End-to-End Path Delay Estimation in Embedded Software Involving Heterogeneous Models

Padma Iyenghar[1], Arne Noyer[1], Joachim Engelhardt[2], Elke Pulvermueller[1], Clemens Westerkamp[3]
[1]Software Engineering Research Group, University of Osnabueck, Germany
[2]Institute for Distributed Systems, Ostfalia University of Applied Sciences, Germany
[3]Institute of Computer Engineering, University of Applied Sciences, Osnabrueck, Germany
{piyengha, anoyer, elke.pulvermueller}@uos.de, jo.engelhardt@ostfalia.de, c.westerkamp@hs-osnabrueck.de

***Abstract*—Extending model-based Non-Functional Property (NFP) analysis approaches to Embedded Software Engineering (ESE) projects cutting across heterogeneous modeling domains is an emerging research challenge. Towards this direction, a generic workflow for timing validation and a methodology for synchronization of timing attributes (*before* performing a timing analysis) in ESE projects developed using heterogeneous modeling domains is proposed in this paper. An experimental evaluation of the proposed approach, in a state-of-the-art timing analysis tool, using a real life, light-weight ESE project, developed using Unified Modeling Language (UML) and Simulink is presented.**

***Keywords*—Embedded software; heterogeneous modeling domains; end-to-end delay analysis; UML; Matlab/Simulink;**

I. Introduction and Motivation

Heterogeneity seems to be the essence in Embedded Software Engineering (ESE): comprising of a subtle combination of hardware and software sub-systems for specifying data and control processing requirements. To address this multi-faceted design aspect, practitioners of model-based ESE are required to use more than one modeling domain/language. Thus, an embedded software system may comprise of several software sub-systems, developed using heterogeneous modeling domains such as the Unified Modeling Language (UML) [1], Matlab/Simulink [2] and Labview [3].

Among the NFPs for embedded systems, the timing properties (e.g. performance, schedulability) gains foremost significance. At this juncture, with heterogeneity being the essence of MDD in ESE, it is intuitive to perceive the need to extend timing analysis approaches for such ESE projects (involving two/more modeling domains). For instance, in such ESE projects, the timing aspect of a *module* in one modeling domain (e.g. a class/operation in UML) could be dependent on the timing property of a *module* in another modeling domain (e.g. a block/sub-system in Simulink). For example, in Fig. 1-(A), the design model of the embedded software system involves software components U1-U6 in the UML domain and M1-M4 from the Simulink domain. In this example, the components U3, U4 (UML domain) are implemented/realized by software components M2, M3 respectively (Simulink). Let us consider an example of an end-to-end path delay estimation for an execution path in this software system. As shown in Fig. 1-(B), an execution path involves the software components U1, U2, U3, U5, U6 from the UML domain. While the execution times (e.g. Core Execution Time (min, max)ms) of the components U1, U2, U5, U6 are known in the UML domain, the execution time of the component U3 is missing. Since U3 is realized in

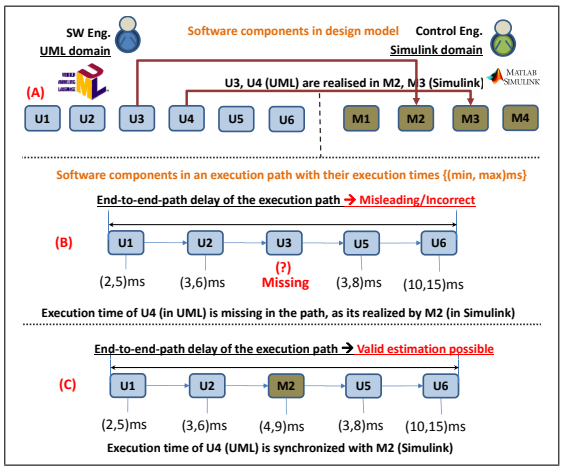

Fig. 1. End-to-end path delay estimation involving software components from UML and Simulink

the Simulink domain, data about its execution time is missing in the UML domain. However, to make a valid timing analysis and estimate the end-to-end path delay of this execution path (Fig. 1-(C)), in one model-based timing analysis tool, a workflow for synchronization of the timing attributes spread across the heterogeneous modeling domains must be in place. Whilst, to the best of our knowledge, such approaches for model-based coupling of related timing attributes and/or model-based timing validation involving heterogeneous modeling domains in ESE projects, have not been proposed. This paper addresses the aforementioned gap, with the following novel contributions:

- A generic workflow and a methodology for synchronization of related timing attributes (*before* performing a timing analysis) in embedded software sub-systems developed using heterogeneous modeling domains.
- An experimental evaluation of the proposed approach, in a state-of-the-art timing analysis tool SymTA/S [4], using a real life, light-weight ESE project example.

The remaining of this paper is organized as follows. Next to this introduction, section II elaborates on the related work. Section III introduces the generic workflow for timing analysis proposed in this paper. An experimental evaluation is presented in section IV. A summary is provided in section V.

II. Related work

Recent developments in model-driven ESE indicate a multi-disciplinary development of real-time embedded systems by combining tools of different disciplines such as UML and Matlab/Simulink [5], [6]. Coupling Simulink and UML models to allow simultaneous simulation is discussed in [5]. Similarly, a mechanism for data interchange among embedded software sub-systems developed using heterogeneous modeling domains is discussed in [6]. However, both [5], [6] do not deal with the coupling of timing properties among sub-systems.

In the case of UML-based ESE, the extensions required for NFP-specific annotations are defined as UML profiles. Modeling and Analysis of Real-Time and Embedded Systems (MARTE) [7] is a popular UML profile for specification of time concepts for real-time and embedded systems [8]. A non-UML based domain, however, finding extensive usage in ESE, for control modeling is Matlab/Simulink [2]. A step towards Worst Case Execution Time (WCET) analysis in Simulink is available in [9]. Control loop timing analysis using Matlab-based analysis tools such as *TrueTime* and *Jitterbug* is discussed in [10]. However, a systematic approach towards specification (as in the UML domain) and validation of timing requirements is not available in Simulink.

In component-based ESE, individual sub-systems are modeled with chains of components that are translated to chains of tasks for scheduling analysis. The timing requirements of such chains, which we coarsely refer to as the *end-to-end delay* can be specified in the component model and also estimated/analyzed during a scheduling analysis. In [11], distinction between different *meanings* of end-to-end timing is presented. Latency requirements can be specified in ESE, for example, using UML-MARTE timing extensions and using specific blocks (e.g. transport delay) in Simulink. The effect of additional functional delays in Simulink models is demonstrated in [12], [13]. Translating such end-to-end timing requirements to a timing analysis model in component-based distributed real-time system is discussed in [14]. However, model-based timing analysis (e.g. end-to-end path delay) of embedded software systems specified using heterogeneous modeling domains is missing in [11], [12], [13], [14].

Specialized timing analysis tools such as SymTA/S [4], MAST [15] provide their own models for describing timing behavior and their validation on a model level. However, often such tools are used for timing analysis/verification only very late in the software development stages. In contrast, an approach towards integration of scheduling analysis into UML based development process through model transformation is discussed in [16]. While [16] can be considered as a first step to perform timing estimates in a schedulability analysis tool, it takes into consideration only one modeling domain. In contrast, the approach presented in this paper considers ESE projects modeled using heterogeneous modeling domains. Of particular interest, is the end-to-end delay analysis of an execution path involving components from two different modeling domains (e.g. UML, Simulink).

Thus, it is seen that there are several modeling techniques to specify, extract and experiment with NFPs. However, an approach for coupling of related timing properties and model-based end-to-end latency analysis of the extracted timing properties, involving more than one modeling domain, in a state-of-the-art timing analysis tool is missing.

III. A Generic Workflow for Timing Evaluation

In this section, an integrated model-based workflow for co-engineering and timing analysis of embedded software sub-systems developed using heterogeneous modeling domains is outlined. The proposed workflow, closely adheres to a general approach for model-based Non-Functional Property (NFP) analysis. For instance, an approach [17] for NPF analysis comprises of the following steps (in brief): (a) add annotations to the design model to describe NFPs (b) define model transformations from annotated software models to formalisms useful for NFP analysis (c) analyze NFPs and (d) give feedback to the designers. However, when the embedded software system involves interdisciplinary teams and heterogeneous modeling domains, the sub-systems may be developed using more than one modeling domain (Fig. 1). In this case, the need arises for co-engineering and exchange/synchronization of related NFPs which are spread/linked across various modeling domains and then perform the steps (c), (d) mentioned above. Following are

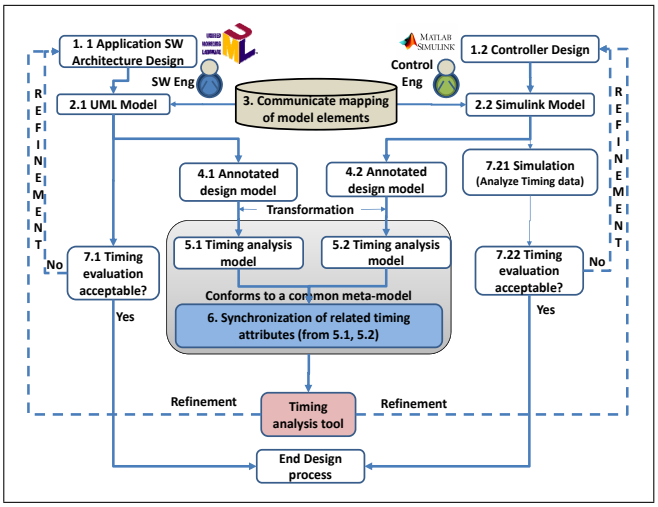

Fig. 2. A generic workflow for timing analysis

some of the aspects considered in the proposed workflow in Fig. 2, to address the aforementioned challenge:

(A) It takes into consideration an ESE scenario, wherein an embedded software system is developed using heterogeneous modeling domains such as UML and Simulink. These two domains are chosen as examples, as they as they are among the two most commonly used modeling domains in ESE projects.

(B) The workflow considers that (a) the overall software system architecture design is created in the UML domain and (b) a majority of the software components are developed in the UML domain (e.g. 70%–90%) and only the components involving control engineering tasks are implemented in Simulink (e.g. 10%–30%). This can be considered as one among the emerging industry practices, in the field of ESE involving UML and Simulink [18] [19].

The basic idea of the workflow is to retain the separation of concerns but only exchange the relevant, related timing attributes which are necessary to perform a timing evaluation. As

seen in Fig. 2, the workflow comprises of two main (vertical) design processes. Steps 1.1 to 7.1 deals with the design of the overall application software architecture (in UML). This overall design in UML may have one or more components requiring implementation in the control engineering domain. Steps 1.2 to 7.22, in Fig. 2, deal with the design of the controller components in Simulink. This controller design may not only implement the components of the overall software architecture (required from step 1.1); but, it could also deal with the integration of this controller (required in the UML domain) with several other hardware aspects for the entire system under consideration. Thereby, the timing attribute(s) of the controller component (required in the UML domain) may depend on further components modeled in Simulink (which are not part of the software architecture design in UML).

Thus, during the design evaluation process, assumptions regarding the quality attributes in the steps 1.1, 2.1 without the knowledge/involvement/inputs about the quality attributes from the Simulink domain may yield incorrect/misleading timing analysis results (Fig. 1-(B)). This is especially true for the components included in overall design in UML (e.g. U3, U4 in Fig. 1), but realised in the control engineering domain. Addressing this aspect, a generic model-driven workflow to synchronize the related timing attributes (among UML, Simulink domains), before performing a timing analysis is described below.

(1) As seen in the steps 2.1 and 2.2 in the workflow in Fig. 2, an initial design model is created based on the requirements from the UML and Simulink domains respectively.
(2) In step 3, both the teams involve in an one-time communication for exchanging information about the related model elements. An example is the exchange of the names and initial NFPs of the components (e.g. M1="Motion-Controller") in the Simulink domain, which implements the respective control software component for the UML domain (e.g. U2="TorqueProtection").
(3) In steps 4.1 and 4.2 the design model is annotated with timing attributes. Using model-to-model transformation, this annotated design model is converted to a timing analysis model (in steps 5.1 and 5.2 respectively). This timing analysis model, conforms with a timing analysis meta-model, which in turn adheres closely with the formalisms/semantics of the timing analysis tool.
(4) A timing analysis could be performed based on the resulting timing analysis model in step 5.1 or 5.2, if only one modeling domain was involved. Since more than one modeling domain is involved here, *before* performing a timing evaluation, the workflow in Fig. 2, introduces a step for model-based co-engineering and synchronization of the timing attributes of software components, which are linked across the two domains.
(5) In step 6 in Fig. 2, synchronization of the related performance attributes in the timing analysis models (created in steps 5.1 and 5.2) is carried out. Examples of synchronizing related timing attributes are provided in section IV.

Note that, once the related timing attributes are synchronized in step 6, a timing validation can be carried out in the timing analysis tool under consideration. An initial iteration of the timing evaluation is considered as complete, once the timing analysis results are fed-back for refinement.

IV. Experimental evaluation

A generic workflow for model-based synchronization of the related timing attributes (e.g. *CET*) and timing evaluation was described in the previous section. In this section, practical examples involving annotation of the design model with the timing attributes for both the UML and the Simulink domains are presented. Aspects such as converting the annotated design model to analysis models and matching of the *tag* attribute in the *runnables* in the analysis models (from UML and Simulink) for synchronizing the related *CET* timing attribute are described with examples. The synchronized timing analysis model is then subject to a timing validation in a state of the art timing analysis tool, namely SymTA/S.

A. A meta-model for timing analysis

A state-of-the-art timing analysis tool SymTA/S [4] is used for timing evaluation of the proposed workflow. SymTA/S supports the exchange of data required for timing analysis using XMI format. This feature enables straight forward means to import/export of timing analysis data between the MDD tools and SymTA/S. On the other hand, there is no standard meta-model for timing analysis defined in advance in SymTA/S. Hence, in this paper a timing analysis meta-model is constructed, which closely adheres with the semantics of the data model used by the SymTA/S tool. Note that the steps 5.1 and 5.2 and step 6 in the workflow in Fig. 2, conforms to a common timing meta-model, described below.

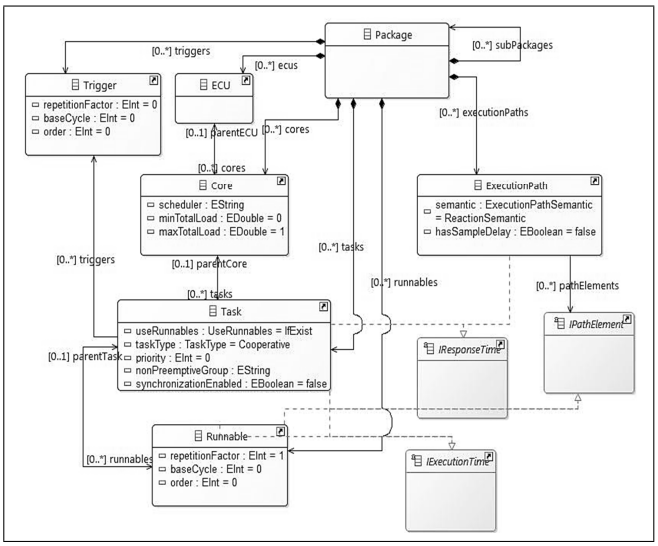

Fig. 3. A meta-model for timing analysis

The meta-model in Fig. 3, created using the Eclipse Modeling Framework (EMF) [20], describes aspects pertaining to timing analysis, which is fully supported by the SymTA/S tool used in this work. Please note that only a simplified (but sufficient) view, of the timing analysis model is presented here, because of space limitations. From Fig. 3, it is seen that the timing analysis meta-model comprises of a package with all the elements 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 *runnables* (e.g.

an operation). Apart from these basic elements, there may be several *execution paths* in a software system, which encompass path elements. The path element may be a *task* or a *runnable* as seen in Fig. 3. 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). Each task and runnable comprise of an attribute to store the execution time (CET). 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.

B. Screwdriver software system

A screwdriver software system is used as a light weight ESE example for experimental evaluation in this paper. Some of the requirements of an electric screwdriver are realised as a screwdriver software system using the MDD approach. The core functionalities of such a screwdriver system considered in the prototype are (a) decoding incoming commands for basic operations and (b) protection mechanisms indicating respective safety hazards (e.g. torque/overheat/undervoltage protection). While the overall embedded software architecture (and the control-logic) is designed in the UML domain (using Rhapsody [21]), the controllers are implemented in Simulink [2].

1) Sub-System modeled using UML: The control-logic sub-system comprises of software components such as *overheat/undervoltage protection*, *torque protection*, *controller*, *bit protection*, *button set* and *task scheduler*. A sub-set of these software components is shown in Fig. 4. The *controller* class

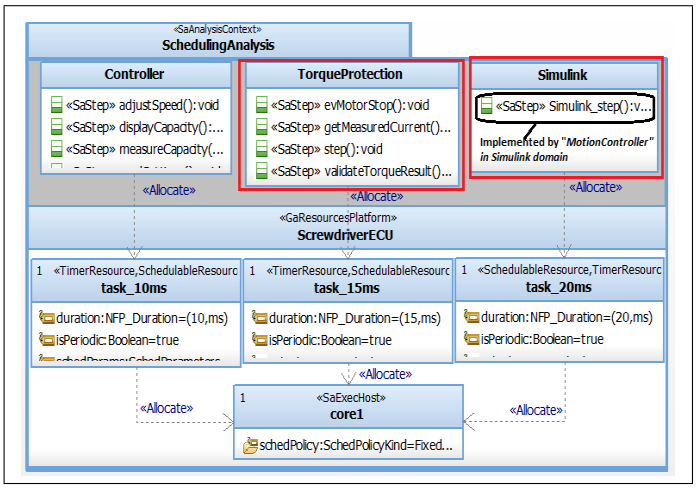

Fig. 4. Screwdriver software system implementation in UML

is responsible for decoding the incoming commands for basic screwdriver operations. The *torque protection* class implements protection mechanism and *Simulink* class is used for interfacing between UML and Simulink domain.

2) Sub-system modeled using Simulink: The closed loop control sub-system for the screwdriver software system is implemented in the Simulink domain (Fig. 5). The controller implementations, handle the queries for parameters received from the discrete modeling domain. For example, upon requirement, an invocation of a controller-step function from the UML domain (e.g. from *torque protection* class) results in communication with the controller (Simulink) to determine if *maximum torque* value (of the motor) is exceeded.

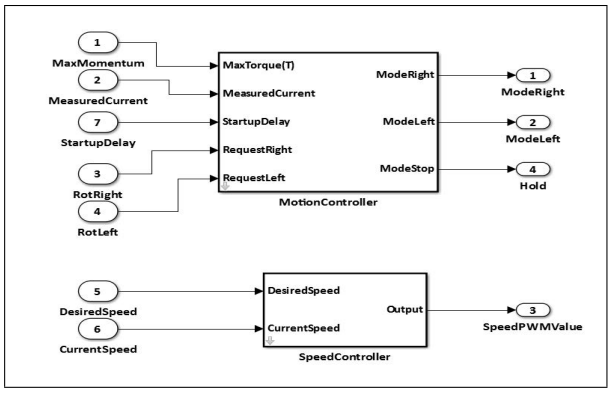

Fig. 5. Screwdriver software system-controller implementation in Simulink

C. Specification of timing properties for screwdriver system

To subject the design model to a timing validation the design specification needs to be annotated with quantitative timing attributes (refer steps 4.1 and 4.2 in Fig. 2). For the software components developed using the UML domain, the existing MARTE [7] profile is used to annotate the design model with the timing properties. As seen in Fig. 4, the aspects necessary for timing analysis such as tasks (*SchedulableResource*), CPU cores (*SaExecHosts*) and scheduling mechanisms (*Scheduler*) are specified with their respective stereotypes, for the screwdriver software system. These elements are placed in a package (stereotyped with *SaAnalysisContext*), to be used for schedulability analysis. Similarly, execution paths are defined by using stereotypes *GaWorkloadBehavior*, *SaEndToEndFlow*, *GaWorkloadEvent*, *SaStep* and *GaScenario* (Fig. 7). Tagged values in the stereotypes such as *deadline* and *execTime* are used for specifying timing properties such as *period* and *CET*.

Extensions such as profiles available in UML are not readily available in Simulink to specify the timing properties. In this work, custom-defined masks are created for blocks/sub-

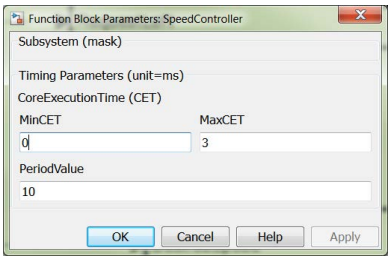

Fig. 6. Custom mask to specify timing attributes for a sub-system in Simulink

systems in Simulink to annotate them with quantitative timing attributes, such as *CET* and *period*, to be used for timing validation. A custom mask created for the *SpeedController* sub-system in Simulink for the screwdriver software system is shown in Fig. 6.

In the next step (steps 4.1 and 4.2 in Fig. 2), based on the timing annotations in the design model, i.e., using MARTE in UML and custom-defined masks in Simulink, instances of the intermediate *timing analysis model* is created for both UML and Simulink domains. In the following, a use case for end-to-end path delay estimation in the screwdriver example is elaborated.

D. A use case for end-to-end path delay estimation

Let us consider the scenario shown in Fig. 7 illustrating the torque protection mechanism in the screwdriver software system. This scenario involving the *torque protection path* is

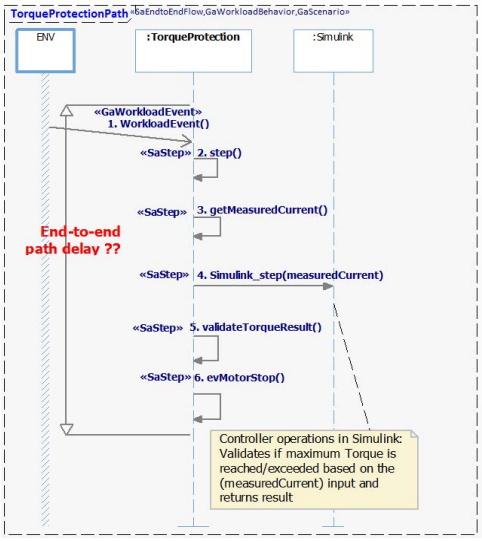

Fig. 7. Series of steps in the torque protection mechanism execution path, in the screwdriver example, involving both UML and Simulink domains

encountered while executing the task involving the *TorquProtection* module. In this execution path (Fig. 7), while the steps 1-3 and 5-6 involves the execution of software components in the UML domain; in the step 4, a function call to the Simulink domain is made. It is straightforward to understand that the end-to-end path latency metric of this execution path involves the timing properties from both the UML and Simulink domain. For instance, the timing property (e.g. CET) of step 4, involving a function call $Simulink\_step(measuredCurrent)$ is dependent on its respective implementation of the controller in the Simulink domain. It is intuitive to understand that the estimation of the end-to-end path latency for such a scenario (Fig. 7), could well be one among the key quality indicators for making proper design decisions; as it involves communication among interdisciplinary, heterogeneous modeling domains. In such scenarios, there arises a need to synchronize the timing properties among the modules developed using different domains and then perform a timing validation.

Applying the generic workflow (in section III) to the screwdriver example, instances of timing analysis meta model created for both the UML and Simulink domain (from steps 5.1 and 5.2) are shown in Fig. 8 and Fig. 9 respectively. These models are created using Model-to-Model (M2M) transformations, wherein, the source is the UML or Simulink design model and the target is the timing meta-model (in Fig. 3). Please note that a Matlab Simulink Integration Framework for Eclipse (MASSIF) [22] is used for converting the annotated Simulink design model, to EMF format. It is then subject to the transformation mentioned above. The M2M transformations are implemented in the Atlas Transformation Language (ATL) [23]. The resulting instances of the UML and Simulink design model conform with the timing meta-model (in Fig. 3).

From Fig. 8, it is seen that the runnable *Simulink_step* comprises of two attributes, namely, the CET and a tag value.

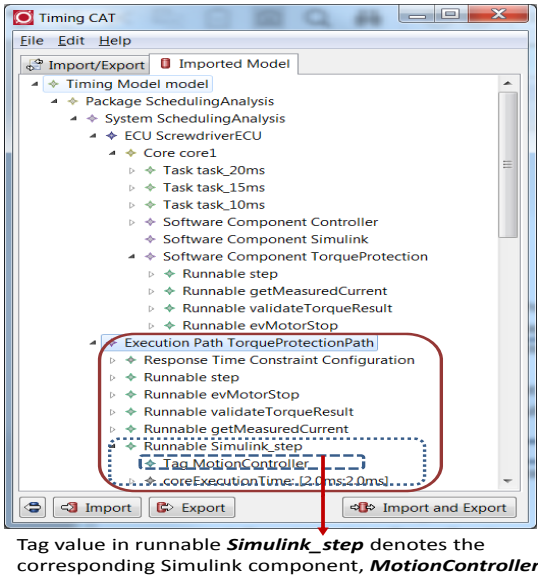

Fig. 8. Instance of the timing analysis model created for the sub-system developed in UML for the screwdriver example

The tag value of *Simulink_step* corresponds to the name of the component in Simulink sub-system which implements this operation/runnable. In this example it is the *MotionController* software component in the Simulink domain, as indicated in Fig. 8. Thus, the CET value of this component *Simulink_step* must be synchronized with the CET value of the *MotionController* from Simulink sub-system (in Fig. 9).

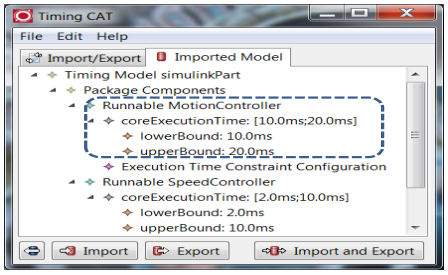

Fig. 9. Instance of the timing analysis model created for the sub-system developed in Simulink for the screwdriver example

Thus, the timing analysis models for both the sub-systems for the screwdriver software system are now available, i.e., the steps 5.1 and 5.2 in Fig. 2 are complete. A synchronization of related timing attributes (step 6 in Fig. 2), must be carried out before performing a timing analysis. In the prototype, the synchronization of *CET* attribute among the runnables from both the domains, is also carried out using simple M2M transformations in ATL. For instance, a synchronization rule (*syncCETValues.atl*) implemented in ATL, takes as input the runnables in the timing analysis model from the UML domain. For every runnable (as input) based on the value of its *tag* field, it checks for the corresponding matching name (of a runnable) in the Simulink domain. Once a match is found, the value of *CET* attribute for the corresponding runnable in Simulink is copied to its respective runnable in UML (e.g. Fig. 8, 9). In our example, the CET value of the runnable *Simulin_step* from the UML domain is synchronized (copied by *synchCETValues*)

with the runnable *MotionController* in the Simulink domain. Once the step involving synchronization of timing attributes is completed (step 6 in Fig. 2), the resulting overall timing analysis model can be subject to a timing validation.

The end-to-end path delay (i.e., the response time) estimated for the *TorqueProtection* mechanism (execution path in Fig. 7), as a result of timing analysis in SymTA/S, is shown as 73.5 ms in Fig. 10. On the other hand, when incorrect

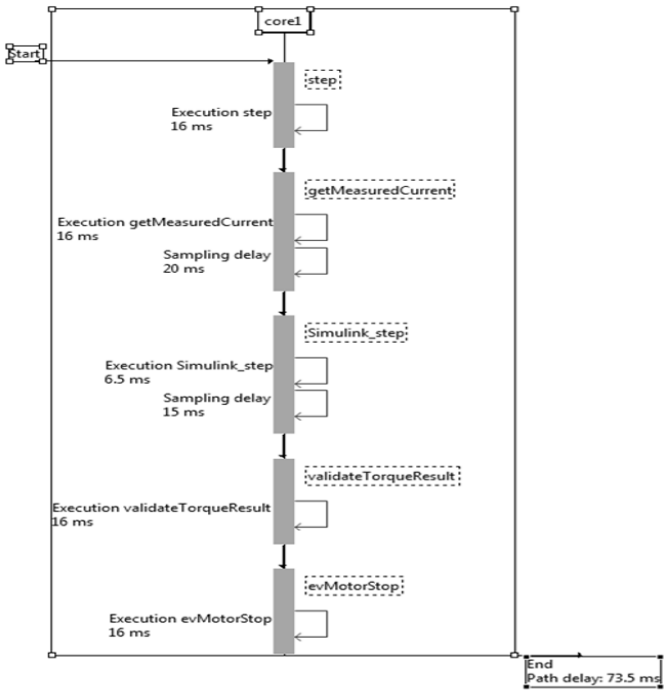

Fig. 10. End-to-end path delay, estimated by the timing analysis tool, for the torque protection mechanism (execution path described in Fig. 7)

CET values are specified for the runnable *Simulink_step* (i.e., (2,2)ms in Fig. 8 instead of (10,20)ms from Simulink-cf. Fig. 9), misleading (lower value) for the response time was estimated as the end-to-end path delay for this execution path. Thus, the need to to synchronize the timing attributes of software components (*before* performing a timing analysis) when they are implemented across heterogeneous modeling domains is reiterated, by this example.

V. Summary

Extending model-based timing analysis approaches to ESE projects cutting across heterogeneous modeling domains, early in the design stages, is an emerging research challenge. A simple example of this is the use case for end-to-end path delay estimation of an execution path involving modules (e.g. operations) from different modeling domains. In such an execution path, the timing properties of software components from different modeling domains may be specified incorrectly, or ignored unintentionally. For instance, this could be because of the involvement of interdisciplinary teams, lack of proper communication among the teams and oversight. To address this challenge, the proposed workflow for timing analysis performs a step to synchronize the timing attributes of the elements which are implemented across various modeling domains (i.e., *before* performing a timing analysis). An experimental evaluation is presented using real-life ESE examples. Future work includes evaluation of the proposed workflow in a sophisticated case study.

Acknowledgment

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] Unified Modeling Language (UML), http://www.uml.org/, 2016.

[2] Matlab and Simulink, http://www.mathworks.com/, 2016.

[3] LabVIEW System Design Software, http://www.ni.com/labview/, 2016.

[4] SymTA/S: Scheduling Analysis & Timing Verification Tool, https://www.symtavision.com/products/symtas-traceanalyzer/, 2016.

[5] J. Hooman, N. Mulyar, and L. Posta, "Coupling Simulink and UML models," in *Proceedings of Symposium FORMATS*, 2004, pp. 304–311.

[6] P. Iyenghar, B. Samson, M. Spieker *et al.*, "A mechanism for data interchange between embedded software sub-systems developed using heterogenous modeling domains," in *3rd International Conference on MDE & Software Development, MODELSWARD'15, France*.

[7] The UML profile for Modeling And Analysis of Real-Time and Embedded Systems (MARTE), http://www.omgmarte.org/, 2016.

[8] M. Z. Iqbal, S. Ali, T. Yue, and L. Briand, "Experiences of Applying UML/MARTE on Three Industrial Projects," in *Proceedings of the 15th International Conference MODELS'12*.

[9] R. Kirner, R. Lang, P. Puschner, and C. Temple, "Integrating WCET Analysis into a Matlab/Simulink Simulation Model," in *Proceedings of 16th IFAC Workshop on Distributed Computer Control Systems 2000*.

[10] A. Cervin, K.-E. Arzen, D. Henriksson, M. Lluesma *et al.*, "Control loop timing analysis using truetime and jitterbug," in *IEEE International Conference Computer Aided Control System Design*, 2006.

[11] N. Feiertag, K. Richter, J. Nordlander, and J. Jonsson, "A compositional framework for end-to-end path delay calculation of automotive systems under different path semantics," in *Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS)*, 2008.

[12] M. Di Natale, L. Guo, H. Zeng, and A. Sangiovanni-Vincentelli, "Synthesis of multitask implementations of simulink models with minimum delays," *IEEE Transactions on Industrial Informatics*, vol. 6/4, 2010.

[13] Z. Al-bayati, H. Zeng, M. Di Natale, and Z. Gu, "Multitask implementation of synchronous reactive models with earliest deadline first scheduling," in *Industrial Embedded Systems (SIES), 2013 8th IEEE International Symposium on*, June 2013, pp. 168–177.

[14] S. Mubeen, J. Mäki-Turja, and M. Sjödin, *Embedded Systems Development: From Functional Models to Implementations*. Springer, 2014, ch. Extracting End-to-End Timing Models from Component-Based Distributed Embedded Systems, pp. 155–169.

[15] MAST tool, http://mast.unican.es/, 2016.

[16] M. Hagner and U. Goltz, "Integration of scheduling analysis into uml based development processes through model transformation," in *Computer Science and Information Technology (IMCSIT), Proceedings of the 2010 International Multiconference on*, Oct 2010, pp. 797–804.

[17] D. C. Petriu, *Software Model-based Performance Analysis*. John Wiley & Sons, Inc., 2013, pp. 139–166.

[18] C. Lauer, R. German, and J. Pollmer, "Fault tree synthesis from uml models for reliability analysis at early design stages," *SIGSOFT Softw. Eng. Notes*, vol. 36, no. 1, pp. 1–8, Jan. 2011.

[19] Willert Software Tools GmbH, http://www.willert.de/, 2016.

[20] Eclipse Modeling Framework (EMF), https://eclipse.org/emf/, 2016.

[21] IBM Rational Rhapsody Developer, Ver 8.2, http://www.ibm.com, 2016.

[22] MATLAB Simulink Integration Framework for Eclipse, http://www.incquerylabs.com/, 2016.

[23] Atlas Transformation Language, https://eclipse.org/atl/, 2016.