From UML/MARTE Specifications to ESL HW/SW Co-Design: Early Functional Verification and Timing Validation

Vittorio Cortellessa
University of L'Aquila
Italy
vittorio.cortellessa@univaq.it

Luigi Pomante
University of L'Aquila
Italy
luigi.pomante@univaq.it

Vincenzo Stoico
University of L'Aquila
L'Aquila, Italy
vincenzo.stoico@graduate.univaq.it

Abstract

The continuous adoption of embedded systems in the most diverse application domains contributes to the increasing complexity of their development. Hardware/Software Co-Design methodologies are usually employed to tackle the challenges deriving from even more stringent functional and non-functional requirements. Using these methodologies, several validation and verification steps can be carried out early in the design process using a unified, technology-independent system model.

This work investigates the possibility of integrating formal functional verification and timing validation in a Hardware/Software Co-Design flow at the system-level of abstraction. Specifically, we introduce Co-V&V, namely an additional step that consists of two phases: (i) a transformation from UML/MARTE to UPPAAL Timed Automata, and (ii) a preliminary functional verification and timing validation that exploits the UPPAAL verifier.

We describe the Co-V&V step through a case study characterized by a component-based architecture and reactive behavior. The verification and validation conducted with UPPAAL indicate that our approach is particularly effective in discovering design flaws located in the communication protocol as well as those arising from the internal behavior of components.

CCS Concepts

• **Software and its engineering → Unified Modeling Language (UML)**; *Formal software verification*; • **Computer systems organization → Embedded systems**.

Keywords

UML, HW/SW Co-Design, UPPAAL, Model Checking, Formal Verification, Timing Validation

ACM Reference Format:
Vittorio Cortellessa, Luigi Pomante, and Vincenzo Stoico. 2023. From UML/MARTE Specifications to ESL HW/SW Co-Design: Early Functional Verification and Timing Validation. In *Companion of the 2023 ACM/SPEC International Conference on Performance Engineering (ICPE '23 Companion), April 15–19, 2023, Coimbra, Portugal.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3578245.3584850

ICPE '23 Companion, April 15–19, 2023, Coimbra, Portugal

ACM ISBN 979-8-4007-0072-9/23/04…$15.00
https://doi.org/10.1145/3578245.3584850

1 Introduction

The rising adoption of embedded systems led designers to deal with more stringent functional and non-functional requirements. This trend has made inefficient the classical design flow in which the hardware and software were developed independently until the final integration step. Hardware/Software Co-Design has been introduced to overcome the shortcomings of such a classical design flow. Its key feature consists in the adoption of a behavioral and implementation-agnostic model to represent the target system. Based on this model, the designer can perform preliminary analyses before the Design Space Exploration (DSE) step and, consequently, reduce the defect density in later stages. Therefore, these activities require modeling languages, sufficiently expressive, to represent system functional and non-functional requirements.

The survey conducted by Liebel et al. [15] shows that UML is among the most used modeling languages in the embedded systems industry, where UML is often enriched with MARTE (Modeling and Analysis of the Real-Time Embedded systems) annotations [18]. MARTE is a standard profile to represent quantitative non-functional properties (e.g., time, performance). UML specification, in some of its sections, delegates to the designer the interpretation of its semantics (i.e., variation points). This aspect enables the designer to explore potential design alternatives, but at the same time it hampers a rigorous analysis of system properties [9]. The introduction of Model-To-Model (M2M) transformations from UML/MARTE to formal languages is a common practice to include formal verification into the design flow [16]. As a consequence, M2M transformations that apply to MARTE-annotated UML models have the potential to generate not only functional models but also non-functional ones.

There is significant evidence in literature about the need to introduce formal analyses in the ESL (Electronic System-Level) design flow [14, 15, 21, 23, 26]. Moreover, the state of art already provides examples of HW/SW Co-Design flows starting from UML/MARTE models that integrate formal verification and validation (V&V) [2, 13, 25]. However, V&V is usually performed after some refinements of the initial model. We intend to anticipate V&V by raising its level of abstraction, thus we add a step dedicated to functional verification and timing validation, namely Co-V&V, early in the design flow. In this paper, Co-V&V is based on a M2M transformation from UML/MARTE to Timed Automata, and the usage of UPPAAL model checker [1] to analyse the generated automata.

Figure 1 shows the V&V-based HW/SW Co-Design flow that we envision in this paper. The flow starts with the Co-Specification step focusing on the construction of an initial model. We envisage a model that results from the composition of three views: Application, Communication, and Time. The Application View embodies the

logical structure and the behavior aspects of the application. In this view, system entities are described at a high abstraction level and cannot be distinguished according to their implementation (i.e., hardware or software). The Communication View includes the communication protocols used for system entities interactions, while the Time View encloses time requirements such as duration constraints.

The Co-Specification is followed by two steps targeting different system properties: Co-V&V and Co-Analysis. Co-V&V purpose is to check control flow correctness and the satisfiability of timing constraints. Co-Analysis focuses on the analysis of system performance exploiting performance models (e.g., Layered Queuing Networks). Co-Analysis provides a first performance estimation, which will be thereafter refined during the DSE phase. In the proposed HW/SW Co-Design flow, the DSE suggests the physical structure of the system (e.g., the processors, the memory architecture, and the physical links connecting them) and the mapping of the model elements onto it. After such mapping, the performance of the new model is estimated through a further Co-Analysis step. The DSE outputs a system model enclosing hardware/software structure and behavior.

In this paper, we investigate the problem of model checking UML/MARTE models to assess control-flow correctness and time constraints satisfiability. These problems are faced during Co-V&V, which is based (as mentioned before) on UPPAAL Timed Automata. We illustrate in some details the Co-Specification and Co-V&V steps through the FIRGCD (Finite Impulse Response Greatest Common Divisor) case study [20]. The major contributions provided by this work are:

- we introduce a novel organization for embedded systems models composed by three views: Application, Communication, Time. We refine the definition of Communication View proposed by Enrici et al. [10] and contribute by adding a Time View;
- we exploit a subset of UML/MARTE for modeling reactive components, message-based communication, and time constraints on the duration of interactions;
- we define a semantic mapping from UML/MARTE to a network of timed automata. The transformation exploits both system architectural and behavioral description as well as the organization in views of the model;
- we provide strategies to perform functional verification and timing validation of embedded systems using UPPAAL.

The paper is organized into the following sections: Section 2 describes UML/MARTE and the main features of UPPAAL, Section 3 illustrates the Co-Specification step, Section 4 deepens the semantic mapping between UML/MARTE and timed automata, Section 5 reports the properties verified using UPPAAL and the results of the verification on the considered case study, Section 6 discusses related work while Section 7 concludes the paper.

2 Background

2.1 UML/MARTE

The Unified Modeling Language (UML) [19] is a general-purpose modeling language, among the most used in the embedded systems domain [15]. Due to its vast semantics, UML enables the representation of structural and behavioral aspects at different levels

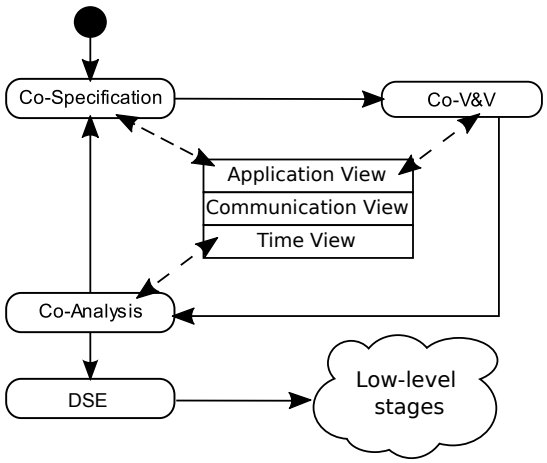


Figure 1: V&V-Based Hardware/Software Co-Design Flow

of abstraction. For example, the UML Component Diagram is often employed to model component-based architectures, while the UML Activity Diagram is used to represent fine-grained behaviors. However, UML, as a general-purpose language, fails to represent domain-specific situations. For example, when modeling real-time behaviors where a more precise representation of timing properties is needed. For this purpose, UML can be extended using the Modeling and Analysis of the Real-Time Embedded systems (MARTE) [18] profile. Indeed, MARTE provides classes such as `TimedProcessing` to assign a duration to system entities. Moreover, MARTE includes the Value Specification Language (VSL) to specify time constants and expressions. For example, the tuple `(100,ms)` denotes 100 milliseconds. This tuple can be used to define a duration constraint such as `(endEvent - startEvent) <= (100, ms)`. This constraint binds the interval between two observed events to last at most 100 milliseconds.

2.2 UPPAAL

The UPPAAL model checker [1] aims at the verification of real-time systems represented via a network of timed automata. Automata execute in parallel and they can synchronize using channels. A rendezvous is modeled by labeling two edges of two different automata with `c?` and `c!`. Moreover, UPPAAL automata are specified as parametric templates that may be extended with discrete and real-valued variables named clocks. A timed automaton is made by a set of locations and a set of edges. A location may have an invariant that denotes the property held while staying at that location. An initial location is unique and describes the starting condition of the automaton, while an urgent one defines a location with no delay. Edges may have guards to allow transitions, channel synchronizations, and variable assignment.

System properties are expressed using Computational Tree Logic (CTL) [7]. Examples of properties are deadlock freedom, program termination, and location reachability. A CTL formula may be defined over the states or the paths of the timed automata network. Properties over paths are written using two CTL quantifiers: All (A) and Exists (E). A ϕ says that ϕ should hold for all paths, while E ϕ describes a property valid for one or more paths. In UPPAAL, a path quantifier should be followed by a linear time operator between

Finally (F) ϕ (i.e., $\Diamond$) and Globally (G) ϕ (i.e., $\Box$). They express statements specific to a path. Indeed, F ϕ is used when ϕ can eventually hold in the path, while, using G, ϕ should be satisfied in all the subsequent states.

Timed automata are insufficient for modeling non-deterministic decisions [7]. For this reason, UPPAAL has been extended to support Stochastic Model Checking (SMC) [7]. Probabilistic branches are defined by labeling outgoing edges of branching points with probabilities. This enhancement allows for more fine-grained analyses. It is possible to write properties over a fixed observation time and number of executions. For example, $Pr[<= 100; 200](\Diamond\phi)$ returns the probability to satisfy ϕ in 100 time units, and the expression is evaluated by performing 200 runs. The precision of the result is proportional to the number of repeated runs. Lastly, the SMC extension provides an operator to calculate the expected value of an integer clock variable with an interval of time. Such an expression is written as $E[<= 100; 200](max : \phi)$ that calculates the average of ϕ by considering, at each run, the maximum value that ϕ can assume.

3 Co-Specification

As explained in the Introduction, the proposed HW/SW Co-Design flow involves an initial modeling step called Co-Specification. Section 3.1 details the constructs of UML/MARTE used, in each view, to describe system structural, behavioral, and time concerns. Instead, Section 3.2 presents a toy case study: FIRGCD.

3.1 UML/MARTE Subset

The three views of the system embed, respectively, the application, the communication protocols, and the timing constraints. Aspects as structure and behavior are cross-cutting to the views. For example, system behavior is partly represented in the Application View, for what concerns a system element internal behavior, and partly represented in the Communication View, for what concerns the interactions among system elements. This section describes the elements of UML/MARTE selected to model FIRGCD structure, behavior, and time.

3.1.1 Structure Specification The description of the logical structure of the system involves system entities and their interconnections. These static aspects are shown using a UML Component Diagram, which consists of a set of UML Components connected through UML Connectors. Components exchange data over connectors through one or more input/output UML Ports. Each component may contain sub-components in a hierarchical fashion. Data are forwarded to sub-components by binding the input ports of a component to those of its sub-components. Data forwarding is modeled annotating a UML association with «`delegate`». Figure 2 reports the structure specification of FIRGCD, which is described in details in Section 3.2.

3.1.2 Behavior Specification A designer can leverage the extensive semantics of UML to define fine-grained behaviors. Such a detailed behavioral specification reproduces the flow of operations executed by system components. In an activity diagram, the UML `CallOperationAction` element represents the action of calling of an operation. Moreover, the execution flow may involve branching points and possible exceptions. Branching points are denoted by a UML `DecisionNode`, while exceptions are indicated using the UML `RaiseExceptionAction` element.

Since embedded systems are typically reactive [24], it is necessary to model behaviors dealing with asynchronous events. The action of waiting for an event has been modeled using the UML `AcceptEventAction` element. This element has an attribute (i.e., Trigger) defining the kind of expected event. However, UML lacks semantics for representing the reception of specific types of messages. We employ the MARTE «`DataEvent`» stereotype to fill this lack. The «`DataEvent`» stereotype extends the UML `AnyReceiveEvent` element by adding a tag denoting the message type. After processing, data are written on ports using the UML `SendObjectAction` element. Instead, signals reception and sending are modeled, respectively, through UML `ReceiveSignalEvent` and `SendSignalEvent`.

3.1.3 Time Specification In the time model of UML, time instants correspond to event occurrences. UML provides the concept of Observation to represent an instant (i.e., UML `TimeObservation`) during an execution. Observations can be combined with value specifications to define temporal constraints on system behavior. For this purpose, it is useful to exploit the syntax of the Value Specification Language (VSL) [18], embedded in MARTE, to define time values. For example, the VSL expression $endEvent - startEvent$ represents the duration between the occurrence of two events. In our model, $endEvent$ and $startEvent$ are two `TimeObservation` associated to the occurrence of two events. Therefore, we define constraints on execution chunks written as $event2 - event1 \backsim x$ where $event1$, $event2$ are instances of `TimeObservation`, $\backsim \in \{>=, <=, >, <, =\}$, and x is a real number or a VSL time value.

3.2 Modeling the case study: FIRGCD

The features of FIRGCD [20] made it suitable for V&V of reactive and distributed computation. FIRGCD [20] is made by a network of processes communicating via point-to-point channels. The processes react upon the reception of data or signals.

3.2.1 Application View Figure 2 shows the structure of FIRGCD. The structure comprises a cluster of components dealing with the generation of inputs for the Application. This cluster includes two Stimulus (i.e., `stim0` and `stim1`) and a Timer. The stimuli generate a random integer every time they receive signals from the timer. The generated integers are passed to the FIR filters: `fir8` and `fir16`. The filters execute whenever an integer shows up at their input port. The GCD component waits until data is ready on both input ports before starting the processing. GCD receives the data from the filters and sends the result to a Display instance. The latter component shows the result of the Application. The behavioral specification of FIRGCD embodies the interactions among system entities (i.e., system-level behavior) plus the description of their internal behavior. The latter is specified by the class to which each element belongs to. For example, the `fir8` and `fir16` components, in Figure 2, have the same behavior since they are instances of the same class (i.e., the FIR class). In system-level behavior, they can be parameterized differently and associated with different time constraints.

3.2.2 Communication View The Communication View describes the exchange of information among components and was first introduced by Enrici et al. [10]. The main reason behind the adoption

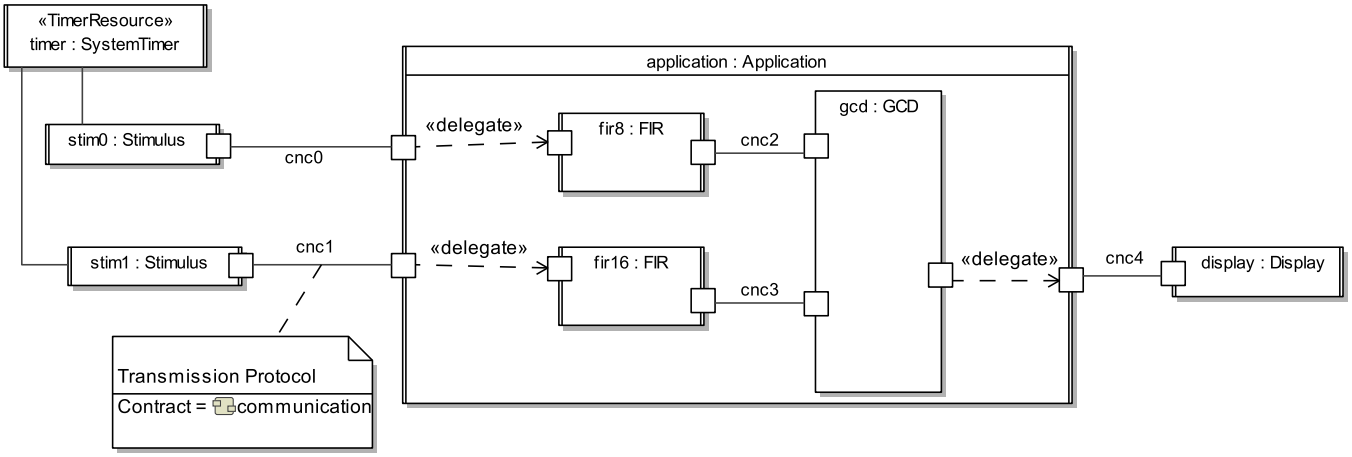


Figure 2: FIRGCD Component Diagram

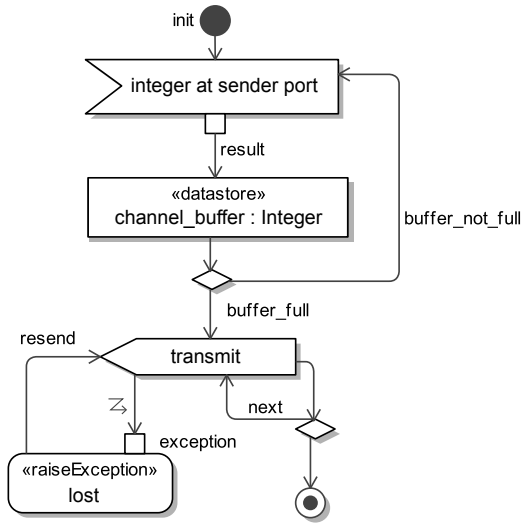


Figure 3: Communication Protocol adopted by the connectors of FIRGCD

of the Communication View is to overcome communication mismatch in the later stages of the design. As highlighted by Enrici et al., communication logic is usually embedded in the application and platform description but it is realized using different models of computation. Therefore, the designer may encounter inconsistency issues during the mapping stage of a HW/SW Co-Design flow. For the sake of simplicity, we consider all the connectors of FIRGCD, i.e. the solid lines in Figure 2, implementing the same protocol. Figure 3 shows the communication protocol of FIRGCD. The protocol implemented in our case study consists of a point-to-point communication. The exchange is unidirectional, so it involves a single sender and receiver. The data is kept until the buffer is filled. Once full, buffer elements are transmitted one at a time to the receiver. Data received while the buffer is full are lost. Moreover, data could be lost during transmission. In this case, the protocol provides for the re-transmission of the data. The flow ends whenever everything is successfully transmitted.

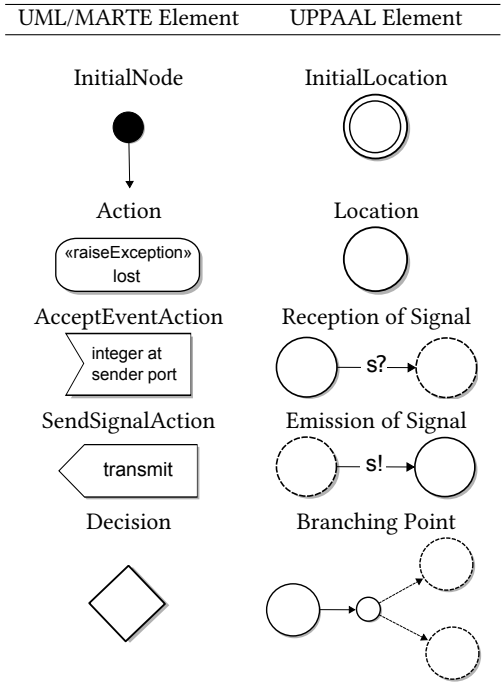
3.2.3 Time View In FIRGCD, time flows continuously. The generation of integers is regulated by a Timer, which periodically sends a signal to `stim0` and `stim1` after a delay of 100 time units. To model the duration of the delay, we used the MARTE `TimedProcessing` stereotype. The activity diagram of the Timer includes an action named `delay` annotated with `TimedProcessing`. This stereotype has the tag `duration` set to 100. Moreover, MARTE helps in constraining the duration of actions and execution chunks. Such a constraint expresses the duration between the occurrence of two events. In FIRGCD, we impose a limit of 100 time units on the execution chunk that begins when the timer is triggered (i.e., receipt of `reset` on `stim0`), and ends when the output is shown to the user. This constraint, named *global_constraint* is defined using the following VSL expression: $displayed - reset < 100$.

4 UML/MARTE to UPPAAL Timed Automata Transformation

In order to exploit UPPAAL for functional verification and timing validation, we defined a semantic mapping from UML/MARTE to UPPAAL timed automata network. The network is built considering the architecture of the model and the behavioral description of each component. Table 1 reports all the elements involved in the generation of the automata. Each activity diagram represents the internal behavior of a system element and it has a corresponding UPPAAL template. A template can be instantiated in one or more UPPAAL Processes. In our case, each process represents the behavior of a component. The node starting the flow of an activity diagram (i.e., `InitialNode`) is converted into the initial location of the target automaton. The remaining locations are obtained transforming each action found in the activity diagram.

We used the architecture of the model, in Figure 2, to define the topology of the network of timed automata. In UPPAAL, we bound connectors by modeling the data exchange between components. Indeed, in the UML/MARTE model, components can write and read data to/from the connectors. This aspect is modeled in UPPAAL using channels. For each connector, we create two channels: `conn` and `data`. The `conn` channels are used to model rendezvous between components and connectors. We use `conn!` to denote a

Table 1: UML/MARTE to UPPAAL Timed Automata Transformation



component sending data to a connector, while conn? embodies a connector waiting for data to be written. The data channels have the same semantics but the roles are inverted. In the same way, we use UPPAAL synchronizations to model the arrival and the emission of signals. In our model, the signal exchange is modeled using `AcceptEventAction` for the reception and `SendSignalAction` for the emission of signals. Finally, we translated UML decision nodes into UPPAAL branching points. Each outgoing edge of a branching point is annotated with a probability chosen by the designer.

5 Co-V&V

The timed automata network resulting from the transformation is supplied to UPPAAL for verification and validation. We defined a set of properties, expressed in Computation Tree Logic (CTL) [7], to check functional correctness and validate the timing constraints defined the in UML/MARTE model. Table 2 reports the CTL formulae along with the results produced by the UPPAAL verifier. These results are used to improve the UML/MARTE model defined during Co-Specification. It is evident by looking at the description of FIRGCD, in Section 3.2, that the results of the V&V are conditioned by the quality of the communication protocol. Indeed, the latter implements the most sophisticated behavior. Thus, in the case of FIRGCD, the quality of the initial design can be improved by simplifying the structure of FIRGCD or the logic of the communication protocol, respectively, in Figure 2 and Figure 3. Therefore, before calculating the results, it is necessary to fix the parameters that can affect the results of the formulae. In FIRGCD, these parameters are the probability values labeling the edges of the branching points in Figure 3. As described in Section 3.2.2, the protocol starts the transmission when the buffer is full. We test the system considering a small capacity buffer to obtain many transmissions during the observation time frame. The situation involving a full buffer is represented setting a likelihood of 80% between `DataArrived` and `Transmit`. The functional correctness of the system can be verified by defining CTL formulae that ensure the reachability of timed automata locations and deadlock freedom. During the simulation of the timed automata network, reaching a specific location can represent a correct or faulty behavior of the system. Besides, whether this is done within a time frame reveals the satisfiability of timing constraints. Before proceeding to deeper analyses, it is important to know if the system execution does not lead to a deadlock. Deadlock freedom of FIRGCD is guaranteed executing Equation 1 in Table 2 in the UPPAAL verifier.

5.1 Reachability of Display

In FIRGCD, Display reachability can be considered evidence of successful execution. Using the Stochastic Model Checking features of UPPAAL, we can calculate the likelihood of reaching the `displayed` location. Equation 2 and Equation 3 express, respectively, the reachability of the `displayed` location in some and all the execution traces. The satisfiability of these formulae suggests whether the `displayed` location is eventually reached. Therefore, in case the formulae are satisfied, a designer can perform a deeper analysis checking the likelihood to reach `displayed`. Equation 4 calculates the probability that the `displayed` location is reached within the first 300 time units. A low probability value can be a symptom of problems occurring before arriving at Display.

The UPPAAL verifier validates the Equation 2, while the compliance of Equation 3 is not guaranteed. The non-compliance of Equation 3 implies that Display might never be reached. This behavior may stem from the communication protocol, since the protocol models the situation in which there is a loss of data. Thus, in the worst case, data could be infinitely lost and never reach Display. The execution of the Equation 4 outputs that the probability to reach the `displayed` location within 300 time units lies between 82% and 85% measured in 2000 runs.

Figure 6a reveals that the `displayed` location is periodically reached. This phenomenon may originate from the behavior of Timer. Indeed, the Timer starts the execution flow by sending a signal to `stim0` and `stim1` each 50 time unit. In order to prove it, we have shortened the Timer by decreasing the delay to 20 time units. The results show that the probability increased to a range of 89% to 92%. This hypothesis is supported by comparing Figure 6a and Figure 6b. Indeed, Figure 6b shows a chart shifted to the left with more concentrated values than Figure 6a. However, the average number of data lost by all the connectors, calculated using the Equation 8, increases from 1.48 to 1.84. This issue can be fixed by changing the structure of the UML/MARTE model, in Figure 2, by merging the connectors linking the filters with the GCD component. Moreover, the resulting connector will implement a larger buffer to store the results of the filters. A larger buffer can be represented by decreasing the probability that the buffer is full from 80% to 60%. Thus, in Figure 4, we change the probability value on the edge connecting `DataArrived` to `Transmit`. The reachability of the `displayed` location becomes almost certain within 300 time units, while the average number of data lost decreases from 1.84 to 0.91.

Table 2: System properties checked using UPPAAL

Property		Expression	Results
Deadlock Freedom	(1)	$A\Box\, not\ deadlock$	Satisfied
Reachability	(2)	$E\Diamond\ display.Displayed$	Satisfied
	(3)	$A\Diamond\ display.Displayed$	Unsatisfied
	(4)	$Pr[<=300;2000](\Diamond\ display.Displayed)$	[0.821109, 0.853893]
Communication	(5)	$E\Diamond\ forall\ (i : conn\_t)\ Connector(i).Transmitted$	Satisfied
	(6)	$Pr[<=200;500](\Diamond\ Connector(4).Transmitted)$	[0.419613, 0.508815]
	(7)	$E[<=300;500](max : sum(i : conn\_t)Connector(i).Transmitted)$	2.078
	(8)	$E[<=300;500](max : sum(i : conn\_t)Connector(i).Lost)$	1.48
Time	(9)	$E[<=100;2000](max : Stopwatch.x)$	53.991

5.2 Communication Correctness

The benefits of introducing a Communication View become evident in the Co-V&V step. Throughout the transformation, each communication protocol is translated into an independent UPPAAL template. In this way, every protocol results in a different timed automaton that can be analyzed separately. This modeling decision reduces V&V complexity since the designer may check communication properties independently from the network behavior. The architecture of FIRGCD, in Figure 2, presents five connectors implementing a common communication protocol. Therefore, the resulting network contains a unique UPPAAL template, embodying the protocol behavior, and a UPPAAL process for each connector. The functional correctness of the protocol implies that data is eventually transmitted. This test can be done by checking the location representing the system state after the transmission of an item. However, due to the shape of the Activity Diagram, the obtained protocol automaton does not include such a location. Therefore, we add a location to the protocol automaton called `Transmitted`. Figure 4 depicts the protocol automaton including the `Transmitted` location. This situation can happen whenever a designer represents a behavior at a high level of abstraction and desires to perform a more fine-grained verification during Co-V&V. In our case, the `Transmit` location denotes the system state when ready to send data. However, it does not inform about what happened before (i.e., if data has just been sent or not). Equation 5 guarantees that data is eventually transmitted by all the connectors. Instead, Equation 6 returns the probability of transmitting data from the Connector(4), namely `cnc4` in Figure 2, within the first 200 time units. Consequently, the same property can be checked when data is lost. Expressions involving time values, like Equation 5, are well suited for checking the satisfiability of timing constraints defined in the Time View.

The UPPAAL verifier outputs a range between 41% and 50% for Equation 6, while a likelihood within 38% and 47% that information is lost. Further insights about the system model may arise by checking the average number of data items transmitted and lost. Equation 7 outputs 2.078 that denotes the maximum average data transmitted within 300 time units. The same property can be written for the loss of data. In this case, the verifier returns an average of 1.48.

5.3 Timing Validation

The Time View describes the definition of constraints on the time elapsing between the occurrence of two events. One example of

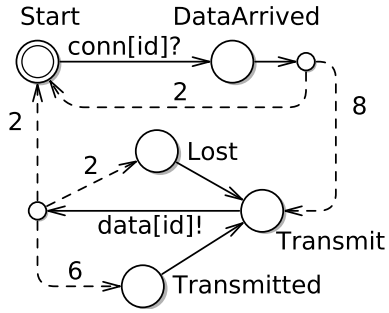


Figure 4: Connector Automata

such a constraint, i.e. the `global_constraint`, is described in Section 3.2.3. This constraint binds time from when `stim0` generates an integer to the moment this integer will be displayed. An event can occur multiple times during an execution. Therefore, to measure a time interval, we need to track the time elapsing between the *ith* occurrences of two events. The introduction of variables or locking mechanisms to control the occurrence of events seemed a solution that would substantially complicate the network of timed automata. For this reason, we introduced an additional timed automaton behaving as a stopwatch. Figure 5 depicts the automaton of the stopwatch. The stopwatch switches to the waiting state upon the arrival of *start*! and exits at the reception of *stop*!. Hence, the duration of intervals can be checked by measuring how much time the stopwatch spends in the waiting state.

We check the satisfiability of *global_constraint* through Equation 9. This formula includes the expression $max : Stopwatch.x$ that denotes the maximum value of x over a run. The variable x is a clock that is reset as the stopwatch enters and exits the waiting state. The UPPAAL verifier evaluates Equation 9 for 2000 runs, each lasting for 300 time units. At each run, the verifier takes the maximum value of x, and then it returns the average of all values chosen in 2000 runs. Equation 9 outputs that the average duration of the interval between `reset` and `displayed` equals to 130.75 time units. So, *global_constraint* is not satisfied since the duration between these two events should not exceed 100 time units. The same thing happens considering the improved structure of FIRGCD as described in Section 5.1. In this case, Equation 9 results in an average of 118.59. It is worth noting that the satisfiability of *global_constraint* is remarkably affected by the delay of the timer as well as the duration of the `shifting` and `evaluation` operations of the filters. Consequently, it is sufficient to reduce these duration

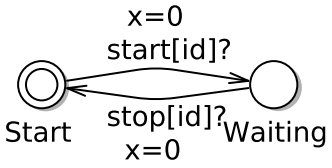


Figure 5: Stopwatch Automaton

values to meet *global_constraint*. This is supported by the UPPAAL verifier that outputs an average of 96.51 just decreasing the duration of the `evaluation` operation from 50 to 25 on `fir8` and from 45 to 25 on `fir16`. Such an analysis suggests, to the designer, possible operation duration values to consider for subsequent design phases.

6 Related Work

UML/MARTE is a widely used language for modeling the characteristics of real-time embedded systems. However, it lacks support for functional verification and validation of time constraints. A popular strategy to achieve verification consists in the creation of a semantic mapping towards UPPAAL timed automata. This approach has already been investigated by several works [3, 4, 8, 11, 12, 22, 27].

Some of the most representative work start from a state-based representation of system behavior for the conversion into timed automata. Surdyevara et al. [22] propose a mapping between UML Statemachine and UPPAAL timed automata. The UML state machines are annotated using MARTE+CCSL to express causal constraints of clock instants. A similar work is Chen et al. [3] which focuses on verification of component-based systems. They propose a novel mapping from MARTE+CCSL to UPPAAL-TIGA timed I/O automata to model components interfaces. These works differ from ours by the level of abstraction of the source model. Their conversion maps a state to a location. In this way, a location of a timed automaton represents the state of the system when it assumes a particular configuration (i.e., values of its variables or set of active objects) and thus when an invariant condition holds. In this paper, a location represents a fundamental unit of executable functionality. Hence, the semantics of actions in activity diagrams. This is necessary to examine specific execution flows, including those internal to the state of a system. Besides, state representation would lead to inaccurate time estimation and thus invalid timing validation. Both [22] and [3] employ CCSL to model causal constraints on event occurrences. They constrain event order or their timing. Our work diverges since the control flow of events is expressed using the constructs of activity diagrams. Instead, we constrain the duration of actions and executions rather than events timing.

Another peculiarity of our work is the adoption of stochastic model checking to analyze the likelihood of reaching a location. Gu et al. [11] has commonalities with our work both for the adoption of stochastic model checking and the usage of activity diagrams. They exploit stochastic modeling to represent user inputs and action duration. Gu et al. use a different strategy to model actions duration. Indeed, they annotate actions with a range denoting their execution time variation. Instead, we express action duration constraints using MARTE/VSL. Both of the approaches use properties written as $Pr[<= T](\psi)$ to check whether constraints are validated. The works differ in how action durations are described in UPPAAL. Gu et al. use a two-dimensional array storing the execution duration variation for each node. We implement the duration constraint as a guard on transition. This choice was made considering that we perform timing validation in the early stages of the design. So, action execution time is estimated and modeled as a delay in UPPAAL. Finally, to the best of our knowledge, it is a novelty to involve both structural and dynamic aspects of a UML/MARTE model to obtain a network of timed automata. Indeed, the component diagram defines the topology of the network, while the activity diagrams induce each automaton composing the network. Moreover, arranging the system model in views allows to reduce verification complexity and analyze system properties independently (e.g., communication).

7 Conclusion

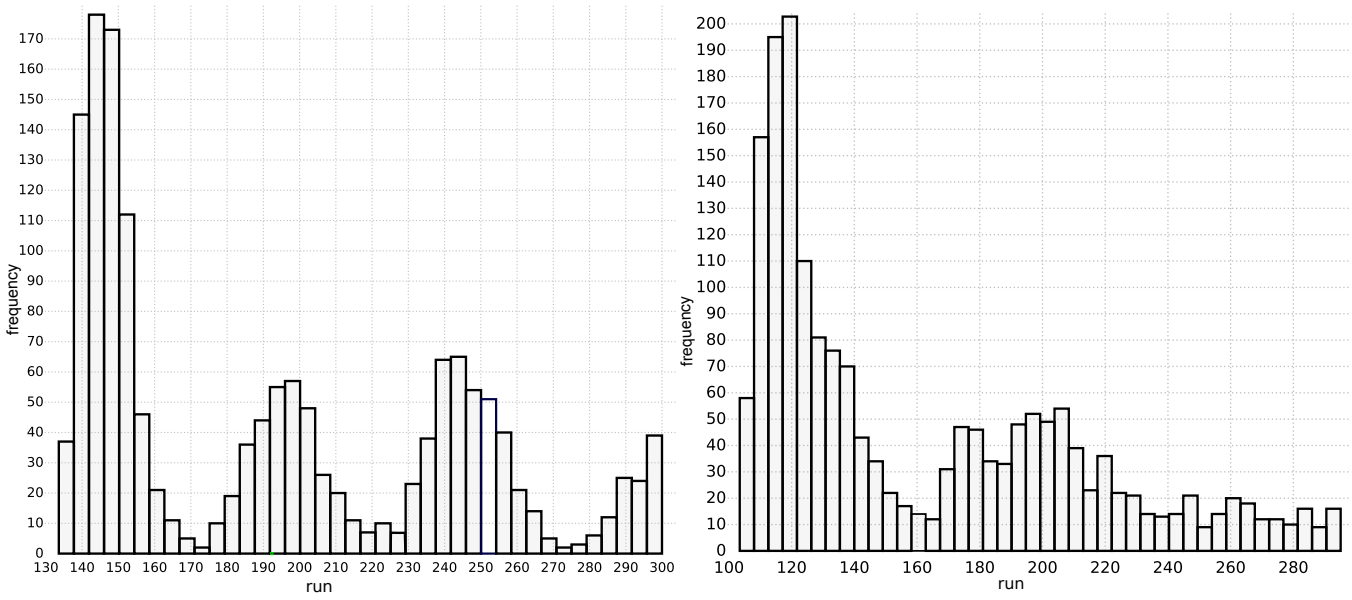
In this work, we presented a novel HW/SW Co-Design flow integrating formal analyses in the early stages of the design. During the first step, i.e. Co-Specification, we exploited UML/MARTE to model the functional and non-functional requirements of the system. The UML/MARTE model is made by three views: Application, Communication, Time. We introduced a new verification and validation step, namely Co-V&V. Co-V&V uses the UPPAAL model checker for functional verification and timing validation of the system. To use UPPAAL, Co-V&V includes an M2M transformation from UML/MARTE to a network of timed automata. We showed Co-Specification and Co-V&V through the FIRGCD case study. FIRGCD is characterized by a component-based architecture where each component runs asynchronously at the reception of data or signals. After the transformation into timed automata, we used UPPAAL to check the reachability of the `displayed` state, the correctness of the communication protocol, and the satisfiability of the timing constraints. The Co-V&V showed how reachability verification can be beneficial to improve the structure of the system. Indeed, we achieved a higher probability value of reaching the `displayed` location eliminating the two connectors linking the FIR filters to the GCD component. We proved that the *global_constraint* is not satisfiable with the initial design of FIRGCD. The validation revealed that the constrained execution chunk is mostly affected by the delay of the timer and the duration of the operations of the two FIR and poorly influenced by the structure of FIRGCD.

In the future, we plan to examine case studies with more stringent non-functional requirements (e.g., real-time constraints [17]) or more complex behavior (e.g., self-adaptive [6]). Moreover, it will be necessary to study how to associate UPPAAL time units with more concrete metrics such as milliseconds. We intend to implement Co-Analysis, shown in Figure 1, for performance estimation and validation. Furthermore, we are investigating a method to automatically improve model performance by exploiting multi-objective optimization approaches [5]. Consequently, after building a model, a designer can check its functional correctness, validate performance requirements, and optimize the design.

References

[1] Gerd Behrmann, Alexandre David, and Kim G. Larsen. 2004. A Tutorial on Uppaal. In *Formal Methods for the Design of Real-Time Systems: International School on Formal Methods for the Design of Computer, Communication, and Software Systems, Bertinora, Italy, September 13-18, 2004, Revised Lectures*. Springer, Berlin, Heidelberg.

[2] Rabie Ben Atitallah, Philippe Marquet, Éric Piel, Samy Meftali, Smail Niar, Anne Etien, Jean-Luc Dekeyser, and Pierre Boulet. 2008. Gaspard2: from MARTE to SystemC Simulation. In *Proceeedings of the DATE'08 workshop on Modeling*



(a) Timer's period equal to 50 time unit

(b) Timer's period equal to 20 time units

Figure 6: Number of times the displayed state is reached in 1000 time units by stimulating the application with different timers.

and Analyzis of Real-Time and Embedded Systems with the MARTE UML profile. Washington, United States.

[3] Bo Chen, Xi Li, and Xuehai Zhou. 2018. Model checking of MARTE/CCSL time behaviors using timed I/O automata. *Journal of Systems Architecture* 88 (2018), 120–125.

[4] Jinho Choi, Eunkyoung Jee, and Doo-Hwan Bae. 2016. Timing consistency checking for UML/MARTE behavioral models. *Software Quality Journal* 24, 3 (2016), 835–876.

[5] Vittorio Cortellessa, Daniele Di Pompeo, Vincenzo Stoico, and Michele Tucci. 2023. Many-objective optimization of non-functional attributes based on refactoring of software models. *Information and Software Technology* 157 (2023), 107159. https://doi.org/10.1016/j.infsof.2023.107159

[6] G. D'Andrea, T. Di Mascio, and G. Valente. 2019. Self-adaptive loop for CPSs: Is the Dynamic Partial Reconfiguration profitable?. In *2019 8th Mediterranean Conference on Embedded Computing, MECO 2019 - Proceedings*.

[7] Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikučionis, and Danny Bøgsted Poulsen. 2015. Uppaal SMC tutorial. *International Journal on Software Tools for Technology Transfer* 17 (2015), 397–415.

[8] Zamira Daw and Rance Cleaveland. 2015. Comparing model checkers for timed UML activity diagrams. *Science of Computer Programming* 111 (2015), 277–299.

[9] P. Derler, E. A. Lee, and A. Sangiovanni Vincentelli. 2012. Modeling Cyber–Physical Systems. *Proc. IEEE* 100, 1 (2012), 13–28.

[10] Andrea Enrici, Ludovic Apvrille, and Renaud Pacalet. 2017. A Model-Driven Engineering Methodology to Design Parallel and Distributed Embedded Systems. *ACM Transactions on Design Automation of Electronic Systems* 22 (2017), 34:1–34:25.

[11] Fan Gu, Xinqian Zhang, Mingsong Chen, Daniel Große, and Rolf Drechsler. 2016. Quantitative Timing Analysis of UML Activity Diagrams Using Statistical Model Checking. In *Proceedings of the 2016 Conference on Design, Automation & Test in Europe* (Dresden, Germany) *(DATE '16)*. EDA Consortium, San Jose, CA, USA, 780–785.

[12] Fenglin Han, Peter Herrmann, and Hien Le. 2013. Modeling and Verifying Real-Time Properties of Reactive Systems. In *Proceedings of the 2013 18th International Conference on Engineering of Complex Computer Systems (ICECCS '13)*. IEEE Computer Society, USA, 14–23. https://doi.org/10.1109/ICECCS.2013.13

[13] Fernando Herrera, Julio Medina, and Eugenio Villar. 2017. *Modeling Hardware/Software Embedded Systems with UML/MARTE: A Single-Source Design Approach*. Springer Netherlands, Dordrecht, 141–185.

[14] K. Keutzer, A. R. Newton, J. M. Rabaey, and A. Sangiovanni-Vincentelli. 2000. System-level design: orthogonalization of concerns and platform-based design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 19, 12 (2000), 1523–1543.

[15] Grischa Liebel, Nadja Marko, Matthias Tichy, Andrea Leitner, and Jörgen Hansson. 2018. Model-based engineering in the embedded systems domain: an industrial survey on the state-of-practice. *SoSyM* 17 (2018), 91–113.

[16] William E. McUmber and Betty H. C. Cheng. 2001. A General Framework for Formalizing UML with Formal Languages. In *Proceedings of the 23rd International Conference on Software Engineering* (Toronto, Ontario, Canada) *(ICSE '01)*. IEEE Computer Society, USA, 433–442.

[17] V. Muttillo, G. Valente, D. Ciambrone, V. Stoico, and L. Pomante. 2018. Hepsycode-RT: A real-time extension for an ESL HW/SW Co-design methodology. In *ACM International Conference Proceeding Series*.

[18] OMG Standards Development Organization. 2023. About the UML Profile for MARTE Specification Version 1.2. https://www.omg.org/spec/MARTE/1.2/About-MARTE/ Last accessed 25 January 2023.

[19] OMG Standards Development Organization. 2023. The Unified Modeling Language 2.5.1 Specification. https://www.omg.org/spec/UML/2.5.1/About-UML/ Last accessed 25 January 2023.

[20] Luigi Pomante, Vittoriano Muttillo, Marco Santic, and Paolo Serri. 2020. SystemC-based electronic system-level design space exploration environment for dedicated heterogeneous multi-processor systems. *Microprocessors and Microsystems* 72 (Feb. 2020).

[21] Ingo Sander, Axel Jantsch, and Seyed-Hosein Attarzadeh-Niaki. 2017. ForSyDe: System Design Using a Functional Language and Models of Computation. In *Handbook of Hardware/Software Codesign*. Springer Netherlands, Dordrecht, 1–42.

[22] Jagadish Suryadevara, Cristina Seceleanu, Frédéric Mallet, and Paul Pettersson. 2013. Verifying MARTE/CCSL Mode Behaviors Using UPPAAL. In *Proceedings of the 11th International Conference on Software Engineering and Formal Methods - Volume 8137* (Madrid, Spain) *(SEFM 2013)*. Springer-Verlag, Berlin, Heidelberg, 1–15.

[23] J. Teich. 2012. Hardware/Software Codesign: The Past, the Present, and Predicting the Future. *Proc. IEEE* 100 (2012), 1411–1430.

[24] Frank Vahid and Tony Givargis. 2001. *Embedded System Design: A Unified Hardware/Software Introduction* (1st ed.). John Wiley & Sons, Inc., USA.

[25] Jorgiano Vidal, Florent de Lamotte, Guy Gogniat, Philippe Soulard, and Jean-Philippe Diguet. 2009. A Co-Design Approach for Embedded System Modeling and Code Generation with UML and MARTE. In *Proceedings of the Conference on Design, Automation and Test in Europe* (Nice, France) *(DATE '09)*. European Design and Automation Association, Leuven, BEL, 226–231.

[26] Jim Woodcock, Peter Gorm Larsen, Juan Bicarregui, and John Fitzgerald. 2009. Formal methods: Practice and experience. *Comput. Surveys* 41, 4 (2009), 19:1–19:36.

[27] Yu Zhou, Luciano Baresi, and Matteo Rossi. 2013. Towards a Formal Semantics for UML/MARTE State Machines Based on Hierarchical Timed Automata. *Journal of Computer Science and Technology* 28, 1 (2013), 188–202.