Model-Based Tool Support For Energy-Aware Scheduling

Padma Iyenghar, Stephan Wessels, Arne Noyer, Elke Pulvermueller
Software Engineering Research Group, University of Osnabrueck, Germany
{piyengha, stwessels, anoyer, elke.pulvermueller}uni-osnabrueck.de

***Abstract*—Energy-aware scheduling is a challenging research problem that has been studied extensively in the recent decade. Results from such analysis could provide hints on the possible alternative configurations for a schedulable and energy-aware embedded software system design. Towards this direction, model-based co-engineering and trade-off analysis among the inter-related NFPs in Embedded Software Engineering (ESE) is an emerging research area. In this paper, a completely model-based and fully automated tool chain for early estimation and evaluation of the various configurations of the aforementioned inter-related NFPs is presented. Contributions are discussed along with some initial results.**

***Keywords*—Embedded software; Unified Modeling Language (UML); energy-aware scheduling; model-based; tool support;**

I. Introduction

Model-based specification and analysis of Non-Functional Properties (NFPs) has enabled the early detection of potential problems in the underlying design, based on its model, *before* implementing code. This is particularly beneficial in Embedded Software Engineering (ESE), as embedded systems are often subject to critical NFPs such as performance, energy-consumption, reliability and safety. For instance, software model-based performance analysis, power-consumption analysis and dependability assessment based on the Unified Modeling Language (UML) are discussed in [1], [2] and [3] respectively. However, the non-functional aspects are often inter-related in embedded software systems. For example, a more power-aware software system could result in compromising the real-time constraints [4], which may even lead to a catastrophic failure in embedded systems. In this context, energy-aware scheduling is a challenging research problem that has been studied for decades, investigating the trade-off between performance and energy consumption [5].

To understand the practical advantages of energy-aware scheduling, let us consider some examples of battery-powered embedded systems deployed in applications such as wearable devices, industrial controllers and wireless sensor networks. Such systems often necessitate efficient energy management algorithms to provide practical/operational advantages. Early decision on the best set of possible configurations for energy-aware scheduling could translate to, for instance, less frequent recharge/replacement of batteries; a crucial factor in battery-powered embedded systems. A longer lifetime of an embedded software system which also meets performance constraints, clearly provides financial and operational advantages.

In this direction, early model-based co-engineering and trade-off analysis among the inter-related schedulability and energy-consumption NFPs could provide hints, on the possible alternative configurations for a schedulable and energy-aware embedded software system design. A general approach for model-driven NFP analysis, outlined in [1], is shown in Fig. 1. It comprises of the following steps (in brief): (a) add annotations to the design model to describe the NFPs (b) define model transformations from annotated software models to formalisms useful for NFP analysis (c) analyze NFPs and (d) assessment and feedback to the designers. In line

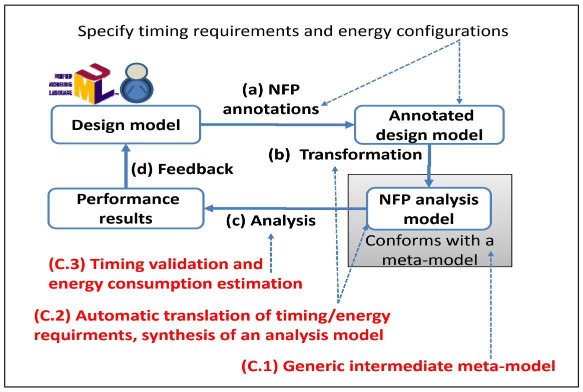

Specify timing requirements and energy configurations
(a) NFP annotations
Design model
Annotated design model
(b) Transformation
(d) Feedback
Performance results
(c) Analysis
NFP analysis model
Conforms with a meta-model
(C.3) Timing validation and energy consumption estimation
(C.2) Automatic translation of timing/energy requirments, synthesis of an analysis model
(C.1) Generic intermediate meta-model

Fig. 1. A general workflow for model-driven NFP analysis and challenges (C.1, C.2, C.3) addressed towards model-based tool support for energy-aware scheduling

with the aforementioned workflow, studies such as [4], [6] carry out UML-based analysis of performance and energy consumption estimation. A system-level multi-view modeling framework, which provides a means to specify functional and non-functional aspects in inter-related views is discussed in [7]. Similarly, a UML-based thermal analysis view of real-time embedded systems is discussed in [8]. While [7], [8] provide an operational framework for joint simulation and analysis, tight integration with state-of-the-art analysis tools is missing. On the other hand, [4], [6] deal with model-based specification and manual analysis of the energy and performance aspects, i.e., steps (a), (c) in Fig. 1. But, they are neither completely model-based nor automatically integrated with timing validation or energy estimation tools.

To summarize the related work, model-based support for a complete tool chain from specification until validation of such inter-related NFPs, integrated with the state-of-the-art model-based design and/or analysis tools, is not yet available. Addressing this gap and in line with the generic workflow for NFP analysis in Fig. 1, the following main aspects are explicitly addressed in this paper towards a complete model-based tool chain support for energy-aware scheduling.

(A) A generic intermediate timing and energy meta-model, as an interface between the design tools (e.g. UML-based tools [9]) and the analysis tools (e.g. SymTA/S [10]). This is referred to as challenge C.1 in Fig. 1.
(B) Support for automated import/export of energy and performance requirements from UML design model to intermediate meta-model and eventually to timing validation/energy estimation tool(s). This aspect is referred to as challenge C.2 in Fig. 1.
(C) Automated support for energy estimation along with timing validation in a timing analysis tool (challenge C.3).

In the remaining of this paper, contributions with examples from a prototype are outlined in section II. A summary is provided in section III.

II. Contributions

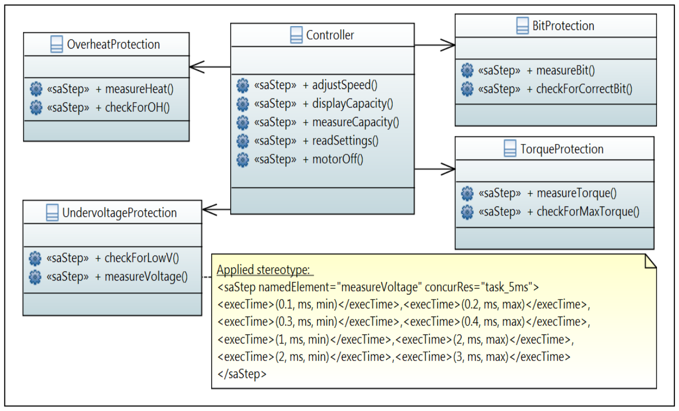
To subject the design model to a timing validation and/or energy estimation, the design specification needs to be annotated with the respective timing and energy attributes. For this model-based energy-aware scheduling analysis two inputs are required (i.e., for step (a) in Fig. 1). First, the underlying CPU configuration[1] modes with power consumption values, which can be obtained from measurements or data sheets (e.g. in the case of Commercial Off-The-Shelf (COTS) products) [6]. Second, a set of timing attributes (e.g. execution times) for the tasks/runnables (i.e., operations/functions) in the underlying software system. For example, the execution times for each runnable/operation can be obtained from Worst Case Execution Time (WCET) analysis tools such as AbsInt [11]. As mentioned earlier, these two inputs are annotated in the UML design model in step (a) in Fig. 1. Given these inputs, the main aim of the proposed contributions to the model-based energy-aware scheduling tool chain is to obtain as output, an analysis of the given configurations indicating the total power consumption per each mode and the schedulability result.

A. *Specification of energy and timing attributes*

Related works on energy-aware scheduling such as [6], [12] employ the existing Modeling and Analysis of Real-Time and Embedded Systems (MARTE) [13] profile for specification of the energy and timing attributes in the UML design model. Whereas, in [4] an extension of the MARTE profile is proposed for providing a power consumption analysis view. The authors consider that the existing stereotypes in MARTE are sufficient for the aims of this work outlined above, rather than reinventing the wheel of NFPs specification using UML.

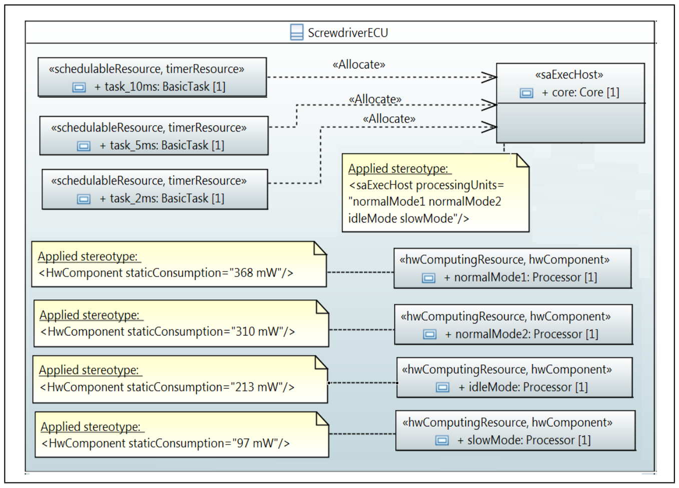
For example, some aspects pertaining to annotation of a UML design model for timing-energy analysis, which is used for prototype evaluation, are shown in Fig. 2 and Fig. 3. Elements required for a timing analysis, such as tasks (*SchedulableResource*), runnables/operations (*SaStep*) and CPU cores (*SaExecHosts*) are specified using their respective stereotypes as seen in Fig. 2. Similarly, the power configuration modes for the CPU are specified using using *staticConsumption* tagged value in the *HWComponent* stereotype as seen in Fig. 3. The execution time of each operation/runnable corresponding to each power mode is specified using the *saStep* stereotype (e.g. using the *execTimes* tagged value, cf. Fig. 2).

[1]Single core architectures are considered for the first prototype evaluation


OverheatProtection
«saStep» + measureHeat()
«saStep» + checkForOH()
Controller
«saStep» + adjustSpeed()
«saStep» + displayCapacity()
«saStep» + measureCapacity()
«saStep» + readSettings()
«saStep» + motorOff()
BitProtection
«saStep» + measureBit()
«saStep» + checkForCorrectBit()
TorqueProtection
«saStep» + measureTorque()
«saStep» + checkForMaxTorque()
UndervoltageProtection
«saStep» + checkForLowV()
«saStep» + measureVoltage()
Applied stereotype:

```
<saStep namedElement="measureVoltage" concurRes="task_5ms">
<execTime>(0.1, ms, min)</execTime>,<execTime>(0.2, ms, max)</execTime>,
<execTime>(0.3, ms, min)</execTime>,<execTime>(0.4, ms, max)</execTime>,
<execTime>(1, ms, min)</execTime>,<execTime>(2, ms, max)</execTime>,
<execTime>(2, ms, min)</execTime>,<execTime>(3, ms, max)</execTime>
</saStep>
```

Fig. 2. An example of a screwdriver software system-UML design model annotated with timing requirements (using MARTE in Papyrus [9])


ScrewdriverECU
«schedulableResource, timerResource»
+ task_10ms: BasicTask [1]
«schedulableResource, timerResource»
+ task_5ms: BasicTask [1]
«schedulableResource, timerResource»
+ task_2ms: BasicTask [1]
«Allocate»
«Allocate»
«Allocate»
«saExecHost»
+ core: Core [1]
Applied stereotype:
Applied stereotype:
Applied stereotype:
Applied stereotype:
Applied stereotype:
«hwComputingResource, hwComponent»
+ normalMode1: Processor [1]
«hwComputingResource, hwComponent»
+ normalMode2: Processor [1]
«hwComputingResource, hwComponent»
+ idleMode: Processor [1]
«hwComputingResource, hwComponent»
+ slowMode: Processor [1]

```
<saExecHost processingUnits=
"normalMode1 normalMode2
idleMode slowMode"/>
<HwComponent staticConsumption="368 mW"/>
<HwComponent staticConsumption="310 mW"/>
<HwComponent staticConsumption="213 mW"/>
<HwComponent staticConsumption="97 mW"/>
```

Fig. 3. The screwdriver software system example annotated with power consumption values for various processing modes of CPU-core

Thus, with the specification of multiple configurations for energy and timing attributes in the UML design model, the inputs for the model-based tool chain (i.e., step (a) in Fig. 1) are now available for analysis. In the next step, these annotated NFPs need to be extracted/translated and an analysis model with energy and timing attributes needs to be synthesized.

B. *Translation of timing/energy NFPs and synthesis of a timing-energy analysis model*

Making use of the benefits of the model-based paradigm, Model-To-Model (M2M) transformations (e.g. implemented using Atlas Transformation Language (ATL) [14]) may be employed to translate the timing and energy requirements in the annotated design model to an analysis model. Such model transformations aid in bridging the large semantic gap between the source (e.g. generic UML-based) and the target model (e.g. with timing/energy specific semantics). In our case, the source model is the UML design model. As a target model, an intermediate timing-energy meta-model as shown in Fig. 4 is implemented and employed during M2M transformations. This implies that the timing and energy NFPs are extracted from the UML design model and a corresponding instance of a timing-energy meta-model is synthesized, i.e., as a result of the M2M transformations. Some advantages of this step are:

(a) The resulting instance of the timing-energy analysis model (based on annotated design model) can be used for analysis across several timing/energy analysis tools.
(b) Coupling of timing/energy properties of two/more sub-design models from one modeling domain (e.g. from UML or Simulink or Labview) based on their resulting instances of the timing-energy analysis models.
(c) Merging/coupling of related timing/energy attributes of the design models created based on completely different modeling domains (e.g. one each from Simulink, UML and Labview)

A simple timing-energy analysis meta-model is implemented in the prototype, using the Eclipse Modeling Framework (EMF) [15], as shown in Fig. 4. The meta-model closely adheres with the semantics and formalisms of generic timing analysis concepts. It also supports the specification of power consumption modes for ECU cores. Please note that, in place of the simplified meta-model proposed in Fig. 4, a recently introduced, however a significantly large common meta-model for multi-core software/hardware modeling (AMALTHEA) available in [16] may be employed. From Fig. 4, it is seen that

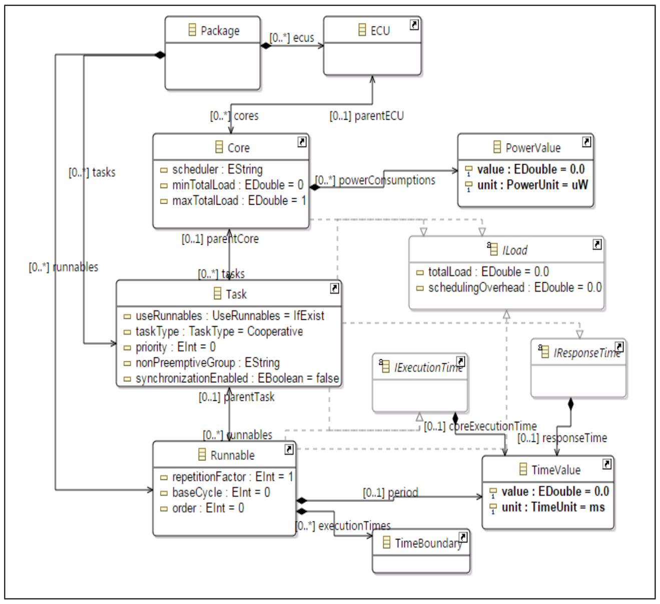
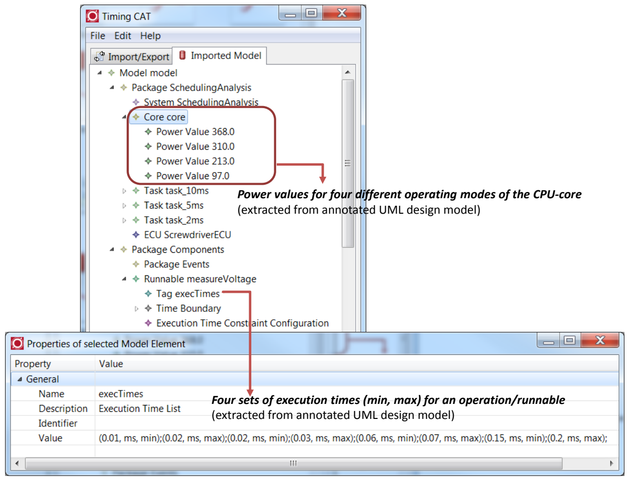


Fig. 4. A simplified view of the timing-energy intermediate meta-model

the 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). 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. Similarly, the power consumption modes specified in the UML design model can be mapped to the attribute *PowerConsumptions* for a core.

In the prototype evaluation, based on the meta-model for both UML [17] and timing-energy analysis (Fig. 4), an instance of the timing-energy analysis meta-model is created for the design model. The source of this M2M transformation is the UML design model annotated with MARTE stereotypes. The


Timing CAT

File Edit Help

Import/Export | Imported Model

- Model model
  - Package SchedulingAnalysis
    - System SchedulingAnalysis
    - Core core
      - Power Value 368.0
      - Power Value 310.0
      - Power Value 213.0
      - Power Value 97.0
    - Task task_10ms
    - Task task_5ms
    - Task task_2ms
    - ECU ScrewdriverECU
  - Package Components
    - Package Events
    - Runnable measureVoltage
      - Tag execTimes
      - Time Boundary
      - Execution Time Constraint Configuration

Power values for four different operating modes of the CPU-core
(extracted from annotated UML design model)

Properties of selected Model Element

Property	Value
General	
Name	execTimes
Description	Execution Time List
Identifier	
Value	(0.01, ms, min);(0.02, ms, max);(0.02, ms, min);(0.03, ms, max);(0.06, ms, min);(0.07, ms, max);(0.15, ms, min);(0.2, ms, max);

Four sets of execution times (min, max) for an operation/runnable
(extracted from annotated UML design model)

Fig. 5. Instance of the timing-energy analysis model created for the annotated UML design models in Fig. 2, 3 (i.e., as a result of M2M transformations)

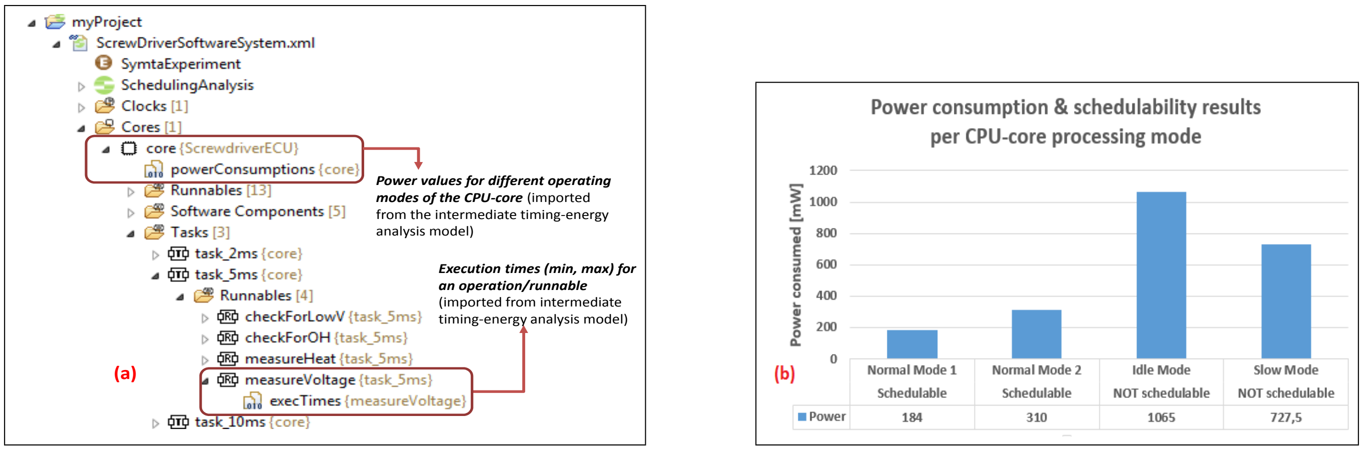
target model is the timing-energy meta-model shown in Fig. 4. The resulting instance of the timing-energy analysis model (corresponding to the design model in Fig. 2, Fig. 3) is as shown in Fig. 5.

C. Timing validation and energy-consumption estimation

The instance of intermediate timing-energy analysis model created corresponding to the UML design model in the previous step is exported using a tool adapter for carrying out a timing validation and energy-consumption estimation. In the prototype, a tool adapter is implemented in the programming language Java for exporting the resulting timing-energy analysis model (persisted in XMI) to the timing validation tool, SymTA/S. The resulting analysis model imported to SymTA/S for validation is shown in Fig. 6-(a). The timing validation is taken care of by the schedulability analysis algorithms implemented in the SymTA/S tool. However, for the energy consumption estimation per CPU operating mode/configuration, an additional plug-in is implemented as part of the prototype implementation in the SymTA/S tool.

Let us consider an example of the timing and power configurations specified in the design model (in Fig. 2, Fig. 3). In this example, four processing modes namely, normal mode 1, normal mode 2, idle mode and slow mode are considered. Their average power consumption ratings are *368, 310, 213* and *97 mW* respectively (taken from data sheet). For each CPU-core processing mode, the corresponding execution times (e.g. obtained from WCET analysis tools) are specified as inputs in the UML design model (e.g. in Fig. 2).

The result of the power consumption estimation and timing analysis are summarized for the above inputs in Fig. 6-(b). The energy estimation script implemented in the prototype, at first calculates the worst-case energy consumption of each runnable, by using the runnable's maximum execution time and the average power consumption of the processing mode. Each task, which is comprised of multiple runnables, then gets a power consumption value corresponding to the energy consumption of all its runnables divided by the task period. The sum of these values form the power consumption rating for each processing mode.



	Normal Mode 1 Schedulable	Normal Mode 2 Schedulable	Idle Mode NOT schedulable	Slow Mode NOT schedulable
■ Power	184	310	1065	727,5

Fig. 6. (a) Imported overall analysis model with power and timing configurations for validation in SymTA/S. (b) Scheduling analysis and power consumption estimation results in SymTA/S

For example, the screwdriver system in Fig. 2, 3 has 13 runnables, which are spread across three tasks with periods 2ms, 5ms and 10ms. Each runnable has the same execution times as seen in the annotation of the *measureVoltage* operation. For the *normalMode* this results in an energy consumption of $73.6mJ$ (i.e., $368mW * 0.2ms$) for each runnable. The 5ms task has four runnables, which results in a power consumption of $58,88mW$ (i.e., $\frac{4*73.6mJ}{5ms} = \frac{294.4mJ}{5ms}$). The power consumption for other tasks are computed in a similar way, thereby resulting in an overall power consumption of $184mW$ (i.e., $73.6mW + 58.88mW + 51.52mW$). Fig. 6-(b) shows the power consumption and schedulability analysis result of each processing mode, calculated by the energy estimation. For instance, the normal modes are both schedulable in contrast to the idle and slow mode. This can be attributed to the significantly higher execution time of the idle/slow modes.

The results from Fig. 6-(b), for instance, could be used to determine the lifespan of a certain battery. It may also be used to estimate the required battery capacity for a certain lifespan of the embedded device under a chosen (schedulable) operating mode of the CPU, for a given embedded software system.

III. Summary

Evaluating a system design before it is built is a good engineering practice. In this context, early model-based evaluation of the NFPs may provide insights regarding the available choices among different configurations for software/system design. Towards this direction, this paper demonstrates a model-based tool chain with complete tool support from specification until validation for energy-aware scheduling, in ESE projects involving the UML domain. In the proposed tool chain, the only manual activity is the annotation of the UML design model with the CPU-core configuration modes (i.e., power values) and the respective execution times for the underlying software system. The timing analysis and energy-estimation based on the inputs, comprise of analysis results which indicate the total power consumption in each CPU configuration mode and if each mode is schedulable. Such early results regarding energy-aware scheduling provide significant advantage for software/system architects to decide on various configurations for the underlying software/hardware system.

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

[2] D.-H. Kim, J.-P. Kim, and J.-E. Hong, *A Power Consumption Analysis Technique Using UML-Based Design Models in Embedded Software Development*. Springer Berlin, SOFSEM 2011, pp. 320–331.

[3] S. Bernardi, J. Merseguer, and D. C. Petriu, *Model-Driven Dependability Assessment of Software Systems*. Springer Inc, 2013.

[4] M. Hagner, A. Aniculaesei, and U. Goltz, "UML-Based Analysis of Power Consumption for Real-Time Embedded Systems," in *10th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, 2011, pp. 1196–1201.

[5] M. Bambagini, M. Marinoni, H. Aydin, and G. Buttazzo, "Energy-aware scheduling for real-time systems: A survey," *ACM Trans. Embed. Comput. Syst.*, vol. 15, no. 1, pp. 7:1–7:34, Jan. 2016.

[6] E. Andrade, P. Maciel, T. Falcão *et al.*, "Performance and energy consumption estimation for commercial off-the-shelf component system design," *Innovations in Systems and Software Engineering*, vol. 6, no. 1, pp. 107–114, 2009.

[7] A. Khecharem, C. Gomez, J. DeAntoni *et al.*, "Execution of heterogeneous models for thermal analysis with a multi-view approach," in *Specification and Design Languages (FDL), 2014 Forum on*, vol. 978. IEEE, 2014, pp. 1–8.

[8] M. Hagner and S. Kolatzki, "UML-Based Thermal Analysis of Real-Time Embedded Systems," in *Power, Energy, and Temperature Aware Real-Time Systems, 33rd IEEE Real-Time Systems Symposium*, 2012.

[9] Papyrus UML Tool, http://www.papyrusuml.org/, 2016.

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

[11] AbsInt toolchain, http://www.absint.com/, 2016.

[12] D. Shorin and A. Zimmermann, "Evaluation of Embedded System Energy Usage with Extended UML Models," in *2nd Workshop EASED@BUIS*, 2013.

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

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

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

[16] AMALTHEA Project:, https://itea3.org/project/amalthea.html, 2016.

[17] Object Management Group, http://www.omg.org, 2016.