Consistency Analysis of AUTOSAR Timing Requirements

Steffen Beringer[1][a] and Heike Wehrheim[2][b]
[1]*dSPACE GmbH, Rathenaustr. 26, 33102 Paderborn, Germany*
[2]*Department Specification and Modelling of Software Systems, Paderborn University, 33102 Paderborn, Germany*

Keywords: AUTOSAR, Consistency Analysis, Timing Analysis, Timing Constraints, Satisfiablilty Modulo Theories, Maximum Satisfiability, Unsatisfiable Core, Timed Automata.

Abstract: Applying formal methods in the automotive industries can significantly increase the correctness and reliability of the developed system architectures. This in particular demands a formal specification and analysis of *requirements* on systems. Automotive software architectures are, however, often described using the (semi-formal) AUTOSAR standard which is based on various meta models as exchange formats. This complicates a formal analysis. In this paper, we provide a formalization of *timing requirements* within the AUTOSAR standard. Timing requirements specify constraints on events of the underlying software architecture. We provide a translation of timing requirements into logical constraints which enable the usage of SMT solvers to analyse requirements. Specifically, we employ this translation to check *consistency* of the requirements and use maximum satisfiability solving for localization of erroneous requirements.

1 INTRODUCTION

For the distributed development of various automotive software, the automotive industry utilizes the features of the de facto standard AUTOSAR (AUTOSAR, 2019). It provides a common infrastructure for automotive systems of all vehicle domains based on standardized interfaces. This also includes the definition of component-based software architectures. It further comprises a methodology how to facilitate and parallelize the development of AUTOSAR architectures.

AUTOSAR provides methods for specifying *timing requirements* on the underlying software architecture by the so called *AUTOSAR timing constraints* meta-model extension. Different types and flavors of timing constraints allow for an easy use for different parts of the software architecture in different development phases.

Nevertheless, when the requirements get numerous, maintaining an overview of them can be hard and hence inconsistencies in the requirements can occur. Inconsistent requirements are an indication for a misunderstanding of the expected system functionality by the software developer. Moreover, when requirements are formally checked against the system architecture, e.g. via timing verification methods as described in (Beringer and Wehrheim, 2016) or (Richter, 2005), inconsistent requirements lead to unnecessary verification runs since verification is then inevitably going to fail for some of the requirements. It is advantageous to know inconsistencies in the requirements beforehand.

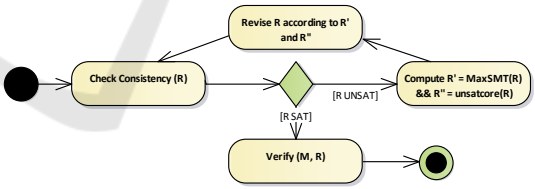

Figure 1: Analysis process described as UML activity diagram.

This work thus presents an approach for supporting the early validation of AUTOSAR software architectures by simplifying the process of AUTOSAR timing requirements specification. It comprises an analysis method which checks a defined set of AUTOSAR timing requirements for inconsistencies by transformation of requirements models into satisfiable modulo theories (SMT) formulae. This enables checking requirements before any further verification task is triggered and assists a requirements engineer in fixing the requirements set. Furthermore, it supports the identification of possible inconsistency

[a] https://orcid.org/0000-0002-8504-7435
[b] https://orcid.org/0000-0002-2385-7512

Beringer, S. and Wehrheim, H.
Consistency Analysis of AUTOSAR Timing Requirements.
DOI: 10.5220/0009766600150026
In *Proceedings of the 15th International Conference on Software Technologies (ICSOFT 2020)*, pages 15-26
ISBN: 978-989-758-443-5

causes. Whenever the set of SMT formulae is unsatisfiable (and hence the requirements inconsistent), we compute the maximal subset of satisfiable formulae (MaxSMT). These represent a set of consistent requirements. All requirements which are not in this set are potential inconsistency causes. By furthermore computing an unsatisfiable core of formulae, we narrow down the requirements to look at for inconsistency resolution. A visualization of both unsat core and MaxSMT result helps the requirements engineer in fixing errors. Note that we do not run the requirements check on any transformed verification model (e.g. timed automata when checking the requirements as described in (Beringer and Wehrheim, 2016)), but directly on the AUTOSAR timing constraints model. This approach is advantageous, since it decouples the concistency check from the specific verification method.

An overview of the proposed steps and their execution sequence is shown in Figure 1 and comprises the following five steps (for a given set of requirements $\mathcal{R}$ and given architecture model M):

1. Check the consistency of requirements $\mathcal{R}$.
2. If the complete set is not consistent, compute the greatest subset of verifiable constraints $\mathcal{R}'$ (via a maximum satisfiability solver) and $\mathcal{R}''$ a minimum subset of constraints, which is inconsistent, otherwise go to step 5.
3. Revise the requirements $\mathcal{R}$ based on the information by $\mathcal{R}'$ and $\mathcal{R}''$.
4. Repeat steps 1-3 until the requirements are consistent.
5. Perform verification process on the requirements set and the architecture model $(M, \mathcal{R})$.

All steps within the analysis process are fully automated and implemented prototypically for the dSPACE AUTOSAR authoring tool SystemDesk®[1]. For step 5, we transform the given AUTOSAR architecture model into timed automata and the requirements into test automata and timed computational tree logic queries (TCTL). The approach for this has been proposed in (Beringer and Wehrheim, 2016). Note that steps 1-4 are performed on the requirements model only. The architecture model is used in step 5.

The paper is organized as follows. The next section 2 introduces the domain of AUTOSAR, the concept of timing requirements, and introduces a first example model. Transformation of requirements into SMT-formulae and our approach for identification and correction of inconsistent requirements are shown in Sections 3 and 4, respectively. Thereafter, runtime measurement results for requirements transformation and verification are presented in Section 5. Finally, Section 6 discusses existing work related to our approach following a conclusion in Section 7, which summarizes the achievements in this work and give hints for future improvements.

2 BACKGROUND

In this chapter we introduce the foundations of AUTOSAR and the AUTOSAR timing extensions used to specify timing requirements. We briefly discuss consistency in general and introduce our running example model.

2.1 AUTOSAR

AUTOSAR[2] is short for **AUT**omotive **O**pen **S**ystem **AR**chitecture and is a joint standard developed by various automobile manufacturers, suppliers, tool vendors, services- and semiconductor companies. AUTOSAR defines the architecture and interfaces of software as meta-model as well as the file format for data exchange. Furthermore, the standard defines its own development methodology. The concepts of this work are based on the current AUTOSAR release version 4.3.

AUTOSAR Software Architecture. On the highest level AUTOSAR defines a layered software architecture, which contains three layers (AUTOSAR, 2019). The *application layer* contains the controller software and is structured as component-based architecture, which means software components can be defined, which communicate via ports. Components contain one or more so-called *runnable entities* (or *runnables* in short) $Re = \{re_1, \ldots, re_n\}$, which contain executable software. The *runtime environment layer* provides standardized APIs to connect the components and modules on the basic software layer, whereas the *basic software layer* provides modules for basic ECU functions like the operating system and communication services and is again subdivided into layers (see e.g. Figure 2).

AUTOSAR Timing Extensions. A subset of the AUTOSAR meta-model addresses the annotation of model elements with timing properties (AUTOSAR, 2019). For a set of model elements so-called *timing events* $E = \{e_1 \ldots e_n\}$ can be specified. A timing event

[1] http://www.dspace.com/en/pub/home/products/sw/\system_architecture_software/systemdesk.cfm

[2] http://www.autosar.org

is the abstract representation of a specific system behavior that can be observed at runtime, e.g. the start of a runnable or the arrival of new data at a port. Furthermore, AUTOSAR timing extensions enable the definition of *timing constraints* $\mathcal{R} = \{r_1, \ldots, r_n\}$, which define a timing relationship between two or more timing events. Timing constraints can be defined by restricting the execution times of runnable entities. The following timing constraints are considered:

Offset Timing Constraint. An offset timing constraint $r_{otc} = (e_s, e_t, min, max)$ constrains the time between a source timing event $e_s \in E$ and target timing event $e_t \in E$ by defining a minimum and maximum offset $min, max \in \mathbb{R}$ between those events.

Latency Timing Constraint. A latency timing constraint $r_{ltc} = (C, min, max)$ describes a minimum and maximum latency in the scope of a timing chain C. The timing chain is an ordered sequence of events, $C = \langle e_1, \ldots, e_n \rangle$, $e_i \in E$ for all $1 \leq i \leq n$, which have to occur in the specified time bound.

Synchronization Timing Constraint. A synchronization timing constraint $r_{stc} = (S, tolerance), S \subseteq E$ specifies a set of timing events, which must occur simultaneously within a tolerance value.

Execution Order Constraint. An execution order constraint $r_{eoc} = \langle re_i, \ldots, re_j \rangle$ constrains the execution order of runnable entities. So for each execution order constraint $r_{eoc} \in \mathcal{R}_{eoc}$ with an ordered set of runnable entities $\langle re_1 \ldots re_n \rangle$ a successor runnable entity re_{i+1} may only be executed after the runnable entity re_i has been executed.

Execution Time Constraint. The execution time constraint $r_{etc} = (re, min, max)$ does not restrict the occurrence of timing events, but the execution time of an executable entity *re* to a minimum duration *min* and maximum duration *max*. The runnable *re* can be either a runnable entity on the application layer or a basic software module on the basic software layer.

A software developer can define a number of such requirements on a given AUTOSAR software architecture. Further timing constraints like *Age Timing Constraint* are not considered since they do not add inconsistencies in our approach, because they only reference one timing event and restrict the timing of a variable implicitly. For modeling of AUTOSAR timing constraints we employ SystemDesk®.

The AUTOSAR Authoring Tool SystemDesk. SystemDesk®is the tooling environment for AUTOSAR models of the company dSPACE. It supports sophisticated and extensive modeling of AUTOSAR architectures by providing a rich graphical user interface as well as code generation for virtual ECUs. Graphical model representations are available for important elements. For example, software components, ports and connections within a software composition can be visualized in a *composition diagram*. Furthermore, single software components with their ports, interfaces and data types can be visualized in a *component diagram*. Other model elements are ordered hierarchically in a tree structure.

2.2 Consistency of Requirements

Our interest here is in checking consistency of such timing requirements. The IEEE recommendations for software requirements specifications (IEEE, 1998) calls a software requirements set consistent "if, and only if, no subset of individual requirements described in it conflict". This means that the set of requirements is consistent, if there exists a model or system, which can satisfy all requirements in this set.

Definition 1 (Consistency). *Let $\mathcal{R} = \{r_1, \ldots, r_n\}$ be the set of timing requirements in an AUTOSAR model and let $\mathcal{M}$ be the set of models. Then $\mathcal{R}$ is consistent iff $\exists M \in \mathcal{M} : \forall r_i \in \mathcal{R} : M \models r_i$.*

In the case of AUTOSAR the set of models is defined by all models which can be modeled using the AUTOSAR meta-model and the requirements are defined by a set of AUTOSAR timing constraints. Thus, the set of AUTOSAR timing constraints is only consistent, if there is a AUTOSAR model which conforms to all restrictions imposed by the AUTOSAR timing constraints. Since the timing constraints restrict the model's behavior only in the sense of timing, the only model elements of interest are the AUTOSAR timing events. Therefore, checking the existence of a valid model is reduced to finding a valid execution order of timing events.

2.3 AUTOSAR Example: The Turn Switch Indicator

In the following we will consider a simple AUTOSAR software architecture. The architecture manages left and right direction indicators of a vehicle according to turn switch and warn lights sensors. The application layer consists of several software components which comprise several so-called runnable entities containing executable software. The example architecture is shown in Figure 2. The two software components on the left read in sensor data and check for errors before forwarding the signal data to the next software

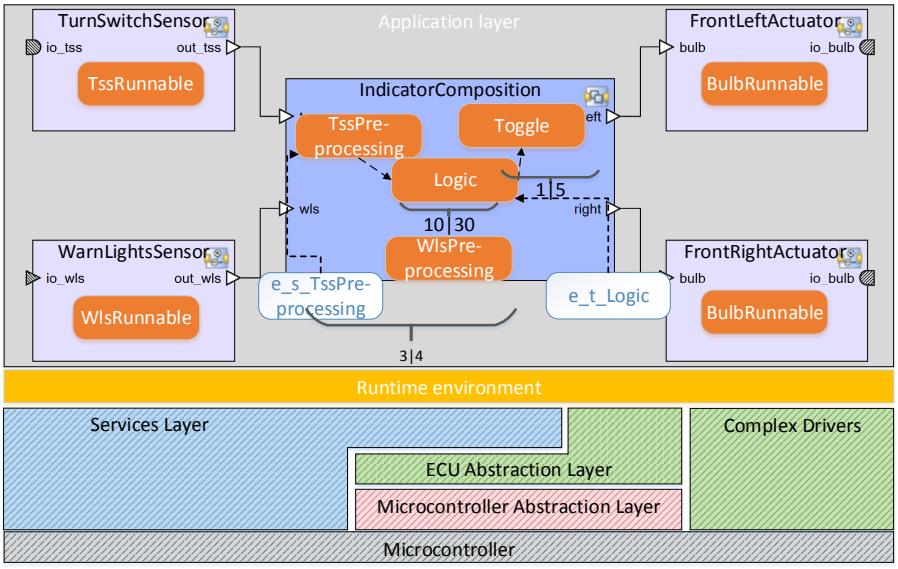

Figure 2: Example Software Architecture.

component. The `Indicator Composition` software component receives the raw sensor values and encapsulates several runnable entities for pre-processing of the signal values as well as the logic of the system. The actuator software components on the right are responsible for activating the left respectively right bulb of the direction indicator. Furthermore, the example contains a configuration of the RTE, and on the *basic software layer* the configuration for the operating system. Other basic software modules are not considered in this example. Furthermore, the example is extended with the following timing description events and timing constraints. The runnable `TssPreprocessing` consists of an event for the start $e^s_{\texttt{TssPreprocessing}}$, and for runnable `Logic` an event for its termination $e^t_{\texttt{Logic}}$ has been added. These timing events are shown as white boxes in the example diagram, connected to the runnable entity they belong to. Additionally, the architecture contains three timing constraints: An execution order constraint constraining the execution of runnable entities, which is shown as dashed lines between runnables; an offset timing constraint constraining the time between newly added timing events, which is reflected by the brackets among the timing events, and an execution time constraint constraining the execution time of the `Logic` runnable (see also Table 1).

This set of constraints is not consistent. The `Logic` runnable must take at least 10ms for execution (r_{etc}), while the calculation of the indicator logic shall be completed between 3ms and 4ms after the turn switch sensor pre-processing has been started (r_{otc}). This includes the computation of the `Logic` runnable, since the runnables `TssPreprocessing` and `Logic` must be executed in order (because of r_{eoc}). Our approach automatically identifies this kind of inconsistencies, while a manual inspection would be quite complex since the inconsistency results from different types of timing constraints. We consciously stick to only three different constraints in our example to keep it comprehensible, although we implemented our approach for all of the presented timing constraints.

3 TRANSFORMATION OF AUTOSAR TIMING CONSTRAINTS

For consistency checking of timing requirements, we use a logical approach. In this section we describe the transformation of AUTOSAR timing requirements into logical formulae of an SMT (satifiability modulo theories) solver. The SMT formulae reflect the temporal constraints predefined by the timing requirements and thus satisfiability conforms to the existence of a valid ordering of timing events satisfying all requirements. In Figure 3 the abstract concept is shown. The AUTOSAR model contains the system model, timing events, which are associated to model elements in the system model and the AUTOSAR timing constraints. The timing events are transformed into SMT variables and the AUTOSAR timing events are transformed into linear constraints over the generated variables, while the system model is not needed for the consistency analysis. The transformations have been

Table 1: Example Timing Constraints.

Description	Requirement
The runnable entities `TssPreprocessing`, `Logic` and `Toggle` must be executed in order.	$r_{eoc} = \langle$`TssPreprocessing, Logic, Toggle`$\rangle$
The calculation of the indicator logic shall be completed between 3ms and 4ms after the turn switch sensor preprocessing has been started.	$r_{otc} = (e^s_{\texttt{TssPreprocessing}}, e^t_{\texttt{Logic}}, 3, 4)$
The calculation of the `Logic` runnable should be completed in between 10ms and 30ms.	$r_{etc} = (\texttt{Logic}, 10, 30)$
The calculation of the `Toggle` runnable should be completed in between 1ms and 5ms.	$r_{etc_2} = (\texttt{Toggle}, 1, 5)$

automated and are directly applied on the AUTOSAR model within SystemDesk.

Some timing constraints are not only based on timing events, but on runnable entities, for example the execution order constraint. To detect inconsistencies between those timing constraints and timing constraints based on timing events, timing constraints based on runnable entities are transformed so that they are also based on timing events. This is possible since AUTOSAR supports timing events related to the execution of runnable entities. Thus in the following we assume that for each runnable entity $re \in \mathit{RunnableEntities}$ there exist corresponding timing events e^s_{re} and e^t_{re}, which represent the execution start and termination of the runnable entity. Some timing constraints also restrict the occurrence between timing events, thus lower and upper bounds itself can be defined by the occurrence of another timing event. Nevertheless, all restrictions can be defined as linear constraints. Since the restrictions on timing events are defined by linear constraints, the existence of a valid model can be checked by searching for a valid valuation $t : E \to \mathbb{R}_{\geq 0}$ of the timing events using a SMT solver with linear arithmetic. For this, we use the SMT solver Z3 (Bjorner and Phan, 2014). For a given AUTOSAR model with timing events $E = \{e_1 \dots e_n\}$ and timing constraints $\mathcal{R} = (r_1, \dots, r_n)$ an SMT formula $\mathcal{F} = \bigwedge_{i=1}^{n} f_{r_i}$ is successively constructed. Each term in $\mathcal{F}$ is either a constant constraint value (e.g. tolerance or minimum / maximum values) or a variable, which conforms to a timing event e and which needs a satisfying assignment into the time domain $t : E \to \mathbb{R}_{\geq 0}$. We denote the sets of different timing constraint types as $\mathcal{R} = \mathcal{R}_{otc} \cup \mathcal{R}_{ltc} \cup \mathcal{R}_{stc} \cup \mathcal{R}_{eoc} \cup \mathcal{R}_{etc}$. In the following the transformation for each type of timing constraint is presented.

Offset Timing Constraint. An offset timing constraint $r_{otc} = (e_s, e_t, min, max)$ constrains the time between a source timing event e_s and target timing event e_t. Thus, for each timing offset constraint $r_{otc} \in \mathcal{R}_{otc}$ two expressions are added to the SMT formula as follows

$$f_{r_{otc}} = e_s + max \geq e_t \wedge e_s + min \leq e_t \tag{1}$$

Latency Timing Constraint. To verify that a latency timing constraint does not conflict with other timing constraints, for example an offset timing constraint, there must be a valid occurrence of timing events where there exists a sequence of timing events, which does not exceed the maximum value of a latency timing constraint. Therefore, for each ltc with chain $C = \langle e_1, \dots, e_n \rangle$ the SMT formula is extended as follows

$$\begin{aligned} f_{r_{ltc}} = \forall i, 1 \leq i \leq n-1 : e_i \leq e_{i+1} \\ \wedge min \leq e_n - e_1 \leq max \end{aligned} \tag{2}$$

Synchronization Timing Constraint. A synchronization timing constraint $r_{stc} = (S, tolerance), S \subseteq E$, specifies a set of timing events, which must occur simultaneously with a tolerance value. So for each syn-

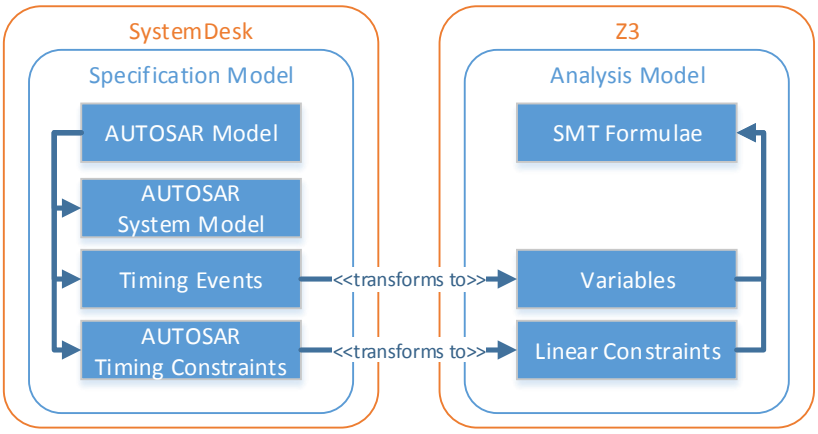

Figure 3: Transformation of AUTOSAR models into SMT formulae.

chronization timing constraint $r_{stc} \in \mathcal{R}_{stc}$, the tolerance value is checked for each combination of events:

$$f_{r_{stc}} = \forall e, e' \in S : e - e' \leq tolerance \quad (3)$$

Execution Order Constraint. For each execution order constraint $r_{eoc} \in \mathcal{R}_{eoc}$ with a sequence of runnable entities $\langle re_1, \ldots, re_n \rangle$ a successor runnable entity re_{i+1} may only be executed after the runnable entity re_i has been executed. Thus, $\forall i, 1 \leq i < n : re_i \leq re_{i+1}$. Since our approach relies on finding a satisfiable assignment based on timing events, we first need to find a modeling alternative, which is equivalent, but is based on timing events instead of runnable entities. Therefore, we model the constrained execution of runnable entities by timing events, which represent the start and stop of a defined runnable and only allow the start event of a succeeding runnable entity to be later than the termination event of the runnable entity before. So let $e^s_{re_i}$ and $e^t_{re_i}$ be the start and terminated event of a runnable re_i. Thus we get

$$\forall i, 1 \leq i \leq n-1 : e^s_{re_i} \leq e^t_{re_i} \wedge e^t_{re_i} \leq e^s_{re_{i+1}} \quad (4)$$

AUTOSAR also allows execution order constraint to be applied to basic software modules. The naming of timing events then is different, but the general transformation method presented can be applied analogously.

Execution Time Constraint. The restriction in an execution time constraint is only within a single AUTOSAR element. Nevertheless, a restriction of the execution time corresponds to having a offset timing constraint between the execution start and termination of the associated runnable. Therefore, a execution time constraint corresponds to having an offset timing constraint $r_{etc} = (r, min, max) = r_{otc}$ with $r_{otc} = (e_{r_s}, e_{r_t}, min, max)$ where e_{r_s} and e_{r_t} are the events associated with the runnable being started and terminated, respectively.

The generated SMT formulae for the example model are shown in Table 2. For the execution order constraint r_{eoc} start events for each runnable must occur before the corresponding termination event and the start event for runnable `Logic` must be before the termination event of runnable `TssPreprocessing`. Analogously the termination event of runnable `Logic` must occure before the start event for the `Toggle` runnable occurs. Therefore, our approach generates five inequalities, which constrain the timings as described. For the Offset Timing Constraint r_{otc} two inequality clauses are generated, which constrain the minimum and maximum offset for the timing events. Finally, for the execution time constraints r_{etc} and r_{etc_2} two clauses are generated for each constraint, which constrain the timing for the start and termination of the `Logic` runnable and the `Toggle` runnable respectively. Since the requirements are not consistent, the generated set of clauses is unsatisfiable.

4 RESOLVING INCONSISTENT REQUIREMENTS

In this section we describe methods to resolve inconsistent requirements. For this, we propose to compute the maximum set of satisfiable clauses as well as an unsatisfiable core and visualize the results.

4.1 Identification of Inconsistency Causes

To identify defects in an unsatisfiable requirements set we investigate several methods, which compute a subset of timing requirements associated with consistency. First we compute the maximum set of satisfiable clauses (MaxSMT) (Bjorner and Phan, 2014).

Table 2: Example Transformations.

Requirement	Generated formulae
$r_{eoc} =$ $\langle$`TssPreprocessing, Logic, Toggle`$\rangle$	$f_1 = e^s_{\texttt{TssPreprocessing}} \leq e^t_{\texttt{TssPreprocessing}}$ $f_2 = e^s_{\texttt{Logic}} \leq e^t_{\texttt{Logic}}$ $f_3 = e^s_{\texttt{Toggle}} \leq e^t_{\texttt{Toggle}}$ $f_4 = e^t_{\texttt{TssPreprocessing}} \leq e^s_{\texttt{Logic}}$ $f_5 = e^t_{\texttt{Logic}} \leq e^s_{\texttt{Toggle}}$
$r_{otc} = (e^s_{\texttt{TssPreprocessing}}, e^t_{\texttt{Logic}}, 3, 4)$	$f_6 = e^s_{\texttt{TssPreprocessing}} + 3 \leq e^t_{\texttt{Logic}}$ $f_7 = e^s_{\texttt{TssPreprocessing}} + 4 \geq e^t_{\texttt{Logic}}$
$r_{etc} = (\texttt{Logic}, 10, 30)$	$f_8 = e^s_{\texttt{Logic}} + 10 \leq e^t_{\texttt{Logic}}$ $f_9 = e^s_{\texttt{Logic}} + 30 \geq e^t_{\texttt{Logic}}$
$r_{etc_2} = (\texttt{Toggle}, 1, 5)$	$f_{10} = e^s_{\texttt{Toggle}} + 1 \leq e^t_{\texttt{Toggle}}$ $f_{11} = e^s_{\texttt{Toggle}} + 5 \geq e^t_{\texttt{Toggle}}$

Definition 2 (Weighted MaxSMT)**.** *Given a set of formulae $\mathcal{F}$ and numeric weights $w \in \mathcal{W}$ associated with each formula by a function $c : \mathcal{F} \to W$, the task of* weighted maximum satisfiability solving *modulo theories (MaxSMT) is to find a subset $I \subseteq \mathcal{F}$ such that*

1. *$\bigwedge_{f \in I} f$ is satisfiable and*
2. *$\sum_{f \in I} c(f)$ is maximal in all subsets of $\mathcal{F}$.*

Thus, if the set of requirements is not satisfiable, MaxSMT finds a subset of formulae which maximizes a defined cost function c. In our case, $c(f) = 1$ for all formulae f, so we give no priorities to certain constraints.

Applying MaxSMT in this case finds a largest set of formulae which are satisfiable. Furthermore, for traceability we keep a mapping $r : \mathcal{F} \to \mathcal{R}$, which maps each formula f back to the timing requirement it originated from. With this information a requirements engineer might fix the requirements set by either discarding or altering / relaxing requirements which do not belong to the MaxSMT set, so that finally the complete requirements set is satisfiable. The MaxSMT computation is performed in Z3. A MaxSMT subset for our example is $I = \mathcal{R} \setminus \{f_7\}$.

Another method for identification of inconsistencies are unsatisfiable cores (*unsat core* in short).

Definition 3. *Let $\mathcal{F}$ be a set of formulae which is unsatisfiable. An* unsatisfiable core *of $\mathcal{F}$ is a subset $F \subseteq \mathcal{F}$ which is unsatisfiable as well.*

SMT solvers typically try to compute *minimal* unsat cores. However, neither unsat cores nor MaxSMT solutions are unique. Given the MaxSMT solution, and in particular a set of requirements R which are not part of this solution (and thus potentially incorrect), we can compute an unsat core to get the set of formulae (requirements) which are together inconsistent. Ideally, this is not the entire set of requirements so that the requirements engineer does not need to look at all requirements when trying to identify and correct wrong requirements.

In our example an unsat core would be $UC = \{f_1, f_4, f_7, f_8\}$. For those constraints there is no satisfiable ordering of the events $e^s_{\texttt{TssPreprocessing}}$, $e^t_{\texttt{TssPreprocessing}}$, $e^s_{\texttt{Logic}}$ and $e^t_{\texttt{Logic}}$. Since r_{etc_2} does not contain any clauses of the unsat core, this requirement is not responsible for the inconsistency. Thus, here we can use the unsat core to narrow down the set of requirements to look at during correction.

4.2 Consistency Visualization

A crucial point for the identification of inconsistency causes is an adequate result visualization of the generated MaxSMT and unsat core clauses. In this regard we provide a graph-based visualization of the MaxSMT and unsat core, where nodes in the graph represent timing events and directed edges represent the timing relations (i.e., the SMT clauses) among the events.

Definition 4 (ResultGraph)**.** *Given a set of timing events E and the set of transformed requirement formulae $\mathcal{F}$, the* result graph $G = (G_V, G_E, c_{min}, c_{max})$ *is the following:*

- *$G_V = E$ is the set of nodes containing one node per timing event,*
- *$G_E \subseteq G_V \times G_V$ is the set of directed edges, where $(e, e') \in G_E$ if the set $\mathcal{F}$ contains a formula $e \leq$*

e' *or* $e + min \leq e'$ *or* $e + max \geq e'$ *for some* $min, max \in \mathbb{R}$,

- $c_{min} : G_E \rightarrow 2^{\mathbb{R}}$ *is the minimum distance labeling function, where* $min \in c_{min}(e, e')$ *if the set* $\mathcal{F}$ *contains a formula* $e + min \leq e'$,
- $c_{max} : G_E \rightarrow 2^{\mathbb{R}}$ *is the maximum distance labeling function, where* $max \in c_{max}(e, e')$ *if the set* $\mathcal{F}$ *contains a formula* $e + max \geq e'$.

For the MaxSMT and unsat core two result graphs are visualized separately. In the MaxSMT result graph G_{max} nodes are labeled green, if the corresponding clauses, which represent the timing relations among the events, belong to I and red otherwise. For the unsat core result graph G_{uc} nodes are labelled green, if they do not belong to UC and green otherwise.

4.2.1 Example Visualization

Figure 4 shows the graphical representation of the (generated) events and the timing relationships among them utilizing our example transformations shown in Table 2. Timing events which belong to satisfiable timing requirements are shown in green, while timing events which belong to unsatisfiable timing requirements are shown in red.

5 EVALUATION

In this section, we evaluate the runtime of our approach based on example architectures. We first evaluate the runtime of the requirements consistency checking approach and try to figure out whether the application to real-world projects is feasible. Afterwards we argue if and how the consistency checking and our timing verification approach can be efficiently combined and whether our proposed process in Figure 1 is reasonable. Since the generated SMT formulae are linear constraints and the considered domain is the set of rational numbers, the model can be solved using the general simplex algorithm and thus the model is solvable in polynomial runtime (Kroening and Strichman, 2008). To identify possible defects in an unsatisfiable requirements set we compute the maximum set of satisfiable constraints (MaxSMT). Since the MaxSMT problem is NP-hard, we presume higher runtime (Garey et al., 1976). Nevertheless, MaxSMT comutation should still be faster than the verification task itself, because the transformed timed automata model has exponential runtime in the number of clocks, which can be rather high, since the full AUTOSAR architecture is transformed. For each test scenario we measure the time for requirements transformation and the SMT solver as well as the verification time for the approach described in (Beringer and Wehrheim, 2016).

5.1 Test Scenarios

First, the runtime for requirements transformation, SMT solving and MaxSMT optimization is measured for each timing constraint type separately and for a set of different combined timing constraint types. With this step we identify, if for a predefined size of requirements the runtime for consistency checking and MaxSMT computation is reasonable. For the measurement of SMT solving and requirements transformation the timing constraints were generated. For an offset timing constraint a source and target timing event with a minimum and maximum offset between 1ms and 10ms were selected randomly. For an execution order constraint 5 runnables were selected randomly and for each synchronization timing constraint 3 timing events were selected randomly. We limit the size to 10, because otherwise the timing verification task takes too long. For the MaxSMT optimization measurement we generated an unsatisfiable set, for which the solver computes the maximum satisfiable set. Afterwards the runtime for timing verification is measured based on two different generated architecture models and compared to the consistency check to have an idea whether consistency checking is generally favorable before the verification task to save time. One model M_1 contains only of a limited set of software components and very few tasks, whereas a second model M_2 is more complex containing more software components and tasks. Table 4 gives an overview of the complexity of those models. Since most of the requirements sets generated are inconsistent, the subsequent timing verification would fail for at least one requirement.

5.2 Evaluation Setup

The SMT solver used is Z3 version 4.6.0-x64. For timing verification UPPAAL version 4.0.13 is used (Behrmann et al., 2004). The test scenario execution and runtime measurement is performed on a Windows 7 Professional system with an Intel Core i7 4800MQ with 8GB memory. UPPAAL was used with BFS search order, conservative state space reduction and DBM state space representation. The runtime is measured in seconds. The symbols used in the results are explained in Table 3.

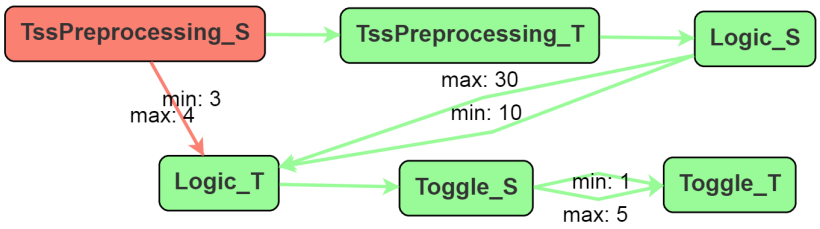

Figure 4: ResultGraph for MaxSMT.

Table 3: Description of symbols.

Symbol	Description
$T(t)$	Runtime for transformation of requirements into SMT in seconds
$T(smt)$	Runtime for SMT solving in seconds
$T(msat)$	Runtime for MaxSMT in seconds
$T(tv)$	Runtime for model transformation into timed automata
$T(v)$	Runtime for timing verification in seconds
$ratio$	Runtime ratio in percentage $\frac{T(t)+T(smt)+T(msat)}{T(tv)+T(v)} * 100$

5.3 Results

Table 5 shows runtime measurements for the computation of the maximum satisfiable set of constraints and the timing verification for both M_1 and M_2. The table shows, that transformation of the AUTOSAR elements into Z3-SMT takes a significant amount of time ($T(t)$). This is partly due the fact that the elements are accessed via an automation interface of the AUTOSAR authoring tool, which is less efficient than a direct access due to technological reasons. Furthermore, the transformation algorithm has not been optimized by now. Nevertheless, the time is acceptable compared to the time needed for timing verification.

T(t), T(smt) and T(msat) are identical for test scenarios with the same requirements $\mathcal{R}$, even on different architecture models (e.g. test scenarios 1 and 5). This is evident since the runtimes, which belong to the requirements checking part, are independent of the rest of the AUTOSAR architecture.

The model transformation into timed automata and the verification runtime is obviously largely dependent on the size and complexity of the overall AUTOSAR architecture model. While transformation and verification of the requirements of M_1 is fast with a maximum runtime of 20.1 and 11 seconds for test scenario 4, the same requirements embedded in architecture M_2 can take up to roughly 2 minutes for transformation and 36 minutes for verification in test scenario 8. Finally, we measured the runtimes for the running example, which consists of 5 software components, 8 runnables and 3 tasks, which is a rather simple model. The transformation and consistency check took 4.5 and 0.1 seconds respectively, while verification took only 1.4 seconds.

Table 4: Test scenarios for verification runtime.

	M_1	M_2
Software Components	5	8
Runnable Entities	10	80
Tasks	5	8

The ratio between requirement checking and verification emphasizes the key result. While in test scenarios 1 to 4, which perform the verification task on the simple model M_1, the consistency check takes roughly 30% to 80% of the verification time, in test scenarios 5 to 8, the consistency check only takes a small proportion with 0.6% to 3.5%. So on more complex architectures consistency checking is valuable, while on simple architectures, consistency checking may be omitted. Nevertheless, since we assume that real world examples may more often be complex, and the set of requirements is usually much greater, it is generally favorable to perform the consistency check before the verification task as we proposed in our process presented in the beginning.

Test results 9-14 show how our approach scales with more timing requirements as for each type we extend the generated set to 100. Furthermore, we evaluate, if there are performance differences, when we use a satisfiable set instead of an unsatisfiable. While SMT solving time is still acceptable for all timing requirements, MaxSMT gets considerably slow for execution order constraints and synchronization timing constraints. The differences between satisfiable and unsatisfiable sets are rather small. MaxSMT is faster on the satisfiable set of offset timing constraints

Table 5: Runtimes for requirements transformation, SMT-Solving, MaxSMT computation and verification.

Id	M	$\lvert\mathcal{R}_{otc}\rvert$	$\lvert\mathcal{R}_{eoc}\rvert$	$\lvert\mathcal{R}_{stc}\rvert$	SAT	$T(t)$	$T(smt)$	$T(msat)$	$T(tv)$	$T(v)$	$ratio$
1	M_1	10	0	0	unsat	5,4	0.06	0.04	13	3.0	34.4
2	M_1	0	10	0	unsat	10.9	0.02	0.07	17.2	5.0	49.5
3	M_1	0	0	10	unsat	8.1	0.04	0.04	15.6	8.0	34.6
4	M_1	10	10	10	unsat	24.1	0.1	0.13	20.1	11.0	78.2
5	M_2	10	0	0	unsat	5.4	0.06	0.04	67.5	88.9	3.5
6	M_2	0	10	0	unsat	10.9	0.02	0.07	65.7	864.4	1.2
7	M_2	0	0	10	unsat	8.1	0.04	0.04	72.9	1229	0.6
8	M_2	10	10	10	unsat	24.1	0.1	0.13	110	2168.7	1.1
9	M_2	100	0	0	unsat	31.9	0.1	1.3	80	-	-
10	M_2	0	100	0	unsat	252	0.42	209	571	-	-
11	M_2	0	0	100	unsat	52.7	0.56	339	328	-	-
12	M_2	100	0	0	sat	33.3	0.1	0.4	81	-	-
13	M_2	0	100	0	sat	219	0.39	154	567	-	-
14	M_2	0	0	100	sat	57.1	0.54	335	331	-	-

and on the execution order constraints, while there are only minor differences on the synchronization timing constraints. Due to the slightly differently generated sets, transformation times also differ a little bit. Since timing verification for all requirements would take too long, we omitted verification time and ratio.

Finally, the efficiency of our process presented in Section 1 also depends on how difficult it is to model the timing requirements and on the modeling skills of the software architect, because this influences the probability of faulty timing requirements. When there is less probability of finding inconsistent requirements sets, it may be favorable to omit the requirements checking process since it always extends the absolute verification pipeline runtime.

6 RELATED WORK

Several methods and languages for the formal specification of timing constraints exist, for example CCSL (André, 2009), (Mallet and Simone, 2015), which has been adopted by EAST-ADL (EAST-ADL Association, 2013) for automotive use cases, TSSD (Klein and Giese, 2007), UPL (Teige et al., 2016), SALT (Bauer et al., 2006) or real-time pattern languages (Gruhn and Laue, 2006). Usually, for verification the approaches define transformations into lower level logics like TLTL, MTL or TCTL, or into observers based on executable C-Code. In our approach we focused on the widely used standard *AUTOSAR* and the integrated AUTOSAR Timing Constraints.

Methods, which check consistency of modeling artifacts, e.g. UML models, are presented in (Rasch and Wehrheim, 2003; Seifert et al., 2005; Kotb and Katayama, 2005; Simmonds and Bastarrica, 2005; Kalibatiene et al., 2013; Derrick et al., 2002; Abdelhalim et al., 2011) or for SysML in (Jacobs and Simpson, 2017). Usually consistency is treated as a property between different model diagram views, different model versions or models at different levels of abstraction (Mens et al., 2005; Engels et al., 2001). This is somewhat different to our approach, since we are only interested in the consistency of a set of requirements, which is captured altogether in one single model. Therefore, our model always complies to the AUTOSAR meta-model. Therefore, many types of inconsistencies, for example syntactical inconsistencies, do not arise. Nevertheless, further rules, which describe the static semantics of AUTOSAR, are defined textually and are only checked partially.

In (Mahmud et al., 2016) structured requirements specifications are transformed into SAT formulae to find logical inconsistencies by identifying antonyms. In the context of product line engineering, Mendonca et al. (Mendonça et al., 2009) check feature models by transformation into propositional logic.

In (Post et al., 2011b) and (Post et al., 2011a) system requirements are formulated in the real-time logic *Duration Calculus* (Chaochen et al., 1991). Verification of the properties is performed by transformation of every requirement into a *Phase Event Automaton (PEA)* and subsequently into an UPPAAL-verifiable timed automaton.

Verification of timed automata-based real-time systems is presented e.g. in (Kim et al., 2015), where a novel analysis framework is shown, which combines symbolic and statistical model checking in UPPAAL to speed up verification runtime. A similar approach for consistency checking timing requirements is presented in (Toennemann et al., 2018). In contrast to our approach the authors use timing requirements

and a system design model to check for consistency, whereas we propose a multi-step approach where we only employ the requirements set itself for the consistency checking and only add the system architecture for verification purpose.

7 CONCLUSION

In this paper we presented an approach to support the early validation of AUTOSAR software architectures by simplifying the process of AUTOSAR timing requirements specifications. We presented an approach for checking consistency of AUTOSAR timing requirements, which improves the requirements quality. Especially by computing the maximum set of satisfiable timing requirements the user can easily identify faulty model elements. We showed that the consistency checking only takes a fraction of time in comparison with the verification task, which has to be performed afterwards. Thus checking consistency before verification can be very beneficial in the case of an inconsistent requirement set, but does not significantly slow down the whole verification process even in the case of consistent requirements sets. In the future we will evaluate our approach on more realistic examples.

REFERENCES

Abdelhalim, I., Schneider, S., and Treharne, H. (2011). Towards a Practical Approach to Check UML/fUML Models Consistency Using CSP. In *ICFEM 2011*, volume 6991, pages 33–48. Springer Berlin Heidelberg.

André, C. (2009). *Syntax and Semantics of the Clock Constraint Specifcation Language (CCSL)*. Phd., Inria Institute, Sophia Antipolis.

AUTOSAR (2019). AUTomotive Open System ARchitecture Methodology. https://www.autosar.org/fileadmin/user_upload/standards/classic/19-11/AUTOSAR_TR_Methodology.pdf.

Bauer, A., Leucker, M., and Streit, J. (2006). SALT - Structured Assertion Language for Temporal Logic. In *ICFEM 2006*, volume 4260 of *LNCS*. Springer.

Behrmann, G., David, R., and Larsen, K. G. (2004). A tutorial on uppaal. In *Formal Methods for the Design of Real-Time Systems*, pages 200–236. Springer.

Beringer, S. and Wehrheim, H. (2016). Verification of AUTOSAR Software Architectures with Timed Automata. In *FMICS-AVoCS 2016*, volume 9933 of *LNCS*, pages 189–204. Springer.

Bjorner, N. and Phan, A.-D. (2014). vZ - Maximal Satisfaction with Z3. In *SCSS 2014*, volume 30 of *EPiC Series in Computing*, pages 1–9. EasyChair.

Chaochen, Z., Hoare, C., and Ravn, A. P. (1991). A Calculus of Durations. *Information Processing Letters*, 40(5):269–276.

Derrick, J., Akehurst, D., and Boiten, E. (2002). A framework for UML consistency. In *UML 2002 Workshop on Consistency Problems in UML-based Software Development*, volume 2460 of *LNCS*, pages 182–196. Springer.

EAST-ADL Association (2013). EAST-ADL Domain Model Specification.

Engels, G., Heckel, R., and Küster, J. M. (2001). Rule-Based Specification of Behavioral Consistency Based on the UML Meta-model. In *UML 2001*, volume 2185 of *LNCS*, pages 272–286. Springer Berlin Heidelberg.

Garey, M. R., Johnson, D. S., and Stockmeyer, L. (1976). Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3):237–267.

Gruhn, V. and Laue, R. (2006). Patterns for Timed Property Specifications. In *Electronic Notes in Theoretical Computer Science*, volume 153, pages 117–133.

IEEE (1998). IEEE 830-1998 Recommended Practice for Software Requirements Specifications.

Jacobs, J. and Simpson, A. (2017). On the formal interpretation and behavioural consistency checking of SysML blocks. *Software & Systems Modeling*, 16(4):1145–1178.

Kalibatiene, D., Vasilecas, O., and Dubauskaite, R. (2013). Rule Based Approach for Ensuring Consistency in Different UML Models. In *Information Systems 2013*, volume 161 of *LNBIP*, pages 1–16. Springer.

Kim, J. H., Larsen, K. G., Nielsen, B., Mikucionis, M., and Olsen, P. (2015). Formal Analysis and Testing of Real-Time Automotive Systems Using UPPAAL Tools. In *FMICS 2015*, volume 9128 of *LNCS*, pages 47–61. Springer.

Klein, F. and Giese, H. (2007). Joint Structural and Temporal Property Specification Using Timed Story Scenario Diagrams. In *FASE 2007*, volume 4422 of *LNCS*, pages 185–199. Springer.

Kotb, Y. and Katayama, T. (2005). Consistency Checking of UML Model Diagrams Using the XML Semantics Approach. In *WWW 2005*, pages 982–983. ACM.

Kroening, D. and Strichman, O. (2008). *Decision Procedures: An Algorithmic Point of View*. Springer Berlin Heidelberg.

Mahmud, N., Seceleanu, C., and Ljungkrantz, O. (2016). ReSA Tool: Structured Requirements Specification and SAT-based Consistency-checking. In *FedCSIS 2016*, pages 1737–1746. IEEE.

Mallet, F. and Simone, R. d. (2015). Correctness issues on MARTE/CCSL constraints. *Science of Computer Programming*, 106:78–92.

Mendonça, M., Wasowski, A., and Czarnecki, K. (2009). SAT-based Analysis of Feature Models is Easy. In *SPLC 2009*, pages 231–240. ACM.

Mens, T., Van der Straeten, R., and Simmonds, J. (2005). A Framework for Managing Consistency of Evolving UML Models. In *Software Evolution with UML and XML*, pages 1–30. IGI Global.

Post, A., Hoenicke, J., and Podelski, A. (2011a). rt-inconsistency: a new property for real-time requirements. In *FASE 2011*, volume 6603 of *LNCS*. Springer.

Post, A., Podelski, A., and Hoenicke, J. (2011b). Vacuous real-time requirements. In *RE 2011*, pages 153–162. IEEE.

Rasch, H. and Wehrheim, H. (2003). Checking Consistency in UML Diagramms. In *FMOODS 2003*, volume 2884 of *LNCS*, pages 229–243. Springer.

Richter, K. (2005). *Compositional Scheduling Analysis Using Standard Event Models: The SymTA/S Approach*. Dissertation, Braunschweig.

Seifert, T., Jug, F., and Rackl, G. (2005). Automated Quality Assurance for UML Models. In *Informatik 2005 - Informatik LIVE!/2*, volume 68 of *LNI*, pages 496–500. GI.

Simmonds, J. and Bastarrica, M. C. (2005). A tool for automatic UML model consistency checking. In *ASE 2005*, page 431. ACM.

Teige, T., Tom, B., and Holberg, H. J. (2016). Universal Pattern: Formalization, Testing, Coverage, Verification and Test Case Generation for Safety-Critical Requirements. In *MBMV 2016*. Albert-Ludwigs-Universität Freiburg.

Toennemann, J., Rausch, A., Howar, F., and Cool, B. (2018). Checking Consistency of Real-Time Requirements on Distributed Automotive Control Software Early in the Development Process Using UPPAAL. In *FMICS 2018*, volume 11119 of *LNCS*, pages 67–82. Springer.