTE) is a response to the OMG RFP to provide a UML profile for real-time and embedded systems [14]. MARTE is a successor of SPT, aligned on UML 2, and with a wider scope. MARTE introduces a number of new concepts, including time concepts.

The underlying model of time is a set of time bases. A time base is an ordered set of instants. Instants from different time bases can be bound by relationships (coincidence or precedence), so that time bases are not fully independent and instants are partially ordered. This partial ordering of instants characterizes the time structure of the application. This model of time is sufficient to check the logical correctness of the application. Quantitative information can be added to this structure when quantitative analyses become necessary. Note that a specification of a temporal behavior may refer to points of time (instants) or to segments of time (durations). In the MARTE meta-model of time, Instant and Duration are two distinct concepts, specialization of the abstract concept of time.

The users of MARTE have access to the time structure through clocks. Here, clocks are not physical devices; they are model elements representing a general concept of time. While in SPT, clocks were implicitly bound to the physical time, in MARTE, a clock can be bound to any recurrent event. Thus, MARTE distinguishes two kinds of clocks:

- chronometric clocks, which make reference to physical time, and
- logical clocks, which focus on the ordering of instants, possibly ignoring the physical duration between instants.

Some classes are stereotyped by the stereotype ClockType, they define the type of a clock. Such a class specifies the nature (dense or discrete) and the kind (chronometric or logical) of the represented time, a set of clock properties (e.g., resolution, maximal value, etc.) a set of accepted time units. For the chronometric clock types, time units are the usual time units: the second (s) and its derived units. Most logical clock types use a generic time unit called tick. In some cases, they may use more specific units: a processor cycle, for instance, or even units for physical quantities, such as for example time measured in angular degree (◦). A Clock (i.e., an instance of a Clock-Type) is characterized by its unit and the values (real numbers) given to its optional properties: resolution, maximalValue, offset. Resolution gives the granularity of the clock; maximalValue is the value at which the clock rolls over; offset specifies the origin instant. The resolution, the maximalValue and the offset are given with the unit of the clock. A predefined Clock is provided in the TimeLibrary of MARTE:

- idealClk. This hypothetical clock reads the dense "physical time". It is used as a reference clock for the (imperfect) chronometric clocks defined by the users of the profile.

A ClockConstraint sets dependencies up amongst clocks.

In the MARTE time model, time-related concepts (e.g., event occurrences and behavior executions) make explicit reference to one or several clocks, through the property on, on identifies the clock and then the unit used. In UML, an Event describes a set of possible occurrences; an occurrence may potentially trigger effects in the system. A TimeEvent is an Event that defines a point in time (instant) when the event occurs. The specification can be either absolute or relative to some other instant. A TimedEvent is a TimeEvent, where the instant specification explicitly refers to a clock. Note, that in the general case it is not possible to compare two events that refer to two different clocks. The comparison is only possible when specific constraints on the clocks (ClockConstraint) or the events (InstantConstraint) are given. A UML Behavior describes a set of possible executions; an execution is the performance of an algorithm according to a set of rules. MARTE associates a duration, an instant of start, an instant of termination with an execution, these times being read on a clock. A TimedProcessing is a Behavior or an Action with explicit references to clocks.

Analysis Capabilities

Simulation and Formal Proof

Timing simulation: Timesquare tool

Schedulability analysis: MAST tool

Associated Tools

MARTE editors: Papyrus [11], IBM Rational Rhapsody [12],

MagicDraw [15]

Scheduling analysis: Marte2Mast [16]

Timing analysis: Timesquare tool [17]

3.1.5 Clock Constraint Specification Language CCSL

Type of Properties

Continuous / discrete / multiform time

Modeling Concepts

The Clock Constraint Specification Language (CCSL) has been first introduced in the annex of the MARTE profile. After many improvements, CCSL has now a formal semantics [5] that can be exploited to process a correct execution, if any. Foundational CCSL constraints are defined in a kernel library. CCSL allows building new libraries and the definition of user-defined constraints by composing existing relations (from the kernel library or from other ones) in order to fit the constraints from a specific domain. CCSL is a mean to specify relations between the evolutions of some clocks. These relations can either be synchronous or asynchronous.

All these relations between clocks are first defined by using a reflexive and transitive instant relation named precedence and noted $\preccurlyeq$.

From *precedence* four new instant relations are derived: *Coincidence, Strict precedence, Independence and Exclusion*.

To express clock relations, one can then use these instant relations. For instance, a strict clock *precedence* relation (denoted $\prec$) between two clocks *a* and *b* is asynchronous and specifies that for all natural number k, the k^{th} instant of *a* occurs before the k^{th} instant of *b*:

The coincidence relation (denoted =) between these two clocks imposes a stronger synchronous dependency: the k^{th} instant of *a* must be coincident with the k^{th} instant of *b*:

The same mechanism applies for all relations. Informally, the *exclusion* relation (denoted #) between two clocks *a* and *b* specifies that no instants of the clock *a* coincide with one of the clock b.

The *alternatesWith* relation (denoted ~) between two clocks *a* and *b* specifies that instants of the clock *b* are interleaving instants of the clock *a*.

Additionnally to constraints some *expressions* can be directly defined on clocks. For instance, the *isPeriodicOn* expression takes three parameters: a clock specifying the super clock, a positive natural number specifying *period* and a positive natural number specifying the *offset*. It results in a clock which is a subclock of the super clock and whose instants are always separated by *period* instants of the super clock. Moreover, the first instant of the resulting clock coincides with the $offset^{th}$ instant of the super clock.

Another useful expression is *delayedFor*. It takes three parameters: a *reference* clock, which represents the clocks we want to delay; a *counter* clock, which represents the clock on which the delay is counted and a positive natural number, which specifies the number of ticks of the counter clock by which the reference clock is delayed.

A CCSL specification is the conjunction of all these constraints. As a result, because synchronous and asynchronous relations are used

conjointly, the execution of a CCSL specification is a partially ordered set of coincidence equivalence classes of instants [6]. It is important to notice that the addition of new constraints to a CCSL specification results in a new partial order that is either the same than the previous one (i.e., the new constraint has no impact) or a subset of the previous partial order (i.e., it is "more ordered").

Because clocks are defined in the MARTE profile and used in the CCSL semantics, they act as an interesting junction between the structural model and its behavior. In some sense, Clocks (that can be applied on every property or instance specification) generalize the concept of UML events, which are used in only very specific parts of the model.

The current version of TimeSquare generates a sequence of steps that satisfies the set of constraints. An inconsistent specification can lead to a deadlock: after a finite sequence of steps, the simulation gets at a point where all the clocks are disabled (not allowed to tick).

Examples of Properties

Intervals: ($t_{FBDC} + 40 \leq t_{IC} \leq t_{FBDC} + 60$) on *crkClk*

Period: T *isPeriodicOn* cc *period* p *offset* o;

Temporal patterns between clocks *camClk* = *crkClk* filteredBy (10)

Mathematical relations $t_{KWE} - t_{KWB}$ = min (sampleNb * Tsampling on idealClk, KAWD on crkClk)

Causal relation $A_j \prec B_j$

Analysis Capabilities

Simulation and Formal Proof

Associated Tools

TimeSquare [17] is the software environment to deal with MARTE time model and CCSL. TimeSquare is an Eclipse plug-in that has four main functionalities: 1) interactive specifications, 2) clock constraint checking, 3) generation of a solution, 4) displaying and exploring waveforms.

TimeSquare has been designed to be used with UML tools applying MARTE profile. In this profile, clocks and clock constraints can be associated with many and various model elements.

A wizard is included in TimeSquare. It facilitates clock definitions, clock constraint specifications, model element browsing, and parameter setting.

The second functionality checks constraint sanity and is called when the above mentioned wizard is not used.

The third functionality relies on a constraint solver that yields a satisfying execution trace or issues an error message in case of inconsistency. The traces are given as waveforms written in VCD format. VCD (Value Change Dump) is an IEEE standard textual format for dump files used by EDA logic simulation tools. The solver intensively uses Binary Decision Diagrams (BDD) to manipulate boolean equations induced by CCSL clock constraints. Waveforms can be displayed with any VCD viewer.

As fourth functionality, TimeSquare has its own viewer enriched with interactive constraint highlighting and access facilities.

3.1.6 Synchronous Languages

Type of Properties

Discrete / multiform time

Modeling Concepts

Synchronous languages [18] like Esterel/SyncCharts, Lustre/Scade and Signal appear in the early 1980th. They are suited to reactive systems programming and their formal semantics allows taking off ambiguous behavioral interpretation and a synchronous design can be proved to be correct. The synchronous languages used in reactive system programming also make use of logical time. In synchronous programming, (physical) time passing is represented by event occurrences; for instance a signal generated by an external device. However, these events do not have any specific status that distinguishes them from other events. Hence, a synchronous program may have statements such as “a task must complete before 10 ms”, and “a car must stop within 50 m”. Both statements express a deadline: “10 ms” for the former, and “50 m” for the latter. This is known as Multiform Time.

Analysis Capabilities

Simulation and model checking

Associated Tools

Simulation and model checking are two ways to validate a system. With synchronous languages, simulation can be used for early bug finding. With simulation, a designer can test scenarios through a user-friendly interface, and moreover, see the internal reactions of the controller.

However, even if the design passes successfully all the simulation tests, it is not sure that a safety property holds. To prove this, an exhaustive simulation of the controller behavior should be provided. Symbolic executions of the model can solve this problem. Symbolic model checker, part of the synchronous languages distribution, does this job very well. A limitation is that signals must not convey values. They use only pure signals (associated with event occurrences) and counters. When a safety property is violated, model checkers generate a counter-example input sequence. This sequence can be played back in order to understand the flaw.

Another technique widely used in synchronous programming [12] is the technique of an observer. An observer is a reactive program expressing the property to verify. The observer is put in parallel with the program and receives the same inputs from the environment. It takes also as input the outputs of the program. The unique output of the observer is a violation signal. The verification consists of checking that the observer never emits the violation signal. Analysis is provided

by calculating reachability analysis onto the synchronous composition of the observer and the program.

3.1.7 Property Specification Language PSL

Type of Properties

PSL can be applied for the definition of assertions, as well as for complex modeling. It can be used for the precise formulation of specifications in the area of hardware design by means of hardware description and modeling languages like VHDL, Verilog, SystemC, or SystemVerilog. The IEEE Conclusion PSL can be seen as a variant of CTL and LTL mainly introducing a different syntax to future based TLs. Thus timing concepts are already covered when CTL and LTL is considered. As PSL also covers timing intervals based on semantics of discrete event simulation along the lines of VHDL, SystemVerilog, and SystemC (delta-cycle with time advance) it also provides insights to TL extension towards this simulator family.

Modeling Concepts

The *Property Specification Language* (PSL) is derived from Sugar, which was originally developed by IBM. PSL as a standard was developed by the industrial consortium Accellera, and it has recently been adopted as IEEE Standard P 1850 [22] [23]. PSL is defined in four layers: boolean, temporal, verification, and modeling layer.

The boolean layer encompasses expressions, as defined for the underlying HDL.

The temporal layer is a central building block of PSL. It allows all expressions from the boolean layer in connection with temporal operators and Sequential Extended Regular Expressions (SEREs).

The verification layer specifies syntactic elements for grouping PSL expressions, as well as mechanisms to bind such expressions to models in an HDL.

The modeling layer, finally, allows behavioral elements of the underlying HDL. This allows, e.g., the calculation of expected results for a simulation. This layer also supports mechanisms, which allow the access to values from previous simulation cycles.

Analysis Capabilities

PSL is a specification language to be used in conjunction with hardware description languages, such as VHDL, Verilog, and SystemC. Verification is performed either dynamically, e.g. by means of simulation, or in a formal static manner, e.g., by means of model checking, or theorem proving.

Associated Tools

There are several tools that support PSL for simulation and formal verification (e.g. from Mentor Graphics, Cadence, and Synopsys). The tools support PSL assertions for use in dynamic simulation verification.

3.1.8 Temporal Logic and CTL

Type of Properties

Some variants of temporal logic allow the specification of time-intervals (CCTL, TCTL, TLTL). Other variants are based on different execution models, e.g. discrete vs. continuous time. Some of these variants restrict themselves to certain kinds of operators for efficiency reasons (ACTL, UPPAAL). Most temporal logics applied today are restricted to future based expressions, though some exist, which also allow past expressions.

Modeling Concepts

A modal logic based on temporal modalities is called temporal logic. Operators of a temporal logic relate to future (Next, Henceforth, Eventually, Until, Unless) or past (Previous, Has-Always-Been, Once, Since, Back-To). Temporal operators usually relate to state sequences. The most frequently applied temporal logics are future oriented. They apply either a branching or a linear time model. In the branching time model, a formula applies from a current state to all possible execution paths simultaneously, whereas in the linear time model, a formula evaluates over a certain path.

Analysis Capabilities

The primary application area of temporal logic is in formal requirements descriptions. The high degree of abstraction supports discrete and continuous specifications in the value and in the time domain. Temporal logics are frequently applied in model-checking [24]. Several techniques have been described for the generation of test cases from a model and from temporal logic specifications. The formal means of temporal logics and its variants CTL and LTL [25] are the most commonly used means for formal verification.

Associated Tools

The described temporal logics are supported by several tools (e.g. SPIN and UPPAAL). Each Tool defines its own, specific syntax for logic expressions. The semantics of each temporal logic are usually defined formally. Temporal logics allow the precise formulation of temporal dependencies.

3.1.9 Testing and Test Control Notation TTCN

Type of Properties

In comparison with its predecessor TTCN-2, which mainly bases on hierarchical tables, TTCN-3 is similar to modern programming languages. TTCN-3 basically differs from established imperative programming languages because it features a compact additional syntax for the description of alternative execution paths and allows the handling of synchronous and asynchronous communication mechanisms. As an administrative feature, test cases can be grouped hierarchically and integrated in a test suite.

Modeling Concepts

The Testing and Test Control Notation (TTCN) is used for the detailed specification of tests and its latest version is TTCN-3 standardized by ETSI and ITU-T [19] [20]. Its main feature is the separation of concern between abstract test suites and an adapter layer which allows full portability of test suites and therefore makes them independent of any platform implementation. The test adapter handles all platform and implementation languages (e.g. C/C++ or JAVA) issues for the communication with a System Under Test (SUT) and also the actual coding and decoding requirements of an application.

Analysis Capabilities

TTCN was mainly applied in the telecommunication sector and most recently was also used for automotive software testing. Its description of alternative execution paths allows the handling of synchronous and asynchronous communication mechanisms. Furthermore, the definition of alternatives triggered by time-outs enables protocol tests in a straightforward manner.

Associated Tools

There are several open source tools as well as commercial compilers, interpreters, and generators for TTCN-3 available. These tools offer automatically generated TTCN-3 test cases and scripts by means of the state model of the SUT, test data, test scripts, and timed tests.

3.1.10 SystemC

Type of Properties

Properties for SystemC can be specified either in the language itself e.g., by annotating the model with assertions or specifying state machines/automatons which react sensitive to state changes of the system model. Moreover, formal temporal languages (e.g. IEEE P1850 PSL) can be applied on the signal level.

Modeling Concepts

System modeling in SystemC (IEEE Std. 1666™-2005) [26] is based on the discrete event modeling concepts. Occurrences of an event are assigned to certain points on a linear time line. Processes are sensitive to events which again notify events immediately or in future. Thus, timing aspects are modeled using *wait statements* blocking a process for a certain amount of time or until an event occurs. In order to model the structure of a system, processes can be grouped to (hierarchical) modules which communicate via ports, interfaces and channels.

Analysis Capabilities

Analysis in SystemC is based on dynamic verification through the executable model (simulation). A pseudo parallel simulation is achieved by means of a cooperative multi-tasking kernel. Simulation output can be either proprietary (e.g. textual) or a standardized trace format (e.g. Value Change Dump IEEE 1364-2001).

Associated Tools

The Open SystemC Initiative (OSCI) [27] reference implementation of the SystemC language and the simulation kernel consists of a collection of C++ macros and class libraries. Thus, the model itself is pure C++ code that must be linked to the SystemC library. Compiling the model for simulation on a host requires the availability of a C++ toolchain (e.g. the GNU toolchain).

3.1.11 SystemVerilog

Type of Properties

SystemVerilog specifies so-called *assertions* (SystemVerilog Assertion, SVA) to verify properties of a design or a system evolving and manifesting over time. These assertions are based on so-called *sequences* and *properties* (supersets of sequences). Sequences consist of boolean expressions augmented with special temporal operators.

Modeling Concepts

SystemVerilog (IEEE Std 1800™-2005) is a unified hardware description and verification language (HDVL) standard based on extensions to the Verilog (IEEE Std 1364™-2002) language [21]. The combination of description and verification in SystemVerilog provides means to handle all important aspects of the design and verification flow: design description, functional simulation, property specification, and formal verification. SystemVerilog is based on a behavioral semantics for discrete event simulation comparable to VHDL, SystemC, and SpecC. This implies causal relationships between events plus advancement of discrete time scalable through user specified time units.

Analysis Capabilities

Analyses and formal verification with SystemVerilog are performed by means of its test bench features which include the specification of functional coverage and constraint-based random test generations. Functional coverage refers to statistics based on sampling events throughout the simulation. It is used to determine when the device under test (DUT) has been exposed to a sufficient variety of stimuli, so that there is a high confidence that the DUT is working correctly. This coverage ensures that all desired corner cases in the design space have been explored. Combined with simulation and SVA this also enables the analysis and verification of the system timing behavior.

Associated Tools

There are several tools that support SystemVerilog for simulation and formal verification (e.g. from Mentor Graphics and Synopsys). The tools support SystemVerilog assertions, functional coverage and constraint-based random test generation for use in dynamic simulation verification.

3.1.12 Modelica

Type of Properties

Continuous / discrete time

Modeling Concepts

Modelica is an object-oriented equation-based general purpose modeling language primarily aimed at physical systems. The model behavior is based on ordinary differential algebraic equation (OAE and DAE) systems combined with discrete events, so-called hybrid DAEs. Such models are ideally suited for representing physical behavior and the exchange of energy, signals, or other continuous-time or discrete-time interactions between system components. Nevertheless it is possible to represent time discrete SW systems or the runtime behavior of embedded systems based on the discrete event semantic.

Modelica models are similar in structure to UML/SysML models in the sense that Modelica models consist of compositions of sub-models connected by ports that represent directed or undirected signal flow. Constraints and timing constraints can be realized by assert statements. Timing constraints in the Model (SysML, EAST-ADL) must therefore be represented as textual constraints on block or function level.

Analysis Capabilities

Simulation of hybrid (continuous and discrete) DAE

Associated Tools

OPENMODELICA, supported by OpenSource Modelica Consortium

- Compiler/Interpreter for the Modelica language
- Model evaluation.

The results of a simulation can be visualized by a plotter, Assert statements can raise warnings, errors.

ModelicaML is a UML profile for Modelica supported by OSMC.

3.2 Timing Analysis Approaches and Tools

In the scope of phase I of the TIMMO-2-USE project, in the predecessor of this document i.e. in the State-of-the-art document of work package 1 (Deliverable D1), several existing ways and approaches dealing with timing have been collected. Corresponding descriptions have been provided from the perspective and experience of contributing partners. Within work package 3, analysis approaches have been discussed and finally structured. As a result of these investigations, in this version of the State-of-the-art document (Deliverable D9) the structuring of the analysis approaches has been aligned with the results from work package 3 (see also Deliverable D10).

Tools supporting the approaches are mentioned in association.

3.2.1 Worst and Best Case Analysis

3.2.1.1 Worst-Case Execution Time Analysis

Type of Properties

Worst-case timing information

Modeling Concepts

Worst-case timing information does not require specific modeling concepts as it can be represented as discrete time.

Worst-case execution time analysis basically focuses on execution times of target code or basic blocks and therefore applies to lower levels of abstraction.

Analysis Capabilities

Embedded systems with hard real-time constraints need reliable guarantees for the satisfaction of their timing constraints. These guarantees can be obtained by sound timing-analysis methods. An overview of timing-analysis methods is provided in [34]. One such method is static program analysis by abstract interpretation, which works by analyzing the program code without actually executing the program. The analysis results obtained in this way are valid for all non-interrupted program runs with all inputs.

Over the last several years, a more or less standard architecture for code-level timing-analysis tools has emerged. One can distinguish three major building blocks:

- Control-flow reconstruction and static analyses for control and data flow find possible values of registers and memory cells, addresses of memory accesses, and (some) loop bounds.
- Micro-architectural analysis determines upper bounds on execution times of basic blocks. It performs an abstract interpretation of the program execution on the particular architecture, taking into account its pipeline, caches, memory buses, and attached peripheral devices. By means of an abstract model of the hardware architecture, the pipeline analysis simulates the execution of each instruction. The cache analysis provides safe approximations of the contents of the caches at each program point. Complex architectural features are the main challenges for this analysis phase.
- Path analysis computes the longest execution paths through the whole program. This can be done by modeling the control flow by an integer linear program so that the solution to the objective function is the predicted worst-case execution time for the input program. The values of the variables are the execution counts of the basic blocks and their links, which together define the worst-case execution path.

Associated Tools

aiT

Description: The commercially available tool aiT by AbsInt implements this architecture, cf. http://www.absint.com/ait. The tool is employed in the aeronautics and automotive industries and has been successfully used to determine precise bounds on execution times of real-time software [35].

Results: aiT determines safe and precise upper bounds for the worst-case execution times of tasks in real-time systems. Here, a task means a sequentially executed piece of code (no threads, no parallelism, and no waiting for external events). aiT operates on binary executables for selected target architectures and produces results valid for all program runs with all inputs. aiT takes as input an executable containing the task to be analyzed, a description of the hardware on which the task is running including a description of (external) memories and buses (i.e. a list of memory areas with minimal and maximal access times), and code annotations providing additional information like targets of indirect jumps, loop bounds, etc. The annotations may be written by the user or generated by other tools.

The tool then computes an upper bound for the runtime of the task (assuming no interference from the outside). Results about the basic-block execution times, the worst-case path, and also the results of auxiliary analyses such as register values are provided as a textual report intended for human reading, an XML report intended for machine reading, and graphical output in the form of an annotated control-flow graph.

SWEET

Description: The Swedish Execution Time Analysis Tool (SWEET) is a research prototype WCET analysis tool from Mälardalen University. It has the same basic architecture as aiT, but its use is mainly for automatic program flow analysis and computing program flow constraints rather than making precise WCET estimates using a micro-architectural analysis. SWEET's program flow analysis can take into account restrictions on possible values for program variables, and can produce tighter results if such restrictions are given. SWEET performs its flow analysis on the "ALF" code format, and can analyze different kinds of code, both on source and binary level, by first translating them into ALF. SWEET can also perform an approximate WCET analysis for source code if a cost model is provided.

Results: SWEET can compute advanced program flow constraints for C code using a C-2-ALF translator. If desired, the constraints can be exported as source-level annotations that are readable by aiT. SWEET can also compute approximate (unsafe) WCET estimates for C source code.

RapiTime

Description: RapiTime by Rapita Systems Ltd is a commercially available tool that computes worst case execution times and reports other timing and code coverage information based on evidence from testing. RapiTme is typically used during system testing, where detailed timing measurements are taken of the software running on the real hardware, with operating system scheduling/interrupts etc.

Static analysis of the source code structure is used in conjunction with test data to compute the WCET for functions in the system. RapiTime is used for timing verification, performance optimization and code coverage measurements. RapiTime operates mostly at the source-code level, supporting C, C++ and Ada on most 8, 16 and 32-bit targets with a variety of on-target tracing options.

Results: RapiTime determines accurate upper bounds for the worst-case execution times of components for real-time systems, the test evidence to support detailed “drill-down” and optimization processes. RapiTime can import and export data and traces in a variety of formats.

3.2.1.2 Worst-Case Response Time Analysis

Type of Properties

Worst-case timing information:

Complete abstract system description including buses, communication layers, ECUs, cores, bus speed, tasks, runnable entities, task priority, execution times, runnable entities order, task chaining, variables, signals, etc.

Modeling Concepts

Worst-case response time analysis basically focuses on the dependencies and interferences of different components with their properties and therefore applies to higher levels of abstraction.

System level:

On system level, worst-case response time analysis is also referred to as Scheduling Analysis which also covers the consideration of best cases, resulting in lower and upper bounds for the parameter under investigation.

Worst-case response times can be applied to and computed for several objects in an embedded system, for example response times of tasks, end-to-end delays spanning multiple components e.g. from sensor to actuator, or communication paths.

Worst-case response times are impacted by scheduling properties like activation periods, delays / execution times of components, interrupts, etc.

CAN communication:

CAN frame transmissions are modeled with a trajectory based approach (like stochastic processes, but without probabilities) so that results about worst case (=longest possible) response times can mathematically be proven and worst case values or upper bounds on them can concretely be computed. This approach allows to verify latency constraints.

[37] introduces and formalizes the notion of meeting n out of m deadline constraints. This type of uncertainty loosens the requirement imposed in TADL by the delay constraint.

Analysis Capabilities

Computation of worst-case response times, or upper bounds, for tasks, path latencies, or communication frames.

Associated Tools

SymTA/S

SymTA/S is used for predicting, optimizing and verifying software integration (embedded controllers), communication integration (field buses), and system integration (controllers and buses).

The SymTA/S tool-suite automatically predicts and verifies worst-case and typical-case timing using efficient models of the system functions, electronic architecture, controller and bus scheduling, and the system environment. A key strength of SymTA/S is the right level of abstraction. SymTA/S focuses on the 'time consumers' in the system at an appropriate level of detail for each development phase. Most data is imported automatically, avoiding unnecessary modeling overhead.

INCHRON Tool-Suite

The INCHRON Tool-Suite with its tool component chronVAL allows constructing system models using an intuitive GUI, supported by an import facility for OIL (OSEK Implementation Language) files. Tasks can be specified as black box or with functions/runnables. The Tool-Suite GUI provides the view of the system from different perspectives, e.g. system view, hardware view, or clocks view,

Main features:

- worst-case response time calculation for tasks and CAN communication
- worst-case response time analysis for end-to-end delays / event chains, also covering distributed systems including CAN and FlexRay communication
- analysis of Residual Bus (Restbus) traffic (chronBUS)
- specification and supervision of different timing requirements
- extensive HTML report containing detailed information about each task, ISR, function, and CAN message, event chains, and timing requirements

NETCAR-Analyzer

The NETCAR-Analyzer tool provides worst-case response time analysis for CAN frames. It requires as input

- Bus Speed
- Frame periods, payloads, priorities, transmission offsets

and offers the following features:

- Computation of worst-case response times of periodic CAN frames
- Takes into account transmission offsets
- Near-optimal offsets assignment algorithms with user-defined performance criteria: e.g. optimize the worst-case response

times for a specific subset of tasks, for instance, the 10 lowest priority frames

- Exhibit the situations leading to the worst-case: results can be checked by simulation (e.g. with RTaW-Sim) or testing
- Enable dimensioning frame transmission queues and buffers at ECU and communication controller level
- Handle both FIFO and prioritized waiting queues at the ECU level
- Fast multi-core implementation: typically, an exact response time computation requires less than 30 seconds for 100 frames on a dual-core system

3.2.2 Worst and Best Seen Cases and Statistics

3.2.2.1 Simulation of CAN Bus Communication

Type of Properties

Discrete timing information

Modeling Concepts

Discrete event based simulation that allows inferring statistics about response times of CAN frames. The simulated events are limited to those strictly needed to reproduce the typical behavior of CAN bus communications.

Type of Properties

The simulation of CAN bus communications requires as input:

- Bus Speed
- Frame periods, payloads, priorities, transmission offsets
- Clock drifts
- Transmission error models

Analysis Capabilities

Statistics about CAN frame response times derived by simulation:

- Min, average, max, quantiles, histogram

Associated Tools

RTaW-Sim

RtaW-Sim provides the following features:

- Simulates frame exchanges on CAN buses
- Provides fine grained statistics (histogram, min, average, max, quantiles) for frame transmission delays
- Takes frame transmission offsets into account

- Models ECU clocks drifting apart
- Takes into account the queuing policies at the ECU and com. controller levels (FIFO, HPF, etc)
- Fault-injection : transmission errors through user-defined error models
- Simulates the total functioning time of a vehicle in a couple of hours

INCHRON Tool-Suite

The INCHRON Tool-Suite with the add-on tool component chronBUS provides analysis capabilities for CAN and FlexRay communication.

Main features:

- automatic import of industrial CAN and FlexRay configuration files (CANdb dbc, FIBEX)
- support of separate ECU clocks in distributed systems
- editing of ECU and message properties
- extensive HTML report containing detailed information about CAN buses and CAN messages

3.2.2.2 System Simulation

Type of Properties

Execution times can be specified as best-case, worst-case, or probabilistic distribution (normal or uniform) between bounds.

- Discrete event descriptions
- Discrete clocks
- Stimulation patterns (periodic, sporadic, burst, jitter) for task activations and external stimuli
- Bus schedules, speeds, and transmission times
- ISR, task, runnable entity, and functions execution times (to be specified as best-case, worst-case, or probabilistic distribution (normal or uniform) between bounds)

Modeling Concepts

Static and probabilistic timing information can be associated to different entities in a distributed system:

- activation patterns for tasks and functions
- execution times of tasks, functions, and basic blocks
- communication means e.g. buses
- hardware clocks

Constructing a system model composed of (hardware) resources, processes (tasks and ISRs) deployed on the resources, operating system and scheduling properties, and equipping these entities with

timing properties like execution times allows to simulate and analyze the system model both in a pessimistic (worst-case) and optimistic (best-case) context, and in any (probabilistic) context between best-case and worst-case.

Simulation and analysis take into account inter-process impact like scheduling-based preemptions or data dependencies which may influence start and end time of processes as well as their execution times.

Simulation results in a variety of diagrams and graphs, e.g. sequence diagram, state diagram, and load diagram. Histograms show statistical distributions of chosen parameters.

Analysis results in graphs showing best-case and worst-case under consideration of inter-process interference according to the specified system criteria.

Analysis Capabilities

Simulation in the scope of an ECU or of distributed systems

Applicable to Analysis, Design, and Implementation levels

Associated Tools

INCHRON Tool-Suite

The INCHRON Tool-Suite including the components chronSIM, chronBUS, chronVIEW, and chronEST allows constructing system models using an intuitive GUI, supported by an import facility for OIL (OSEK Implementation Language) files. Tasks can be specified in different granularity, from black box via functions/runnables to basic blocks, and even target code. Furthermore a profile for IBM Rational Rhapsody® is available which enhances a UML model by timing characteristics and which allows to automatically generate a simulation model out of a UML model.

The Tool-Suite GUI provides the view of the system from different perspectives, e.g. system view, hardware view, or clocks view,

Main features:

- support of separate clocks in distributed systems
- specification of execution time in time units (granularity from picosecond to second) or in clock ticks
- estimation of execution time of target C / C++ code (chronEST)
- consideration of CAN, FlexRay, and Ethernet buses
- simulation of Residual Bus (Restbus) traffic (chronBUS)
- simulation and visualization of event chains and their segments
- specification and supervision of different timing requirements
- extensive HTML report containing detailed information about each task, ISR, function, and CAN message, event chains, and timing requirements

- visualization of trace logs with timing relevant events (chronVIEW)

Website

http://www.inchron.com/tool-suite-inchron.html

3.2.3 Probabilistic Timing Analysis

3.2.3.1 Distributions Analysis and Typical Case Analysis

Type of Properties

Probabilistic

Modeling Concepts

One possibility to reduce the pessimism inherent to worst-case analysis is to introduce uncertainty in the model of the system. This approach applies only to soft real-time systems, as the result of the analysis will also be uncertain.

In probabilistic analysis, the uncertainties on task/resource parameters are modeled by probability distributions, obtained for example by using statistical analysis of execution traces. As the behavior of a system that has a probabilistic choice at every instant is called a stochastic process, probabilistic timing analysis is often referred to as stochastic analysis.

Analysis Capabilities

The key idea of probabilistic timing analysis is to model at least one parameter in the task/resource model by a random variable. In the case of execution times for instance, this parameter is ideally represented by a function which associates with each possible execution time, relative to a given sampling scenario, a probability value. In practice, such a function is approximated by a set of pairs (execution time, probability value).

Then compositional techniques must be found to provide guarantees (expressed using probabilities) on the global system based on the properties of its constituting tasks.

Since the 1990's probabilistic timing analysis has been extensively studied. Earlier approaches include Gardner and Liu [29], Lehoczky [30] and Manolache [28] and all make some worst-case assumptions such as restrictions on preemption, restrictive load conditions etc.

Díaz et al. [30] [31] have greatly contributed to the state of the art in stochastic analysis of real-time systems under various scheduling policies. Assuming a periodic and independent task model, their approach provides safe bounds for response time distributions. Their

use of correct approximations renders the complexity of the analysis "manageable" while preserving soundness of the result. Besides no worst-case of restrictive condition is needed anymore.

Furthermore, dependencies arising from shared resources can be taken into account. However, other dependencies are not dealt with. More precisely, interdependency between different tasks of a system can be classified as follows:

- Inter-stream dependency: several tasks sharing a resource (e.g. executing on the same processor) interfere with each other.
- Intra-stream dependency, which can be twofold:
  1. Data dependency: the output value of some task may have an impact on the execution time of another task. This is due to the fact that the execution time of a task may depend on the input data it is provided with.
  2. History dependency: successive activations of the same task may not be independent of each other (e.g., bursts may never occur). Typically, if the result of a computation is stored, then the second activation of a task will be much faster than the first one.

When supposing independence of tasks, it is possible to combine tasks' execution times using convolution. However, this hypothesis does not hold in many practical situations and may lead to major errors in the results of timing analysis, as shown by Ivers and Ernst [32]. When dependencies are unknown, Bernat et al. showed in [37] that execution times should be combined using supremal convolution and Ivers integrates this result into the scheduling analysis approach followed by Díaz.

Associated Tools

INCHRON Tool-Suite

The INCHRON Tool-Suite allows to model probabilistic distributions for process/function/runnable execution times and for periodic event patterns. Such models can be both simulated and statically validated.

RapiTime

Part of the analysis that RapiTime does is based on measuring and analyzing the distributions of execution time and producing probability distributions of worst case execution time: evidence-based probabilities of different execution times occurring when the worst case path is executed.

3.2.3.2 Probabilistic Timing Information Visualization and Analysis

Modeling Concepts

Understanding system timing and performance are key when testing real-time systems. Tracing is regularly applied today to log the timing

of relevant events. A challenge is to efficiently analyze such trace data and quickly identify timing problems and their root causes.

Analysis Capabilities

Visualize and analyze traces

Associated Tools

TraceAnalyzer

Description: The SymtaVision TraceAnalyzer analyzes and visualizes network and ECU traces and enables engineers to find the cause of timing problems. TraceAnalyzer can create timing models for scheduling analysis in SymTA/S.

Results:

- Visualize controller and bus schedules with events, blocking, and preemptions.
- See the flow of signals and data through a system.
- Automatically create overviews of key timing parameters, e.g. load over time.
- Automatically generate key statistics, e.g. distribution of task response times.
- Create timing models of existing systems as input for SymTA/S

INCHRON Tool-Suite

The INCHRON Tool-Suite with its component chronVIEW allows processing of trace logs in a way that activation, start, termination, entry and exit events are assigned to the corresponding processes (tasks, ISRs, functions/runnables), resulting in the possibility to transform this information into diagrams known from the tool chronSIM. Once the diagrams are available, statistical information can be retrieved and probabilistic distributions can be visualized e.g. in histograms. Furthermore, timing requirements can be defined or imported to supervise the traced behavior of the measured system with respect to constraints.

Main features:

- visualization of trace logs with timing relevant events (chronVIEW)
- specification and supervision of different timing requirements
- extensive HTML report containing detailed information about each task, ISR, function/runnable, event chains, and timing requirements

3.2.3.3 Reasonable-Case Timing Analysis

Type of Properties

Uncertain

Modeling Concepts

[36] presents a method based on reasonable heavy-load case analysis for dealing with a-periodic tasks. The idea behind a reasonable heavy-load case is to be less pessimistic than a worst-case analysis while being more pertinent for validation of timing constraints than an average case analysis. However, dependencies between tasks (frames in the paper) are not handled.

Analysis Capabilities

Simulation and Formal Proof

There is no need for specific analysis tools with this approach: worst-case analysis tools can be used. However, there must exist modeling tools providing the expected type of uncertain timing information.

script "R"

Description: This *modeling* tool is used for statistical inference about inter-arrival of event and the computation of the reasonable worst case curve. It is freely available.

Associated Tools

Not applicable

3.3 Other tools

The tools described in this section are applied in industrial tool chains for automatic code generation from function models.

3.3.1 Simulink®

Type of Properties

Continuous, discrete and hybrid timing information

Modeling Concepts

Simulink® is a mathematical multi-domain modelling and simulation, design, implementation, and integration tool based on the MATLAB® environment. This tool is suitable for modelling and simulating heterogeneous systems (linear and non-linear, continuous, discrete and hybrid) with different implementation levels and solvers, based on the MATLAB® environment.

Within a Simulink® model, Stateflow blocks can be used to model state charts and flow graphs.

Selecting and Customizing Blocks

Simulink® software includes an extensive library of functions commonly used in modeling a system. These include:

- Continuous and discrete dynamics blocks, such as Integrator and Unit Delay
- Algorithmic blocks, such as Sum, Product, and Lookup Table
- Structural blocks, such as Mux, Switch, and Bus Selector

You can customize these built-in blocks or create new ones directly in Simulink®. Additional blocksets (available separately) extend Simulink® with specific functionality (aerospace, communications, etc.). You can also model physical systems such as those with mechanical, electrical, and hydraulic components.

Incorporating MATLAB® Algorithms and Hand-Written Code

With the MATLAB® code, you can call MATLAB® functions for data analysis and visualization. Additionally, you can design embedded algorithms that can then be deployed through code generation with the rest of your model. You can also incorporate hand-written C, Fortran, and Ada code directly into a model, enabling you to create custom blocks in your model.

Defining and Managing Signals and Parameters

Simulink® enables you to define and control the attributes of signals and parameters associated with your model. You can define the following signal and parameter attributes:

- Data type: single, double, signed or unsigned 8-, 16- or 32-bit integers; boolean; and fixed-point
- Dimensions: scalar, vector, matrix, or N-D arrays
- Complexity: real or complex values
- Minimum and maximum range, initial value, and engineering units

Managing temporal aspects with solvers

Simulink® software provides some features like fixed-step and variable-step solvers for simulating these models. Solvers are numerical integration algorithms that compute the system dynamics over time using information contained in the model. The solvers support the simulation of a broad range of systems, including continuous-time (analog), discrete-time (digital), hybrid (mixed-signal), and multirate systems of any size. These solvers can simulate stiff systems and systems with state events, such as discontinuities, including instantaneous changes in system dynamics. Simulation options can be configurable in the sense that it has a number of adjustable parameters such as the type and properties of the solver, simulation start and stop times, amplitude of signal, noise, etc, and whether to load or save simulation data. Optimization and diagnostic information for the simulation can be also defined, together with different combinations of options.

Simulations in Simulink® are based on the so-called zero execution time assumption, that means, Simulink® does not take into account the execution time of an algorithm on a given target processor. An ideal-fast processor is assumed and the behavior of the model is computed at the points in time determined by the solver.

Analysis Capabilities

Model-Based Design/Simulation/Analysis

Associated Tools

MATLAB Simulink® tool, a commercial product commercialized by Mathworks. It provides a graphical user interface (GUI) for building models as block diagrams. After you define a model, you can simulate it, using a choice of mathematical integration methods, either from the Simulink® menus or by entering commands in the MATLAB® Command Window. The menus are convenient for interactive work, while the command line is useful for running a batch of simulations. As an analysis tool, Simulink® includes also linearization and trimming tools, which can be accessed from the MATLAB® command line.

Using scopes and other display blocks, you can see the simulation results while the simulation runs. The simulation results can be put in the MATLAB® workspace for post-processing and visualization.

The output files are with “mdl” extension.

3.3.2 Production Code Generation from Simulink®

Type of Properties

Discrete time Simulink/Stateflow® model (see 3.3.1)

Modeling Concepts

An ECU function is modeled as a subsystem in a Simulink/Stateflow model® which describes the behavior of the function. The subsystem can be embedded in an environment model which allows for closed loop simulations and tests in the Simulink®.

For safety and efficiency reasons, only a limited subset of Simulink® blocks is applied in production code models of ECU functions. This subset contains mainly discrete-time blocks, algorithmic blocks, state charts, and structural blocks, but no continuous-time blocks.

In addition to the normal Simulink® block parameters, some further properties are specified to prepare a model for production code generation. These additional properties include:

- Datatypes, ranges and scaling factors for fixed-point variables.
- Variable classes, which control the implementation of variables in the code, for example, the memory section and type qualifiers like `extern`, `static`, `volatile`, and `const`. Variable classes are also used to specify, if a variable is accessible for a calibration system.
- Function classes to control the code partitioning, for example, the name of a C code function and the name of the generated source code file.
- Implementation options for lookup tables, for example, the search algorithm. The optimization of lookup tables has a large impact on the performance of the generated code.
- AUTOSAR properties, for example, to assign a Simulink Inport/Outport to the data access of a runnable at an AUTOSAR SW-C port.

Most of these implementation properties do not change the behavior of the model during offline simulations, but they have a huge impact on the performance of the generated code on the target processor, with respect to RAM, ROM, and execution time.

Analysis Capabilities

The data flow and control flow in the model is analyzed by the code generator. Typically, each basic block is translated into an intermediate representation which contains expressions and assignments to variables. Several optimizations are applied to this intermediate representation before the final production code is generated, for example, expressions are combined and re-ordered to avoid intermediate variables. The code generator also analyzes the range of signal values to select data types for intermediate variables. Furthermore, call graphs are analyzed and code functions may be in-lined (embedded) to improve the code efficiency. In multi-rate models, each root function can be assigned to a task. Several communication mechanisms are available for inter-task communication. Depending on the task priorities, the inter-task communication may be optimized. The final production code is assigned to code modules. If possible, the scope of variables will be limited to the code module where they are used.

Associated Tools

TargetLink® is a production code generator for Simulink/Stateflow® models provided by dSPACE. It is widely used in the automotive industry for model-based development of ECU functions. TargetLink® is focused on safety and code efficiency and supports all modeling techniques and optimization methods described above. Further characteristics of TargetLink® are:

- A special, enhanced blockset which is seamlessly integrated in Simulink and which allows the user to specify the required implementation properties for each block in the model.
- A data dictionary which may (optionally) be referenced by the blocks in the model. The data dictionary provides all required implementation properties. Thus, the behavioral model can be separated from the implementation details.
- After code generation, the data dictionary contains detailed meta-information about the generated code. This meta information can be used, for example, to export an A2L variable description for the generated component, or to generate a HTML documentation.
- The generated code can be customized according to company-specific coding style guides, for example, the names of code files, functions, variables, data types etc. The formatting of the source code can be controlled using XSL style sheets.
- The generated production code can be simulated in MIL, SIL, or PIL mode in closed loop with a Simulink® environment model. Thus, regression tests can be performed between the behavior of the original model and the generated code. In PIL mode, execution times of the code can be measured for a given target compiler on an evaluation board.

- TargetLink® is compatible to the MISRA guidelines for model-based ECU development [MISRA] and it has a “fit for purpose” certification according to the IEC 61508 and ISO 26262 standards.

The results of a TargetLink® code generation are:

- The source code files and optionally an AUTOSAR software component description.
- A detailed meta description of the generated code in the Data Dictionary.
- Simulation results which are recorded during MIL, SIL, and PIL simulations, including execution time measurements for code functions.
- A documentation of the generated code and an A2L variable description for calibration systems.

Especially the dSPACE Data Dictionary provides an excellent basis for timing analysis tools, for example, to compute the worst-case execution time of the code on a given target processor. This will be exploited in the TIMMO-2-USE project.

Website

http://www.dspace.de/en/pub/home/products/sw/pcgs/targetli.cfm

3.3.3 AUTOSAR ECU Design and Implementation

Type of Properties

AUTOSAR System (see 4.1).

AUTOSAR SWC implementations, e.g. from TargetLink (see 3.3)

AUTOSAR basic software modules from ECU suppliers

Modeling Concepts

AUTOSAR has defined a standardized ECU software architecture, consisting of

1. an application software layer with cross-platform reusable software components,
2. a basic software layer providing services for the application components and managing access to the ECU resources, and
3. a run-time environment (RTE) which realizes the communication between the different modules and the integration into the operating system.

AUTOSAR supports a component-based development process by separating the external ports and interfaces of a software component from the internal behavior and implementation on the code level.

On the application layer, several software components can be combined to compositions, which reflect higher level functions or a logical grouping. Thus both, top-down and bottom-up designs, are possible.

In an AUTOSAR system, the global software architecture, the network communication, and the hardware topology are clearly separated. Mappings are used to describe which system element instances exist on a concrete ECU.

Analysis Capabilities

The AUTOSAR description of an ECU provides all relevant information about the software architecture, network communication and hardware topology, including the configuration of the operating system and the communication stack. The software is described down to the level of internal behavior and implementation, i.e. the code files for a given target. This information can be used, for example, to analyze the communication, or to perform a schedulability analysis, if the execution times of runnable entities are known.

Associated Tools

SystemDesk is an AUTOSAR authoring tool provided by dSPACE. It supports the major steps in the AUTOSAR methodology:

- Modelling of the software architecture.
- Definition of the network communication, including import from AUTOSAR, FIBEX, DBC, LDF files.
- Definition of the hardware topology.
- System mappings, such as SWCs to ECUs, data elements to system signal instances, and runnables to tasks.
- Basic software configuration.
- RTE generation based on TargetLink technology, including A2L file export.
- Offline simulation of the ECU network in a virtual environment (SIL and PIL mode).

SystemDesk is seamlessly integrated with TargetLink (see 3.3) as a production code generator for software components. It can be extended with plug-ins. All features are fully automatable to support integration in a company-specific tool-chain. Thus, external tools have full access to the AUTOSAR data in SystemDesk.

SystemDesk's data model supports all basic software modules, including OS and COM. The AUTOSAR Timing Extensions or TADL are currently not included. Within the TIMMO-2-USE project, the management of timing information can be realized, for example, by plugins.

Website

http://www.dspace.de/en/pub/home/products/sw/system_architecture_software/systemdesk.cfm

4 Project Results in the Field of Time Modeling

4.1 AUTOSAR

Project Name

AUTOSAR

Website

http://www.autosar.org

TIMMO-2-USE Participants

Bosch, Continental, dSPACE, INCHRON, Volvo

General Topics

The Automotive Open System Architecture (AUTOSAR) has been formed with the goals of

- Implementation and standardization of basic system functions as an OEM wide "Standard Core" solution
- Scalability to different vehicle and platform variants
- Transferability of functions throughout network
- Integration of functional modules from multiple suppliers
- Consideration of availability and safety requirements
- Redundancy activation
- Maintainability throughout the whole "Product Life Cycle"
- Increased use of "Commercial off the shelf hardware"
- Software updates and upgrades over vehicle lifetime
- The AUTOSAR scope includes all vehicle domains.

The AUTOSAR standard will serve as a platform upon which future vehicle applications will be implemented and will also serve to minimize the current barriers between functional domains. It will, therefore, be possible to map functions and functional networks to different control nodes in the system, almost independently from the associated hardware.

Timing Related Topics

In the specification of timing extensions for AUTOSAR, which was introduced in AUTOSAR 4.0, the notion of *event* is the main entity. It is used to refer to an observable behavior within a system (e.g. the activation of a runnable entity, the transmission of a frame etc.) at a certain point in time. AUTOSAR introduced a new abstract type TimingDescriptionEvent as a formal basis for the timing extensions. Depending on the concrete model entity and the associated observable behavior, specific timing events are defined and linked to the different views. In order to relate timing events to one another, a

further concept called TimingDescriptionEventChain is introduced in AUTOSAR. It is important to note that for the events referred to within an event chain a functional dependency is implicitly assumed. This means that an event of a chain somehow causes subsequent chain events. Based on events and event chains, it is possible to express various specific timing constraints derived from the abstract type TimingConstraint. These timing constraints specify the expected timing behavior. As timing constraints shall be valid independently from implementation details, they are also expressed on an abstract level by referencing the TimingDescriptionEvents and TimingDescriptionEventChains. Thus, by means of events, event chains and timing constraints defined on top of these, a separate central timing specification can be provided, decoupling the expected timing behavior from the actually implemented behavior. This approach supports timing contracts for AUTOSAR systems in a top-down as well as bottom-up approach.

Results Connected to TIMMO-2-USE Topics

The idea of events, event chains, and timing constraints are common to AUTOSAR and T2U. T2U will further advance TADL while keeping the current alignment between TADL and AUTOSAR timing concepts and adapting TADL in future if future changes of the AUTOSAR timing concepts occur.

4.2 EAST-ADL

Project Name

ATESST2, MAENAD

Website

http://www.atesst.org, http://www.maenad.eu

TIMMO-2-USE Participants

Continental, Volvo

General Topics

EAST-ADL provides an information structure and ontology that makes the development of stand-alone automotive embedded systems more systematic and predictable. EAST-ADL is an architecture description language with means for capturing the requirements, characteristics and configurations of automotive systems and the related analysis and V&V. A Methodology and guidelines support language/tool adoption and cost-efficient development and V&V. EAST-ADL is harmonized with relevant standards including AUTOSAR and SysML.

The model-based development and V&V approach contributes to improving communication among system stakeholders, documentation, and V&V capabilities. This is a shift from today's document-driven testing and simulation procedures, to a model-based way of working. This provides means for stakeholders to deal with the complexity and risk management of active safety systems.

Timing Related Topics

EAST-ADL provides support for model-specific engineering information, including non-functional properties that are relevant for the timing of automotive functions. Conceptually, timing information can be divided into timing requirements and timing properties, where the actual timing properties of a solution must satisfy the specified timing requirements.

The EAST-ADL timing package originates from the TIMMO project (see Section 4.3) and is therefore also closely connected to the timing concepts of AUTOSAR (see Section 4.1). Concretely, modelling of timing requirements and properties on the functional abstraction levels of the architecture description language is done by means of the “Timing Augmented Description Language” TADL developed by the TIMMO project. The Implementation Level, i.e., AUTOSAR, is addressed by the “AUTOSAR Timing Extensions”, which are introduced in AUTOSAR release 4.0. These extensions are based on TADL concepts, too.

Timing constraints are defined separately from the structural modelling and reference structural elements of the EAST-ADL. The requirements support in EAST-ADL allows for tracing from solutions as modelled in the structural model to requirements, and from verification cases to requirements. The TADL constraints fit in the requirement support as refinements of the requirements.

The fundamental concept for describing timing constraints is that of Events and Event Chains (see Section 4.2). On every level of abstraction, events can be identified, i.e., a stimulus that causes a reaction and such a reaction leads to another observable event, i.e., a response.

Timing requirements can be imposed on Event Chains, for example, specifying that the time between the occurrence of a stimulus event and the occurrence of the expected response event shall not exceed a specific amount of time – i.e., an end-to-end delay from a sensor to an actuator, or the response event shall not occur before a specific amount in time and not later than a specific amount of time after the point-in-time the stimulus event has occurred. In addition, requirements regarding the synchrony of events can be expressed as well, stating that a number of events shall occur “simultaneously” in order to cause a reaction, or be considered as valid response of a system function.

For example, in case of a passenger vehicle, its brake system shall apply the brakes simultaneously; or the exterior light system shall simultaneously turn on and off the rear- and front turn signal indicators.

Results Connected to TIMMO-2-USE Topics

In the ATESST2 project, the TADL (as a result from the TIMMO project) were integrated into EAST-ADL and further advanced. Also in the MAENAD project, timing support will be further investigated, in close relation with TIMMO-2-USE.

4.3 TIMMO

Project Name

TIMing MOdel

Website

http://www.timmo.org/

TIMMO-2-USE Participants

Continental, Volvo, Bosch, SymtaVision, Chalmers, Paderborn University C-Lab

General Topics

The European research project TIMMO (TIMing MOdel) developed an automotive system timing management approach using a common, standardised way for handling all timing-related information during the development process. The complexity — and the cost — of the development cycle is reduced significantly, while reliability is improved. TIMMO is about developing a Timing Augmented Description Language (TADL) and an accompanying methodology that provide:

- a formal and standardised specification, analysis and verification of timing constraints across all development phases, avoiding over- or under-dimensioned systems and unnecessary iterations in the development process;
- a formal and standardised specification, analysis and verification of timing constraints at all levels of abstraction enabling, e.g., timing requirements to be traced across all abstraction levels;
- an improved and predictable development cycle enabling a common, standardised infrastructure for handling timing to shorten the development cycle and increase its predictability.,

TIMMO had three major results: A formal language for modelling timing aspects, an accompanying methodology that describes how to apply the language in the development process, and a set of case studies serving as example applications and as validators for the language and methodology.

Timing Related Topics

Modeling of timing requirements and properties is done by means of the “Timing Augmented Description Language” (TADL). Please refer to Section 4.2 for further details.

Results Connected to TIMMO-2-USE Topics

TIMMO-2-USE will take the results of the TIMMO project as a basis and advance and improve them.

5 Conclusion

This state-of-the-art document is centered on referencing modeling and analysis techniques, tools and projects in the context of a multi-level design process for automotive applications. A particular interest is paid to the modeling of temporal behaviors of such real-time embedded systems. This document aims at identifying and clarifying the different notions of time handled at the different levels of an automotive design process and the capabilities of models and tools to handle the different timing characteristics.

A first result of this state-of-the-art analysis is the possibility to establish the complementarities between adopted modeling language and standard such as EAST-ADL, AUTOSAR and TADL and other existing modeling languages concerning the capabilities of modeling statistical, discrete, continuous and multiform timing aspects.

Concerning the continuous time, Modelica, Simulink are used for modeling continuous combined (or not) with discrete events (hybrid systems). Such modeling of continuous behavior can be connected with the structural parts of a design (external behavior of EAST_ADL functions) and some timing constraints applied on SysML/UML blocks translated in Modelica concepts. Tools for simulating and detecting errors of the continuous, discrete or mixed behaviors are available. Simulink and Modelica can be used at analysis and design levels of a development process. At the implementation level, discrete Simulink models can be taken as input for the dSPACE tool TargetLink for producing ECU function code and characterizing low levels parameters such as data types and ranges.

Concerning the discrete time, EAST-ADL, AUTOSAR and TADL are models capable to express timing constraints such as repetition rates, end to end delays, and synchronization. Measurements are constant values associated with two possible time units (ms and °crank). The UML profiles MARTE and SysML make it possible to integrate in a design, more complex algebraic expressions for manipulating time. Discrete time is also a common concept for formal languages such as CTL, PSL, and CCSL for the specification of parameters/variables, constants, and expressions. These models are independent from any abstraction level of a design. Tools for analysis of such models are available such as Uppaal for simulation and formal verification of CTL specifications or Timesquare which allows timing analysis activities on CCSL specifications.

Different tools provide WCET analysis. Some of them can be used at different levels (analysis, design) such as SymTA/S from SymtaVision, INCHRON Tool-Suite, or RTAW at design and implementation levels, Some need implementation code as input which could be the implementation code for the application (for aiT) or any code generated by intermediate tools (INCHRON Tool-Suite and TargetLink from dSPACE).

Statistical and uncertain time refer to modeling uncertainties on task/resource parameters. Such information can be provided or processed by tools or can be inputs of analysis tools (RTAW tools). For example, with TimeAnalyser from SymtaVision, RTaw-Sim or

INCHRON Tool-Suite, statistical results on system behavior or bus response times can be obtained by analyzing execution. In addition, the INCHRON Tool-Suite takes this information as input for providing system simulation. Modeling, composing and verifying statistical information is an open issue. Statistical/uncertain information can be integrated at the design and implementation levels.

Logical/Multiform time is a concept introduced by synchronous languages. Manipulating multiple time bases in a common model can be convenient for modeling timing requirement which mix for example periods related to physical distance (meters, camshaft) and temporal distance (ms). The UML profile MARTE allows such modeling and the algebraic support for solving and simulating such “hybrid” expressions.

6 Bibliography

[1] F. A. Schreiber, *Is Time a Real Time? An Overview of Time Ontology in Informatics*, in W. A. Halang, A. D. Stoyenko (Eds.) Real Time Computing, Springer Verlag NATO-ASI, vol. F127, 1994, pp.283-307.

[2] D. G. Messerschmitt, *Synchronization in Digital System Design*, IEEE Journal on Selected Areas in Communications, vol. 8, no 8, October, 1990, pp.1404-1419.

[3] L. Lamport, *Time, Clocks, and the Ordering of Events in a Distributed System*, Communications of the ACM 21, (7), 558-565, 1978.

[4] R. Schwarz and F. Mattern, *Detecting Causal Relationships in Distributed Computations* – in Search of the Holy Grail, Distributed Computing 7, 149-174, 1994.

[5] C. André. *Syntax and semantics of the clock constraint specication language*. Technical Report 6925, INRIA, 2009.

[6] C. André, F. Mallet, and R. de Simone, *Modeling time(s)*, in MoDELS. Lecture Notes in Computer Science, G. Engels, B. Opdyke, D. C.Schmidt, and F. Weil, Eds., vol. 4735. Springer, 2007, pp. 559–573.

[7] OMG. *UML 2.2 Superstructure Specification*, Feb 2009. OMG document number: formal/09-02-02.

[8] B. Selic. *On the semantic foundations of standard UML 2.0*. In SFM-RT 2004, volume 3185 of LNCS, pages 181–199. Springer-Verlag, 2004.

[9] OMG. *UML Profile for Schedulability, Performance, and Time Specification*, January 2005. OMG document number: formal/05-01-02 (v1.1).

[10] Enterprise Architect, Sparx System http://www.sparxsystems.com/products/ea/index.html

[11] Papyrus : Open source UML editor, CEA http://www.papyrusuml.org

[12] IBM Rational Rhapsody http://www-01.ibm.com/software/awdtools/rhapsody

[13] OMG. *Systems Modeling Language (SysML) Specification 1.1*. Object Management Group, May 2008. OMG document number: ptc/08-05-17.

[14] OMG UML *Profile for Modeling and Analysis of Real-time and Embedded Systems*, MARTE V1.0. Object November 2009. OMG document number: formal/2009-11-02.

[15] Magic Draw MARTE editor http://www.magicdraw.com

[16] Marte2Mast converter http://mast.unican.es/umlmast/marte2mast/index.html

[17] TimeSquare Model Development Kit, INRIA, http://www-sop.inria.fr/aoste/dev/time square/

[18] A. Benveniste, P. Caspi, S. A. Edwards, N. Halbwachs, P. L. Guernic, and R. de Simone. *The synchronous languages 12 years later*. Proceedings of the IEEE, 91(1):64–83, 2003.

[19] ETSI/TTCN Homepage, http://www.ttcn-3.org/ , March 2011.

[20] ETSI ES 201 873-1 V2.2.1, *Methods for Testing and Specification (MTS)*; The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language, Sophia Antipolis Cedex, France, Feb. 2003.

[21] McGrath, Dylan. *IEEE approves SystemVerilog, revision of Verilog*, EE Times, 2005-11-9. Retrieved on 2007-1-31.

[22] Acellera Consortium Homepage. http://www.accellera.org/, March 2011

[23] IEEE P1850 - *Standard for PSL* - Property Specification Language Homepage http://www.vhdl.org/ieee-1850/, March 2011.

[24] E.M. Clarke, O. Grumberg, and D.A. Peled: *Model Checking*. MIT Press, Cambridge MA, USA, 1999.

[25] A. Pnueli. *A temporal logic of concurrent programs*. Theoretical Computer Science, 13:45-60, 1980.

[26] IEEE - P1666 SystemC WG Homepage http://www.vhdl.org/systemc/, March 2011.

[27] Open SystemC Initiative (OSCI) Homepage http://www.systemc.org/home/, March 2011.

[28] Manolache, S., *Schedulability analysis of real-time systems with stochastic task execution times*, Citeseer, 2002.

[29] M. Gardner and J. Liu. *Analyzing stochastic xed-priority real-time systems*. Tools and Algorithms for the Construction and Analysis of Systems, pages 44-58, 1999.

[30] J. Lehoczky. *Real-time queueing theory*. Proceedings of the 18th IEEE real-time systems symposium, pp 58-67, 1997.K. Kim, JL Diaz, L.L. Bello, J.M. Lopez, C.G. Lee, and S.L. Min. *An exact stochastic analysis of priority-driven periodic real-time systems and its approximations*. Computers, IEEE Transactions on, 54(11):1460-1466, 2005.

[31] JL Diaz, DF Garcia, K. Kim, C.G. Lee, L. Bello, J.M. Lopez, S.L. Min, and O. Mirabella. Stochastic analysis of periodic real-time systems. In Real-Time Systems Symposium, 2002. RTSS 2002. 23rd IEEE, pages 289-300. IEEE, 2003.

[32] José María Lopez, José Luis Díaz, Joaquín Entrialgo and Daniel García. *Stochastic analysis of real-time systems under preemptive priority-driven scheduling.* In Real-Time Systems, 2008, volume 40-2, pages 180-207.

[33] M. Ivers and R. Ernst. Probabilistic Network Loads with Dependencies and the Effect on Queue Sojourn Times. Quality of Service in Heterogeneous Networks, pages 280-296, 2009.

[34] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. The worst-case execution time problem - overview of methods and survey of tools. Trans. on Embedded Computing Sys., 7(3):1-53, 2008.

[35] Christian Ferdinand and Reinhold Heckmann. Worst-case execution time - a tool provider's perspective. In 11th IEEE ISORC 2008, Orlando, Florida, USA, May 2008.

[36] Dawood A. Khan and Nicolas Navet and Bernard Bavoux and Jörn Migge, Aperiodic traffic in response time analysis with adjustable safety level, 2009.

[37] Guillem Bernat and Alan Burns and Albert Llamosi, Weakly hard real-time systems, IEEE Transactions on Computers, 2001, vol 50, 308-321.

[38] Modelica Homepage, https://www.modelica.org/, March 2011.

[39] Open Modelica Homepage, http://www.openmodelica.org/, March 2011.