Early Model-Driven Timing Validation of Embedded Software Systems Developed Using Matlab/Simulink

Padma Iyenghar, Elke Pulvermueller
Software Engineering Research Group, University of Osnabreuck, Germany
{piyengha,elke.pulvermueller}@uos.de

ABSTRACT

This paper presents a model-driven workflow for specification of timing constraints in Matlab/Simulink (ML/SL), their extraction and synthesis of a timing analysis model and its subsequent validation in a specialized timing validation tool (e.g. SymTA/S). We present a light-weight interfacing tool framework, adhering to the proposed workflow, and evaluate it in a real-life automotive engine model example. A main benefit of this work is the feedback at early design stage, about the performance characteristics (e.g. load, schedulability) of the system.

KEYWORDS

Model-Driven Timing Analysis, Matlab/Simulink, Embedded Software Engineering (ESE), Model-to-Model (M2M) Transformations

1 INTRODUCTION

Many software-implemented functions in embedded systems may involve time delays, which may critically affect the performance of the system. These delays can stem from time lags such as transport delays, computational delays and sampling effects. Such delays may critically affect the performance of a system. For instance, the closed loop responses of a system could become oscillatory and unstable if the various delays (e.g. communication delays) are not taken into account. A typical case where such computational and/or sampling delays have pronounced effects is in the case of the performance of a control system. A motivational example outlining these effects was discussed in [5]. Therefore, during modeling and systems engineering, the additional delays need to be included to arrive at an accurate design and to avoid undesirable performance outcomes. Ignoring the specification of such additional delays in a functional model of a system, may result in changing the behavior of the model, thereby affecting its performance. In the worst case, it may endanger the safety of the system as a whole leading to catastrophic consequences.

Matlab/Simulink (ML/SL) [9] is a widely used tool to design Embedded Software, involving control functions in various fields including the automotive and avionics. In Simulink, the individual functions may be implemented using atomic function blocks or subsystems (a block encompassing a set of blocks). Simulink supports the creation of open/closed loop models of control systems with delays and analyzes their stability and performance. Based on simple experiments on examples from the Simulink library, one can infer that the control quality and performance are degraded because of the additional delays involved in the actuation and sensing paths. To compensate for the effects of delays and sampling, controller re-design approaches (e.g. controller tuning), may be employed and verified using simulation.

1.1 Motivation

In line with verification using simulation, equally important are the real-time scheduling analysis of the system, under various timing constraints. A step forward in this direction is to incorporate the real-world effects into functional modeling in Simulink and an early performance analysis. For instance, when individual systems are modeled as chains of components with delays, they can be translated to chains of tasks with timing constraints for scheduling analysis [5]. The timing parameters of the system design may further be tuned during early design stages, based on the analysis results from specialized timing validation tools (e.g. SymTA/S [12]). The aforesaid aspects serve as the main motivation for this paper, towards a workflow for early model-driven timing validation of embedded software systems developed using ML/SL.

To perform a schedulability analysis, the various function blocks in the system design model need to be mapped as elements such as tasks. Further, the timing constraints available in the design model need to be extracted, based on which a timing analysis model could be synthesized. The motivational background for this aspect is elaborated with an ML/SL example in a work-in-progress paper (poster) published in [5]. The challenges for translating the timing requirements in a Simulink model to a timing analysis model are outlined, in line with a general approach for Non-Functional Property (NFP) analysis in [5]. A methodology to synchronize the timing properties spread across heterogeneous modeling domains such as the Unified Modeling Language (UML) and ML/SL, is dealt with in [6]. Further, adhering to the general approach for NFP analysis, Reliability analysis of Simulink models using Fault Trees is presented in [7].

1.2 Novel contributions

A motivational background for annotating the timing aspects in a ML/SL model and their translation is briefly dealt with in [5]. However, aspects pertaining to implementing the generic NFP workflow for *timing analysis of a ML/SL model*, *synthesis of an overall system-level timing analysis model*-its validation in a specialized timing validation tool employing a *real-life ML/SL example* are missing. Addressing the aforesaid aspects and extending the motivational ideas in [5] the following novel contributions are presented in this paper.

(1) A prototype implementation of the workflow for timing validation of Simulink models, using a light-weight interfacing tool framework (with plug-ins) developed using Eclipse Modeling Framework (EMF) and the Eclipse plugin development environment.
(2) An experimental evaluation of the importer/exporter framework and the proposed workflow in a real-life automotive engine model example implemented in ML/SL.

In the remaining of this paper, section 2 discusses the related work. In section 3 the proposed workflow is elaborated along with experimental evaluation. Section 4 concludes this paper.

2 STATE-OF-THE-ART PRACTICE FOR TIMING ANALYSIS OF SIMULINK MODELS

There are some existing approaches to estimate the Worst Case Execution Time (WCET) of various components in Simulink models [8]. However, they are primarily based on the generated code from the function blocks and/or depend on Matlab-based simulation tools for timing analysis such as *TrueTime* [13], *Jitterbug* [2], *T-RES* [3] and *SimEvents*[9].

Jitterbug, for instance, is used to compute a performance criterion for a control loop under various timing conditions to determine the controller's sensitivity.

TrueTime is a ML/SL-based simulator which enables detailed co-simulation of process dynamics, control task execution and network communication in a distributed real-time control system [2]. A software centric approach using TrueTime is described in [4], wherein it facilitates the simulation of control software close to its temporal behavior.

T-RES [3] is a more recently introduced tool for incorporating timing delays in code and verifying the impact on control software. It provides a modular approach to design the controller model, thereby enabling to define the controller code apart from the model of the task. Yet, tool support for specification of timing constraints at a higher level of abstraction (e.g. at the model), during early design stages, is not available in [2], [3], [4].

On the other hand, *SimEvents* in ML/SL provides a discrete-event simulation engine and component library for analyzing event-driven system models. and optimizing performance characteristics in the Simulink domain. The elements from this library may be used as blocks in a Simulink model to study the effects of task timing and resource usage with support for basic scheduling policies. However, a major drawback is the lack of support for code generation from the system model comprising of SimEvents blocks. Hence, this approach does not provide a realistic solution towards using a single design model for both code generation and performance analysis.

Apart from these tools, various related approaches concentrate on addressing a certain aspect of simulation and analysis only. For instance, execution time aware simulink blocks presented in [10] demonstrate the scheduling of such blocks and show that preemption has effects already in the simulation. Thus, such methods and tools help the designer only with the study of a particular part of a system under some specific timing delays. They do not consider the overall functionality of a system under development during a performance analysis. Then, for systems engineering, the system designer needs to incorporate these analysis results in the actual design model (which could be a totally different model) and generate target specific code in the next steps.

In summary, a systematic workflow or tool support towards model-driven specification of timing requirements and their validation, in *specialized timing analysis tools such as SymTA/S*, is not supported for ML/SL. This makes it difficult to include the timing requirements into the same model (and later validate timing), which also incorporates the envisaged system behavior and is subsequently used for target code generation.

3 WORKFLOW AND EXPERIMENTAL RESULTS

The proposed workflow for early model-driven timing analysis of Simulink models is illustrated in Fig. 1. The workflow comprises

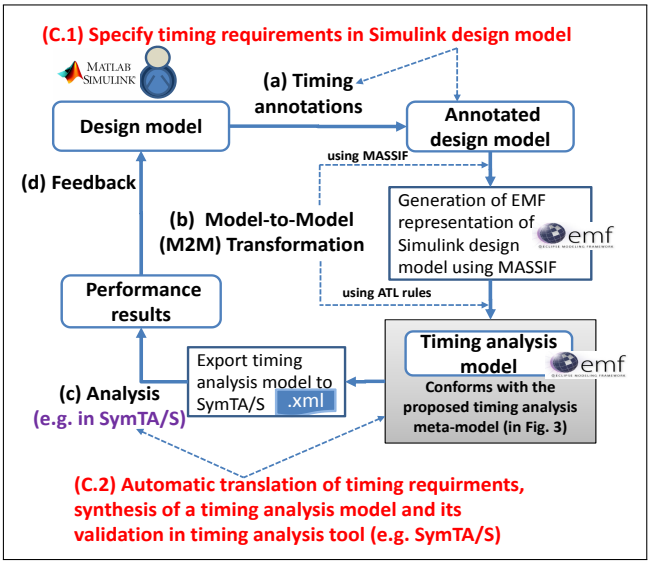


Figure 1: Workflow and challenges (C.1, C.2) addressed in this paper

of the following steps: (a) add annotations to the design model to describe the timing properties, (b) define model transformations from annotated software design models to formalisms useful for timing analysis, (c) analyze timing (i.e., in a timing validation tool) and (d) feedback to the design. In the following, the above steps are explained in detail together with examples from an experimental evaluation.

3.1 Annotation of timing requirements (Step (a) in Fig. 1)

To employ the workflow in Fig. 1, for timing analysis of a Simulink design model, the components of the ML/SL design model needs to be annotated with the timing properties. However, tailored extensions (e.g. UML profiles) required for timing specification are not available in Simulink. Addressing this aspect, a simple but effective mechanism for the specification of timing requirements, using *masks* (a custom user interface for a block), in the components of the Simulink design model is proposed (challenge C.1 in Fig. 1).

The underlying assumptions for annotating the design model are described below. Then the proposed solution for annotation of timing requirements is explained with an example.

(1) A general assumption that, for a basic schedulability analysis in a specialized timing validation tool (e.g. SymTA/S [12]) a set of tasks and runnables (e.g. operations) with timing properties are required, is taken into consideration

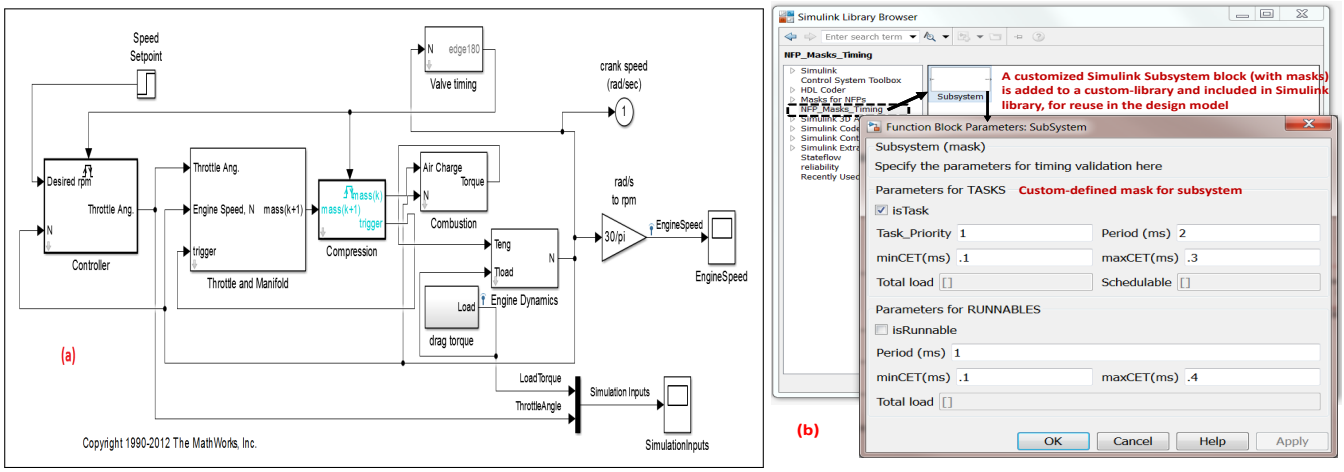


Figure 2: (a) Simulink model of a four-cylinder spark ignition internal combustion engine and (b) Custom-defined mask library to specify timing requirements

in this work. The timing properties of tasks such as *priority*, *execution time* and *period* are considered for analysis. Hence, the components in the ML/SL design model need to be mapped as a set of tasks and runnables for a schedulability analysis in SymTA/S.

Please note that only a simplified, yet sufficient, set of timing parameters (e.g. *priority*, *execution time* and *period* of tasks) for a basic schedulability analysis are discussed in this paper. The proposed approach can be extended for annotating further timing parameters required for timing analysis in the timing validation tool under consideration.

(2) During systems engineering, the design model of an embedded software implemented in Simulink needs to adhere to the following guideline. Related blocks (atomic elements in Simulink) which represent one functionality, should be kept together with the aid of a subsystem in Simulink. Then, the timing properties of such a subsystem may be specified using the proposed custom mask-library.

The design model, as per the assumption above (i.e., section 3.1-(1)), comprises of subsystems annotated with timing properties. The Model-to-Model (M2M) transformations (step (b) in Fig. 1), take as input the ML/SL design model and automatically allocate each subsystem in the design model to a task. The annotated elementary blocks (if present) inside the subsystem are mapped as runnables for that task. Thus, in the proposed approach, multiple elementary blocks inside a subsystem and the subsystem are mapped to runnables and task respectively.

Please note that, these tasks are mapped to a single core which is in turn mapped to one ECU. This is also automatically assigned during M2M transformations, for the sake of simplicity[1]. The masks for the Simulink subsystem introduced in Fig. 2-(b) can be further extended to specify multiple cores and their mapping to multiple ECUs.

[1]Observed in the generated timing analysis model in Fig. 5

Please note that the usage of multiple ECUs and cores is supported by the underlying meta-model, introduced in Fig. 3. The mapping of elements in the Simulink design model to the timing analysis model, explained above, is shown in Table 1.

Table 1: Mapping of elements in Simulink model to Timing Analysis model

Element	
Simulink model	***Timing Analysis model***

`Model`
`└─ Subsystem`
`└─ Block`
	`ECU,Core`
	`└─ Task`
	`└─ Runnable`

Sequence of Subsystems	Execution path
Sequence of Runnables	Execution path

The native options in ML/SL for specification of delay (e.g. integer/transport delay) or simulation engines based on ML/SL (cf. section 2), do not readily support annotation of the aforementioned timing parameters in the design model (e.g. for subsystems). To address this gap, an existing feature in Simulink is used to propose a simple but effective mechanism (using *masks*) for the specification of the aforementioned timing requirements in the components of the design model.

3.1.1 Engine model example. Let us consider the embedded software implementation of an electronic engine controller functionality, in Simulink. This relatively complex example design model of a four-cylinder spark ignition internal combustion engine (Fig. 2-(a)) from the Simulink library, which is employed for experimental evaluation, can be considered as a representative of the following aspects: (a) A design model comprising of subsystems (representing various sub-functionalities) collaborating to achieve an overall functionality and (b) Native options in ML/SL for specification of

delay (e.g. integer/transport delay) or simulation engines based on ML/SL (cf. section 2), do not readily support annotation of the aforementioned timing parameters in the subsystems (i.e., the design model) for a schedulability analysis in specialized timing validation tools such as SymTA/S.

3.1.2 Custom-defined mask library. Some salient features supported by Simulink w.r.t the blocks and libraries are:

- Customizable blocks which can be added to a custom-defined block library, for reuse
- Availability of customizable *masks* for blocks

A *mask* is a custom user interface for a block. By *masking* a block, one can encapsulate the block to have its own user-defined interface which may have a customized appearance, block description and parameter prompts. Making use of these advantages, the components in the design model can be annotated with the timing requirements using custom-defined masks.

Further, to avoid the repeated manual effort involved in creating masks for blocks, a custom-defined Simulink library may be created (for reuse). A custom defined mask for a subsystem, which is then included in a custom-defined Simulink library (*NFP_Masks_Timing*) is shown in Fig. 2-(b). Thus, the timing attributes of a subsystem, which will be mapped to a task or a runnable for timing analysis, can be specified as shown in Fig. 2-(b).

The non-editable elements in the mask in Fig. 2-(b), namely *Total load* and *Schedulable* are used to store the results of the schedulability analysis provided as feedback from the timing validation tool.

3.2 Intermediate timing analysis model

The timing requirements in the annotated design model need to be extracted and translated to a timing analysis model, in order to carry out a timing analysis (i.e., steps (b), (c) in Fig. 1). M2M transformations can be used to convert the annotated design model to a timing analysis model. Such model transformations aid in bridging the large semantic gap between the source and the target model. In our case, the source model is the Simulink design model and the target model is a timing analysis model. While employing the M2M transformations, both the source and the target models must conform with their respective meta-models.

In the prototype, an open source Matlab Simulink Integration Framework for Eclipse[2] (*MASSIF*) is used for converting the annotated design model in Simulink to the EMF format. An advantage of this step is that, when the Simulink design model is available in EMF format (now conforming with the MASSIF meta-model), the elements in the design model may be elementarily subject to further transformations.

As a target meta-model, a timing analysis meta-model (which adheres with the semantics of timing analysis concepts is defined in this paper. From Fig. 3, it is seen that the timing analysis meta-model comprises of a package with all the elements (e.g. *ECUs*, *cores*, *tasks*, *runnables*, *trigger*) required for a basic timing evaluation of a software system, in a hierarchy.

The package comprises of *ECUs*, which in turn may encompass *cores*. Each core may have *task*(s) and each task could comprise of

[2]http://www.incquerylabs.com/

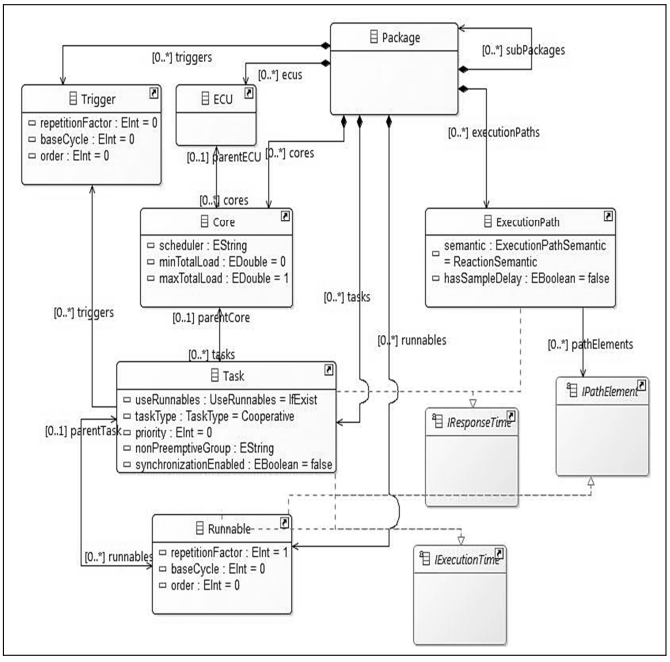


Figure 3: A meta-model (created using EMF) for timing analysis based on the general semantics of timing analysis.

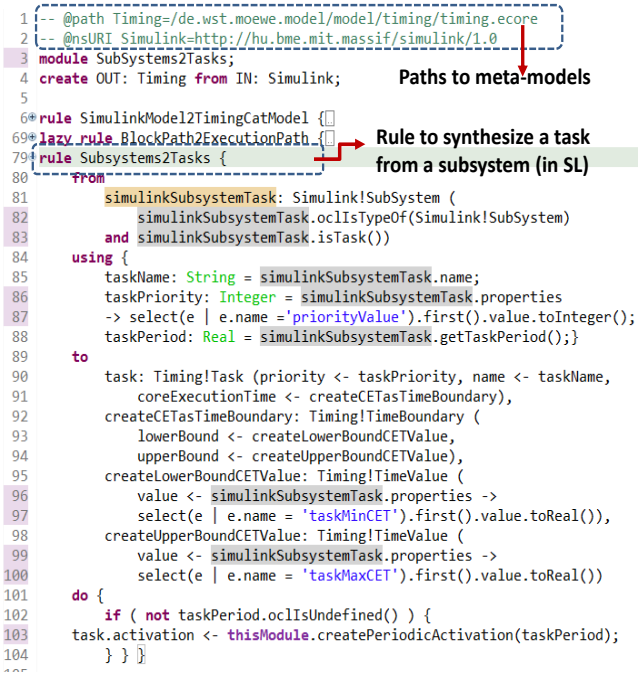
runnables (e.g. an operation). A task may or may not have a *trigger*, depending on its activation. In addition, each *task* and *runnable* comprises of a *tag* element, which may be used to store additional useful information (e.g. as a string value). Apart from these basic elements, there may be several *execution paths* in a software system. The execution path comprises of path elements, denoted as *IPathElement* in Fig. 3. For instance, the path elements can be *tasks* or *runnables*. Each task and runnable comprise of an attribute to store the execution time (*IExecutionTime* in Fig. 3). This is used as an input for timing analysis. A result of timing validation, namely the *response time* is an attribute for tasks and execution paths.

Further, the element *Core* comprises of an attribute *Scheduler: EString* which represents the scheduling policy for timing analysis. Please note that in our example, a scheduling policy is not specified explicitly. A default scheduling policy (supported by the timing validation tool SymTA/S) is automatically assigned during the M2M transformations (from design model to a timing analysis model). The masks can be extended to provide a drop down list of the schedulers supported by the timing analysis tool under consideration. Therefore, if a scheduler (i.e., a scheduling policy) is not chosen explicitly in the masks in the design model, a default scheduling policy (e.g. Generic OSEK in SymTA/S) is chosen for analysis. This value, if required, can be changed in the (resulting) timing analysis model in SymTA/S tool.

3.3 Synthesis of timing analysis model

In the prototype, the timing requirements in the annotated design model are extracted using M2M transformations implemented in Atlas Transformation Language (ATL) [1]. The transformations are invoked as part of a prototype implementation of an importer/exporter

plug-in environment. An excerpt of transformation rules for extracting timing analysis related aspects from the EMF representation of the ML/SL design model to synthesize a timing analysis model is shown in Fig. 4. The paths to the meta-models corresponding to




```
1  -- @path Timing=/de.wst.moewe.model/model/timing/timing.ecore
2  -- @nsURI Simulink=http://hu.bme.mit.massif/simulink/1.0
3  module SubSystems2Tasks;
4  create OUT: Timing from IN: Simulink;
5
6  rule SimulinkModel2TimingCatModel {
69 lazy rule BlockPath2ExecutionPath {
79 rule Subsystems2Tasks {
80     from
81         simulinkSubsystemTask: Simulink!SubSystem (
82             simulinkSubsystemTask.oclIsTypeOf(Simulink!SubSystem)
83         and simulinkSubsystemTask.isTask())
84     using {
85         taskName: String = simulinkSubsystemTask.name;
86         taskPriority: Integer = simulinkSubsystemTask.properties
87         -> select(e | e.name ='priorityValue').first().value.toInteger();
88         taskPeriod: Real = simulinkSubsystemTask.getTaskPeriod();}
89     to
90         task: Timing!Task (priority <- taskPriority, name <- taskName,
91             coreExecutionTime <- createCETasTimeBoundary),
92         createCETasTimeBoundary: Timing!TimeBoundary (
93             lowerBound <- createLowerBoundCETValue,
94             upperBound <- createUpperBoundCETValue),
95         createLowerBoundCETValue: Timing!TimeValue (
96             value <- simulinkSubsystemTask.properties ->
97             select(e | e.name = 'taskMinCET').first().value.toReal()),
98         createUpperBoundCETValue: Timing!TimeValue (
99             value <- simulinkSubsystemTask.properties ->
100            select(e | e.name = 'taskMaxCET').first().value.toReal())
101    do {
102        if ( not taskPeriod.oclIsUndefined() ) {
103    task.activation <- thisModule.createPeriodicActivation(taskPeriod);
104        } } }
```

Figure 4: An excerpt of transformation rules for extracting timing analysis related aspects from EMF representation of ML/SL design model to synthesize timing analysis model

the Simulink design model in EMF format (i.e., the meta-model of MASSIF) and the timing analysis model are provided in lines:1-2 in Fig. 4. A main rule *SimulinkModel2TimingCatModel* in line 6, invokes further ATL rules such as *Subsystems2Tasks* (line: 79) and *BlockPath2ExecutionPath* (line: 69).

For instance, as the name implies, the rule *Subsystems2Tasks* in Fig. 4, is used to synthesize a *task* element for timing analysis from a subsystem (in ML/SL). As seen in Fig. 4, in the rule *Subsystems2Tasks*, the annotated timing parameters in the subsystem such as task priority and period are parsed and assigned to a *task* element in the timing analysis model using simple ATL rule statements (cf. lines 84-88 in Fig. 4).

A challenging aspect in the M2M transformations is, extracting the execution paths [3] with closed-loops (Fig. 2). In the example in Fig. 4, the lazy rule *BlockPath2ExecutionPath* (line: 69) is implemented to convert a given set of blocks in ML/SL model to an execution path in the timing analysis model. In order to estimate the response time of possible execution paths, they first need to be extracted from the design model. This aspect is a challenge in models with closed-loops (such as in Fig. 2), as there is no fixed start/end block. In the prototype discussed in this paper, the start of the execution path is chosen as the first block of the logical execution (e.g. Controller block in Fig. 2). Then, the subsequent path elements are obtained by parsing the connected blocks.

3.3.1 Empirical results. The prototype implementation of the transformations in step (b) of the workflow (Fig. 1-(a)) are invoked as part of an importer/exporter framework implemented in the programming language Java. The experiments were carried out on a standard X-86 based host with Windows-7 OS. The prototype is evaluated primarily with the help of the Simulink model of a four-cylinder spark ignition internal combustion engine use case shown in Fig. 2. Further, for analyzing the performance of the prototype, the transformations were invoked for several examples with varying number of blocks (e.g. subsystems) in the Simulink design model. Please note that, the subsystems are the elements mapped to tasks in the timing analysis model. The number of tasks (and their properties) in the timing analysis model, may be considered as a primary factor for computing complexities involved in schedulability analysis of systems [11], [12].

For varying input size, i.e., number of subsystems in the Simulink design model, the time and memory requirement of the ATL module to generate the respective instance of the timing analysis model (e.g. with tasks) is determined (Table 2). The number of subsystems

Table 2: Set of inputs, time & memory requirement on a standard X-86 based host (with Windows-XP OS), for *SimulinkModel2TimingCatModel* ATL module

Subsystems	Time (s)	Memory(MB)
7	10.54	0.85
10	17.67	1.59
13	22.52	2.56
33	35.34	3.23
46	54.38	5.60

for this experiment were chosen keeping in mind the complexity involved in modeling embedded software systems/use cases. For instance, the Simulink model of a four-cylinder spark ignition engine use case (in Fig. 2) comprises of 7 main subsystems. A more complex, industrial case study of a Simulink-based system currently under development in collaboration with industrial partners (by the authors), comprises of 46 classes.

The results indicate that the ATL transformations terminates once the generation of the timing-energy analysis model is completed. The generation time and memory requirement is bounded for varying input sizes, including those from real-world scenarios. This demonstrates the applicability and suitability of the steps involved in the proposed workflow for automatic derivation of a timing-energy analysis model for energy aware timing analysis.

3.4 Export of timing model and timing analysis in validation tool

The source and target for the M2M transformations in Fig. 4, are the Simulink model (in EMF format) and the timing analysis meta-model respectively. The output is an instance of the timing analysis

[3]Please note that the response time of possible execution paths can be obtained as a result of timing analysis, from SymTA/S.

model as seen in Fig. 5-(1). The intermediate timing analysis model comprises of only the timing properties of the design model in Fig. 2-(1) and conforms with the meta-model in Fig. 3.

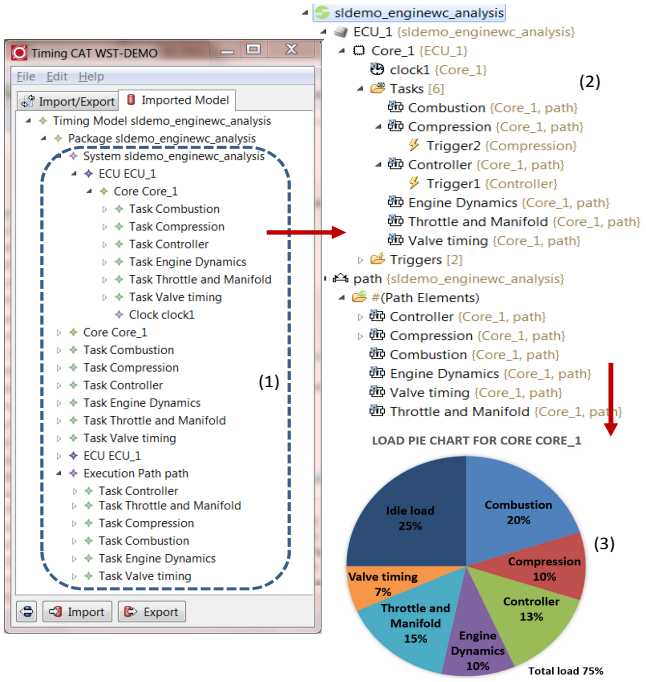


Figure 5: (1) Imported timing analysis model instance for the design model in Fig. 2-(a), (2) Timing analysis model exported to SymTA/S for validation and (3) Worst case load analyzed by SymTA/S

The timing analysis model in Fig. 5-(1) is persisted in XMI and exported to SymTA/S (using plug-ins implemented in Java and EMF). The timing analysis model corresponding to the design model in Fig. 2-(1) is now ready for timing validation in SymTA/S, as seen in Fig. 5-(2). This resulting timing analysis model includes the real-world effects modeled using delays (e.g. using timing budgets) and incorporated among various timing constraints using *masks* for subsystems in ML/SL model.

Extensive analysis of the timing properties may be obtained as a result of the validation in SymTA/S, as it is a specialized timing analysis tool. In the approach presented in this paper, only the timing requirements from design model are extracted and subject to validation in specialized timing validation tools. This is performed directly on the system design model (from which target code is generated), without affecting either the functional aspects of the design model or distorting the timing properties. These aspects provide significant advantages over state-of-the-art practices (section 2) wherein, only limited timing analysis is possible as such tools [2], [3] are not specialized ones timing validation.

For example, the worst-case load per task for the engine model is shown in Fig. 5-(3). The non-editable elements in the mask in Fig. 2-(b), namely *Total load* and *Schedulable* are used to store the results of the schedulability analysis provided as feedback from the timing validation tool.

4 CONCLUSION

A model-driven workflow for timing analysis of ML/SL models annotated with timing requirements, in specialized timing validation tools (e.g. SymTA/S) is proposed. A prototype implementing various aspects of the workflow and its evaluation on a real-life engine model from the Simulink library is presented. The results indicate that the timing parameters can be incorporated in the same Simulink design model which is subsequently used for code generation without affecting the functional aspects of the design model.

In summary, the two main advantages of the proposed approach are: (a) the timing parameters specified using the custom-defined mask library may be used to incorporate real-world effects of various delays during design stages and carry out an early model-driven timing validation of the system design model and (b) the timing parameters may be further tuned, already during early design stage, based on the feedback from timing analysis tools. Future work includes an experimental evaluation with support for timing analysis of multi/many cores and communication aspects.

ACKNOWLEDGMENTS

This work is part of a project supported by a grant (id: KF2312004KM4) from BMWi-ZIM co-operation, Germany. This project work is carried out in close cooperation with Willert Software Tools GmbH and SymtaVision GmbH.

REFERENCES

[1] Atlas Transformation Language. Last accessed: 13.10.2017. https://eclipse.org/atl/. (Last accessed: 13.10.2017).
[2] A. Cervin, K. E. Arzen, D. Henriksson, M. Lluesma, P. Balbastre, I. Ripoll, and A. Crespo. 2006. Control loop timing analysis using truetime and jitterbug. In *IEEE International Conference Computer Aided Control System Design.*
[3] Fabio Cremona, Matteo Morelli, and Marco Di Natale. 2015. TRES: A Modular Representation of Schedulers, Tasks, and Messages to Control Simulations in Simulink. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing, SAC 2015.*
[4] Dan Henriksson, Anton Cervin, and Karl Arzen. 2003. TrueTime: Real-time Control System Simulation with MATLAB/Simulink. In *Proc. of the Nordic MATLAB Conference, 2003.*
[5] P. Iyenghar, A. Noyer, J. Engelhardt, and E. Pulvermueller. 2016. Translating timing requirements of Embedded Software systems modeled in Simulink to a timing analysis model. In *21st IEEE International Conference on Emerging Technologies and Factory Automation (ETFA).* 1–4.
[6] P. Iyenghar, A. Noyer, J. Engelhardt, E. Pulvermueller, and C. Westerkamp. 2016. End-to-end path delay estimation in embedded software involving heterogeneous models. In *11th IEEE Symposium on Industrial Embedded Systems (SIES).* 1–6.
[7] P. Iyenghar, S. Wessels, A. Noyer, E. Pulvermueller, and C. Westerkamp. 2016. A novel approach towards model-driven reliability analysis of Simulink models. In *21st IEEE International Conference on Emerging Technologies and Factory Automation (ETFA).* 1–6.
[8] Raimund Kirner, Roland Lang, Peter Puschner, and Christopher Temple. Integrating WCET Analysis into a Matlab/Simulink Simulation Model. In *Proceedings of 16th IFAC Workshop on Distributed Computer Control Systems 2000.*
[9] Matlab and Simulink. 2017. http://www.mathworks.com/. (2017).
[10] Andreas Naderlinger. 2012. Execution-time Aware Simulink Blocks. In *Proceedings of the 2012 SpringSim Poster & Work-In-Progress Track.*
[11] J. A. Stankovic. 1988. Misconceptions about real-time computing: a serious problem for next-generation systems. *Computer* 21, 10 (Oct 1988), 10–19. DOI: http://dx.doi.org/10.1109/2.7053
[12] SymTA/S: Scheduling Analysis & Timing Verification Tool. Last accessed: 05.05.2017. https://www.symtavision.com/products/symtas-traceanalyzer/. (Last accessed: 05.05.2017).
[13] TrueTime. 2017. Matlab/Simulink-based simulator for real-time control systems. http://www.control.lth.se/truetime/. (2017).