




Systematic Synthesis of Energy-Aware Timing Models in Automotive Software Systems

Padma Iyengar^(✉) 

Software Engineering Research Group, University of Osnabrueck, Osnabrück, Germany
piyengha@uos.de

Abstract. In automotive embedded software, functions have several performance requirements such as timing, energy, safety and reliability. For such complex software architectures, an early evaluation and decision on the best set of performance configuration (e.g. timing vs energy trade-offs) could save costly corrections of potential errors in the design. For example, appropriate performance analysis workflows and frameworks if employed already during early design stages, allow us to understand the performance aspects and behavior of the system depending on software and hardware characteristics. The main input required for such analysis is the performance-analysis model based on the underlying design model. In this context, this chapter presents a workflow for synthesis of energy-aware timing analysis models for AUTOSAR-based embedded software systems developed using the Unified Modeling Language (UML)/Systems Modeling Language (SysML) domains. A prototype of the model transformations for the synthesis of the energy-aware timing models and its evaluation in an automotive use case is presented.

Keywords: Energy-aware timing model · AUTOSAR · Unified modeling language (UML) · Synthesis · Meta-model · Model transformation

1 Introduction

The Automotive Open System ARchitecture (AUTOSAR) [3] has been created as a worldwide development partnership of vehicle manufacturers, suppliers, service providers and companies from the automotive electronics, semiconductor and software industry. To achieve the technical goals of modularity, scalability, transferability, and function reusability, AUTOSAR provides a common software infrastructure based on standardized interfaces for the different layers [37]. While doing so, AUTOSAR employs component-based software architecture, for the design and implementation of automotive software systems. With the standardized layer between application software and the hardware of an Electronic Control Unit (ECU)¹, the software is largely independent from any chosen micro controller and car manufacturer, making it reusable for several individual ECU systems.

¹ An embedded system that controls one or more of the electrical systems or subsystems in a vehicle.

At this juncture, the automotive industry not only continues to expand rapidly but also is becoming increasingly complex and heterogeneous with the adoption of multi and many-core processors systems. Further, in automotive embedded architectures, functions have several performance requirements such as timing and energy. For such complex software/system architectures, an early evaluation and decision on the best set of performance configuration (e.g. timing, timing vs energy trade-offs), could save costly corrections of potential errors in the design. For example, appropriate performance analysis workflows and frameworks if employed already during early design stages, allow us to understand the performance aspects and behavior of the system depending on software and hardware characteristics. Further, they help to explore different design architectural choices and quantitatively evaluate their implications on system performance.

On the other hand, the scientific effort provided by academic institutions often does not match the needs of the industry as the proposed solutions fail to consider the state-of-the-practice challenges [36]. Some emerging challenges in the context of performance analysis are, integrating (specification/modeling) performance aspects in the early design stage, an automated synthesis of performance models (e.g. timing, reliability, safety, energy) and early model-based performance analyses in modeling tools or specialized performance analyses tools. In this context, this chapter contributes to the particular aspect of early model-based synthesis of energy-aware timing models in AUTOSAR-based embedded software systems modeled using UML/SysML domain.

1.1 State-of-the Practice by Automotive Organizations

In the race to provide model-based tool support (e.g. architecture design, automatic code generation) for AUTOSAR-based Embedded Software Engineering (ESE) in the Unified Modeling language (UML) [44]/Systems Modeling Language (SysML) [41] domain, UML tools such as Enterprise architect (EA) [8] and IBM Rhapsody Developer [17] emerged as front runners. For instance, AUTOSAR-related UML/SysML profiles for the architectural description of an AUTOSAR model that uses the native AUTOSAR concepts is supported by Rhapsody and EA. At this juncture, a majority of the state-of-the-practice in the automotive industry is that, UML is used at higher abstraction levels, for instance, to create descriptive UML models that describe the overall software and system architecture.

The descriptive models produced in the UML/SysML domain, are then used for various purposes such as (a) to produce more fine grained architecture of the prescriptive models (e.g. using Simulink) and (b) debugging using model execution frameworks² in the context of realistic mock-ups of the intended user interface. On the other hand, the automotive software is loaded with numerous non-functional requirements. During the software architecture design of such systems, several non-functional parameters need to be taken into consideration, optimized and fine tuned. Some examples are, studying timing versus energy trade-offs and minimizing CPU load vs meeting safety goals. To achieve this, the non-functional properties such as timing, energy and safety need to be specified in the UML/SysML-based early design model. With this annotated design

² <https://www.nomagic.com/product-addons/magicdraw-addons/comeo-simulation-toolkit>.

model as input, a performance analysis model (such as timing/energy/safety model) needs to be synthesized. Such an analysis model can then be used for early performance validation (such as timing analysis in specialized timing analysis tool [11]) and trade-off studies.

1.2 Relation to Author's Previous Work and Novel Contributions

In the above context, a systematic series of steps towards extraction and synthesis of timing analysis models in AUTOSAR-based embedded system design models which are developed in UML tools has been presented in [22]. In this book chapter, the work in [22] is extended and the following novel contributions are presented.

- Extension of the framework introduced in [22] to include energy properties in the AUTOSAR-design model (developed in UML/SysML tools) with the help of stereotypes from the MARTE profile [26].
- Mapping of the energy properties to a generic timing-energy meta model.
- A prototype implementation of the model transformations using Atlas Transformation Language (ATL) [2] in Eclipse Modeling Framework (EMF) [7] for synthesis of energy-aware timing analysis model from AUTOSAR-based design model.
- Evaluation of the above prototype in a practical automotive use case (introduced in [22]).

In the remainder of this paper, background and related work is presented in Sect. 2. The proposed approach for synthesis of energy-aware timing analysis model for AUTOSAR-based design model developed in UML/SysML domain is presented in Sect. 3. An experimental evaluation in an automotive case study is presented in Sect. 4. Section 5 concludes the paper.

2 Background and Related Work

In this section, background and related work pertaining to general modeling options for automotive embedded software systems is presented in Sect. 2.1. In Sect. 2.2, related work on model-based timing and energy specifications and a brief background on AUTOSAR-TE and the MARTE profile are provided. In Sect. 2.3, related work and background on model-based timing and energy analysis is presented.

2.1 Modeling Automotive Embedded Software Systems

Automotive embedded software applications are different than typical embedded software applications that we find on smart devices such as phones, gadgets, etc. In the automotive applications, real-time complex interactions across multiple-systems such as braking, steering, suspension, power-train, body-electronics, etc. are extremely crucial. A single feature might need interactions across 20 or more automotive embedded software applications spread across multiple ECU connected over multiple networking protocols. No single automotive embedded software application performs on its own, it

is always part of a much bigger system of systems [27]. To address the increasing complexity in development of such systems, Model Driven Development (MDD) [31], is considered as the next paradigm shift. In MDD, the requirements are specified as models at a higher abstraction level (e.g. using UML diagrams). They are then refined, starting from higher and moving to lower levels of abstraction, via model transformations.

Further, MDD methodology also provides support for analysis of non-functional properties such as timing and reliability parameters. For instance, UML supports generic system and software modeling and also UML profiles for specific aspects such as quality analysis. Some examples of employing UML for MDD and examining quality properties such as timing, energy and reliability are available in [21, 23, 35].

Matlab/Simulink (M/S) [28] is a popular example for a modeling tool with non-UML modeling language, which is established in the industry, including the automotive domain [10]. It is primarily employed for simulation studies and model-based development of control loops. Further, the Rubus Component Model (RCM) [5] and EAST-ADL are among other established solutions used within the vehicular domain.

AUTOSAR Framework. A promising approach is the standardization of the software architecture used in ECU development [29]. A comprehensive and well-established solution used in the automotive sector is the AUTOSAR standard [3]. It emphasizes to shift the ECU development from an ECU-centric approach to a functionality-based approach. AUTOSAR uses a component-based software architecture, with central modeling elements called *Software Components* (SWCs or SW-Cs). The SWCs describe a completed, self-contained set of functionality. The AUTOSAR methodology describes various steps, namely, *System configuration*, *ECU configuration and component implementation* involved in the development process. It also describes the artifacts created and interchanged between the steps. In between these steps, the ARXML file format [3] is used for the exchange of development artifacts, which is an XML-based file format. The functionality-based approach aims to specify the functions of the complete vehicle first in the so-called *system configuration*, and afterwards extract specifications for the suppliers to implement an ECU. This way, the automotive software can be interchanged on a function level instead of the ECU level, which increases its reusability.

The various components of the AUTOSAR framework are illustrated together with the mapping of software components to ECUs, in the system configuration step, in Fig. 1. The software components (seen at the top of Fig. 1, e.g., *SW-C1*) are used to structure the AUTOSAR model and group functionality into individual components. These components can be connected together, oblivious of the hardware they will be running on. This is handled by the *Virtual Function Bus* (VFB), which provides an abstraction layer for the SWC to SWC communication. Components distributed over different ECUs however, may use the network bus for communication. This is determined automatically by the *Run-Time Environment* (RTE), which is a communication interface for the software components. The lower part of the Fig. 1 represents the mapping of ECUs to SW-Cs in the system configuration step. Here, the ECUs 1, 2..n are seen communicating over a network bus (e.g. FlexRay, CAN). In each ECU (e.g. ECU 1 in lower part of Fig. 1), the RTE provides interfaces between SW-Cs (e.g. AUTOSAR

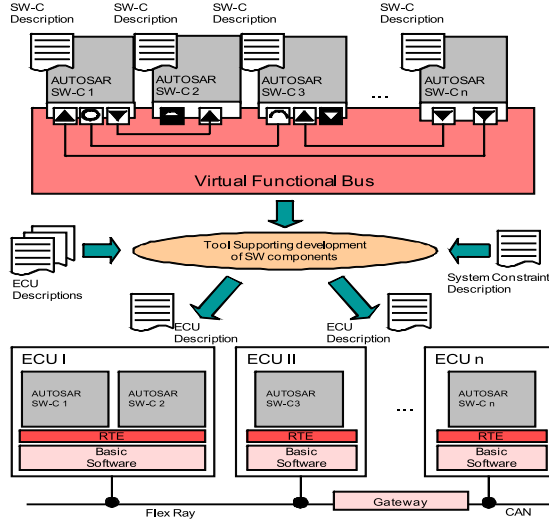


Fig. 1. Mapping of software components to ECUs [3,22].

SW-C 1 and AUTOSAR SW-C 2 in ECU 1) and between SW-C and basic software (BSW). Further it provides the BSW services (as API abstraction) to SW-C.

The underlying software functions which implement the given requirements are contained inside the SW-Cs. These are later on implemented manually by the software developers. The RTE and *Basic Software* (BSW) which are provided by third-party AUTOSAR software vendors are at the disposal of the developer for communication and hardware abstraction. The inner functionality of the application and sensor/actuator SWCs is defined in *Internal Behavior* elements. They encapsulate *Runnable Entities*, which correspond to atomic functions on the code level that are implemented later in the development process. The communication between the SWCs is modeled by using communication ports. In this paper, we deal with the system configuration step and specification of timing and energy properties in the SW-Cs.

2.2 Model-Based Timing Specification

Alternatives for specifying timing behavior in the UML domain have been introduced more than a decade ago [31]. Modeling and Analysis of Real-Time and Embedded Systems (MARTE) [26] is a standardized UML profile, which extends UML and provides support for modeling the platform, software and hardware aspects of an application. There are several approaches in the direction of model-based timing specification in the literature [1, 20, 33]. But, modeling constraints using AUTOSAR-TE and an automated extraction of timing parameters, synthesis of an analysis model and analysis of the timing analysis model in a state-of-the-art timing analysis tool [11], is missing. In this direction, a workflow for early synthesis of timing models in AUTOSAR-based automotive embedded software systems has been proposed by the author in [22]. In

this book chapter, the work in [22] is extended. Thereby, a workflow for synthesis of energy-aware timing models in AUTOSAR-based embedded software developed using UML/SysML domains is presented in this paper along with experimental evaluation.

There are also several modeling alternatives in non-UML domains such as SystemC [4], Event-B³ and Matlab/Simulink [28] to name a few. Unlike UML-based profiles, support for specification and analysis of timing properties is very limited in SystemC and Event-B. The newly introduced *System Composer* toolbox in M/S [42] provides system engineering capabilities in M/S. It supports creation of custom-defined profiles and custom-defined scripts to analyze the models based on the stereotype values as in the case of UML profile mechanisms and tools. However, there are no studies available yet on the usage of the new features in M/S for energy-aware timing analysis of automotive embedded software models.

Further, several modeling languages, domain-specific languages and a number of generic approaches have emerged that include timing behavior. *PTIDES* [6,45] and *Giotto* [15] provide a good basis for defining an approach to model timing requirements. However, these are often used to analyze system behavior rather than specification of timing properties. In the following a brief background on AUTOSAR-TE and MARTE are provided as they are used in this paper to annotate the AUTOSAR design model with timing and energy properties respectively.

AUTOSAR-Timing Extensions (TE). The AUTOSAR-Timing Extensions (TE) metamodel is separate from the AUTOSAR metamodel, in order to leave the option whether to provide timing specifications or not. They feature an event-based model for the description of the software's temporal behavior and can be defined on top of a system architecture. The AUTOSAR release with timing extensions and own timing model, finds extensive usage in the automotive industry. This is supported by studies including [9, 13, 34].

The TE metamodel (Fig. 2) provides five different views for timing specification, depending on what kind of timing behavior of the AUTOSAR model is described [3]. The five views are *VfbTiming*, *SwcTiming*, *SystemTiming*, *BswModuleTiming* and *EcuTiming*. In the experimental evaluation, the *SwcTiming* view is employed, as in the system configuration step and timing specification step the SWCs are employed (cf. Sect. 2.1). *SwcTiming* view describes the internal behavior timing of software components. Further explanation of AUTOSAR methodology and AUTOSAR-TE are not provided here because of space limitations (interested readers are referred to [3]).

MARTE Profile. The MARTE profile [26] standardized by the OMG [31] is primarily aimed at modeling and analysis of real time and embedded systems. It is a popular standard which introduces a domain view for time modeling and defines standard UML elements to express timing concepts of real time and embedded systems. MARTE also enhances the UML to support value units with the aid of a Value Specification Language (VSL). Further, the profile extends the UML to be able to model a platform, on which a software application is executed and how the deployment of the software to the platform

³ <http://www.event-b.org/index.html>.

2.3 Model-Based Timing and Energy Analysis

The specified timing behavior in the design model can be analyzed using dedicated timing analysis tools. There are several open source tools such as Cheddar [40] and MAST [14]. Some popular proprietary timing analysis tools include chronSIM [19], Gliwa T1. timing suite [11] and Timing Architect [43]. These tools are independent of the modeling languages used. Therefore, they require the timing specifications to be in a particular format, although some provide import functions for common modeling languages. However, the timing analysis carried out in such tools are very late in the development process. It is imperative to note that the design errors realised from such late timing analysis would be costly to fix at a later development stage. Hence, an early model-based timing analysis is necessary to overcome this drawback.

On the other hand, there is no tool support for automated synthesis and export of AUTOSAR-based timing analysis model (from AUTOSAR-based application design model in UML tools) to these timing analysis tools. In the literature, AUTOSAR-TE were used for a model-based timing analysis in works such as [24] and [38]. Further, a review of the literature shows that there is no systematic model-based approach for timing or energy analysis of AUTOSAR-based systems. Except for [22], there exists no related work on early synthesis of timing models for model-based timing analysis of AUTOSAR-based systems.

Further, a related work in [21] deals with a model-driven workflow for energy-aware scheduling analysis of IoT enabled use cases. It carries out energy-aware timing analysis (of UML models) of IoT use cases in state-of-the-art (timing) analysis tools. However, ready made support for energy-aware timing analysis is not available in any of the state-of-the-art timing analysis tools. Hence, in [21], an additional tool-plugin is implemented in a timing analysis tool to include the energy properties and carry out energy-aware timing analysis. Note that the workflow in [21] deals with the synthesis and analysis of energy-aware timing models from hand-written IoT code. Thus, in the literature, there is no published work dealing with the synthesis of energy-aware timing models in AUTOSAR-based embedded software systems developed using UML/SysML domain.

Addressing this gap and in line with the novelties outlined in Sect. 1.2, in the remainder of this paper, the proposed workflow for synthesis of energy-aware timing models in AUTOSAR-based systems and an experimental evaluation are presented in Sect. 3 and 4 respectively.

3 Workflow for Synthesis of Energy-Aware Timing Models

The proposed workflow for a systematic integration of the energy and timing performance requirements in the AUTOSAR-design model and the automated synthesis of an AUTOSAR-based energy-aware timing analysis model is presented in this section. A series of steps involved in this systematic synthesis of energy-aware timing analysis models incorporated in the AUTOSAR development process shown in Fig. 3. The steps involved in the workflow are described in Sect. 3.1. The custom-defined generic timing-energy metamodel used in the workflow is described in Sect. 3.2. The mapping among elements in the AUTOSAR metamodel and the custom metamodel (from Sect. 3.2) w.r.t

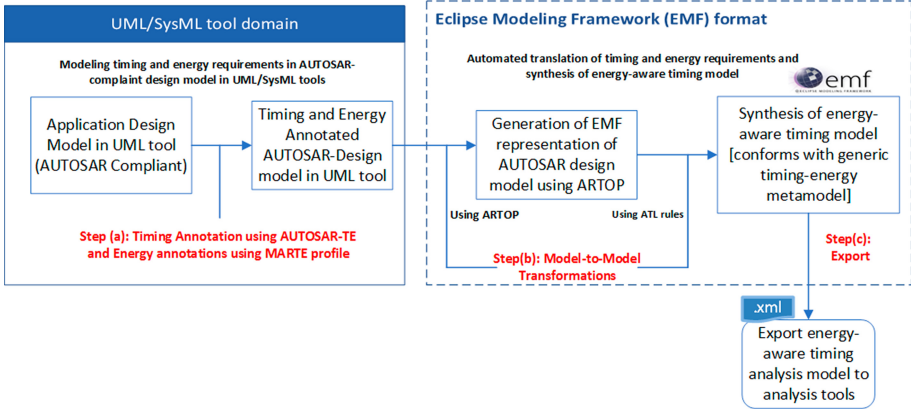


Fig. 3. Steps involved in synthesis of energy-aware timing analysis model incorporated in AUTOSAR development process.

timing properties is described in Sect. 3.3. Similarly, mapping between MARTE stereotypes and custom-defined metamodel (from Sect. 3.2) for energy properties is described in Sect. 3.4. An overview of the M2M transformations used in the workflow is provided in Sect. 3.5.

3.1 Steps Involved in the Synthesis of an Energy-Aware Timing Analysis Model

The proposed workflow for integrating the energy and timing performance requirements in the AUTOSAR-design model and the automated synthesis of an AUTOSAR-based energy-aware timing analysis model is shown in Fig. 3. It comprises of the following steps:

1. In the first step (step (a) in Fig. 3), it is considered that an initial AUTOSAR-based design model of the automotive embedded software application under consideration is already modeled in an UML/SysML tool [8, 17]. Note that step-(a) in Fig. 3 is applied in an early stage of development process. It involves the specification of the timing and energy requirements in the AUTOSAR-based design model using AUTOSAR-TE and MARTE profile respectively. The output of this step is a timing and energy-annotated AUTOSAR-based design model.
2. In line with the main scope of this paper, an AUTOSAR-based energy-aware timing analysis model needs to be synthesised based on the inputs from step-(a) in Fig. 3. For this purpose, given the energy-aware timing annotated design model as input, Model-to-Model (M2M) transformations are implemented for extracting the timing and energy properties. This results in the synthesis of the AUTOSAR-based energy-aware timing analysis model (conforming to a generic metamodel, cf. Sect. 3.2). Thus, the output of step-(b) in Fig. 3 is the synthesized energy-aware timing analysis model.

Note that this resulting model from step (b) can be used for performance validation such as energy-aware timing analysis and trade-off studies. Thus the output (model)

from step (b) may be exported (cf. step (c) in Fig. 3), for instance in XML format, to industry standard analysis tools [11,43].

3.2 Generic Timing-Energy Metamodel

A metamodel comprising a set of timing and energy properties is required for the Model-to-Model (M2M) transformations in step-(b) in Fig. 3. A generic, custom-defined metamodel for energy-aware timing analysis introduced in [21], for energy-aware timing analysis of IoT-compliant use cases, is employed in this paper to synthesize an energy-aware timing analysis model for AUTOSAR-based embedded software systems developed using UML/SysML tools. This metamodel bears similarity to the AUTOSAR metamodel with respect to the software and hardware architecture elements. It can be termed as a generic metamodel, as it closely adheres with timing models used in several timing validation tools [11,43]. A simplified view of the custom-defined timing-energy metamodel is shown in Fig. 4.

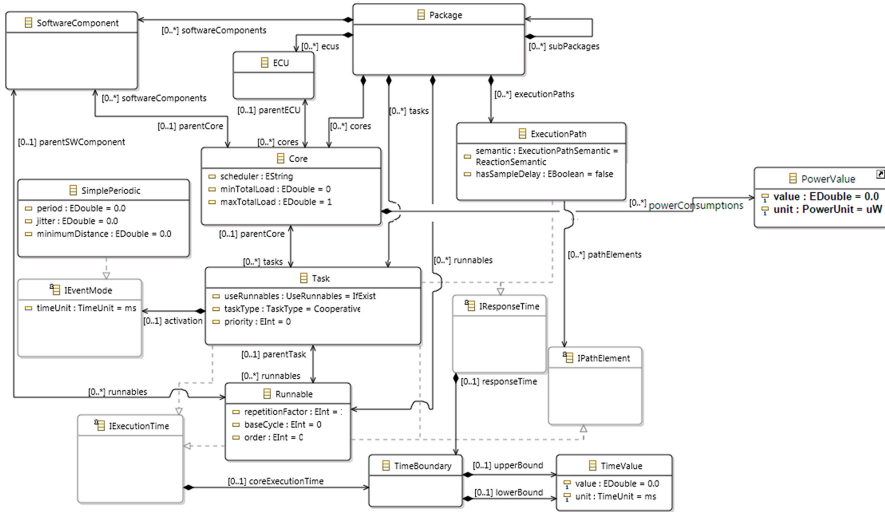


Fig. 4. Excerpt of the timing-energy metamodel [21].

From Fig. 4, it can be seen that the metamodel comprises a package with the elements required for an energy-aware timing evaluation of a software system, in a hierarchy. It consists of elements such as *Packages*, containing the different model elements such as *Runnables* (e.g. an operation), *SoftwareComponents*, *Tasks*, *Cores*, *ECUs* and *ExecutionPaths*. A task may or may not have a *trigger*, depending on its activation. Each task and runnable comprises an attribute to store the execution time. This is used as an input for timing analysis. A result of timing validation, namely the *response time* is an attribute for tasks.

Similarly, the power consumption modes (along with their average power rating) specified in the UML design model can be mapped to the attribute *PowerValue* for a core. Please note that only a simplified (yet sufficient) view of the timing-energy analysis model is presented here, because of space limitations. Interested readers are referred to [22] for a detailed description of the timing elements in the metamodel.

3.3 Mapping Among Metamodels for *Timing Properties*

In this section, the relevant metamodel elements from the custom-defined intermediate timing-energy metamodel (cf. Sect. 3.2, Fig. 4) are mapped to their counterparts in the AUTOSAR-TE metamodel [3]. The AUTOSAR Tool Platform⁴ provides an EMF model, which contains the element names as per specification. An evaluation version of this AUTOSAR EMF model is used in this paper for mapping the timing metamodel elements to the AUTOSAR metamodel elements. It is also used as an input metamodel for the automated model transformations (cf. Sect. 3.5). A summary of relevant mappings of elements is shown in Table 1. In the following, these mappings are described in more detail.

Table 1. Mapping of timing-related elements in proposed generic metamodel (in Fig. 4) to AUTOSAR elements.

Nr	Timing element in Fig. 4	AUTOSAR element	Description
1	Model	AUTOSAR	Top-level model element
2	Package	ARPackage	Structuring element
3	SoftwareComponent	AtomicSwComponentType	Encapsulates functionality
4	Runnable	RunnableEntity	Executable operation
	period	Period of TimingEvent	Period of operation
	coreExecutionTime	LatencyTimingConstraint	Execution time of runnable
	order	RtePositionInTask	Execution order of runnable
	baseCycle	RteActivationOffset	First runnable execution
	repetitionFactor	runnable period/task period	How often it is executed
5	ECU	EcuInstance	Electronic control unit
6	Core	HwElement	Processing core
	period	OsSecondsPerTick	Seconds per clock tick
7	System	System	Network of ECUs
8	Task	OsTask	Schedulable unit
	priority	OsTaskPriority	Fixed priority of task
	taskType	OsTaskSchedule	Preemptability of task
	synchronizationMechanism	OsAlarmCounterRef	Reference clock
	synchronizationOffset	OsAlarmAlarmTime	Offset for the reference clock
	activation	OsAlarmCycleTime	Periodic task activation
9	ExecutionPath	TimingDescriptionEventChain	End-to-end path

⁴ <https://www.artop.org/>.

1. The top-most element of every AUTOSAR model is the *AUTOSAR* element. It denotes the AUTOSAR revision and links to the corresponding XML schema definition. This element is mapped to the *Model* element, as it represents a dedicated model. Note that this element in Table 1 is not shown in the Fig. 4.
2. The *ARPackage* element gets mapped to the *Package* timing element, as it structures the different AUTOSAR elements in packages and subpackages.
3. The mapping of software components is straightforward, because these elements exist similarly as central modeling elements in the AUTOSAR standard. Every *AtomicSwComponentType* of the AUTOSAR application model is mapped to a *SoftwareComponent* in the timing metamodel. This includes *SensorActuatorSwComponentTypes* and *ApplicationSwComponentTypes*, as they inherit from the atomic software component type.
4. The *Runnable* timing elements exist in AUTOSAR inside the *InternalBehavior* of an *AtomicSwComponentType* as *RunnableEntities*. They represent the executable operations of the software components.
5. The *ECU* elements can be mapped to the AUTOSAR *EcuInstance*. This is used for linking the software components, and therefore runnables, to their dedicated ECUs, on which they are later on implemented and executed.
6. The *Core* elements are mapped to *HwElements* in the AUTOSAR model. They need to be linked to a *HwCategory* of the type *ProcessingCore*. Each core belongs to an ECU and is linked to it in the system mapping.
7. The *System* element in timing metamodel corresponds to a *System* element AUTOSAR model. Overall, they represent a top-level element corresponding to a network of ECUs.
8. *Task* elements are created in the AUTOSAR *Os* configuration as *OsTasks*. A task is defined as a schedulable unit in timing analysis.
9. The end-to-end *ExecutionPaths* in the timing metamodel can be represented in the AUTOSAR model as *TimingDescriptionEventChains*. These event chains group a set of events belonging to the activation and termination of runnable entities.

Note that in place of the custom-defined but generic metamodel used in this paper, an open source metamodel namely, AMALTHEA⁵, may be employed for M2M related to timing properties. However, it does not provide ready made support (i.e., elements) for modeling energy characteristics. Hence, in this paper we have employed our custom-defined generic metamodel.

3.4 Mapping Between MARTE Stereotypes and Custom-Defined Timing-Energy Metamodel for *Energy* Properties

For annotating the energy properties, the underlying CPU configuration modes (with power consumption values of the microcontroller) are taken into consideration. This can be obtained from measurements or from data sheets, for instance in the case of Commercial Off-the-shelf (COTS) products. The mapping between the power configuration modes for the CPU core can be added using the *HWComponent* stereotype from MARTE profile (with tagged value *staticConsumption*) [26]. The

⁵ <https://www.eclipse.org/app4mc/>.

HWComputingResource stereotype (with tagged value *resMult*) indicating multiplicity of the processing modes, can be additionally used to link the processing modes to the cores with the tag value *processingUnits* of *SaExecHost* stereotype. Please note that, the aforementioned stereotypes are selected based on an analysis of support for energy modeling in MARTE profile and the requirements for a first hand energy-aware timing analysis of the AUTOSAR-based design models, proposed in this paper.

Table 2. Stereotypes used from MARTE profile for energy/power annotations in the design model and their mapping to elements in Fig. 4.

MARTE Stereotype	Tagged Values	Description	Mapping to element in Fig. 4
SaExecHost	mainScheduler, schedPolicy, utilization, isSched	CPU core and related configuration	Core (also <i>HWElement</i> in AUTOSAR), runnable, task
HwComponent	staticConsumption	Average power consumption per processing mode	PowerValue
HwComputingResource	resMult	Linking various core configurations	powerConsumptions in PowerValue

The mapping between the MARTE stereotypes mentioned above and the corresponding elements in the custom-defined timing-energy metamodel in Fig. 4 are shown in Table 2. Each core element in Fig. 4 may comprise of a *PowerValue* denoting the power consumption values of the underlying microcontroller. Thus, the power consumption values specified using the *HwComponent* stereotype from MARTE profile with tagged value *staticConsumption*, are mapped to the *PowerValue* element denoted in Fig. 4. The various core configurations and their power values from *HwComponent* stereotype represented by the *HwComputingResource* stereotype correspond to the multiplicity *powerConsumptions* in *PowerValue* element in the metamodel in Fig. 4. These are required to link the various core configurations (e.g., power configuration modes). Note that, for a first hand energy-aware timing analysis, the power configuration models of the underlying hardware element (one or more cores) are taken into consideration.

3.5 Model-to-Model (M2M) Transformations

As seen in Fig. 3, after step (a), a timing and energy annotated AUTOSAR-based design model is now available in the UML/SysML tool under consideration. It can be exported from the tool as an ARXML file [3] as input for step (b) in Fig. 3. Note that while employing Model-to-Model (M2M) transformations, both source and target models must conform with their respective metamodels. Here the *source model* is the *timing and energy annotated AUTOSAR design model* obtained from the system description specification in the UML/SysML tool in ARXML format. This conforms with the AUTOSAR metamodel [3]. The *target metamodel* is the *custom-defined generic timing-energy metamodel* introduced in Sect. 3.2. During the M2M transformations the timing and energy properties are extracted from the annotated AUTOSAR-based design model (source model) and a corresponding instance of the energy-aware timing analysis target metamodel is synthesized. Note that here both the metamodels are available in EMF format.

The synthesized analysis model is also available in EMF and XML formats. This model may now be used for performance validation such as energy-aware scheduling.

In this work, the ATLAS transformation language (ATL) [2] is used for implementing the M2M transformations. ATL is a widely used M2M transformation language and readily available as a plug-in for Eclipse development environment. Thus, using ATL a set of rules can be written to transform the AUTOSAR-based design model to an instance of the intermediate timing-energy meta model, based on the mappings listed in Table 1 and 2. The ATL implementation of the transformations in the prototype implementation of the workflow follows the regular structure of ATL transformations [2]. As stated earlier, the source model, M2M transformation and target models each have their own separate metamodels, which are each based on a common metamodel (ECORE) [7]. The model transformations are implemented as an ATL module, *AUTOSARinUML2TimingEnergy.atl*. These are generic implementations which can be applied across any use case satisfying the source and target models used in the ATL implementations. The implementation specifics of this module are explained in detail in the next section along with examples from the use case.

4 Autonomous Emergency Braking System (AEBS)

This AEBS use case is introduced already in [22]. Since this book chapter is an extension of the work done in [22], only a brief introduction about the AEBS use case is provided here. The main purpose of AEBSs is to warn the driver in case of an imminent frontal collision. This happens through visual and acoustic warning signals as a first step, followed by a tactile warning as the next level. The AEBS in cars use the Time-To-Collision (TTC) value [16,25] to estimate the danger of the situation. It is defined as the *time left until a collision happens*, if every object continues to move at the same speed. To calculate TTC, AEBS needs data such as the distance to frontal objects (e.g. from radar sensors) and wheel speed sensor input at certain speed ranges.

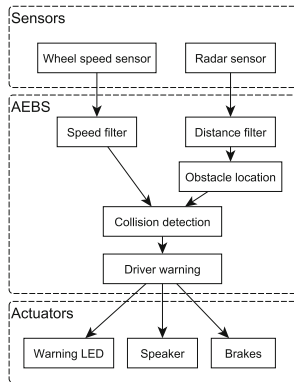


Fig. 5. Control flow and modules of the AEBS.

The control flow and modules of the AEBS are shown in Fig. 5. The AEBS is connected to sensors such as speed and radar sensors and actuators such as the warning LED, speaker and brakes via a software interface. Thus information such as the speed of the car in ms^{-1} (from *wheel speed sensor*), distance in m and relative speed in ms^{-1} (from *radar sensor*) are provided as inputs to the AEBS system. The output from AEBS system can be referenced using port interfaces, which must be processed by the corresponding actuator and issue a corresponding output (e.g. applying brakes, issuing warning signal).

4.1 AUTOSAR Design Model

The AUTOSAR system description of the AEBS is modeled using the IBM Rational Rhapsody Developer modeling tool [18]. Rhapsody is among the most popular UML modeling tool with AUTOSAR support used in the automotive industry. It also supports straight forward usage of the MARTE profile for energy annotations required for the workflow in Fig. 3. The MARTE profile can be added to the model and its stereotypes can be applied to the model elements directly, hence the choice of the tool.

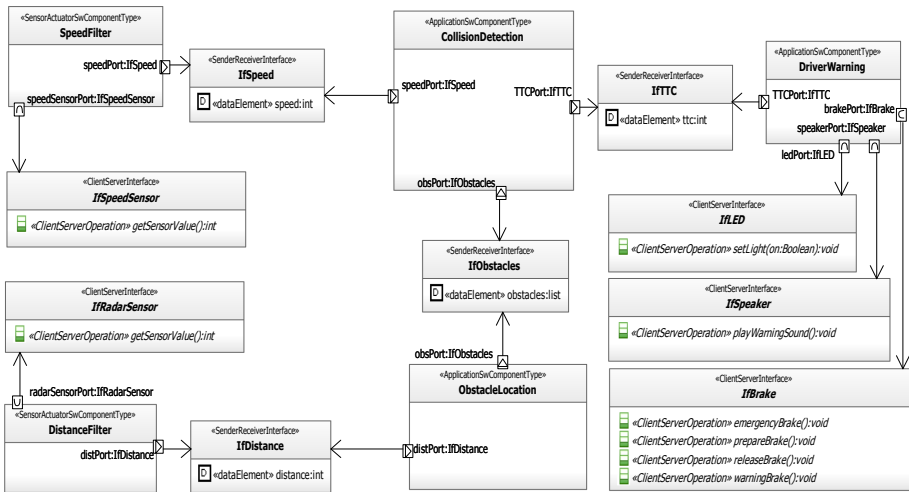


Fig. 6. Software components of AEBS in software component diagram modeled in Rhapsody.

The first step in implementing the AUTOSAR design model is to define the software components, of which the system is composed of as shown in Fig. 6.

- The sensor filter modules on the left-hand side are modeled as *SensorActuatorSwComponentTypes*. They have client ports (*speedSensorPort*, *radarSensorPort*) to be able to connect to the corresponding sensors. These ports are typed by *ClientServerInterfaces* that provide an operation for retrieving the sensor value. This

is illustrated by the association between the ports and the interfaces, which is stereotyped as a *portType*. The rest of the modules are modeled as *ApplicationSwComponentTypes*, as they do not directly represent a sensor or an actuator.

- The communication between the sensor filters and the *CollisionDetection* and *ObstacleLocation* components happens through sender/receiver ports. The filtered *dataElements* get sent to the processing components. Equally, the *ObstacleLocation* sends a list of obstacles (comprising of distance and relative speed) to the *CollisionDetection*. The communication between *CollisionDetection* and *DriverWarning* is also typed as sender/receiver and the corresponding *dataElement* is the TTC value.
- In the end, the *DriverWarning* component is connected by client ports (*ledPort*, *speakerPort* and *brakePort*) to the three actuators. The corresponding interfaces provide the necessary operations for the different levels of driver warning, e.g., setting the warning LED light status (*setLight*), playing a warning sound (*playWarningSound*) or performing an emergency brake (*emergencyBrake*).

Thus, the modules for the AEBS use case shown in Fig. 5 are modeled as AUTOSAR software components in the UML tool [17], as seen in Fig. 6.

4.2 Timing Specification

The timing constraints of the AEBS are added to the model in Fig. 6 with the help of AUTOSAR-TE in the UML tool Rhapsody. Figure 7 shows a latency constraint for the *checkTTC* runnable entity of the *DriverWarning* software component (seen at top-right of Fig. 6). An *SwcTiming* is created for each software component in the AEBS, which link to the component’s internal behavior with the *Lbehavior* association. Inside these elements, two *TDEventSwcInternalBehaviors* are defined for each runnable entity (in this case, *checkTTC* of *IBDriverWarning*). The first event highlights the activation of the runnable, while the second highlights the termination. This is defined by setting the

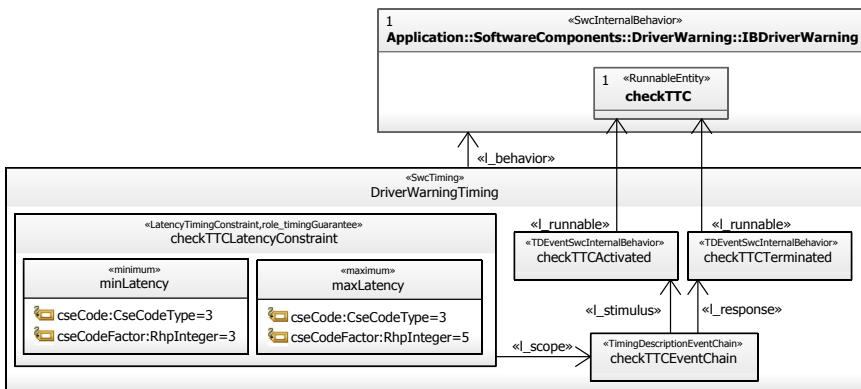


Fig. 7. Timing attributes for the *checkTTC* runnable entity.

tag *tdEventSwcInternalBehaviorType* of the timing event to either *runnableEntityActivated* or *runnableEntityTerminated*. Both these events are now used to form a *Timing-DescriptionEventChain*, in which the event chain stimulus is the runnable activation and the event chain response is the runnable termination.

Finally, the core execution time of the runnable *checkTTC* is specified by the *checkTTC*LatencyConstraint that links to its event chain with *l_scope*. The role *timing-Guarantee* stereotype declares that this constraint is the expected execution time instead of a requirement (*role_timingRequirement*). The related timing information can be given as maximum and minimum execution time and is specified by ASAM CSE codes [39]. The *cseCode* specifies the time base (e.g., 2 = 100 μs, 3 = 1 ms and 4 = 10 ms) and the *cseCodeFactor* determines an integer scaling factor. Thus, in this case, the execution time of the *checkTTC* runnable entity lies between 3 ms and 5 ms.

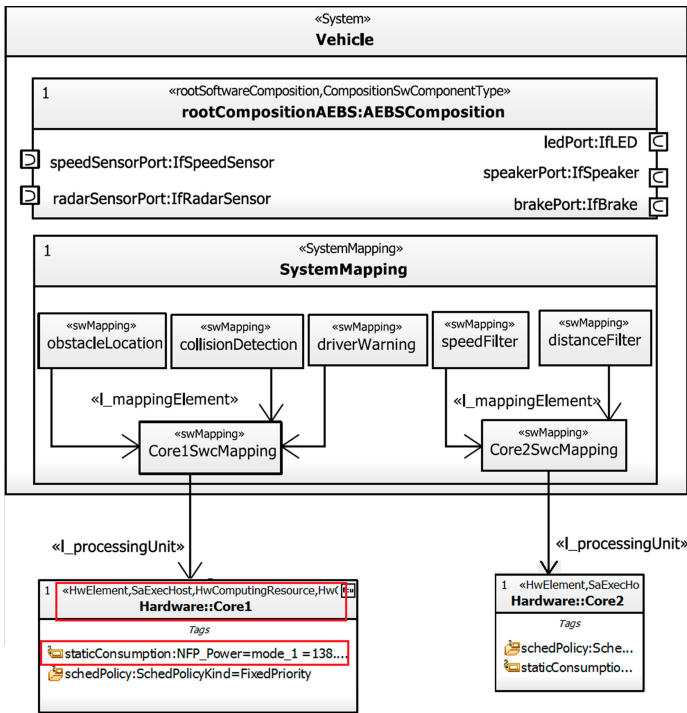


Fig. 8. System diagram containing the system mapping, root software composition and energy annotations for the hardware elements with processing modes for AEBS use case.

4.3 Specification of Energy Properties

In the custom-defined generic metamodel in Fig. 4, each core may comprise of a power value denoting the power consumption values of the underlying microcontroller. Based on the data sheet of the target, the various power ratings for different processor clock

rates can be obtained. These power consumption values are specified in the design model using the *staticConsumption* tagged value of *HWComponent*, as per the mapping introduced in Table 2. For the AEBS use case example in this paper, the power consumption modes of an ARM processor [30] is taken into consideration (cf. Table 3). Note that a simple example is used here to demonstrate the direct usage of power consumption values from a COTS product data sheet. For instance, the power consumption during three active power modes shown in Table 3 namely, 23.1 mW, 76.59 mW and 138.6 mW (corresponding to 12 MHz, 48 MHz and 100 MHz clock frequencies) are specified in the design model as shown in Fig. 8.

Table 3. Power consumption modes for an ARM single core processor [30].

Processing Mode	Clock Frequency	Power
1	100 MHz	138.6 mW
2	48 MHz	76.59 mW
3	12 MHz	23.1 mW

4.4 Model Transformations

The generic M2M transformations are implemented in an ATL module, *AUTOSAR-inUML2TimingEnergy.atl*. It can be applied to any use case (e.g. AEBS) which satisfies the source and target model criteria as in the workflow in Fig. 3. In this module, there are 9 matched rules for all conditional mappings and 8 lazy rules for all unconditional mappings. In addition, 15 helpers are implemented which may be invoked by the transformation rules. Th helpers are often used as *getter()* and *setter()* methods. In the prototype, the helpers are implemented, for instance to resolve computation units (e.g. nano/milli seconds and milli/micro watts) and to provide assertions for type of model and timing elements (e.g. a *softwareComponent* and a *runnable*). An example for each type of rule (matched and lazy) and helper, from the prototype implementation of the M2M transformations in *AUTOSARinUML2TimingEnergy.atl* is described below.

Matched Rule. The rules consist of a source pattern in the `from` section and a target pattern in the `to` section. The source pattern specifies the type of the source model element to be matched and the target pattern contains the output model element that will be created by the transformation for each source element. In the ATL module *AUTOSAR-inUML2TimingEnergy.atl*, for synthesis of energy-aware timing analysis models, the matched rules are used for source elements such as model, package, classes and for the elements with applied stereotypes from AUTOSAR profile shown in Table 1.

Listing 1.1. An example of an ATL matched rule.

```

1 -- @atlcompiler emftvm
2 -- @path TimingEnergy=/de.uos.te.model/model/timingEnergy.ecore
3 -- @nsURI UML=http://www.eclipse.org/uml2/5.0.0/UML
4 -- @nsURI MARTE=http://www.eclipse.org/papyrus/MARTE/1
5 -- @nsURI AR=http://autosar.org/schema/r4.0/autosar40
6
7 module AUTOSARinUML2TimingEnergy;
8 create OUT: TimingEnergy, from IN : AR
9 rule AtomicSWC2SWComponent extends
10 Identifiable2ICATObject{
11 from
12   input : AR!AtomicSwComponentType
13 to
14   output : TimingEnergy!SoftwareComponent(
15   runnables <- input.internalBehaviors
16   ->collect(ib | ib.runnables)
17   -> flatten())}

```

A simple example of an ATL matched rule is shown in Listing 1.1. Note that in lines 2–5 the various paths of the metamodels invoked in the ATL module are specified (either local or at URI-repository resource). The `AtomicSWC2SWComponent` rule extends the parent rule `Identifiable2ICATObject` and thus, its target pattern is inherited. This means that, the target element `SoftwareComponent` automatically receives the name and description attributes from parent rule (i.e., `Identifiable-2ICATObject`-not listed here).

In this matched rule, as seen in lines 11–14, a software component in the source AUTOSAR (meta) model (`AR!AtomicSwComponentType`) is matched to a target software component element (`TimingEnergy!SoftwareComponent`) in the timing-energy (meta) model. Thereby, an instance of the target element (i.e., a software component corresponding to the timing-energy analysis meta-model) is created.

Additionally, it receives the `runnables` (lines 15–17) attribute specified in the new target pattern, to link to the software component’s runnables. The `collect` operation iterates through all internal behavior elements (`ib`) and returns the list of runnables for each. As this statement returns a two-dimensional list, the `flatten` operation ensures that a list directly containing the runnables is returned and assigned to the `runnables` attribute. Use an example to describe here all the rules or later on.

Lazy Rule. Lazy rules are used for source elements that satisfy specific conditions and must be called explicitly for creating target elements. Listing 1.2 shows an example of an ATL lazy rule which is used to create a `powerValue` from a `String`. It may be recalled that the power consumption values are specified in the tag values of the respective MARTE stereotype (as a string). The ATL rule in Listing 1.2 converts this specified power value as a string to a corresponding model element `powerValue` in the generic metamodel (cf. Fig. 4, Sect. 3.3, Table 3, Sect. 4.3 & Fig. 8).

Listing 1.2. An example of an ATL Lazy rule.

```

1 lazy rule StringToPowerValue {
2   from
3     string: String
4   using {
5     splitted : Sequence(String) = string.splitPowerConsumption();
6     value: Real = thisModule.valueFromSplitPowerConsumption(splitted);
7     unit: TimingEnergy!PowerUnit =
8     thisModule.unitFromSplitPowerConsumption(splitted);
9   }
10  to
11    timingEnergyElement: TimingEnergy!PowerValue (
12      unit <- unit,
13      value <- if value.oclIsUndefined() then
14        OclUndefined
15      else
16        value
17      endif
18    )
19 }

```

Let us consider an example of power value of processing mode 1, namely 138.6mw (cf. Table 3). This is specified in the design model using the tagged value *staticConsumption* of the *HWComponent* MARTE stereotype (cf. Table 2 & Fig. 8). The lazy rule *StringToPowerValue* in Listing 1.2, splits the above input string *138.6mw* employing the *using* keyword and expressions in ATL (lines 5–9). The *using* keyword and expression can be used to define complex target pattern elements, thus employed in this lazy rule. Thus, the resulting variables namely *unit* and *value* are assigned to the corresponding target elements in *powerValue* (cf. *powerValue* in Fig. 4) in lines 11–17 of Listing 1.2.

Thus in the example in Fig. 8, the lazy rule *StringToPowerValue* returns the *power value* from a string specified in the tagged value in the stereotypes in model elements. Thus, the power rating for each processing mode such as 138.6mW, 76.59mW and 23.1mW specified in Fig. 8 are returned as output for further calculations.

Helpers. Helpers can be used to define (global) variables and functions. Some examples of include *setter()*, *getter()* methods and functions to resolve attributes involving repetitive pieces of code in one place (e.g. resolving metric units). Helper functions are Object Constraint Language (OCL) [31] expressions. They can call each other by recursion or they can be called from within rules.

Listing 1.3. An example of an ATL Helper.

```

1 helper def: resolveStaticConsumptionFromElement(processingUnit:
2   UML!Element): String = let hwComponent : UML!Stereotype =
3   processingUnit.getHwComponentStereotype() in
4   if not hwComponent.oclIsUndefined() then
5     processingUnit.getValue(hwComponent, 'staticConsumption')
6   else
7     OclUndefined
8   endif;

```

In Listing 1.3, an example of a helper to resolve the tag value `staticConsumption` from the MARTE stereotype element `HWComponent` (cf. Table 2) is presented. In the example in Fig. 8, this helper reads the input value of the tagged value `staticConsumption`, which is highlighted in Fig. 8 in `Hardware::Core1` element and returns the corresponding value of the processing unit.

Similar to the above rules, for the remaining elements in Table 1, a total of 17 ATL rules (9 matched and 8 lazy) and 15 helpers are implemented in the `AUTOSARinUML2TimingEnergy.atl` module.

4.5 Synthesis of Energy-Aware Timing Analysis Model of AEBS

In the above steps, the AUTOSAR-based design model and its corresponding timing and energy annotated AUTOSAR-based design model are created in the UML modeling tool Rhapsody (cf. step(a) in Fig. 3). This model is exported from the UML tool in the interchangeable AUTOSAR ARXML format for M2M transformations (cf. step (b) in Fig. 3). The M2M transformations in `AUTOSARinUML2TimingEnergy.atl` module are invoked in the experimental evaluation directly from the Eclipse development environment. The synthesized AUTOSAR-based energy-aware timing analysis model of the AEBS use case is shown in Fig. 9.

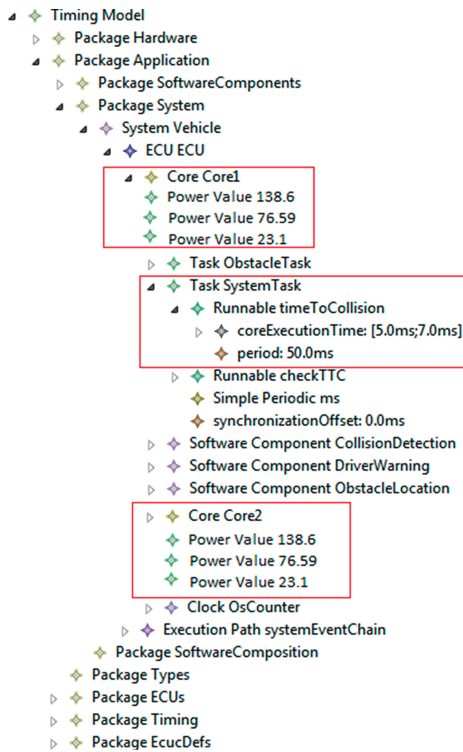


Fig. 9. Synthesized energy-aware timing model of AEBS use case.

The necessary elements for a timing analysis were extracted from the AUTOSAR design model annotated with timing properties (cf. Fig. 6, 7) according to the mapping in Table 1. As seen in Fig. 9, the AEBS model is structured by different *Packages* and the *System* element contains the complete software and hardware elements in a hierarchy. For example, the runnable *timeToCollision* with its corresponding execution time [5 ms, 7 ms] can be seen highlighted in Fig. 9. This runnable is allocated to the *SystemTask*, which is in turn allocated to *Core1* of the ECU. Further, the power consumption modes of the hardware cores are also created corresponding to the annotations in design model. This is highlighted for core1 and core2 in Fig. 9 with the respective power values (138.6, 76.59, 23.1) for each processing mode.

4.6 Performance Analysis

A quantitative performance analysis of the prototype implementation of the workflow in Fig. 3 has been carried out by invoking the transformations for the AEBS use case with varying number of SWCs in the AUTOSAR-based UML design model. This is because, the number of software components (apart from tasks) may be considered as a primary factor for computing complexities involved in schedulability analysis of systems. Further, the number of cores and power consumption modes were also varied to invoke respective M2M transformations for resolving the power consumption modes.

For varying input sizes namely, SWCs and hardware cores in annotated design model), time and memory requirement of the ATL module to synthesize the respective instance of the AUTOSAR-based energy-aware timing analysis model is determined (cf. Table 4). For varying inputs of SWCs, the number of cores were set to two, each having three power consumption modes as described in Sect. 4.3. This is because, the number of SWCs in an AUTOSAR design model can be up to several hundreds. Whereas, the number of cores and their power consumption modes would not scale to such values, hence not provided as a separate set of input in Table 4. The aforesaid experiments were carried out on a standard X-86 based host with Windows-XP OS. The results indicate that the ATL transformations terminate once the generation of the timing analysis model is completed. The generation time and memory requirement is bounded for varying input sizes. This demonstrates the applicability and suitability of the steps involved in the proposed approach for early model-based synthesis of AUTOSAR-based energy-aware timing analysis model from AUTOSAR-based design models developed in UML/SysML tools.

Table 4. Set of inputs, time & memory requirement on a standard X-86 based host for the *AUTOSARinUML2TimingEnergy.atl* ATL module.

SWCs	Time (s)	Memory (MB)
10	26.3	4.1
18	54.2	6.3
23	66.34	8.7
43	136.4	20.7

5 Conclusion

In this book chapter, a systematic workflow for integration of energy and timing requirements in the AUTOSAR-based design model in UML/SysML tools has been presented. Thereby, employing a series of steps, an automated and *early synthesis of energy-aware timing analysis models* is incorporated in the automotive embedded software development process. These performance analysis models may be employed for an early evaluation and decision on the best set of performance configuration and trade-off analysis (e.g. timing vs energy). Thus, employing such performance analysis workflows not only allow us to understand the performance aspects and behavior of the systems during early design stages, but also help to explore different design architectural choices and quantitatively evaluate their implications on system performance. Fine tuning the modeling of energy and timing parameters, such as specification of energy consumption and timing budget per function call is one among the items for future work.

References

1. Anssi, S., Gérard, S., Kuntz, S., Terrier, F.: AUTOSAR vs. MARTE for enabling timing analysis of automotive applications. In: Ober, I., Ober, I. (eds.) SDL 2011. LNCS, vol. 7083, pp. 262–275. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25264-8_20
2. Atlas Transformation Language (ATL) Technology. <https://www.eclipse.org/atl/>. Accessed 20 June 2020
3. Automotive Open System Architecture. <http://www.autosar.org/>. Accessed 20 June 2020
4. Bhasker, J.: A SystemC Primer. Star Galaxy (2010)
5. Bucaioni, A., Cicchetti, A., Ciccozzi, F., Mubeen, S., Sjödin, M.: A metamodel for the Rubus component model: extensions for timing and model transformation from EAST-ADL. IEEE Access **5**, 9005–9020 (2017)
6. Derler, P., Eidson, J., Lee, E.A., Matic, S., Zimmer, M.: Model-based development of deterministic, event-driven, real-time distributed systems. In: Workshop on Model-Based Design with a Focus on Extra-Functional Properties (2011)
7. Eclipse Modeling Framework. <https://www.eclipse.org/modeling/emf/>. Accessed 20 June 2020
8. Enterprise Architect tool. <http://www.sparxsystems.com/>. Accessed 25 June 2020
9. Ficek, C., Feiertag, N., Richter, K., Jersak, M.: Applying the AUTOSAR timing protection to build safe and efficient ISO 26262 mixed-criticality systems. In: Proceedings of ERTS (2012)
10. Franco, F.R., et. al: Workflow and toolchain for developing the automotive software according AUTOSAR standard at a Virtual-ECU. In: 2016 IEEE 25th International Symposium on Industrial Electronics (ISIE), pp. 869–875 (2016)
11. GLIWA Embedded Systems-Timing suite T1. <https://www.gliwa.com/>. Accessed 20 June 2020
12. Hagner, M., Aniculaesei, A., Goltz, U.: UML-based analysis of power consumption for real-time embedded systems. In: IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications, pp. 1196–1201 (2011)
13. Hans, B., Rolf, J., Henrik, L.: Annotation with timing constraints in the context of EAST-ADL2 and AUTOSAR-the timing augmented description language. In: STANDRTS 2009 (2009)

14. Harbour, M.G., García, J.G., Gutiérrez, J.P., Moyano, J.D.: Mast: Modeling and analysis suite for real time applications. In: 13th Euromicro Conference on Real-Time Systems, pp. 125–134. IEEE (2001)
15. Henzinger, T.A., Horowitz, B., Kirsch, C.M.: Giotto: a time-triggered language for embedded programming. In: Henzinger, T.A., Kirsch, C.M. (eds.) EMSOFT 2001. LNCS, vol. 2211, pp. 166–184. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45449-7_12
16. van der Horst, R., Hogema, J.: Time-to-collision and collision avoidance systems. In: Proceedings of the 6th ICTCT Workshop (1993)
17. IBM Software: IBM rational rhapsody developer. <https://www.ibm.com/products/systems-design-rhapsody>. Accessed 25 June 2020
18. IBM Software: IBM rational rhapsody developer (2019). <https://www.ibm.com/software/products/en/ratirhap>. Accessed Nov 2019
19. INCHRON: chronSIM (2019). <https://www.inchron.com/tool-suite/chronsim.html>. Accessed Nov 2019
20. Iqbal, M.Z., Ali, S., Yue, T., Briand, L.: Experiences of applying UML/MARTE on three industrial projects. In: Proceedings of the 15th International Conference MODELS 2012 (2012)
21. Iyengar, P., Pulvermueller, E.: A model-driven workflow for energy-aware scheduling analysis of IoT-enabled use cases. IEEE Internet Things J. **5**(6), 4914–4925 (2018). <https://doi.org/10.1109/JIOT.2018.2879746>
22. Iyengar, P., Huning, L., Pulvermüller, E.: Early synthesis of timing models in AUTOSAR-based automotive embedded software systems. In: Proceedings of the 8th International Conference on Model-Driven Engineering and Software Development, MODELSWARD 2020, pp. 26–38. SCITEPRESS (2020)
23. Iyengar, P., Noyer, A., Engelhardt, J., Pulvermüller, E., Westerkamp, C.: End-to-end path delay estimation in embedded software involving heterogeneous models. In: 11th IEEE Symposium on Industrial Embedded Systems, SIES, 2016, pp. 183–188 (2016)
24. Kim, J.H., Kang, I., Kang, S., Boudjadar, A.: A process algebraic approach to resource-parameterized timing analysis of automotive software architectures. IEEE Trans. Ind. Inf. **12**(2), 655–671 (2016)
25. Kusano, K.D., Gabler, H.: Method for estimating time to collision at braking in real-world, lead vehicle stopped rear-end crashes for use in pre-crash system design. SAE Int. J. **4**(1), 435–443 (2011)
26. MARTE profile. <https://www.omg.org/spec/MARTE/About-MARTE/>. Accessed 25 June 2020
27. Martinez, L.R., Prieto, M.D.: New Trends in Electrical Vehicle Powertrains, 1st edn. Intech Open, London (2019)
28. Mathworks Products. <https://www.mathworks.com/>. Accessed 20 May 2020
29. Navet, N., Simonot-Lion, F. (eds.): Automotive Embedded Systems Handbook. CRC Press, Boca Raton (2009)
30. NXP LPC1768 MCU datasheet. http://www.nxp.com/documents/data_sheet/LPC1769_68_67_66_65_64_63.pdf. Accessed 20 May 2020
31. Object Management Group. <http://www.omg.org>. Accessed 25 June 2020
32. Papyrus UML Tool. <http://www.papyrusuml.org/>. Accessed 05 May 2017
33. Peraldi, M., Sorel, Y.: From high-level modelling of time in MARTE to realtime scheduling analysis. In: First International Workshop on Model Based Architecting and Construction of Embedded Systems (2008)
34. Peraldi-Frati, M.A., Blom, H., Karlsson, D., Kuntz, S.: Timing modeling with autosar-current state and future directions. In: Design, Automation Test in Europe Conference, DATE (2012)
35. Petriu, D.C.: Software Model-based Performance Analysis, pp. 139–166. Wiley, Hoboken (2013)

36. Saidi, S., Steinhorst, S., Hamann, A., Ziegenbein, D., Wolf, M.: Special session: future automotive systems design: Research challenges and opportunities. In: 2018 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), pp. 1–7 (2018)
37. Sangiovanni-Vincentelli, A., Natale, M.D.: Embedded system design for automotive applications. *Computer* **40**(10), 42–51 (2007)
38. Scheickl, O., Ainhauser, C., Gliwa, P.: Tool support for seamless system development based on AUTOSAR timing extensions. In: Proceedings of Embedded Real-Time Software Congress (ERTS) (2012)
39. Scheid, O.: AUTOSAR Compendium, Part 1: Application & RTE. CreateSpace Independent Publishing Platform (2015)
40. Singhoff, F., Legrand, J., Nana, L., Marcé, L.: Cheddar: a flexible real time scheduling framework. In: ACM SIGAda Ada Letters, vol. 24–4. ACM (2004)
41. SysML specification. <http://www.omg.sysml.org/>. Accessed 20 May 2020
42. System Composer toolbox. <https://www.mathworks.com/products/system-composer.html>. Accessed 20 May 2020
43. Timing Architects Tool. <https://www.timing-architects.com/>. Accessed 20 June 2020
44. UML specification. <https://www.omg.org/spec/UML/About-UML/>. Accessed 20 June 2020
45. Zhao, Y., Liu, J., Lee, E.A.: A programming model for time-synchronized distributed real-time systems. In: Proceedings of 13th IEEE Real Time and Embedded Technology and Applications Symposium, pp. 259–268. RTAS (2007)