A Model-Driven Framework for Hardware-Software Co-design of Dataflow Applications (extended version)

Waheed Ahmad, Bugra M. Yildiz, Arend Rensink, and Mariëlle Stoelinga

University of Twente, The Netherlands
{w.ahmad, b.m.yildiz, arend.rensink, m.i.a.stoelinga}@utwente.nl

Abstract. Hardware-software (HW-SW) co-design allows to meet system-level objectives by exploiting the synergy of hardware and software. Current tools and approaches for HW-SW co-design face difficulties coping with the increasing complexity of modern-day application due to, e.g., concurrency and energy constraints. Therefore, an automated modeling approach is needed which satisfies modularity, extensibility, and interoperability requirements. Model-Driven Engineering (MDE) is a prominent paradigm that, by treating models and model transformations as first-class citizens, helps to fulfill these requirements. This paper presents a state-of-the-art MDE-based framework for HW-SW co-design of dataflow applications, based on synchronous dataflow (SDF) graph formalism. In the framework, we introduce a reusable set of three coherent metamodels for creating HW-SW co-design models concerning SDF graphs, hardware platforms and allocation of SDF tasks to hardware. The framework also contains model transformations that cast these models into priced timed-automata models, the input language of the well-known model checker UPPAAL CORA. We demonstrate how our framework satisfies the requirements of modularity, extensibility, and interoperability in an industrial case study.

1 Introduction

Hardware-software (HW-SW) co-design is an engineering practice that allows to meet system-level objectives by exploiting the synergy of hardware and software through their simultaneous design. For instance, HW-SW co-design allows exploring design alternatives, and helps to improve the development cost and time-to-market. However, current tools and approaches for HW-SW co-design have difficulties coping with the concurrency and increasing complexity of modern-day systems. As a result, the time and effort needed for modeling and validating such designs are negatively affected. In fact, it has been widely recognized that a HW-SW co-design approach must have the following features [4, 10, 15]:

– *Modularity* [4, 10]: The modeling approach should separate different aspects — such as hardware, software and their mappings — to keep their various concerns modular. This allows convenient exploration of design alternatives

concerning hardware and software. Modules targeting different concerns are better maintainable and reusable.
- *Extensibility* [10]: The HW-SW co-designing approach should have convenient extension mechanisms allowing rapid implementation of possible future requirements. This reduces not only the development cost of new products, but also their time-to-market.
- *Interoperability* [4,15]: HW-SW co-designing often involves tools serving different purposes, such as model designing, simulating, integrating etc. The HW-SW modeling approach should support interoperability between these tools, enabling system designers to explore design alternatives rapidly.

Current HW-SW co-designing approaches [4, 10, 12, 15] do not fulfill all of these criteria, leading to following limitations:

- *Modularity:* If the hardware platform, software application, and mapping of the application to the platform are not designed as separate models, the final model includes a combined representation of all these three aspects. As a result of this, changes that occur in any of these aspects will require modifications in the combined representation. For instance, if one wishes to analyze a particular software application on various platforms, there is no software model that is explicitly defined as a separate module to be reused. A similar problem occurs if one needs to analyze several applications on a particular platform.
- *Extensibility:* No systematic extension mechanism for satisfying future requirements causes the addition of the new functionality to the existing co-design to be more costly. This also causes delay in time-to-market.
- *Interoperability:* As mentioned earlier, the HW-SW co-designing process often includes tools serving different purposes. Furthermore, these tools use different models of computation and programming languages. No systematic support for interoperability results in lack of flexible orchestration between several tools to work together. Moreover, lack of systematic interoperability also makes it difficult to integrate new tools in the framework.

Model-Driven Engineering (MDE) is an approach that helps to fulfill the aforementioned requirements [26]. In MDE, the important concepts of the target domain are formally captured in a so-called, *metamodel*. Separate metamodels for the domains of interest help to keep the design modular. All models are instances of a metamodel, or possibly an integrated set of metamodels. Moreover, models can be transformed to the other via *model transformations*, defined at the metamodel level.

In this paper, we model software applications as Synchronous Dataflow (SDF) graphs [16] which are partitioned into tasks, with inter-task dependencies and their synchronization properties. SDF graphs are well-known computational models for real-time streaming and dataflow applications. This paper presents a novel HW-SW co-design framework based on the principles of MDE. Our framework allows model-driven HW-SW co-designing of SDF applications mapped on multiprocessor hardware platforms, and generate energy-optimal schedules for these

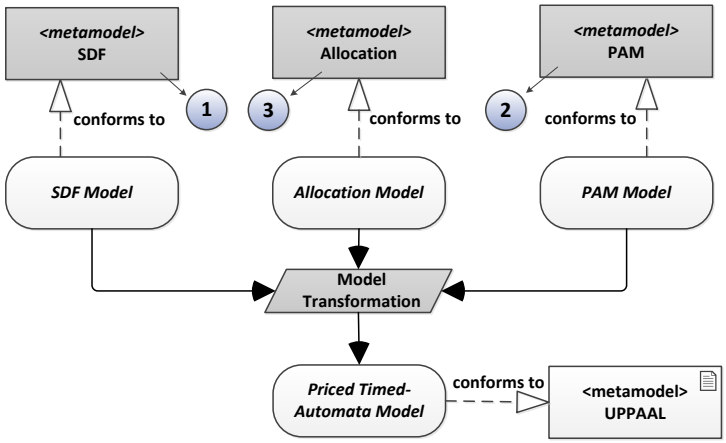

Fig. 1: Overview of our framework

SDF applications. To generate energy-optimal schedules, our framework transforms the co-designed SDF and hardware models to priced timed-automata models. The problem of finding energy-optimal schedules (while satisfying minimal throughput requirements) is encoded as an optimization problem, defined as a reachability property over priced timed-automata models. The property is then checked by the model checker UPPAAL CORA [5] that generates an energy-optimal schedule.

Our framework consists of three metamodels, as shown in Figure 1: (1) a metamodel for SDF graphs; (2) a metamodel for Platform Application Models (PAMs), which describe the processor types and their power levels, and the cost of switching between the power levels; and (3) a metamodel for expressing potential allocations of the tasks in an SDF graph to the processor types in a PAM. As mentioned earlier, our framework considers the model checker UPPAAL CORA for generating energy-optimal schedules. Therefore, for supporting the generation of UPPAAL CORA models, we also use an existing UPPAAL metamodel developed at the University of Paderborn [1]. The models conforming to three metamodels explained earlier, are transformed to UPPAAL CORA models automatically via *model transformations* in the framework. We have already described our method of using priced timed-automata for the purpose of energy optimization in [2], and therefore out of scope of this paper. Rather the novelty of this paper is the design prospects of using MDE. Later in Section 5, we demonstrate our framework as an evidence to show the benefits of MDE namely, modularity, extensibility, and interoperability. The main contributions of this paper are as follows:

– We introduce the insights of state-of-the-art model-driven engineering approach into the embedded systems community, in particular for the domain of HW-SW co-design.

– We propose a reusable set of three coherent, extensible metamodels for HW-SW co-design. [1]
– We define and apply model transformations from the dataflow domain to the model-checking domain, obtaining an automated tool to compute energy-optimal schedules for dataflow applications.
– We demonstrate that our fully automated framework provides modularity, extensibility and interoperability between tools, via an industrial case study.

The rest of the paper is structured as follows: Section 2 provides the related work. Section 3 gives an overview of our framework and Section 4 describes the framework components in detail. Section 5 evaluates our framework using an industrial case study, and Section 6 concludes the paper.

2 Related Work

There exists a plethora of commercial and academic tools for HW-SW co-designing [4, 6, 9, 10, 12, 15]. Due to space limitations, we only present the closest related work.

The state-of-the-art toolsets in the realm of HW-SW co-design of dataflow applications, are Octopus [4] and Ptolemy [15]. The Octopus toolset [4], in comparison to our approach, does not consider any metamodels. Furthermore, this toolset uses Java libraries for model transformation. Rather than Java, which is a general-purpose language, we use ETL that is specifically designed as a domain-specific language for model transformations. The lack of metamodels and model transformation language cause challenges in extensibility and maintainability, which are in fact stated as a future directions of the work in [4]. Ptolemy [15] is another well-known toolset for supporting HW-SW co-design of dataflow applications. However, similar to Octopus, it is not based on MDE which poses challenges in reusability and maintainability.

The closest works to ours are presented in [6] and [9]. Both of these papers utilize MDE techniques for HW-SW co-designing of embedded systems. In contrast to our work, these papers consider generic software applications and hardware models. We, on the other hand, analyze real-life software applications and hardware models enabled with power management.

To the best of our knowledge, this paper presents the first model-driven HW-SW co-design framework for dataflow applications that provides modularity, interoperability, and extensibility.

3 The Model-Driven Framework

3.1 Model-Driven Engineering

Models are powerful tools to express behavior, structure and other properties in many domains such as mathematics, engineering, and other natural sciences.

[1] All metamodels, model transformations, and case studies discussed in this paper can be found at `https://github.com/utwente-fmt/COMET`. An instruction manual for replicating the experiments is also given in this repository.

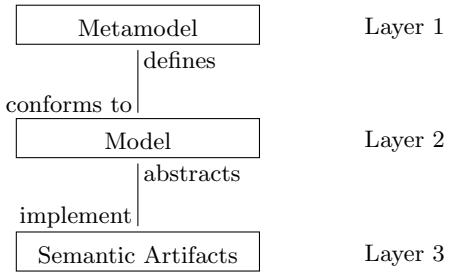

Fig. 2: Metamodeling layered approach [18]

Model-Driven Engineering (MDE) is a software engineering approach that considers models not only as documentation, but also adopts them as the basic abstraction to be used throughout all engineering disciplines and in any application domain [7]. The models in MDE are closer to some particular domain concepts rather than the computing concepts. These models are considered equal to the code since they are formally defined and have execution semantics.

To define models, we need to specify their language as a model of these models at a more abstract level that is so-called *metamodel*s. Metamodeling is the modeling of models. In their common use, metamodels define the permitted structure and behaviour, to which models must adhere. Therefore, metamodels describe the syntax of models [20]. The layered approach used in metamodeling [18] is shown in Figure 2. In Figure 2, the artifact in each layer conforms to, or is abstracted by the adjacent layer. Therefore, semantic artifacts in Layer 3 are abstracted in models defined in Layer 2, which further conform to the metamodels given in Layer 1. For example, some information —such as a task graph— is a semantic artifact, located at Layer 3. The XML file representing this information is a model of this semantic artifact, which is located at Layer 2. Finally, the DTD or XSD schema that this XML file conforms to can be counted as a metamodel, located at Layer 1. The design process that utilizes metamodeling first abstracts the concepts of some domain or application (Layer 3) into the metamodel (Layer 1). Afterwards, the models (Layer 2) defining the domain conforming to their metamodels are generated.

There can be multiple domains in a project. The tools in these domains have typically their own I/O format for their models such as XML, JSON or even plain text. MDE allows interoperability between different domains (and tools in these domains) via *model transformation*s. Models transformations satisfy interoperability and furthermore save effort and reduce errors by automating the model derivation and modification process.

3.2 Overview of Model-Driven Framework

Figure 3 shows the detailed overview of our framework introduced earlier in Figure 1. The HW-SW co-design of the application consists of the first four steps:

- In step 1, an SDF model of the software application is created using the SDF^3 tool [23] in an XML format specific to the tool.
- In step 2, the SDF model is transformed to an SDF model that conforms to the metamodel we defined for SDF graphs.
- In step 3, the hardware platform model is created using *PAM Visual Editor* that is a graphical editor for specifying Platform Application Models (PAMs). This model conforms to the PAM metamodel we defined for PAMs.
- In step 4, an allocation model is created for specifying the mapping of the tasks in the SDF model to the processor types in the PAM.

The analysis of the co-design for energy-optimal schedules is conducted using the UPPAAL CORA model checker. This is achieved in the last three steps:

- In step 5, the co-design is transformed to a priced timed-automata model that conforms to the UPPAAL metamodel.
- In step 6, the priced-timed-automata model is transformed to the format accepted by the model checker.
- In step 7, we analyze the resulting model to compute the energy-optimal schedule using the UPPAAL CORA model checker.

Further details related to the elements of the framework are described in Section 4.

Although the steps in Figure 3 show a general guideline for a HW-SW co-design of a system from scratch, a different strategy can be adopted according to the requirements of the system design. For example, if a system designer needs to analyze how a software application runs on various hardware platforms, s/he can create an SDF model by follow steps 1 and 2 only once and then create several PAM models by conducting step 3 multiple times.

3.3 Tooling Choices

To realize the model-driven approach, we have created metamodels using ECore in Eclipse Modeling Framework (EMF) [21]. EMF provides a plethora of plugins to support various functionalities, such as querying, validation, and transformation of EMF models. For instance, using the EuGENia plugin [14], we have created PAM Visual Editor based on Eclipse Graphical Editing Framework (GMF). To ensure the well-formedness of the metamodels, we have defined The Object Constraint Language (OCL) rules. OCL is a precise text language to express constraints on metamodels that cannot otherwise be expressed by diagrammatic notation [17].

The model transformations have been implemented using Epsilon Transformation Language (ETL) [13], which is one of the domain-specific languages provided by the Epsilon framework. ETL supports many input-to-many output model transformations; it also allows the users to inherit, import and reuse other Epsilon modules in the transformations.

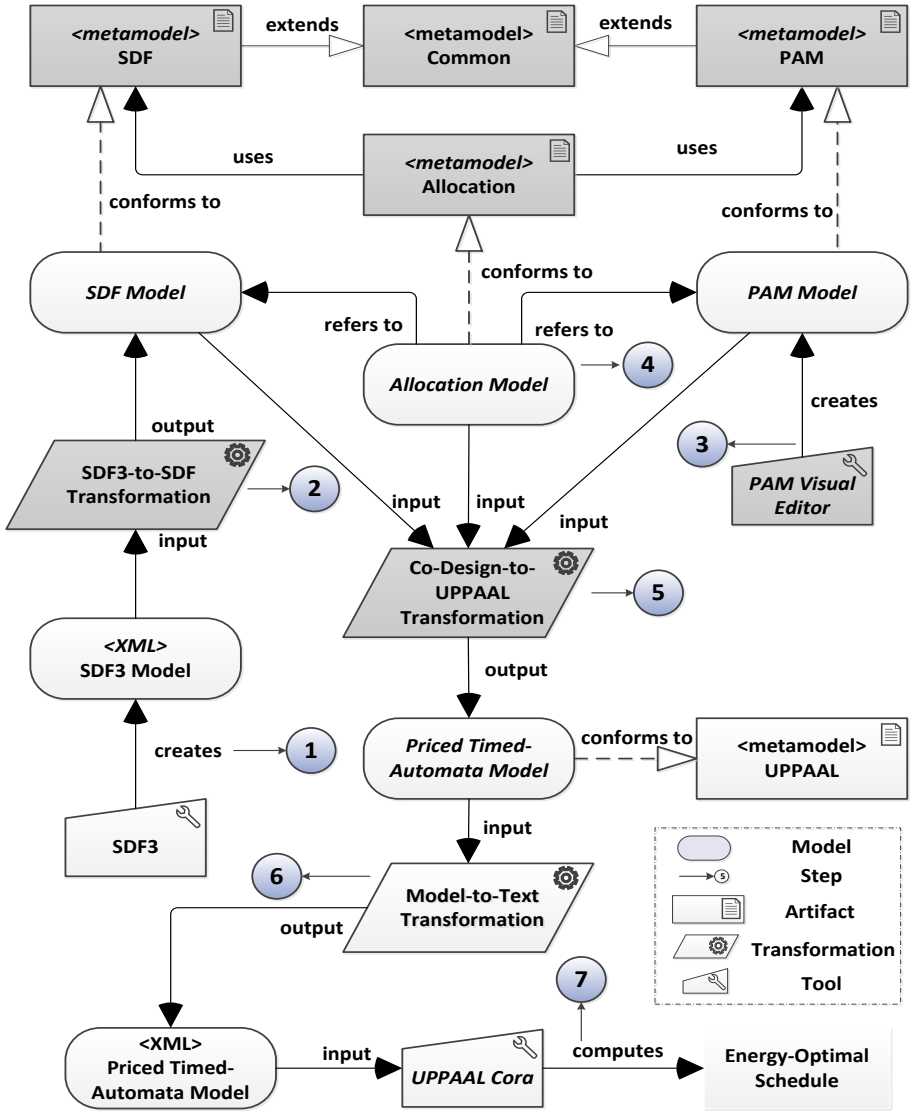

Fig. 3: Detailed overview of our framework. The elements with dark background color represent the new contributions.

3.4 Priced Timed Automata

Timed automata are a popular and powerful formalism to model and analyze real-time systems [3]. A timed automaton is basically a finite automaton extended with real-valued clocks. The clocks can be used in location invariants and edge guards to restrict and guide the behavior of a timed automaton. The

clocks can be also reset. Priced timed automata extend timed automata with costs. Costs can either be accumulated at locations as a rate r such that the accumulated cost in that location over a time period d grows with $r.d$; or a constant value associated with an edge.

We use $B(C)$ to denote the set of clock constraints for a finite set of clocks C. That is, $B(C)$ contains all of conjunctions over simple conditions of the form $x \bowtie c$ or $x - y \bowtie c$, where $x, y \in C$, $c \in \mathbb{N}$ and $\bowtie \in \{<, \leq, =, \geq, >\}$.

Definition 1. *A* priced timed automaton *over clocks C is a tuple $(L, l^0, Act, E, \mathcal{P}, Inv)$, where*

- *L is a set of locations;*
- *$l^0 \in L$ is the initial location;*
- *Act is a finite set of actions, co-actions and internal λ-actions;*
- *$E \subseteq L \times Act \times B(C) \times 2^C \times L$ is a set of edges;*
- *$\mathcal{P} : (L \cup E) \to \mathbb{N}$ assigns costs to edges and locations; and*
- *$Inv : L \to B(C)$ assigns an invariant to each location.*

Example 1. Figure 4 shows an example of a priced timed-automaton of a lamp and an user. The priced timed-automaton of the lamp has three locations, i.e., off, dim and full. Initially, the lamp is in off location. The priced timed-automaton of the user contains one location only, i.e., idle. If the user presses a switch once and synchronizes with press, then the lamp is on and emits dim light, while consuming power equal to 4 kW/h (indicated by the differential equation $\mathsf{p}' == 4$). The user has to press again to switch off the lamp, or to get full light. If full light is required, the switch must be pressed rapidly (in less than 5 time units indicated by the condition $\mathsf{y} < 5$), in which case the lamp consumes more power (8 KW/h). But to switch off the lamp, the user has to wait at least 5 time units (indicated by the condition $\mathsf{y} \geq 5$). The clock y is used to detect if user is fast ($\mathsf{y} < 5$) or slow ($\mathsf{y} \geq 5$). The cost variable is represented by p.

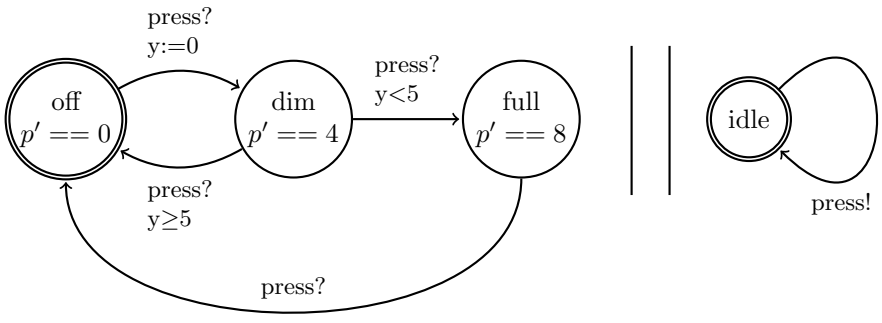

Fig. 4: Priced timed-automaton of a lamp and an user

Using priced timed automata formalism, optimal scheduling problem can be formulated as a cost-optimal reachability problem that is to find the minimum cost of reaching a target state of a system.

4 Details of the Model-Driven Framework

This section presents our concrete instantiation of the model-driven framework by describing our modeling choices in some detail. We recall the formal (mathematical) definitions of the domain concepts and discuss how we have chosen to translate them to metamodel elements.

4.1 SDF Graphs

Typically, real-time streaming applications execute a set of periodic tasks, which consume and produce a fixed amount of data. Such applications are naturally modeled as SDF graphs.

Definition and Metamodel An SDF graph is a directed, connected graph in which tasks are represented by *actors*. Actors communicate with each other via streams of data elements, represented by *channels* (the streams) that carry *tokens* (an abstraction of the data elements). Each channel (a, b, p, q) connects a producer actor a to a consumer actor b, and specifies *production* and *consumption* rates p and q, respectively, both given by integer values. The execution of an actor a is known as the *firing* of a; as a result, q tokens are removed from any channel (b, a, p, q) of which a is the consumer, and q tokens are added to any channel (a, b, p, q) of which a is the producer. Formally:

Definition 2. *An* SDF graph *is a tuple* $G = (A, D, \mathsf{Tok}_0)$ *where* A *is a finite set of actors,* $D \subseteq A^2 \times \mathbb{N}^2$ *is a finite set of channels, and* $\mathsf{Tok}_0 : D \to \mathbb{N}$ *denotes the initial number of tokens on each channel.*

Some notation: given an SDF graph G as above, the sets of *input* and *output channels* of an actor $a \in A$ are defined respectively as $In(a) = \{(b, a, p, q) \in D \mid b \in A, p, q \in \mathbb{N}\}$ and $Out(a) = \{(a, b, p, q) \in D \mid b \in A, p, q \in \mathbb{N}\}$.

Example 2. Figure 5 shows the SDF graph of the Viola-Jones face detector [25], kindly provided by the company Recore Systems, that is used in their face recognition system. The SDF graph contains seven actors (*im_read*, *dupl_im*, *integral*, *haar_det*, *haar_scal*, *col_obj* and *grp_rect*) representing the tasks performed in face detection. For example, *im_read* captures the scene containing one or more faces, *haar_det* detects the regularities in the human face called *Haar features*, and *grp_rect* groups the rectangles having similar Haar features.

The *SDF Metamodel* capturing the concepts of Definition 2 is shown in Figure 6. Recall that an SDF graph is a tuple $G = (A, D, \mathsf{Tok}_0)$.

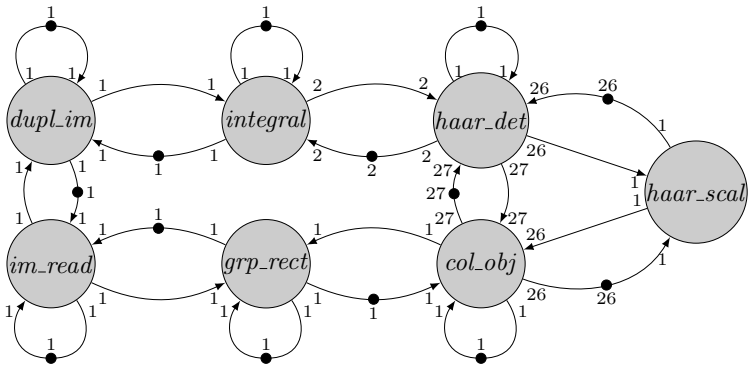

Fig. 5: SDF graph of Viola-Jones face detector

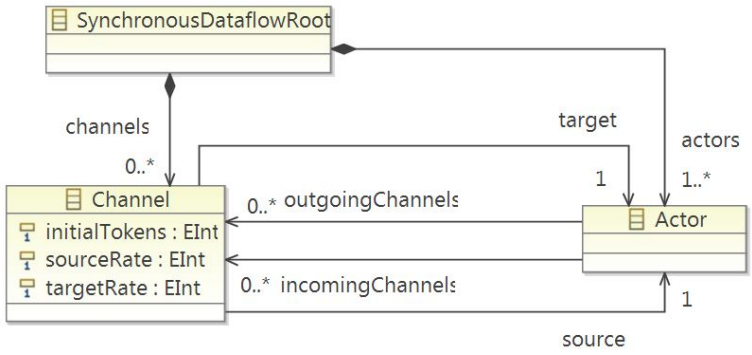

Fig. 6: SDF Metamodel

– *SynchronousDataFlowRoot* is the root of a model, in which everything else is contained; it corresponds to G.
– *Actor* corresponds to the set A; the associations *incomingChannels* and *outgoingChannels* represent the derived functions *In* and *Out* from A to sets of channels.
– *Channel* corresponds to the set D. The 4-tuples $(a, b, p, q) \in D$ are represented in the metamodel by the *source* and *target* associations (for a and b), respectively the *sourceRate* and *targetRate* attributes (for p and q). *initialTokens* represents the function Tok_0; thus, it has been modeled as an attribute of *Channel*, rather than as a separate function.

With respect to the mathematical definition, there are two differences:

(i) whereas a channel (a, b, p, q) is completely determined by its constituent values, due to the nature of metamodels a *Channel* has its own identity (and so conceivably there could be two *Channel*s with the same 4-tuple of values), which can not occur in the mathematical set up in Definition 2; (ii) the function Tok_0 has been combined with *Channel*. This removes some of the modularity of the mathematical model, at the benefit of simplicity.

Model Creation In our framework, SDF models are created in steps 1 and 2 of Figure 3. The starting point is an SDF graph created using the well-known open-source SDF^3 tool [23] (step 1). This tool produces output in the form of an XML document, adhering to its own schema (fixed in an XSD). To bring such documents into our framework, we have defined an SDF^3*-to-SDF Transformation* which produces models conforming to the SDF metamodel of Figure 6. The transformation definition involves a systematic mapping of the SDF^3 concepts to our SDF metamodel concepts.

4.2 Platform Application Models

A Platform Application Model (PAM) models the multi-processor platform to which the application, modeled as an SDF graph, is mapped. Our PAMs support several features, including

(1) heterogeneity, i.e., there can be multiple processors with different types, (2) frequency levels each processor can run on, (3) a partitioning of the processors in voltage/frequency islands, (4) power consumed by a processor at a certain frequency, both when in use and when idle, and (5) power overhead required to switch between frequency levels.

Partitioning processors into voltage/frequency islands (VFIs) allows us to run a group of processors at a common voltage/frequency. The clock voltage/frequencies supplies of a VFI may differ from other VFIs. In absence of VFIs, we are left with two options only, i.e., either running each processor at an individual voltage/frequency, or running all processors at the same voltage/frequency. Hence, VFIs provide better control over energy optimization.

Definition and Metamodel

Definition 3. *Given an SDF graph $G = (A, D, \mathsf{Tok}_0)$ with a set of actors A, a* platform application model *(PAM) is a tuple $\mathcal{P} = (\Pi, \zeta, F, P_{idle}, P_{occ}, P_{tr}, \tau_{act})$ consisting of*

- *a finite set of processors $\Pi = \{\pi_1, \ldots, \pi_n\}$. We assume that Π is partitioned into disjoint blocks of voltage/frequency islands (VFIs) such that $\bigcup \Pi_i = \Pi$, and $\Pi_i \cap \Pi_j = \emptyset$ for $i \neq j$,*
- *a function $\zeta : \Pi \to 2^A$ indicating which processors can handle which actors,*
- *a finite set $F = \{f_1, \ldots, f_m\}$ of discrete frequencies available to all processors,*
- *a function $P_{occ} : \Pi \times F \to \mathbb{N}$ denoting the power consumption (static plus dynamic) of a processor operating at a certain frequency $f \in F$ in the operating state,*
- *a function $P_{idle} : \Pi \times F \to \mathbb{N}$ denoting the power consumption (static) of a processor operating at a certain frequency $f \in F$ in the idle state,*
- *a partial function $P_{tr} : \Pi \times F^2 \nrightarrow \mathbb{N}$ denoting the transition overhead between frequencies for each processor $\pi \in \Pi$, and*
- *a function $\tau_{act} : A \times F \to \mathbb{N}_{\geq 1}$ denoting the actual execution time of each actor (in A) mapped to a processor at a certain frequency level (in F).*

No.	Frequency(MHz)	P_{idle}(W)	P_{occ}(W)
1	1400	0.4	4.6
2	1222	0.3	3.2
3	1033	0.1	1.8

Table 1: Example platform description

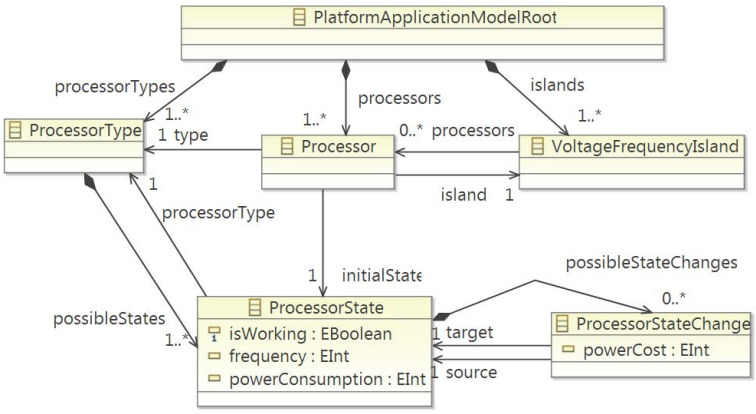

Fig. 7: PAM Metamodel

Example 3. Exynos 4210 is a state-of-the-art processor used in high-end platforms such as Samsung Galaxy Note, SII, etc. Table 1 shows three frequencies (MHz) $\{f_1, f_2, f_3\} \in F$ [19] and corresponding experimental power consumption. We assume that our PAM contains four Exynos 4210 processors, i.e., $\Pi = \{\pi_1, \pi_2, \pi_3, \pi_4\}$. The processors are partitioned into 2 VFIs, i.e., $\Pi_1 = \{\pi_1, \pi_2\}$ and $\Pi_2 = \{\pi_3, \pi_4\}$. We assume that the power overhead (W) of all $\pi \in \Pi$ is, $P_{tr}(\pi, f_1, f_2) = P_{tr}(\pi, f_2, f_3) = 0.2$ and $P_{tr}(\pi, f_3, f_2) = P_{tr}(\pi, f_2, f_1) = 0.1$.

The *PAM Metamodel* capturing most of the concepts of Definition 3 is shown in Figure 7. A brief explanation can be given as follows:

- *PlatformApplicationModelRoot* stands for the PAM as a whole.
- *ProcessorType* collects the characteristics of a set of processors. In the metamodel, the power and frequency characteristics of a processor are associated with its *type*, creating a reusable layer of indirection with respect to the mathematical model.
- *Processor* stands for the elements of Π. Each *Processor* has a *type* association to the corresponding *ProcessorType*.
- *VoltageFrequencyIsland* stands for the clusters Π_i in the VFI partitioning of Π. The element-of relationship between a processor and its VFI is captured by the (opposite) *island* and *processors* associations.

– *ProcessorState* associates the working/idle state of a processor (type) (the boolean *isWorking* attribute), combined with a *frequency* level, to a *powerConsumption* value. This encodes the P_{occ} and P_{idle} functions of the mathematical definition.
– *ProcessorStateChange* encodes the P_{tr} function of the definition: each instance associates a *powerCost* with a certain pair of *source* and *target ProcessorState*s.

In a major change with respect to the mathematical definition, we have chosen not to include the ζ and τ_{act} functions in the PAM, but to isolate them in a separate allocation model. This enhances the modularity of the modeling framework. Apart from this change, all elements of Definition 3 are clearly recognizable in the metamodel, though sometimes encoded in a different manner. In particular, we have introduced the processor types as an intermediate level to enhance modularity; P_{occ} and P_{idle} are combined in *ProcessorState*; and P_{tr} is encoded as *ProcessorStateChange*.

Model Creation The creation of PAMs corresponds to step 3 in Figure 3. Although EMF provides a default tree-based model editor, we have built PAM Visual Editor, a domain-specific visual editor for PAMs, by benefiting from state-of-the-art MDE techniques. To build PAM Visual Editor, we have used EuGENia, which can automatically generate a visual editor from an annotated ECore metamodel. We show an example PAM created using this visual editor in Section 5.

4.3 Allocation Models

In a heterogeneous system, the freedom of assigning actors $a \in A$ to processors $\pi \in \Pi$ is constrained by which processors can be utilized to execute a particular actor. Thus, in order to run an SDF model on a PAM, we need to know (1) which SDF actors can be run on which processors of the PAM and (2) what their execution times are at given frequencies. This information is encoded in an allocation model, which relates both the SDF and PAM models. Allocation models conform to *Allocation Metamodel* that we define to represent this concern.

The information related to allocation concern was part of Definition 3, but we have chosen to put this into a separate *Allocation Metamodel* for the sake of modularity, making the PAM metamodel fully independent of the SDF metamodel.

Definition and Metamodel The information to be represented in the Allocation metamodel consists of the ζ and τ_{act} functions of Definition 3. The *Allocation Metamodel* is shown in Figure 8. It contains:

– *AllocationRoot*, which stands for the combined allocation functions ζ and τ_{act} of Definition 3.

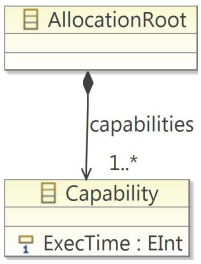

Fig. 8: Allocation Metamodel. *Capability* refers to *Actor* in the SDF metamodel and *ProcessorState* in the PAM metamodel.

– *Capability*, following $\tau_{act} : A \times F \to \mathbb{N}_{\geq 1}$ in Definition 3, refers to *Actor* in the SDF metamodel, and *ProcessorState* (defining the frequency of the processor) in the PAM metamodel, and yields the time needed to execute the actor at the processor state. At the same time, *ProcessorState* also encodes which processor type an actor can be executed on.

The metamodel is in fact more expressive than the mathematical definition: for instance, the execution time of an actor is not constrained to be always the same for a given frequency level; instead, it may also depend on the processor type.

Model Creation The creation of Allocation models corresponds to step 4 in Figure 3. It is supported out-of-the-box via the default tree-based model editor provided by EMF.

4.4 Common Metamodel

In addition to those discussed above, Figure 3 also shows an element called *Common Metamodel*. This demonstrates an MDE technique for reuse: our common metamodel defines the general concept of *Identifiable*, which has a string-valued *identifier* attribute; *Actor*s and *ProcessorType*s are subtypes of *Identifiable* and thereby inherit this feature. Whenever (during extension of the framework) additional reusable concepts are introduced, these can be added to the common metamodel.

4.5 Priced Timed-Automata Models

As shown earlier in Figure 1, we use priced timed-automata for energy optimization. Once the SDF, PAM and allocation models are available, one can generate the priced timed-automata model using the *Co-Design-to-*UPPAAL *Transformation* and successive model-to-text transformation. These correspond to steps 5 and 6 in Figure 3. In Step 7 in Figure 3, the energy-optimal schedule of an SDF graph can be generated. The ideas in [2] related to how the entities in SDF, PAM and allocation models can be mapped to priced timed-automata models and how the energy-optimal schedule can be calculated are reused in steps 5 and 7.

5 Case Study and Evaluation

In this section, we show the effectiveness of our framework for HW-SW co-design by applying it on a case study. We also demonstrate how our framework satisfies the features stated in Section 1, namely: (1) modularity, (2) extensibility, and (3) interoperability. We also evaluate the timing performance of our framework with the help of some other case studies.

5.1 Case Study

As a case study, we consider the dataflow application of the Viola-Jones face detector in Example 2 mapped to a platform with 4 processors of the type Exynos 4210 in Example 3.

Following step 1 in Figure 3, we have created the SDF graph of the Viola-Jones face detection using SDF^3. This SDF graph was already given in Figure 5. In step 2, we apply SDF^3-to-SDF transformation to generate the SDF model conforming to our metamodel for the SDF graph.

In step 3, we create the PAM using the visual editor, as described in Section 4.2. The created PAM is given in Figure 9. The top part of the figure shows 4 processors partitioned into 2 VFIs. The big rectangle at the bottom part of the figure shows the processor type, which is Samsung Exynos 4210 for our case. The oval shapes inside the processor type represent the processor states. For example, at the frequency level (MHz) $f_1 = 1400$, a processor $\pi \in \Pi$ consumes $P_{idle}(\pi, f_1) = 0.4$W if it is idle, or $P_{occ}(\pi, f_1) = 4.6$W if it is working. The arrows between processor states represent transitions between states, and the power overhead of each transition are assigned to the respective arrows. For example, as $P_{tr}(\pi, f_1, f_2) = 0.2$W, the arrow pointing from State 2 (representing the idle processor at f_1) to State 4 (representing the idle processor at f_2) has value 2 assigned to it.

After we have the PAM and SDF models, we create the allocation model that assigns the actors to the processor states with their execution times in step 4.

Once we have the SDF, PAM, and allocation models, we apply the Co-Design-to-UPPAAL transformation in step 5 and the model-to-text transformation in step 6 to generate the priced-timed-automata model that is compatible with UPPAAL CORA. In step 7, we follow the approach presented in [2] to compute the energy-optimal schedule for some given throughput requirements. Figure 10 shows the energy-optimal schedule, for the time per graph iteration constraint of 650 ms for our example. The schedule shows the execution order of the actors at the specific frequency and processors.

5.2 Evaluation

a) Modularity: Modularity helps to satisfy the desirable separation of concerns criterion in development. When concerns are well-separated, the individual modules can be reused and maintained independently. For example, the modularity of our framework allows system designers to model hardware platforms and

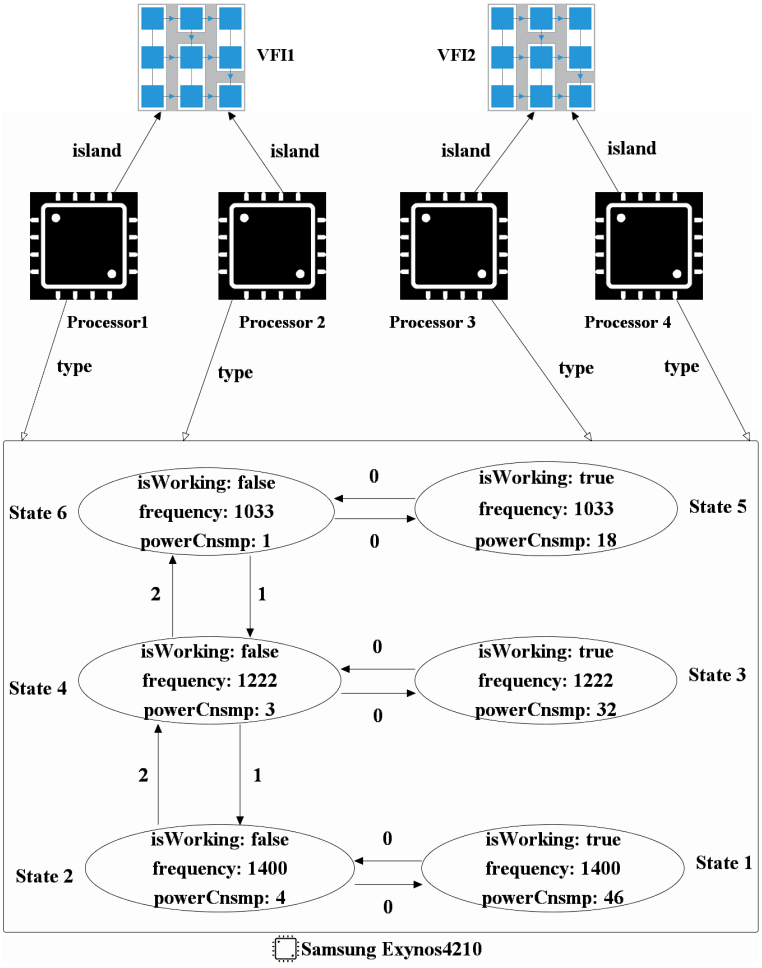

Fig. 9: PAM created using PAM Visual Editor

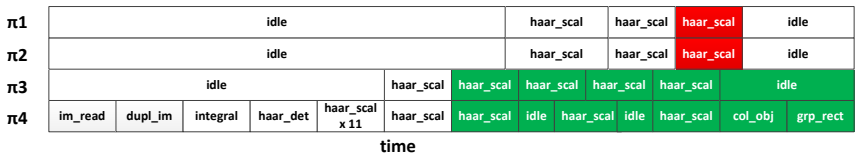

Fig. 10: Energy-optimal schedule on four Exynos 4210 processors $\pi_1, \pi_2, \pi_3, \pi_4$. The white, red, and green blocks denote the frequencies f_1, f_2, and f_3 respectively.

software components separately. In this way, if any change is required either in

Fig. 11: Energy-optimal schedule on four Intel Core2 Duo E6850 processors $\pi_1, \pi_2, \pi_3, \pi_4$

software or hardware models, only the related part needs to be updated independently from the other.

To show this for our framework with a concrete example, let us consider the following scenario: We want to analyze the energy consumption of the face detection application on a different hardware platform, viz., Intel Core2 Duo E6850. Table 2 shows the frequencies and corresponding power consumption of this new processor [19]. For this scenario, we only change the processor type while keeping the number of processors and VFI distributions the same. Now, all we have to do is to develop a new PAM corresponding to this new platform specification and generate the corresponding priced timed-automata model. We reuse the existing SDF model of the application without making any modifications. Using the framework, we derive the energy-optimal schedule shown in Figure 11 on the new hardware platform, for time per graph iteration constraint of 650 ms.

No.	Freq.	P_{idle}	P_{occ}
1	3006	0.4	55
2	2338	0.3	34
3	1776	0.1	22

Table 2: Platform description

b) Extensibility: Due to the adaptation of MDE techniques, our framework provides systematic extensions for future requirements. One can extend the framework using the following mechanisms: introducing new models with related metamodels and new transformations; extending existing transformations or metamodels.

As an example, suppose we want to extend our hardware platform models with the concept of "battery". The current version of the hardware platform assumes energy source to be ideal such that the system never runs out of energy. However, we want to include batteries in our HW-SW co-design as resources. This extension can be achieved through the following steps:

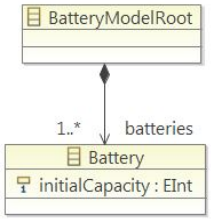

Fig. 12: Battery Metamodel

– *Adding a metamodel for batteries*: An example battery metamodel is shown in Figure 12. This metamodel defines the number of batteries in the system with their initial capacities (Coulomb).

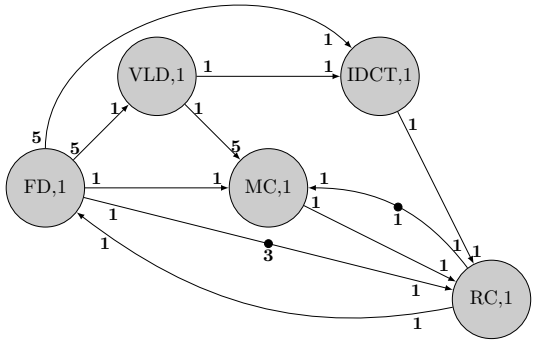

Fig. 13: MPEG-4 Decoder [24]

– *Extending the Co-Design-to-*UPPAAL *model transformation*: In order to include the battery model in the scheduling analysis, we have to transform the concepts in the battery metamodel to the concepts in the priced timed-automata domain. This is achieved through extending the Co-Design-to-UPPAAL model transformation in step 5. The extension to the transformation will create the dependency of the processors on the batteries, in such a way that the processors consume charge from these batteries. The extension will further generate separate templates for battery and battery scheduler. The template of the battery keeps track of the current charge. When the battery gets empty, it informs the battery scheduler via synchronization. In that case, the battery scheduler template activates the next available battery. When all batteries are out-of-charge, the processors cannot run anymore. We can extend the Co-Design-to-UPPAAL model transformation without modifying it since ETL allows to extend and reuse transformation modules. Please note that the model-to-text transformation in step 6 stay unaffected by this extension.

c) Interoperability: As explained in Section 1, HW-SW co-designing involves tools having different domains and focus areas. There must be interoperability between these tools to work together efficiently. In our framework, we utilize SDF^3 for creating SDF graphs and UPPAAL CORA for deriving energy-optimal schedule. To automatically generate UPPAAL CORA models from SDF^3 models, we have implemented model transformations in our framework, thus providing interoperability.

5.3 Timing Performance

To determine the timing performance of our framework, we consider five real-life case studies namely, a Viola-Jones face detector in Figure 5, a MPEG-4 Decoder [24], an Audio Echo Canceller [11], an MP3 Decoder [22], and an MP3 playback application [27]. We also used an artificial bipartite SDF graph [8] with 4 actors. We assume that these case studies are mapped on Exynos 4210 processors having two frequencies.

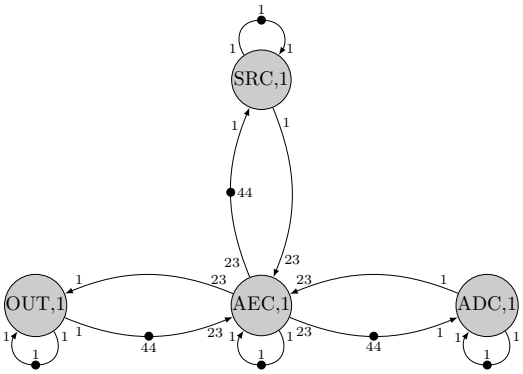

Fig. 14: Audio Echo Canceller [11]

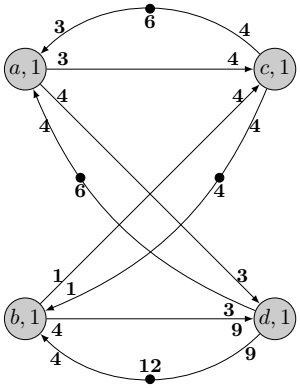

Fig. 15: Bipartite Graph [8]

We examine the timing performance of our framework in two parts: the first part is the timing performance of our framework, i.e., cumulative computation time of steps 2 (SDF^3-to-SDF transformation), 5 and 6 (Co-Design-to-UPPAAL and model-to-text transformations). The second part is the timing performance of obtaining the optimal schedule via UPPAAL CORA model checker, i.e., step 7. The experiments were conducted on a dual-core 2.8 GHz machine with 8 GB RAM. Table 3 shows the results of the experiments. The first column shows the number of processors available for the experiment. The second column shows the timing performance (s) of the first part. The columns 3-5 show the timing performance of the second part. For time per graph iteration constraint (ms) in column 3, the optimal energy consumption (mWs) derived by UPPAAL CORA is shown in column 4, and column 5 shows the time (s) taken to compute the results in column 4.

As one can realize, the time that step 2, 5 and 6 take in total does increase insignificantly as the number of available processor increases. This is due to the slight increase in the model size with the addition of processor instances. For step 7, the time required to complete increases exponentially as the number of

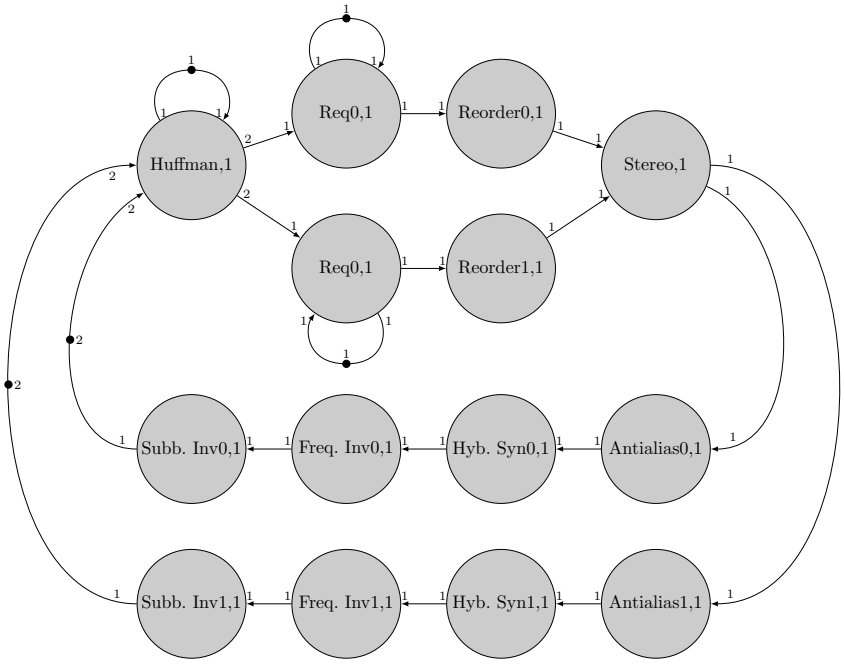

Fig. 16: MP3 Decoder [22]

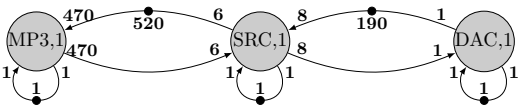

Fig. 17: MP3 Playback Application [27]

processor increases, which is because of the fact that the size of the state-space created by the model checker increases exponentially with the size of the model itself.

6 Conclusions and Future Work

In this paper, we have presented a model-driven framework for HW-SW co-design of dataflow applications. In our framework, we have proposed a reusable set of three coherent metamodels for HW-SW co-design domain. To provide interoperability among domains, we have defined a reusable set of extensible model transformations. We have demonstrated that our framework satisfies the modularity, extensibility, and interoperability requirements with a industrial case study.

As future direction of our work, we plan to extend our framework with other analysis techniques such as simulation and automated HW-SW partitioning. We also plan to add code generation functionality to our framework.

Processor	Steps 2+5+6	Time per Iteration	Energy Consumption	Step 7
Viola-Jones Face Detector in Figure 5				
4	0.171	480	2289.4	8076·98
3	0.071	492	2251.4	269·9
2	0.055	522	2505.6	6·15
1	0.048	756	2419.2	0·51
MPEG-4 Decoder in Figure 13				
5	0.253	42	345.3	18826·46
4	0.253	42	345.3	47·03
3	0.145	44	338.5	3·33
2	0.142	51	333.1	0·41
1	0.14	73	335.8	0·21
Audio Echo Canceller in Figure 14				
4	0.119	23	324.2	13·78
3	0.108	24	322.3	0·71
2	0.106	35	322	0·71
1	0.108	73	335.8	0·41
Bipartite Graph in Figure 15				
4	0.123	42	345.3	366·04
3	0.11	44	338.5	145·12
2	0.102	51	333.1	4·95
1	0.102	73	335.8	0·71
MP3 Decoder in Figure 16				
4	0.361	9	71.1	45·48
3	0.337	9	70.2	11·3
2	0.281	9	64.6	1·41
1	0.256	15	64.4	0·21
MP3 Playback Application in Figure 17				
2	0.101	1880	9907	28800·05
1	0.081	2118	9742.8	71·15

Table 3: Timing performance of our framework

Acknowledgements

This research is funded by the EU FP7 project SENSATION (318490) and NWO project BEATS (612.001.303). The authors are grateful to Kim Sunesen from Recore Systems B.V. for providing the case study.

References

1. Software Enginering Group, University of Paderborn. `https://www.hni.uni-paderborn.de/en/software-engineering/`. Accessed: 2016-01-14.
2. W. Ahmad, P.K.F. Hölzenspies, M.I.A. Stoelinga, and J. van de Pol. Green computing: Power optimisation of VFI-based real-time multiprocessor dataflow applications. In *DSD'15*, pages 271–275, Aug 2015.
3. R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.
4. T. Basten, R. Hamberg, F. Reckers, and J. Verriet. *Model-Based Design of Adaptive Embedded Systems*. Springer Publishing Company, 2013.
5. G. Behrmann, K. G. Larsen, and J. I. Rasmussen. Optimal scheduling using priced timed automata. *SIGMETRICS Perform. Eval. Rev.*, 32(4):34–40, Mar. 2005.
6. L. Bondé, C. Dumoulin, and J.-L. Dekeyser. *Advances in Design and Specification Languages for SoCs*, chapter Metamodels and MDA Transformations for Embedded Systems, pages 89–105. 2005.
7. A. R. da Silva. Model-driven engineering: A survey supported by the unified conceptual model. *Computer Languages, Systems & Structures*, 43:139–155, 2015.
8. M. Geilen, T. Basten, and E. Stuijk. Minimising buffer requirements of synchronous dataflow graphs with model checking. In *DAC '05*, pages 819–824, 2005.
9. I. Gray, N. Matragkas, N. Audsley, L. Indrusiak, D. Kolovos, and R. Paige. Model-based hardware generation and programming - the MADES approach. In *ISORCW'11*, pages 88–96, March 2011.
10. K. Grüttner, P. A. Hartmann, K. Hylla, S. Rosinger, W. Nebel, F. Herrera, E. Villar, C. Brandolese, W. Fornaciari, G. Palermo, C. Ykman-Couvreur, D. Quaglia, F. Ferrero, and R. Valencia. The COMPLEX reference framework for HW/SW co-design and power management supporting platform-based design-space exploration. *Microprocessors and Microsystems - Embedded Hardware Design*, 37:966–980, 2013.
11. J. P. Hausmans, S. J. Geuns, M. H. Wiggers, and M. J. Bekooij. Compositional temporal analysis model for incremental hard real-time system design. In *EMSOFT '12*, pages 185–194, 2012.
12. P. Herber and S. Glesner. A HW/SW co-verification framework for SystemC. *ACM TECS*, 12(1s):61:1–61:23, Mar. 2013.
13. D. S. Kolovos, R. F. Paige, and F. A. Polack. The Epsilon transformation language. In *ICMT '08*, pages 46–60, 2008.
14. D. S. Kolovos, L. M. Rose, R. F. Paige, and F. A. C. Polack. Raising the level of abstraction in the development of GMF-based graphical model editors. In *MISE '09*, pages 13–19, 2009.
15. E. A. Lee. Embedded software. *Advances in Computers*, 56:56–97.
16. E. A. Lee and D. G. Messerschmitt. Synchronous data flow: Describing signal processing algorithm for parallel computation. In *COMPCON'87*, pages 310–315, 1987.
17. Object Management Group. OMG Object Constraint Language (OCL), Version 2.3.1, 2012.
18. Object Management Group. Meta Object Facility (MOF) Core Specification, version 2.5, 2015.
19. S. Park, J. Park, D. Shin, Y. Wang, Q. Xie, M. Pedram, and N. Chang. Accurate modeling of the delay and energy overhead of dynamic voltage and frequency scaling in modern microprocessors. *TCAD*, May 2013.

20. J. Sprinkle, B. Rumpe, H. Vangheluwe, and G. Karsai. Metamodelling. In *Model-Based Engineering of Embedded Real-Time Systems*, pages 57–76. Springer, 2010.
21. D. Steinberg, F. Budinsky, E. Merks, and M. Paternostro. *EMF: Eeclipse Modeling Framework*. Pearson Education, 2008.
22. S. Stuijk. *Predictable Mapping of Streaming Applications on Multiprocessors*. PhD thesis, 2007.
23. S. Stuijk, M. Geilen, and T. Basten. SDF^3: SDF For Free. In *ACSD'06*, pages 276–278, June 2006.
24. B. Theelen, M. C. W. Geilen, T. Basten, J. P. M. Voeten, S. V. Gheorghita, and S. Stuijk. A scenario-aware data flow model for combined long-run average and worst-case performance analysis. In *MEMOCODE'06*, pages 185–194, 2006.
25. P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *CVPR'01*, pages I–511–I–518 vol.1, 2001.
26. M. Völter, T. Stahl, J. Bettin, A. Haase, and S. Helsen. *Model-driven software development: technology, engineering, management*. 2013.
27. M. H. Wiggers. *Aperiodic multiprocessor scheduling for real-time stream processing applications*. PhD thesis, 2009.