Combining UML-MARTE and Preemptive Time Petri Nets: An Industrial Case Study

Irene Bicchierai, Giacomo Bucci, *Member, IEEE*, Laura Carnevali, and Enrico Vicario, *Member, IEEE*

***Abstract*—We present an approach for integration of formal methods within an industrial SW process, illustrating results obtained in a real scenario subject to Military Standard 498 (MIL-STD-498). On the one hand, the formal nucleus of preemptive Time Petri Nets (pTPNs) is used to support design and verification activities of the development process; on the other hand, the Unified Modeling Language (UML) profile for Modeling and Analysis of Real-Time and Embedded (MARTE) systems is adopted to manage the documentation process prescribed by MIL-STD-498. The two cores are integrated by providing guidance for translation of UML-MARTE specifications into equivalent pTPN models, with specific reference to concurrency control and synchronization mechanisms. This permits to attain a smooth transition from the standard artifacts of MIL-STD-498 to pTPN models and analyses, facilitating deployment of the formal core of pTPNs with a limited impact on the industrial practice. The experience proves practical feasibility and effectiveness of the approach, comprising a step towards industrial applicability of formal methods and practices.**

***Index Terms*—Execution Time profiling, Military Standard 498 (MIL-STD-498), model-driven development, preemptive time Petri Nets (pTPNs), real-time code, real-time systems, SW development process, Unified Modeling Language Modeling and Analysis of Real-Time and Embedded (UML-MARTE), V-model.**

I. Introduction

IN several application domains, the development of safety-critical SW is subject to certification standards such as RTCA/DO-178B [46], Military Standard 498 (MIL-STD-498) [52], CENELEC EN 50128 [22], ECSS E-40 [47], and IEC 62304 [31]. Some of these standards explicitly recommend the introduction of formal methods as a means to improve the rigor of development and the quality of SW, provided that the adoption of these techniques does not radically upset the consolidated practice. Hence, an increasing attention is focused on any measure aimed at smoothing the impact of formal methods so as to facilitate an effective integration within the development life cycle.

Various efforts have been pursued to accommodate the two issues by compiling UML specifications [41], [42] into formal models used for performance prediction [2] and dependability analysis [9]. In many of these approaches, UML diagrams are translated into Petri Net models [6]–[8], [26], [38]. In [38], a compositional approach derives a Generalized Stochastic Petri Net (GSPN) from a UML State Machine based on StateChart Diagrams, defining a formal semantics for a significant subset of State Machine elements. The approach is extended in [7] by applying the translation also to UML Sequence Diagrams, providing a more complete representation of system behavior. The method proposed in [8] combines State Machines and Activity Diagrams to derive a Stochastic Well-formed Net for evaluation of performance metrics, such as sojourn time and response time. In [26], performance of an SW architecture is evaluated through a two-phase methodology which first annotates a UML specification with tags and stereotypes of the UML profile for Schedulability, Performance, and Time Specification (SPT) [39] and then generates a corresponding Non-Markovian Stochastic Petri Net (NMSPN) model. In [6], a Time Petri Net (TPN) model is derived from a UML-based SW specification enriched with annotations of the UML profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE) [40], which is specifically targeted to capture nonfunctional properties of real-time embedded systems. The resulting TPN is used to assess the risk of timing failures in early stages of SW life-cycle.

Several other approaches address translation of UML specifications into performance models based on Queuing Networks (QN) and process algebrae (PA) [3], [23], [24], [28], [29], [36], [44], [45], [50], [56]. The approach proposed in [44] builds a Layered Queuing Network (LQN) from a UML description of system architecture made of Class/Object Diagrams and Sequence Diagrams, by converting each architectural pattern into a performance submodel. In [24], QN models are incrementally built from UML diagrams early available during SW development, providing fast feedback to the designer. The approach is extended in [28] to encompass mobility-based paradigms in the SW architecture of an application. In [36], a set of annotated Use Case, Activity, and Deployment Diagrams is translated into a discrete-event simulation model used to derive performance indexes. The methodology is improved in [3] using QN analysis to derive performance bounds. In the approach of [50], annotated UML specifications are exported and analyzed as QN models, using an XML-based interchange format which allows flexibility in design and analysis stages. SW performance analysis is also applied in [23] in the context of an industrial case study from the telecommunication domain, translating Sequence Diagrams and StateChart Diagrams first into flow graphs and then into a specification based on Æmilia [10], an Architectural Description Language (ADL) defined upon a Stochastic Process Algebra (SPA). In [45], a metamodel named Core Scenario Model (CSM) is defined which supports derivation of various kinds of performance models from a UML diagram annotated with UML-SPT stereotypes. The approach is implemented

Manuscript received September 30, 2011; revised January 23, 2012; accepted February 22, 2012. Date of publication June 20, 2012; date of current version October 14, 2013. This work was supported by the SW-Initiative of the FinMeccanica Group. Paper no. TII-11-602.

The authors are with the Dipartimento Sistemi e Informatica, Università di Firenze, 50139, Firenze, Italy (e-mail: irene.bicchierai@unifi.it; giacomo.bucci@unifi.it; laura.carnevali@unifi.it; enrico.vicario@unifi.it).

Digital Object Identifier 10.1109/TII.2012.2205399

in the Performance by Unified Model Analysis (PUMA) tool architecture [56], which provides a unified interface between SW design models and performance models. An intermediate meta-model is used also in [29] to derive performance models from UML diagrams. The transformation framework is based on a kernel language called KLAPER and helps in bridging the gap between design-oriented and analysis-oriented notations.

Integration of formal methods along the development process of real-time SW has been practiced in various Model Driven Development (MDD) approaches and related tools [1], [13], [20], [30], [34], [51], supporting formalization of system requirements and design choices through Domain Specific Modeling Languages (DSMLs), and automated derivation of concrete artifacts such as real-time code, documentation, and tests [33], [48]. The model-based SW development process presented in [13] supports simulation and testing of complex embedded systems in automotive applications. To this end, an executable specification of the entire system is generated during early design phases and then iteratively refined throughout the design process. The Palladio model-driven approach [32] supports prediction of Quality of Service (QoS) properties of component-based SW architectures, providing a meta-model for specification of performance-relevant information [5] and a simulator for derivation of performance, reliability, maintainability, and cost metrics. It is implemented in a well-established tool which enables integration within a component-based development process [34]. In [20], an MDD framework is presented that integrates the core theory of preemptive TPNs (pTPNs) [15], [16] in a tailoring of the V-Model SW life cycle [19], enabling automated derivation of pTPN models from a semiformal specification, automated compilation of models into real-time code running on RTAI [25], and measurement-based Execution Time evaluation. As a characterizing trait, pTPNs encompass temporal parameters varying within an assigned interval and support representation of suspension in the advancement of clocks. This attains an expressivity that compares with StopWatch Automata [21], Petri Nets with hyper-arcs [43], and Scheduling-TPNs [35], enabling convenient modeling of usual patterns of real-time concurrency [18].

In this paper, we extend the formal methodology of [20] according to the experience of application in a one-year-long project of development at the FinMeccanica site of Selex Galileo in Florence. The approach introduces UML-MARTE [40] diagrams to manage the documentation process prescribed by MIL-STD-498 [52], providing guidance for translation into equivalent pTPN models. This provides a base ground that fits the industrial practice subject to MIL-STD-498 and facilitates subsequent deployment of the formal nucleus of pTPNs; at the same time, this also attains a limited impact on both the development process and life cycle data. We illustrate the experimented methodology and exemplify its application to the case study, discussing specific peculiarities and complexities in depth while avoiding disclosure of classified details subject to industrial secrecy constraints.

The remainder of this paper is organized as follows. In Section II, we present an industrial tailoring of the V-Model framework (Section II-A), and we illustrate the experimented methodology (Section II-B). In Section III, we report on a notation similar to Class Responsibility Cards (CRC) employed to document analysis of SW requirements (Section III-A); we show how UML-MARTE has been conveniently introduced to manage the documentation process prescribed by MIL-STD-498 (Section III-B), and we illustrate how UML-MARTE diagrams have been converted into timeline schemata [18] prior to pTPNs to provide a synthetic description of SW design, extending the notation of [20] to model one-shot tasks, branches, and rejoins (Section III-C). In Section IV, we illustrate application of the core theory of pTPNs to SW development. Specifically: we extend the process of translation of timeline schemata into pTPN specifications to include one-shot tasks, branches, and rejoins (Section IV-A). We discuss automated verification of sequencing and timing constraints in Section IV-B. We provide guidance for disciplined implementation of real-time code that conforms with pTPN semantics and runs on VxWorks 6.5 [55] in Section IV-C, and we report on profiling of temporal parameters of the model in Section IV-D. Conclusions are finally drawn in Section V.

II. Integrating Formal Methods in an Industrial SW Process

We introduce here a methodology that integrates UML-MARTE [40] and pTPNs [16] in an industrial SW process, illustrating its application in a real project subject to MIL-STD-498 [52].

A. Industrial Tailoring of the V-Model Life Cycle

Fig. 1 shows the general structure of a V-Model SW life cycle [19] (inner scheme) and the specific industrial tailoring (outer scheme) at our experimentation site, emphasizing the artifacts of the documentation process prescribed by MIL-STD-498 [52] and possible iterative refinements along the development. The steps are briefly recalled to introduce the concepts that are significant for the proposed methodology.

SD1 (System Requirements Analysis), SD2 (System Design), and the first part of SD3 (SW-HW Requirements Analysis) are integrated in a single activity named *System/Subsystem Analysis and Design*. This develops on the outcomes of the *Planning and Budget* activity (out of the scope of the V-Model) and produces the SSDD (*System/Subsystem Design Description*) document, specifying system decomposition into units made of CSCIs (*Computer Software Configuration Items*), HCIs (*Hardware Configuration Items*), and FCIs (*Firmware Configuration Items*). The second part of SD3 is mapped on *SW Requirements Analysis*, which lists all functional and non-functional SW requirements in the SRS (*Software Requirements Specification*) document, and defines inter-unit communication requirements of each unit interface in individual IRS (*Interface Requirements Specification*) documents.

SD4-SW (Preliminary Software Design) and SD5-SW (Detailed Software Design) are integrated in *SW Design*, which produces the SDD (*Software Design Description*) document, specifying the *dynamic architecture* of each CSCI as a set of

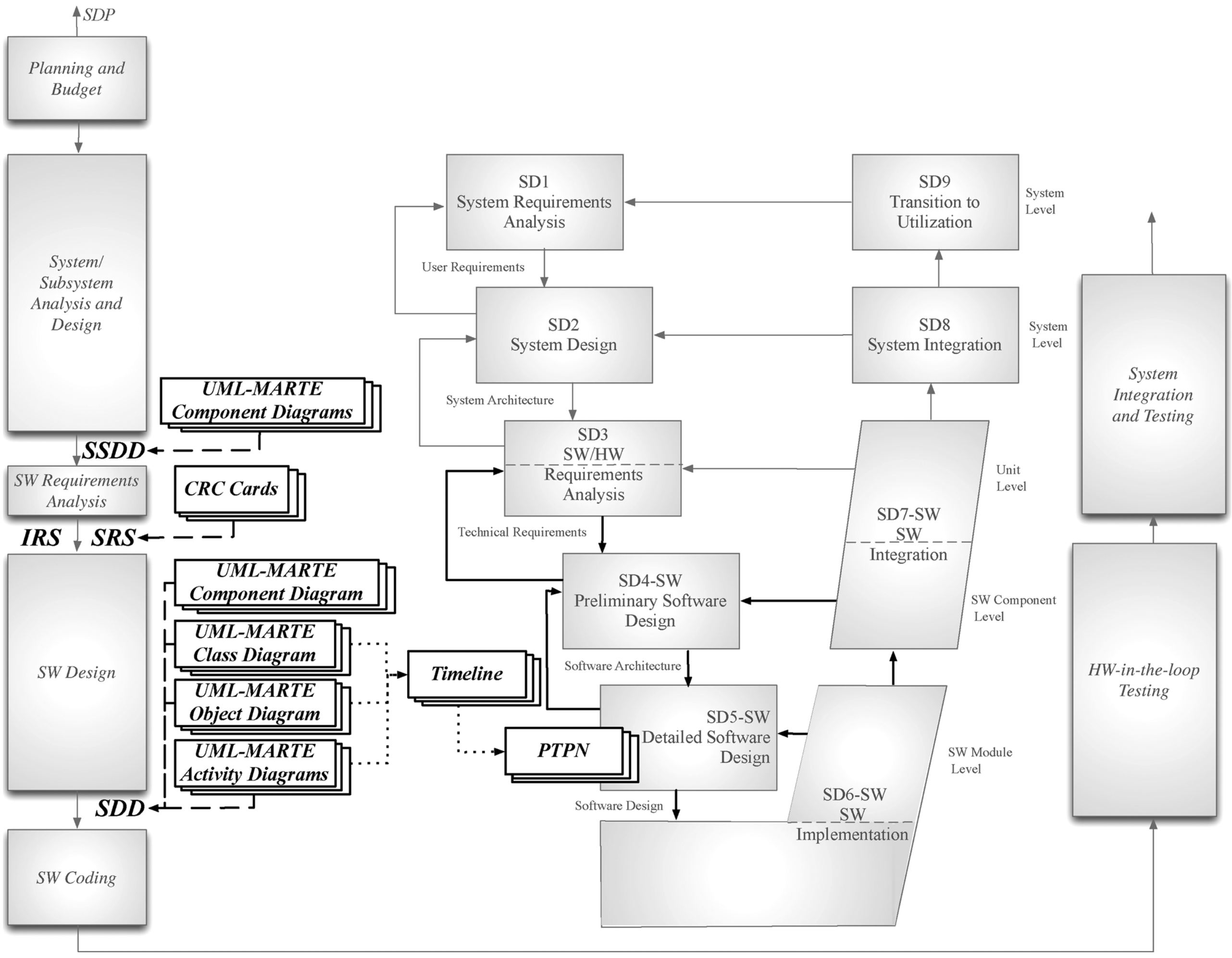

Fig. 1. Scheme of the System Development (SD) submodel of the V-Model life cycle [19] tailored according to MIL-STD-498 [52]. The picture highlights artifacts of the proposed methodology (white boxes), inclusion of artifacts within MIL-STD-498 documents (dashed arrows), translation of documentation artifacts into a formal specification (dotted arrows), and development iterations supported by the approach (bold arrows).

concurrent tasks with allocated resources and prescribed time requirements.

SD6-SW (SW Implementation) is covered by *SW Coding*, which implements the dynamic architecture of each CSCI and their functional behavior, and by the first part of *HW-in-the-loop Testing*, which addresses testing of low-level modules. SD7-SW (SW Integration) at the SW Component Level is mapped on the second part of *HW-in-the-loop Testing*, which verifies the integration of low-level modules within each CSCI.

SD7-SW (SW Integration) at the Unit Level and SD8 (System Integration) are aggregated in *System Integration and Testing*, which tests first the integration of CSCIs, HCIs and FCIs within each unit, and then the integration of all units within the system; SD9 (Transition To Utilization) puts the completed system into operation at the intended application site. These activities are out of the scope of the industrial tailoring described here.

B. Formal Methodology Based on UML-MARTE and pTPNs

Certification standards for safety-critical SW, such as RTCA/DO-178B [46], usually encourage the adoption of formal methods as a means to improve the degree of rigor attained by the development process, especially when SW includes complex behavior characterized by concurrency control, synchronization mechanisms, distributed processing, and nondeterministic timings. Formal methods can actually contribute to increase the quality of SW components by supporting multiple activities along the development life cycle. Formal modeling provides a well-defined semantics, which removes inconsistencies of natural language and permits definition of a non-ambiguous specification. This enables rigorous analysis through comprehensive exploration of system behaviors, supporting derivation of a proof of correctness of SW design. As a relevant point, early assessment of requirements allows early feedback at design stage, which may have an impact on the quality and the cost of the final product. Formal specification also supports MDD, including derivation of code that preserves model semantics, fast prototyping, incremental integration, and testing of low-level modules. The formal description supports the testing stage as well, providing the basis for automation of the testing process and for generation of a test oracle.

At the same time, certification standards also require that technological transfer of formal methods into industry be achieved at a reasonable cost and with a limited impact on the SW life cycle. Unfortunately, the characteristics of an industrial development process comprise various hurdles per se, impeding the introduction of advanced formal techniques without disrupting conventional practices. An industrial process of SW development is usually subject to a documentation standard defining the procedure that should be followed for document production as well as structure, information content, and presentation of each document. These artifacts are traditionally written in natural language and illustrated through domain-specific visual notations, which result from the consolidated experience rather than from an established standard. This comprises not only the design practice which domain experts are best skilled at, but also what is often expected from them in a certification process. As a matter of fact, industrial developers would encounter major troubles in managing formal techniques, due to the complexity of notations, the difficulties in properly understanding analysis outputs, and the limited familiarity with existing tools. For all of these good reasons, straight introduction of formal specifications in an industrial documentation process is not viable. Hence, formal methods should be combined with formalisms and tools that permit to attain a smooth transition from standard artifacts of the documentation process to formal modeling and analysis techniques, guaranteeing conservative representation, ease of use, and rapid configuration.

In the approach proposed here, we extend [20] by combining UML-MARTE diagrams [40] and pTPN theory [16] both to manage the documentation process prescribed by MIL-STD-498 [40] and to support development activities. In particular, UML-MARTE provides a semiformal specification that is sufficiently practical to meet the needs of an advanced industrial domain and sufficiently structured to allow mapping on pTPNs. This enables integration of the two core processes in a unified methodology, yielding an effective ground for deployment of pTPN theory while attaining a smooth impact on the consolidated practice. The methodology provides guidance for translation of UML-MARTE diagrams into equivalent pTPN models, using timeline schemata as an intermediate artifact supplying a synthetic and compact representation of SW design.

In a different perspective, not developed in this paper, integration of documentation and development processes could be achieved the other way round, by compiling pTPN/timeline models into UML-MARTE diagrams to be exposed in the documentation process. The approach is actually feasible and represents the viewpoint of MDD, where a formal specification is transformed not only into code and tests but also into documentation artifacts [48].

The methodology also provides a quantitative ground that drives feedback cycles allowed by the SW development process. As illustrated in Fig. 1, some iterations of the V-Model framework may be performed until contractual SW Requirements are satisfied. During SW Design, if the analysis detects a deadline miss or a deadline satisfied with a very small laxity, then the dynamic architecture is refined to fix the identified flaw. If this cannot be performed without relaxing some SW Requirement, then the development process is restarted from SW Requirements Analysis. During HW-in-the-loop Testing, if an unsequenced execution or a time-frame violation [20] is detected, then pTPN formal specification is used to identify task programming defects or cycle stealing in the real-time code, and a first attempt is done to fix and/or refine the implementation. If this does not succeed, for instance, because a function with an Execution Time out of its nominal range cannot be further optimized, the development process is restarted from SW Design and then, if necessary, from SW Requirements Analysis.

III. Supporting the Documentation Process Through UML-MARTE

Here, we show how UML-MARTE [40] and other notations can be used to support the documentation process prescribed by MIL-STD-498 [52], providing a bridge towards deployment of advanced formal methods.

The proposed methodology starts with System/Subsystem Analysis and Design, and it accompanies the development up to SW Coding and HW-in-the-Loop Testing. For space limitations, we avoid to report here on the way how UML-MARTE is employed in System/Subsystem Analysis and Design and we start from SW Requirements Analysis, assuming that the SSDD document provides the definition of User Requirements and enables the allocation of system functionalities to units and, in turn, the allocation of unit functionalities to CSCIs, HCIs, and FCIs.

In the industrial case study addressed in this paper, an electro-optical system is developed as a part of the equipment of a military vehicle to guarantee battlefield advantage through the use of visual, infra-red and thermal imaging, long-range target acquisition and illumination, and precise aiming. Therefore, the system is decomposed into: an Optical Unit (OU) made of sensors, cameras, and servo-motors; an Electronic Unit (EU) responsible for sensor control and image processing; and, a System Monitoring Unit (SMU) managing the entire system. We focus here on the development of EU, which is sufficient to illustrate the methodology of the approach and the complexities of the case study. According to this, we illustrate decomposition of OU into HCIs only to make explicit the connections with HCIs of EU, and we leave SMU out of the scope for this paper.

EU plays the role of a bridge in the communication between SMU and OU, forwarding the commands periodically sent by SMU to OU and sending back the corresponding replies. EU also processes images acquired by OU and sends obtained results to SMU. Therefore, EU functionalities are allocated to two CSCIs: System Control (SC), responsible for communication with OU and SMU, and Image Tracking (IT), responsible for image processing. In turn, each CSCI is associated with a real-time task-set and allocated to an HCI. Specifically: SC is allocated to Main Board (MB), which embeds a PowerPC MPC 5200B processor [27] and runs the VxWorks 6.5 [55] Real-Time Operating System (RTOS); and, IT is allocated to Video Processor (VP), which runs a proprietary commercial RTOS. OU is made of six HCIs: Servo-Motor (SM); InfraRed Camera (IRC); and, a Laser Visual Device collecting four HCIs named TeleVision Camera (TVC), Laser Sensor (LS), Optical Sensor (OS), and Temperature Sensor (TS).

TABLE I
SW REQUIREMENTS ANALYSIS: SRS DOCUMENT.
CRC CARD OF SYSTEM CONTROL CSCI (SC)

CAPABILITY	DESCRIPTION	COLLABORATION
Init	HW and SW initialization	-
LS-IRC_Power	Switching on/off LS and IRC	LS, IRC
TVC_Config	Management of the TVC configuration	TVC
SMU-OU_Cmd	Management of the messages exchanged by SMU and OU	-
SM_Comm	Communication with SM	SM
SMU_Comm	Communication with SMU	SMU
IT_Comm	Communication with IT	IT
IRC_Comm	Communication with IRC	IRC
TVC_Comm	Communication with TVC	TVC
LS_Comm	Communication with LS	LS

TABLE II
SW REQUIREMENTS ANALYSIS: SRS DOCUMENT. DECOMPOSITION OF *SMU-OU_COMMANDS* IN SUB-CAPABILITIES

SUB-CAPABILITY	DESCRIPTION	COLLABORATION
SMU_Cmd	Management of the SMU commands and the OU replies	-
TVC_Cmd	Management of the TVC configuration parameters	-
HCIs_Trasm	Activation of the data transmission to IRC, TVC, and LS	-
LS-IRC_State	Management of the switched on/off state of IRC and LS	-
HCIs_Data	Processing of the HCIs data	-
SM_LocationData	Processing of the SM location data elaborated by IT	-
OperationModes	Management of Operation Modes	-

A. SW Requirements Analysis

SW Requirements Analysis and subsequent activities proceed separately for each CSCI up to the final integration. In the SRS document of a CSCI, a conventional structure similar to Class Responsibility Collaboration (CRC) cards [4] can be used to specify its functional behavior as a set of capabilities. Each capability reflects a CSCI functionality, is associated with one or more collaborating HCI/CSCI/Unit, and may be decomposed in sub-capabilities.

From now on, the proposed methodology is illustrated with reference to SC, i.e., the CSCI of EU that implements more complex behavior and provides more stimuli for discussion. Table I specifies its capabilities. *Init* initializes HW and SW; capabilities from *IT_Comm* up to *SM_Comm* manage buses connecting SC with IT, SMU, and four HCIs of OU (i.e., LS, IRC, TVC, and SM); *LS-IRC_Power* and *TVC_Config* control LS, IRC, and TVC sensors; *SMU-OU_Cmd* manages communication between SMU and OU and, since it requires the most computational effort, it is decomposed in subcapabilities, shown in Table II.

B. SW Design: Semi-Formal Specification Through UML-MARTE

In the proposed methodology, the SDD document produced by SW Design specifies the *dynamic architecture* of a CSCI in the form of a set of concurrent tasks [18] following a pre-defined structure. In addition to the task-set structure described in [20], a task may be either *recurrent* or *one-shot*. A one-shot task is a single job activated in reaction to an internal event (e.g., the release of a semaphore) or an external event (e.g., the arrival of a signal), with deadline less or equal to the minimum inter-occurrence time of the event. The model of a task-set can be conveniently documented through a UML-MARTE Class Diagram, as illustrated in Fig. 2. Tasks are specified by the *SwSchedulableResource* stereotype (i.e., a resource that executes concurrently with other resources under the supervision of a scheduler according to a scheduling policy); chunks are modeled through the *EntryPoint* stereotype (i.e., a routine to be executed) and the association between a task and its chunks is modeled as a dependency; binary semaphores are represented by the *SwMutualExclusionResource* stereotype (i.e., a resource used to synchronize the access to shared variables).

During SW Design, capabilities identified during *SW Requirements Analysis* are allocated to tasks, enabling definition of their functional and non-functional requirements. Referring to the industrial case study, the six capabilities managing buses that connect SC with OU, SMU, and IT (in Table II, *Comm* capabilities) are allocated to separate tasks named *Tsk*2, *Tsk*3, …, and *Tsk*7, respectively; the remaining four capabilities (in Table II, *Init*, *LS-IRC_Power*, *TVC_Config*, and *SMU-OU_Cmd*) are assigned to a single task named *Tsk*1. According to this, the SC task-set is made of seven tasks and *Tsk*1 comprises its central element. *Tsk*2, *Tsk*4, *Tsk*5, *Tsk*6, and *Tsk*7 interface the associated HCI/CSCI with *Tsk*1 which, in turn, is interfaced to SMU through *Tsk*3.

According to the proposed methodology, SW Design proceeds through *i)* the definition of non-functional requirements through a UML-MARTE Object Diagram and *ii)* the specification of functional requirements through UML-MARTE Activity Diagrams.

1) Specification of Non-Functional Requirements: In the definition of the dynamic architecture of a task-set, non-functional requirements are derived from contractual prescriptions, or obtained from previous artifacts, or autonomously chosen by the developer. Minimum inter-release times and deadlines are directly prescribed by User Requirements, while task periods are usually design choices. The number of chunks constituting a task reflects the number of sub-capabilities allocated to the task, and it may be refined during development iterations depending on the number of branches in the structure of the task. The Execution Time of a chunk can be first tentatively guessed through analogy with previous or similar realizations, and it is progressively refined during development iterations. Semaphore synchronizations necessary to access shared data directly come from tasks interactions.

A UML-MARTE Object Diagram can effectively capture the dynamic architecture of a task-set, enabling representation of non-functional properties, as exemplified in Fig. 3 with reference to the SC task-set of the industrial case study. *Tsk*1 and *Tsk*2 are periodic tasks with period and deadline of 10 and 20 *ms*, respectively; *Tsk*3 and *Tsk*4 are sporadic tasks with minimum inter-release time and deadline of 20 and 40 *ms*, respectively; *Tsk*5, *Tsk*6, and *Tsk*7 are one-shot tasks with deadline of

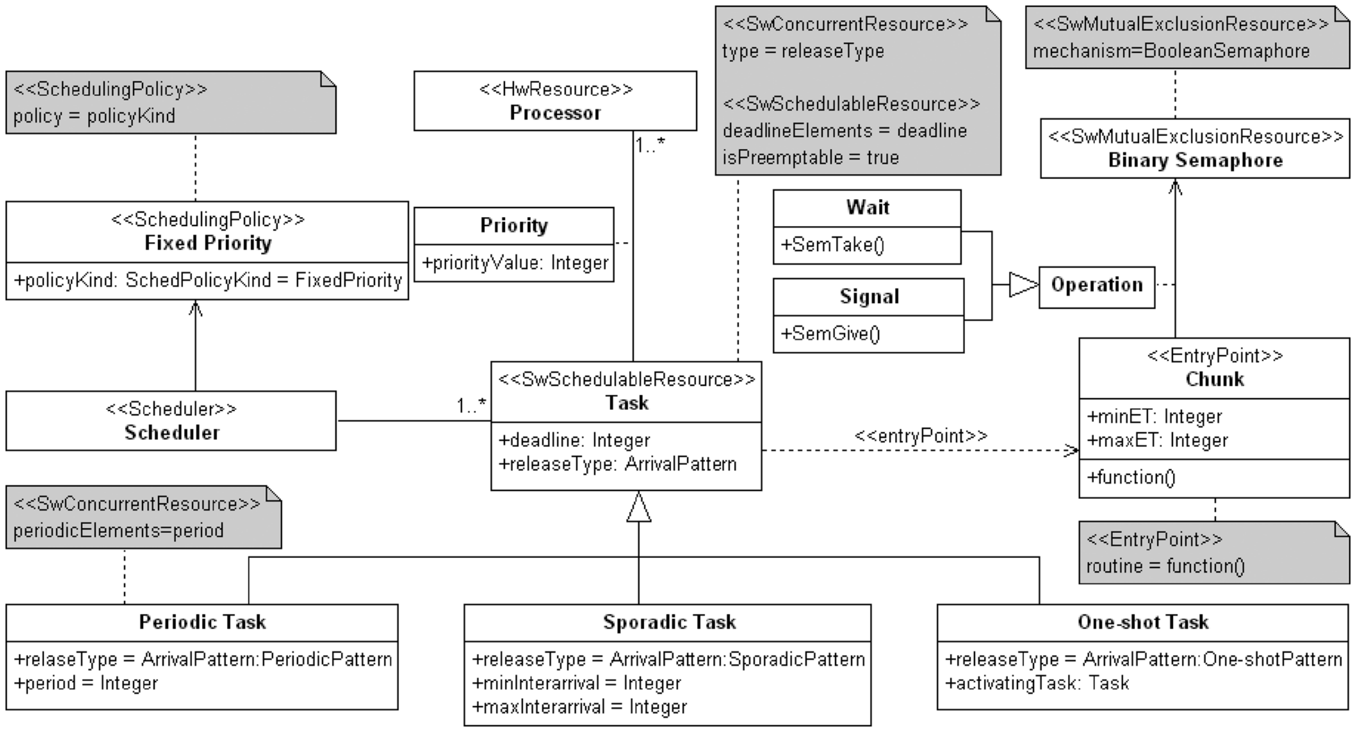

Fig. 2. SW Design: SDD document. UML-MARTE Class Diagram of the task-set model.

10 *ms*. *Tsk*1 requires *cpu* with priority level 1; the other tasks require *cpu* with priority level 2. Specifically, minimum inter-release times and deadlines of *Tsk*3 and *Tsk*4 directly come from User Requirements constraining timeliness of image processing and system management; *Tsk*5, *Tsk*6, and *Tsk*7 are modeled as one-shot tasks since communication with IRC, TVC, and LS is activated on demand by *Tsk*1, depending on the HCI/CSCI addressed by the current SMU command; *Tsk*1 and *Tsk*2 are modeled as periodic tasks to guarantee recurrent control on servo-motors and SMU-OU messages; *Tsk*2 period and deadline are chosen equal to *Tsk*3 deadline so as to timely actuate SMU commands addressing servo-motors; *Tsk*1 period and deadline are selected equal to half *Tsk*3 deadline as a result of the subsequent analysis.

A UML-MARTE Object Diagram also permits to specify the chunks of each task and their semaphore synchronizations. This is exemplified in Fig. 3 with reference to the SC task set of the industrial case study, avoiding representation of every chunk and semaphore to reduce the cluttering. For instance, *Tsk*1 is made of 22 chunks, which result from the four capabilities assigned to SC, the subcapabilities of *SMU-OU_Commands*, and some branches introduced during refinement of entry-points; chunk *C*11 executes entry-point `f11` with an Execution Time constrained within [0.005, 0.100] *ms*, and it is synchronized with chunk *C*23 on semaphore *sem*1 to access data that *Tsk*1 shares with *Tsk*2.

2) Specification of Functional Requirements: The procedural aspects of a task-set can be conveniently specified using UML-MARTE Activity Diagrams according to the following methodology.

- Each task is represented by a separate swimlane labeled with the task name.
- Releases of periodic, sporadic, and one-shot tasks are modeled by input signals labeled with the period, the inter-release interval, and the activating event, respectively.
- Chunk computations are specified by actions labeled with the chunk name.
- Private data structures of a task are represented by objects lying within the task swimlane.
- Data structures shared with other tasks are represented by objects lying on the border of the task swimlane.
- Wait and signal semaphore operations are represented through input and output signals, respectively, labeled with the semaphore name.

Fig. 4 illustrates the concept with reference to the first part (up to the first branch) of task *Tsk*1 of the industrial case study. The task is periodically activated every 10 *ms*; after activation, it performs the sequence of chunks *C*11, *C*12, *C*13, and *C*14, synchronizing on semaphores *sem*1, *sem*2, *sem*3, and *sem*4, respectively, to access shared memories. Afterwards, different paths are followed depending on OU and EU configuration parameters.

For reasons of space, the rest of the diagram and the UML-MARTE Activity Diagrams of *Tsk*2 through *Tsk*7, each composed of a task swimlane, are not shown here.

C. SW Design: Semi-Formal Specification Through Timelines

In practical applications, UML-MARTE diagrams often tend to explode in complexity, as illustrated by Figs. 3 and 4. To circumvent the problem, the methodology of development can conveniently integrate a domain specific notation based on the concept of timelines [18]. These provide a synthetic and intuitive description of the dynamic architecture, acting as an intermediate model that helps in bridging the gap between

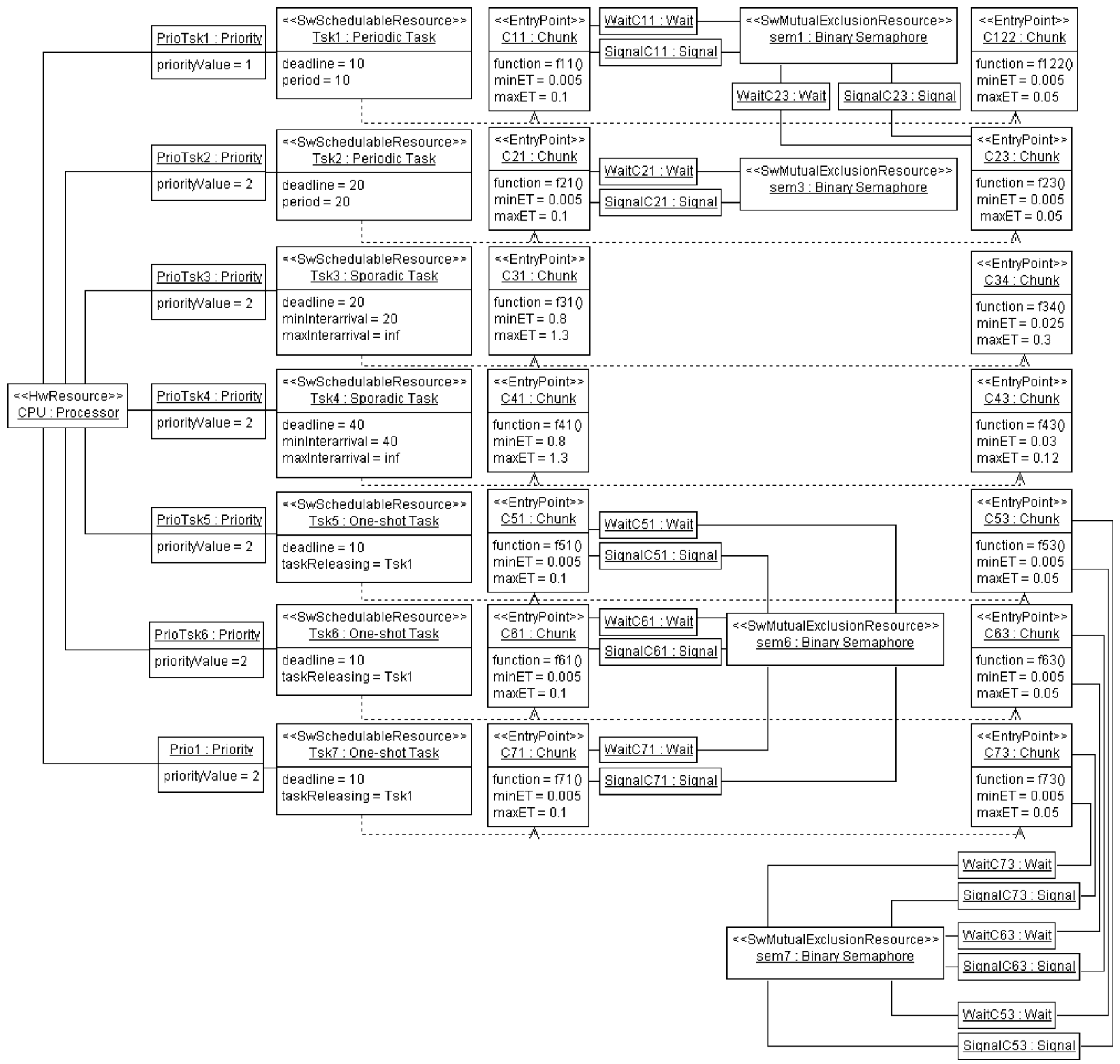

Fig. 3. SW Design: SDD document. UML-MARTE Object Diagram of the SC task-set (times expressed in *ms*). For the sake of readability, only the first and the last chunk of each task are represented, e.g., *Tsk*1 is made of 22 chunks from *C*11 up to *C*122.

a semiformal specification suitable for SW documentation and a formal specification supporting correctness verification through analysis. In this perspective, the proposed approach exposes similarities with [26], [29], and [45], where intermediate artifacts are used as an interface between design-oriented and analysis-oriented models. In addition to the formalism of timelines described in [20], we introduce the following:

- a double-headed or a single-headed arrow over the open box of a task to indicate whether the task is recurrent or one-shot, respectively;
- diamonds to specify branches and re-joins in a sequence of chunks;
- a dotted-arrow from a chunk to each one-shot task activated in reaction to an event thrown by the chunk.

Fig. 5 exemplifies the concept with reference to the SC task-set of the industrial case study, making explicit the sequence of chunks executed by each task and their semaphore synchronizations. Note that the single schema of Fig. 5 replaces the Class Diagram of Fig. 2, the Object Diagram of Fig. 3, the Activity Diagram of Fig. 4, and the remaining Activity Diagrams of *Tsk*2 through *Tsk*7 (not reported here).

IV. Supporting the Development Process Through pTPNs

Here, we illustrate how the formal nucleus of pTPNs [16] is used to support design and verification activities of the development process, providing guidance for derivation of pTPN models from timeline schemata to achieve integration with the

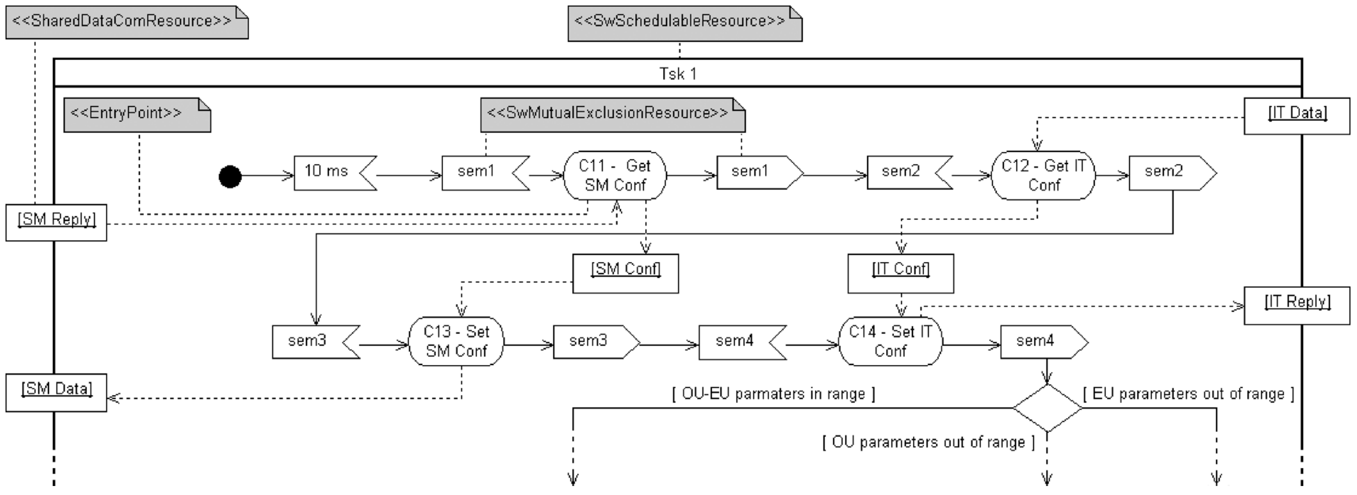

Fig. 4. SW Design: SDD document. A fragment of the UML-MARTE Activity Diagram of task *Tsk*1 of SC, showing the flow of control up to the first branch (times expressed in milliseconds).

documentation process prescribed by MIL-STD-498 [52]. It is worth remarking that the translation process can be automated and the resulting pTPN can even remain transparent to the designer, who will be only concerned with the construction of the timeline schema.

A. SW Design: Formal Specification Through pTPNs

A pTPN [15], [16] extends the model of TPNs [11], [37], [53] with a set of preemptable resources whose availability conditions the progress of timed transitions. According to this, each transition is associated with a firing interval, delimited by a static *Earliest Firing Time* (EFT) and a static *Latest Firing Time*, and may request a set of resources with a *priority level*. A transition is *enabled* if all its input places contain at least one token: in this case, it is associated with a dynamic *time-to-fire* taking a non-deterministic value within its static firing interval. An enabled transition is *progressing* and reduces its time-to-fire if every of its associated resources is not requested by any other enabled transition with a higher priority level; otherwise, it is *suspended* and maintains the value of its time-to-fire, which is resumed when the transition is assigned all its associated resources again. A progressing transition is firable if its time-to-fire is not higher than that of any other progressing transition. When a transition fires, a token is removed from each of its input places and a token is added to each of its output places.

Note that the form of syntax and semantics of pTPNs could be reasonably extended so as to account for weights associated with pre-conditions (i.e., arcs from a place to a transition) and post-conditions (i.e., arcs from a transition to a place). In general, this can help in representing contexts where places account for resources and where multiple resources may be needed to perform semaphore actions. However, in the proposed methodology, this element of expressivity is not needed as transitions account for actions that always depend on boolean conditions.

A pTPN model can be derived from a timeline schema either manually or automatically, following a procedure steered by the model structure. In general, the translation associates a place with each logical condition of each job and with each semaphore, and uses transitions to account for job releases, chunk completions, branches, semaphore and priority operations. With respect to the derivation process described in [20], we add the translation of one-shot task releases, branches, and rejoins. Releases of one-shot tasks are represented by a transition preconditioned by the output place of the transitions that model the completion of the activating chunk. Its firing interval accounts for the time spent in the elaboration of the activating signal. Branches are modeled by immediate transitions preconditioned by the output place of the preceding chunk; conversely, rejoins are accounted by making the chunks share the same output place. For space limitations, Fig. 6 shows only the first part (up to the first branch) of task *Tsk*1 of the SC task-set of the industrial case study. Periodic releases of *Tsk*1 are modeled by transition *t*10 with neither input places nor resource requests; therefore, it fires repeatedly with inter-firing times falling within its static firing intervals. Job chunks are modeled by transitions with static firing intervals equal to the Execution Time range, with requested resources and static priorities. For instance, transition *t*12 models the completion of chunk *C*11 of *Tsk*1. Transitions *t*113, *t*146, and *t*157 are preconditioned by place *p*113 to represent a branch among the mutually exclusive chunks *C*15, *C*16, and *C*17 of *Tsk*1.

According to the priority ceiling emulation protocol [49], low-priority tasks *Tsk*2, *Tsk*3, …, and *Tsk*7 undergo a priority boost and synchronize on a semaphore in the sections where they access memories shared with the high-priority task *Tsk*1. Binary semaphores are modeled as places initially marked with one token. Since experimental results prove that the time spent in priority boost/deboost and semaphore wait/signal operations is not negligible with respect to the Execution Time range of the SC entry-points, these operations are represented by separate transitions with nonpoint-like firing interval. For instance, *sem*1 represents a binary semaphore synchronizing the access to a memory shared between chunks *C*11 and *C*23; *t*11 accounts for *sem*1 wait operation; *t*12 represents the completion of *C*11; *t*13 models *sem*1 signal operations. This differs from [20], where priority boost and semaphore wait operations are represented by immediate transitions, while priority deboost and semaphore

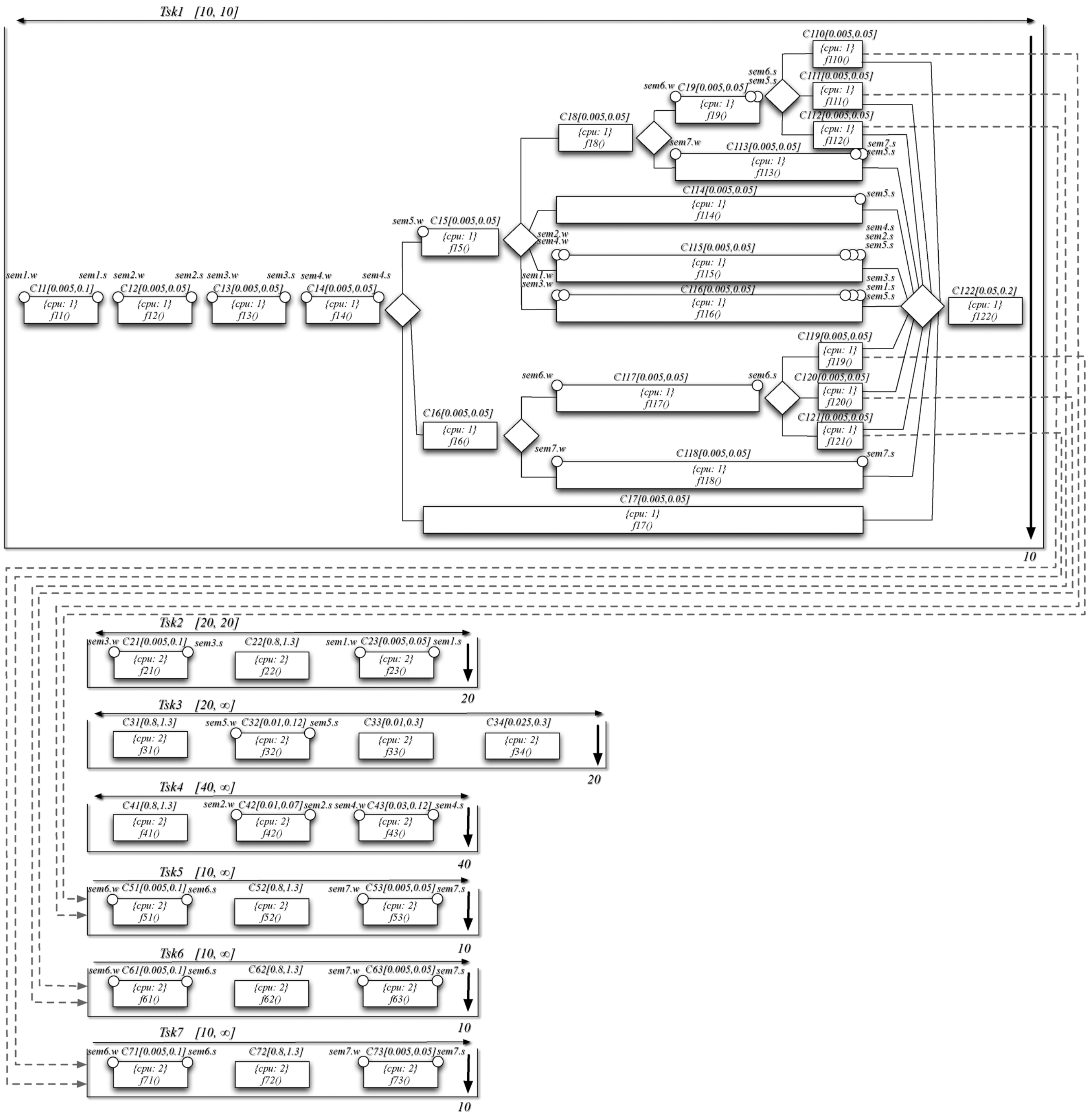

Fig. 5. SW Design: SDD document. The timeline schema of the SC task-set (times expressed in milliseconds).

signal operations are accounted by transitions also modeling chunk completions. The abstraction of [20] is motivated by the fact that, on the RTOS in use there, the time spent in priority and semaphore operations is negligible with respect to the Execution Time range of entry-points under development. Thus, since preemption by a different task within the priority ceiling cannot occur at deboost, the model of [20] does not need to distinguish chunk completions from semaphore signal and priority deboost operations.

Note that deadlines do not have a direct counterpart in the pTPN model, although they could be explicitly represented through additional watch transitions as proposed in [12]. However, this would considerably increase the degree of concurrency of the model and thus the complexity of the analysis. Moreover, in most of the cases, deadlines are coincident with minimum inter-release times, so that deadline misses can be easily identified as task releases occurring while a task-job is still pending.

B. SW Design: Architectural Verification

The pTPN representation of a task-set opens the way to automated verification of non-functional requirements through state-space analysis. This comprises the step of development where the proposed methodology permits to achieve major

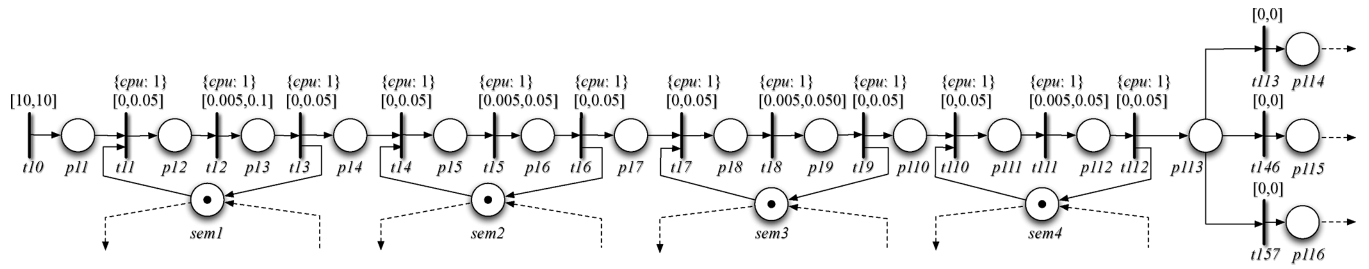

Fig. 6. SW Design: SDD document. A fragment of the pTPN of the dynamic architecture of SC, showing the model of task *Tsk*1 up to the first branch (times expressed in *ms*).

results, which would be significantly hard to perform without relying on a rigorous formal basis.

Verification of non-functional requirements develops on the enumeration of the space of *state-classes*, which is called *state-class-graph* [15], [16]. A *symbolic run* is a path in the state-class-graph representing the dense variety of runs that execute a sequence of transitions with a dense variety of timings between subsequent firings. Selection and timeliness analysis of all symbolic runs that start with a task release and terminate with its completion, which we call *task symbolic runs*, enable the derivation of the *Best Case Completion Time* (BCCT) and the *Worst Case Completion Time* (WCCT) of each task. This supports verification of deadlines as well as derivation of the minimum laxity which they are attained with.

Architectural verification can be performed through the Oris Tool [14], which implements state-space enumeration, selection of paths attaining specific sequencing and timing constraints, and their tight timeliness analysis. In the case of industrial application, the first round of verification detected a *deadline miss* by one-shot tasks *Tsk*5, *Tsk*6, and *Tsk*7, which are triggered by *Tsk*1. Reduction of Execution Times of chunk entry-points was not feasible, since allocated ranges had already been narrowed up to an acceptable trade-off between *precise estimates* and *safe bounds* [54]. Therefore, the dynamic architecture was redesigned by raising *Tsk*1 period from its initial value of 5 *ms* up to 10 *ms*, as shown in the final specification depicted in Figs. 3 and 5. Architectural verification finally yielded the following results: state-space analysis enumerated 4041 state-classes in nearly 1 second; selection and timeliness analysis of task symbolic runs spent nearly 5 seconds to derive 5941, 5660, 5135, 4100, 46 paths for *Tsk*1, *Tsk*2, *Tsk*3, *Tsk*4, *Tsk*5/*Tsk*6/*Tsk*7, respectively, with a WCCT of 1.55, 5.67, 4.02, 7.76, 8.56 *ms*, respectively. This proved that all deadlines were met with minimum laxity of 8.45, 14.33, 15.98, 32.24, 1.44 *ms* for *Tsk*1, *Tsk*2, *Tsk*3, *Tsk*4, *Tsk*5/*Tsk*6/*Tsk*7, respectively.

C. SW Coding: Implementation of Real-Time Code

During SW Coding, the proposed methodology permits to compile the pTPN model of the dynamic architecture of a CSCI into a skeleton of *control code*, i.e., the code that performs job releases, manages semaphore and priority handling operations, and invokes *functional code* represented by chunk entry-points. The control code conforms with pTPN semantics and can be implemented manually, following a programming discipline steered by the model structure which could be easily automated.

We address here translation of pTPN models into real-time code running on VxWorks 6.5 [55], which comprises a common platform for industrial applications. Each task of the timeline specification can be implemented as a real-time task with a priority and an associated entry-function. In particular, each periodic task is triggered by a periodic alarm and it is actually made recurrent through an explicit loop control structure programmed in its entry-function. At each iteration of the loop, the entry-function synchronizes on the alarm and performs a single job execution. In a similar manner, a loop control structure is also programmed in the entry-functions of sporadic and one-shot tasks. Specifically, at each loop repetition, the entry-function of a sporadic task synchronizes on an external signal, whereas the entry-function of a one-shot task synchronizes on an additional semaphore instrumental to one-shot activation. This semaphore is created by the `init` function and signaled by the activating task.

The architecture of the implementation is further extended to enable observation of possible re-entrant job releases, i.e., the situations in which a job is released before the previous one is completed. Therefore, releases of each task are performed by a single high-priority real-time task that spawns a separate task for each job execution. This keeps the Execution Time of each loop of the high-priority task sufficiently short to avoid the completion after the subsequent release. Though useful for testing purposes in early implementation stages, this solution is not suitable for deployment code, since the dynamic creation of tasks is deprecated by most regulatory standards for safety-critical SW, e.g., the Ravenscar profile [17].

In the industrial case study, the SC task-set was implemented as a kernel module of VxWorks 6.5 [55]. The `init` function of the kernel module creates semaphores *sem*1, *sem*2, and *sem*7 which are explicitly represented in the timeline schema of Fig. 5. It also invokes the primitive `sysClkRateSet` to set the period of the system clock equal to the minimum value that can be imposed on the Main Board, i.e., 1 *ms*. To obtain fine-grained time measurements, a hardware counter was used that attains 1 *ns* granularity.

D. HW-in-the-Loop Testing: Execution Time Profiling

During HW-in-the-loop Testing, the proposed methodology supports a disciplined and focused testing that uses the model as an oracle to reveal defects pertaining to concurrency control and timing. In particular, this enables verification of design assumptions about temporal parameters through profiling, with specific emphasis on Execution Times of implemented chunks and timings actually provided by the RTOS. Inconsistencies between assumptions and evidences can be managed through different approaches: by fixing implementation so as to fit specification assumptions; by repeating formal verification on a refined specification that accounts for actually observed parameter values; by providing a recommendation that draws attention on aspects of the implementation that may be not completely covered by formal verification.

The code of a CSCI can be instrumented so as to produce a time-stamped log of each event corresponding to each transition in the pTPN model of the task-set. The impact of logging on a real-time queue is evaluated by estimating its Execution Time through several repetitions of the operation. The operation of logging is conveniently allocated to the dynamic architecture in order to keep instrumentation code separate from functional code of chunk entry-points. This supports automation of the procedure of code generation, leaving the developer only the responsibility of implementing functional entry-points. At the end of each run of the implementation, the sequence of time-stamped logs is sent to the desktop machine for off-line analysis. Logs support reconstruction of the sequence of states visited during execution, evaluation of the sojourn time in each state, and identification of progressing/suspended transitions in each state. This permits to determine whether the execution log comprises a feasible behavior of the pTPN specification, enabling off-line derivation of the Execution Time of any event as the sum of sojourn times in the visited states where the corresponding transition is progressing. As a salient trait, measurements are carried out by letting chunk entry-points execute in interrupted mode, thus taking into account preemption events, HW/SW interrupts, pipeline and cache effects [54].

In the case of industrial application, we performed 10,000 repetitions of the logging operation and we measured the difference between subsequent logged time-stamps: 99.5% of the values fall in the range [0.00306, 0.00456] ms, with a mean value of 0.003282 ms and a standard deviation of 0.000165 ms; recurrent peaks in the interval [0.017, 0.022] ms occur in 0.5% of the cases and can be ascribed to timing uncertainties due to HW effects, which are usually in the order of a few tens of μs. Unfortunately, the time spent for logging turned out to be not negligible with respect to the granularity of temporal parameters of the task-set, which in fact are in the order of 0.005 ms to 40 ms. To circumvent overestimation of Execution Times, which may be caused by the logging overhead, firing intervals of temporal parameters were enlarged during iterative refinements of the dynamic architecture of the task-set.

In the industrial case study, the SC code was integrated with functional entry-points of its chunks and finally tested in a simulated environment, where selected HCIs/CSCIs of the system were emulated by a SW application running on a desktop processor connected to the Main Board through five serial buses. The first round of profiling detected an *un-sequenced execution* during which the high-priority task *Tsk*1 was overtaken by the low-priority task *Tsk*2. Inspection of functional code of the two tasks revealed that the failure was caused by a *task programming defect*, consisting of two chunks (i.e., chunk *C*21 of *Tsk*2 and chunk *C*13 of *Tsk*1) synchronizing on a semaphore that was not explicitly represented in the dynamic architecture. The inconsistency was fixed by adding a semaphore named *sem*3 to the SC task-set and by repeating formal verification.

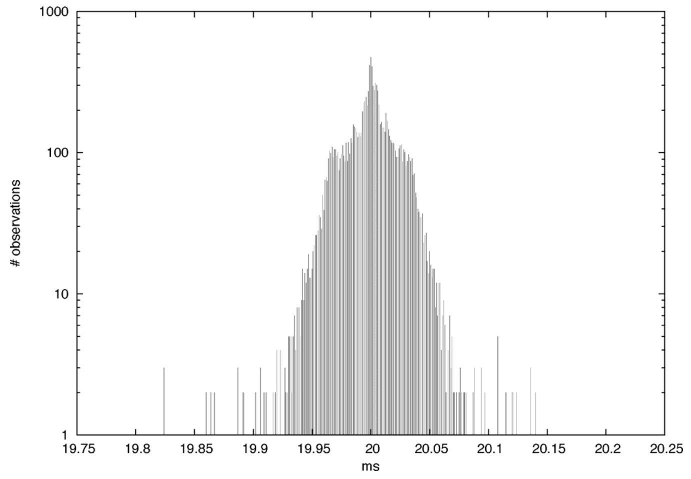

Fig. 7. Histogram of observed inter-release times of periodic release of *Tsk*2.

During subsequent executions, *time-frame violations* were detected on different chunks of different tasks. Optimization of chunk entry-points did not succeed in fixing the problem. Finally, we found out that the failure was due to a *cycle stealing* by a VxWorks task named tNetTask, which provides packet-processing network services and runs at priority level 50. The issue was circumvented by refining the model and repeating formal verification. In particular, the priority of SC tasks was raised from their initial values higher than 100, as usual for user tasks, to values lower than 50, as shown in the final specification of Fig. 6.

We measured inter-release times of task *Tsk*2 which is periodic with period of 20 ms. The n-th inter-release time δ_n is equal to

$$\delta_n = \big((1+n)\cdot T + \varepsilon_{n+1}\big) - (n\cdot T + \varepsilon_n) = T + \varepsilon_{n+1} - \varepsilon_n \quad (1)$$

where $T = 20$ ms is *Tsk*2 period and ε_n is the duration that elapses between the time $n \cdot T$ at which the nth task job should be released and the nth time-stamp. We can fairly assume that $\varepsilon_1, \ldots, \varepsilon_N$ are independent and identically distributed random variables. If $\varepsilon_1, \ldots, \varepsilon_N$ were uniformly distributed over an interval $[0, \gamma]$, then δ would be triangularly distributed over $[T - \gamma, T + \gamma]$. However, in the practice, they are not uniformly distributed due to processor, bus, and cache effects. The histogram of observed inter-release times plotted in Fig. 7 reveals that 98.9% of cases fall within [19.920, 20.069] ms with a peak on 20 ms, while the remaining 1.1% fall within [19.784, 19.920] ms and [20.069, 20.217] ms. Fixing the implementation so as to conform with the design assumption of period 20 ms was not a viable option, as release time jitters are dependent on the interaction between the RTOS and the Main Board. Repetition of the analysis on a refined model was not a convenient approach as well, since asynchronous releases largely increase the state space. Therefore, in this case, the most appropriate action seemed to be a warning to subsequent testing stages.

V. Conclusion

We have proposed a comprehensive methodology for integration of UML-MARTE and pTPNs within an industrial process of SW development. Experimentation in a one-year-long industrial project has proved feasibility and effectiveness of the approach, showing that the joint use of formal methods and advanced practices of SW engineering can largely help to afford the development of case studies of real complexity.

The structure of the SW life cycle addressed by the proposed approach is resumed in Fig. 1, with emphasis on development artifacts and iterations. UML-MARTE is conveniently used to manage the documentation process prescribed by MIL-STD-498, providing a semiformal specification that is sufficiently practical to fit the industrial practice and sufficiently structured to enable subsequent application of advanced formal methods. This provides an effective ground for deployment of pTPN theory, supporting the steps of design, implementation and verification. Integration of documentation and development processes in a unified methodology is achieved by providing support for translation of UML-MARTE diagrams into pTPN models, using timeline schemata as an intermediate artifact providing a synthetic and intuitive representation.

As recommended by main regulatory standards for safety-critical SW, the methodology achieves a limited impact on the mainstream practice. In fact, the difficulties in properly understanding and managing pTPN theory are largely mitigated not only by the adoption of UML-MARTE but also by the possibility to automate the overall approach, which would permit to maintain pTPNs completely transparent to the developer. Moreover, the use of timeline schemata also easies the effort on the part of the developer when practical factors of complexity make specification through UML-MARTE more difficult.

As a remarkable trait, the code of the implementation follows usual patterns of real-time concurrency, providing a clear structure which can be easily controlled and extended by the developer. The main requirement that the methodology brings along in the implementation stage is the necessity to keep functional and control code separate. This can be done at a reasonable cost and comprises one of the aspects that contribute most to code readability, which is essential to achieve industrial acceptance.

Acknowledgment

The authors would like to thank P. Viliani and S. Orsi of Selex Galileo, Florence, Italy, for guidance in understanding the practice of industrial development.

References

[1] R. Alur, I. Lee, and O. Sokolsky, "Compositional refinement for hierarchical hybrid systems," in *Hybrid Systems: Computation and Control*. Berlin, Germany: Springer-Verlag, 2001, vol. 2034, LNCS, pp. 33–48.

[2] S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni, "Model-based performance prediction in software development: A survey," *IEEE Trans. Softw. Eng.*, vol. 30, no. 5, pp. 295–310, May 2004.

[3] S. Balsamo, M. Marzolla, and R. Mirandola, "Efficient performance models in component-based software engineering," in *Proc. 32nd EUROMICRO Conf. SW Eng. Adv. Applications*, Washington, DC, 2006, pp. 64–71.

[4] K. Beck and W. Cunningham, "A laboratory for teaching object oriented thinking," *SIGPLAN Not.*, vol. 24, no. 10, pp. 1–6, 1989.

[5] S. Beckera, H. Koziolekb, and R. Reussner, "The Palladio component model for model-driven performance prediction," *J. Syst. SW*, vol. 82, pp. 3–22, 2009.

[6] S. Bernardi, J. Campos, and J. Merseguer, "Timing-failure risk assessment of UML design using Time Petri Net bound techniques," *IEEE Trans. Ind. Inf.*, vol. 7, no. 1, pp. 90–104, Feb. 2011.

[7] S. Bernardi, S. Donatelli, and J. Merseguer, "From UML sequence diagrams and statecharts to analysable Petri Net models," in *Proc. 3rd Int. Workshop Software Performance*, New York, 2002, pp. 35–45.

[8] S. Bernardi and J. Merseguer, "Performance evaluation of UML design with Stochastic Well-formed Nets," *J, Syst. Software*, vol. 80, pp. 1843–1865, Nov. 2007.

[9] S. Bernardi, J. Merseguer, and D. C. Petriu, "Dependability modeling and analysis of software systems specified with UML," *ACM Computing Survey*, accepted for publication.

[10] M. Bernardo, L. Donatiello, and P. Ciancarini, "Stochastic process algebra: From an algebraic formalism to an architectural description language," in *Performance Evaluation of Complex Systems: Techniques and Tools*. Berlin, Germany: Springer, 2002, vol. 2459, Lecture Notes in Computer Science, pp. 173–182.

[11] B. Berthomieu and M. Diaz, "Modeling and verification of time dependent systems using time Petri Nets," *IEEE Trans. Softw. Eng.*, vol. 17, no. 3, pp. 259–273, Mar. 1991.

[12] B. Berthomieu and M. Menasche, R. E. A. Mason, Ed., "An enumerative approach for analyzing time Petri Nets," in *Inf. Process.: Proc. IFIP Congress*, 1983, vol. 9, pp. 41–46.

[13] C. Bodenstein, F. Lohse, and A. Zimmermann, "Executable specifications for model-based development of automotive software," in *Proc. IEEE Int. Conf. Syst., Man Cybern.*, Oct. 2010, pp. 727–732.

[14] G. Bucci, L. Carnevali, L. Ridi, and E. Vicario, "Oris: A tool for modeling, verification and evaluation of real-time systems," *Int. J. SW Tools Technol. Transfer*, vol. 12, no. 5, pp. 391–403, 2010.

[15] G. Bucci, A. Fedeli, L. Sassoli, and E. Vicario, "Modeling flexible real time systems with preemptive time Petri Nets," in *Proc. 15th Euromicro Conf. Real-Time Syst.*, Jul. 2003, pp. 279–286.

[16] G. Bucci, A. Fedeli, L. Sassoli, and E. Vicario, "Timed state space analysis of real time preemptive systems," *IEEE Trans. Softw. Eng.*, vol. 30, no. 2, pp. 97–111, Feb. 2004.

[17] A. Burns, B. Dobbing, and T. Vardanega, "Guide on the use of the ADA Ravenscar profile in high integrity systems," *ADA Lett.*, vol. XXIV, no. 2, pp. 1–74, 2004.

[18] G. Buttazzo, *Hard Real-Time Computing Systems*. Berlin, Germany: Springer, 2005.

[19] "V-Model 97, Lifecycle Process Model-Developing Standard for IT Systems of the Federal Republic of Germany," BWB – Federal Office for Military Technology and Procurement of Germany, 1997, General Directive No. 250.

[20] L. Carnevali, L. Ridi, and E. Vicario, "Putting preemptive Time Petri Nets to work in a V-Model SW life cycle," *IEEE Trans. Softw. Eng.*, vol. 37, no. 6, pp. 826–844, Nov./Dec. 2011.

[21] F. Cassez and K. G. Larsen, in *The Impressive Power of Stopwatches*, Aug. 2000, vol. 1877, LNCS.

[22] "EN 50128 – Railway Applications: SW for Railway Control and Protection Systems," CENELEC, 1997.

[23] D. Compare, A. D'Onofrio, A. D. Marco, and P. Inverardi, "Automated performance validation of software design: An industrial experience," in *Proc. ASE*, 2004, pp. 298–301.

[24] V. Cortellessa and R. Mirandola, "PRIMA-UML: A performance validation incremental methodology on early UML diagrams," *Sci. Comput. Programming*, vol. 44, pp. 101–129, Jul. 2002.

[25] "RTAI: Real Time Application Interface for Linux," Dept. of Aerospace Eng. Polytechnic of Milan [Online]. Available: https://www.rtai.org

[26] S. Distefano, M. Scarpa, and A. Puliafito, "From UML to Petri Nets: The PCM-based methodology," *IEEE Trans. Softw. Eng.*, vol. 37, no. 1, pp. 65–79, Jan./Feb. 2011.

[27] *"MPC5200B Data Sheet,"* Freescale Semiconductor, 2010.

[28] V. Grassi and R. Mirandola, "PRIMAmob-UML: A methodology for performance analysis of mobile software architectures," in *Proc. 3rd Int. Workshop SW and Performance*, New York, 2002, pp. 262–274.

[29] V. Grassi, R. Mirandola, and A. Sabetta, "Filling the gap between design and performance/reliability models of component-based systems: A model driven approach," *J. Syst. SW*, vol. 80, pp. 528–558, 2007.

[30] T. A. Henzinger, B. Horowitz, and C. M. Kirsch, "Giotto: A time-triggered language for embedded programming," *Proc. IEEE*, vol. 91, no. 1, pp. 84–99, 2003.

[31] P. Jordan, "IEC 62304 international standard edition 1.0 Medical device software – Software life cycle processes," in *Proc. IET Seminar Softw. Medical Devices*, 2006, pp. 41–47.

[32] "Palladio: A Software Architecture Simulation Approach," Karlsruhe Inst. Technol., FZI Res. Ctr. Inf. Technol, University of Paderborn [Online]. Available: http://www.palladio-simulator.com

[33] G. Karsai, J. Sztipanovits, A. Ledeczi, and T. Bapty, "Model-integrated development of embedded software," *Proc. IEEE*, vol. 91, no. 1, pp. 145–164, Jan. 2003.

[34] H. Koziolek and J. Happe, "A QoS driven development process model for component-based software systems," in *Proc. 9th Int. Symp. Component-Based SW Eng.*, Berlin, Germany, 2006, vol. 4063, LNCS, pp. 336–343.

[35] D. Lime and O. H. Roux, "Formal verification of real-time systems with preemptive scheduling," *Real-Time Syst.*, vol. 41, no. 2, pp. 118–151, 2009.

[36] M. Marzolla and S. Balsamo, "UML-PSI: The UML performance SImulator," in *Proc. 1st Int. Conf. Quantitative Evaluation Syst.*, Enschede, The Netherlands, Sep. 2004, pp. 340–341.

[37] P. Merlin and D. Farber, "Recoverability of communication protocols," *IEEE Trans. Commun.*, vol. COM-24, no. 9, pp. 1036–1043, Sep. 1976.

[38] J. Merseguer, J. Campos, S. Bernardi, and S. Donatelli, "A compositional semantics for UML state machines aimed at performance evaluation," in *Proc. 6th Int. Workshop Discrete Event Syst.*, Washington, DC, 2002, pp. 295–302.

[39] "UML Profile for Schedulability, Performance and Time Specification," Object Management Group, 2005.

[40] "UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems v1.0," Object Management Group, 2009.

[41] *"Unified Modeling Language: Infrastructure,"* ver. 2.3, Object Management Group, 2009, 2.3 ed. .

[42] *"Unified Modeling Language: Superstructure,"* ver. 2.3, Object Management Group, 2009.

[43] H. R. Olivier and L. Didier, "Time Petri Nets with inhibitor hyperarcs: Formal semantics and state-space computation," in *25th Int. Conf. on Theory and Application of Petri Nets*, 2004, vol. 3099, pp. 371–390.

[44] D. Petriu, C. Shousha, and A. Jalnapurkar, "Architecture-based performance analysis applied to a telecommunication system," *IEEE Trans. Softw. Eng.*, vol. 26, no. 11, pp. 1049–1065, Nov. 2000.

[45] D. B. Petriu and M. Woodside, "An intermediate metamodel with scenarios and resources for generating performance models from UML designs," *SW Syst. Modeling*, vol. 6, pp. 163–184, 2007.

[46] "DO-178B, Software Considerations in Airborne Systems and Equipment Certification," Radio Technical Commission for Aeronautics, 1992.

[47] "ECSS-E-ST-40C Space Engineering Software," Requirements & Standards Division, ESA-ESTEC, ECSS Secretariat, 2009.

[48] D. C. Schmidt, "Model-driven engineering," *IEEE Computer*, pp. 1–2, Feb. 2006.

[49] L. Sha, R. Rajkumar, and J. P. Lehoczky, "Priority inheritance protocols: An approach to real-time synchronization," *IEEE Trans. Comput.*, vol. 39, no. 9, pp. 1175–1185, Sep. 1990.

[50] C. U. Smith, C. M. Lladó, V. Cortellessa, A. D. Marco, and L. G. Williams, "From UML models to software performance results: An SPE process based on XML interchange formats," in *Proc. 5th Int. Workshop SW and Performance*, 2005, pp. 87–98.

[51] *"Simulink,"* The Mathworks [Online]. Available: www.mathworks.com/products/simulink

[52] "Military Standard for Software Development and Documentation," USDoD, MIL-STD-498, 1994, .

[53] E. Vicario, "Static analysis and dynamic steering of time dependent systems using time Petri Nets," *IEEE Trans. Softw. Eng.*, vol. 27, no. 8, pp. 728–748, Aug. 2001.

[54] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Statshulat, and P. Stenstroem, "Priority inheritance protocols: The worst case execution-time problem: Overview of methods and survey of tools," *ACM Trans. Emb. Comp. Sys.*, vol. 7, no. 3, pp. 1–53, 2008.

[55] *"VxWorks,"* Wind River [Online]. Available: www.windriver.com/products/vxworks

[56] M. Woodside, D. Petriu, D. Petriu, H. Shen, T. Israr, and J. Merseguer, "Performance by unified model analysis (PUMA)," in *Proc. 5th Int. Workshop SW and Performance*, Jul. 2005, pp. 1–12.

Irene Bicchierai received the B.S. and M.S. degrees in informatics engineering from the University of Florence, Florence, Italy, in 2005 and 2008, respectively, where she is currently working toward the Ph.D. degree in informatics, multimedia, and telecommunications engineering.

Her research interests focus on methods for design and testing of real-time embedded systems. Her activity mainly addresses the integration of formal methods in the development life cycle of safety critical software.

Giacomo Bucci (M'94) received the degree in electrical engineering from the University of Bologna, Bologna, Italy, in 1968.

From 1970 to 1982, he was with the University of Bologna, Bologna, Italy. During 1975, he was a Visiting Researcher with IBM T. J. Watson Research Center, Yorktown Heights, NY. Since 1986, he has been a Full Professor with the Faculty of Engineering, University of Florence, Florence, Italy, where he teaches a course in computer architectures and a course in software engineering. He has held several academic positions. Currently, he coordinates the doctoral program in Informatics, Systems and Telecommunications at the University of Florence. His current research interests include computer architecture, real-time systems, distributed systems, software development methodologies, and computer performance evaluation.

Laura Carnevali received the B.S. and M.S. degrees in informatics engineering and the Ph.D. degree in informatics, multimedia, and telecommunications engineering from the University of Florence, Florence, Italy, in 2004, 2006, and 2010, respectively.

She is currently a Postdoctoral Fellow with the Department of Systems and Informatics, University of Florence, Florence, Italy. Her research is focused on correctness verification and performance evaluation of real-time systems, with specific interest on integration of formal methods in the development life cycle of real-time software and stochastic characterization of timed models.

Enrico Vicario (M'95) received the Ph.D. degree in electronics engineering and Ph.D. degree in informatics and telecommunications engineering from the University of Florence, Florence, Italy, in 1990 and 1994, respectively.

He is a Full Professor of Computer Science with the School of Engineering, University of Florence, Florence, Italy. His research is presently focused on formal methods for model driven development, correctness verification of real-time systems, and quantitative evaluation of concurrent non-Markovian models.