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 UML (Unified Modeling Language) profile for MARTE (Modeling and Analysis of Real-Time and Embedded 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.


I. INTRODUCTION

In several application domains, the development of safety-critical SW is subject to certification standards such as RTCA/DO-178B [46], 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 [42], [41] 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 [38], [7], [8], [26], [6]. 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 a SW architecture is evaluated through a two-phase methodology which first annotates a UML specification with tags and stereotypes of the UML profile for SPT (Schedulability, Performance, and Time Specification) [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 MARTE (Modeling and Analysis of Real-Time and Embedded systems) [40], which is specifically targeted to capture non-functional properties of real-time embedded systems. The resulting TPN model 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 Algebras (PA) [44], [24], [28], [36], [3], [50], [23], [45], [56], [29]. 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 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 AEmilia [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 an UML diagram annotated with UML-SPT stereotypes. The approach is implemented in the PUMA (Performance by Unified Model Analysis) tool architecture [56], which provides a unified interface between SW design models and performance models. An intermediate metamodel 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 [51], [1], [30], [13], [34], [20], 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 metamodel 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 Time Petri Nets (pTPNs) [16], [15] in a tailoring of the V-Model SW life cycle [19], enabling automated derivation of pTPN models from a semi-formal 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 rest of the paper is organized as follows. In Sect. II, we present an industrial tailoring of the V-Model framework (Sect. II-A) and we illustrate the experimented methodology (Sect. II-B). In Sect. III: we show how UML-MARTE has been conveniently introduced to manage the documentation process prescribed by MIL-STD-498 (Sects. III-A and III-C); we report on a notation similar to Class Responsibility Cards (CRC) employed to document analysis of SW requirements (Sect. 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 (Sect. III-D). In Sect. 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 (Sect. IV-A); we discuss automated verification of sequencing and timing constraints (Sect. IV-B); we provide guidance for disciplined implementation of real-time code that conforms with pTPN semantics and runs on VxWorks 6.5 [55] (Sect. IV-C); and, we report on profiling of temporal parameters of the model (Sect. IV-D). Conclusions are finally drawn in Sect. 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. An 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 FCLs (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 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 FCIIs 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. A 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 non-deterministic 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, 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 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 semi-formal specification that is practical enough 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

In this Section, 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.

A. System/Subsystem Analysis and Design

During System/Subsystem Analysis and Design, definition of User Requirements enables identification of system functionalities and their allocation to system units. 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. 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.

A UML-MARTE Component Diagram can effectively capture system decomposition, as exemplified in Fig. 2: SMU and EU are modeled through the ProcessingResource stereotype (i.e., a resource allocated to the execution of schedulable resources); OU is represented through the DeviceResource stereotype (i.e., an external device that may be manipulated through specific services); and, the two communication channels connecting EU with SMU and OU are specified through the CommunicationMedia stereotype (i.e., a mean to transport information from one location to another).

At this stage, functionalities of each system unit are allocated to CSCIs, HCIs, and FCIs. With regard to the industrial experience, 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 is responsible for controlling devices of OU, sending them commands elaborated by SMU and forwarding their replies to SMU. 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); IT is allocated to Video Processor (VP), which runs a proprietary commercial RTOS. EU also embeds a battery. 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).

Also the aspect of unit decomposition can be effectively represented through a UML-MARTE Component Diagram, as illustrated in Fig. 3. A CSCI is represented by the rtiUnit stereotype (i.e., a real-time application that owns one or more schedulable resources); a real-time task-set is modeled by the SchedulableResource stereotype (i.e., an active resource that performs actions using the processing capacity brought from a processing resource managed by a scheduler); an HCI is specified through the HwCard stereotype (i.e., a printed circuit board typically made by chips and electrical devices); and, a battery is represented by the HwPowerSupply stereotype (i.e., a hardware component supplying the hardware platform with power). Association of a CSCI with a real-time task-set is modeled by the allocate stereotype (i.e., an allocation relation between elements of the application model and those of the execution platform, represented by the ApplicationAllocationEnd and ExecutionPlatformAllocationEnd stereotypes, respectively, denoted by ap_allocated and ep_allocated for short, respectively). The allocate stereotype is also used to represent allocation of a real-time task-set to an HCI.

An UML-MARTE Component Diagram also permits to make explicit relationships between HCIs, CSCIs, and FCIs, as exemplified in Fig. 3. Specifically, MB and VP are connected through bus B7, specified by the HwBus stereotype (i.e., a particular wired channel with specific functional properties). MB is also connected to SMU through bus B6 and to HCIs of OU through buses B1, B2, B3, B4, and B5. Note that every bus is bidirectional and is represented by a provided and a required interface in the diagram of Fig. 3.

B. 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 HCl/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_Communication up to SM_Communication manage buses connecting SC with: IT (B7), SMU (B6), and four HCIs of OU, i.e., LS (B1), IRC (B5), TVC (B3), and SM (B4); LS_IRC_Power and TVC_Configuration control LS, IRC, and TVC sensors; SMU_OU_Commands manages communication between SMU and OU and, since it requires the most computational effort, it is decomposed in sub-capabilities, shown in Table II. Communication with the other two HCIs of OU (i.e., OS and TS) is performed through bus B2 which is directly managed by MB (i.e., the HCI which SC is allocated to).

C. 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 predefined structure.

- A task may be either recurrent or one-shot. A recurrent task is an infinite sequence of identical activities called jobs activated with periodic or sporadic policy (i.e., with release time deterministic or bounded by a minimum value, respectively), with deadline less or equal to the minimum inter-release time. 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.
- A job is a sequence of chunks running under static priority preemptive scheduling; each chunk requires one or more resources with a priority level (low priority numbers run first) and has an associated entry-point method implementing its functional behavior with a non-deterministic range of Execution Time.
- Chunks belonging to jobs of different tasks may share resources (e.g., memory space) and synchronize on semaphores under priority ceiling emulation [49], i.e., raising their priority to the highest priority of any chunk that ever uses that semaphore. Wait and signal semaphore operations are constrained to occur at the beginning and at the end of chunks, respectively.

The model of a task-set can be conveniently documented through a UML-MARTE Class Diagram, as illustrated in
Fig. 3. System/Subsystem Analysis and Design: SSDD document. UML-MARTE Component Diagram of system units.
Fig. 4. 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. According to the proposed methodology, this proceeds through three steps: i) identification of structural relations among tasks through a UML-MARTE Component Diagram; ii) definition of non-functional requirements through a UML-MARTE Object Diagram; and, iii) specification of functional requirements through UML-MARTE Activity Diagrams.

1) Specification of inter-task relations: A UML-MARTE Component Diagram can be conveniently used to represent relations among tasks of a CSCI as well as relations between a task of a CSCI and the collaborating HCI/CSCI/Unit, as illustrated in Fig. 5 with reference to the SC task-set of the industrial case study. Specifically, the six capabilities managing buses that connect SC with IT, SMU, and OU (in Table II, Communication capabilities) are allocated to separate tasks named $Tsk_2$, $Tsk_3$, ..., and $Tsk_7$; the remaining four capabilities (in Table II, Init, LS-IRC_Power, TVC_Configuration, and SMU-OU_Commands) 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 $Tsk_3$. $Tsk_1$ is responsible for processing SMU commands, producing data for the addressed HCIs/CSCIs, and writing them on shared memories. These data are then sent to HCIs/CSCIs by the associated tasks, which are also responsible for writing back the replies. Finally, $Tsk_1$ forwards replies to SMU via $Tsk_3$. Communications through buses are described by the HwBus stereotype, while shared memories are represented by the SharedDataComResource stereotype (i.e., a specific resource used to share the same area of memory among concurrent resources).

2) 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. 6 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

| **TABLE I** |
| **SW Requirements Analysis: SRS Document. CRC Card of System Control CSCI (SC).** |
| **CAPABILITY** | **DESCRIPTION** | **COLLABORATION** |
| Init | HW and SW initialization | - |
| TVC_Communication | Communication with TVC | TVC |
| SMU_Communication | Communication with SMU | SMU |
| LS_Communication | Communication with LS | LS |
| IRC_Communication | Communication with IRC | IRC |
| TVC_Configuration | Communication with TVC | TVC |
| SM_Communication | Communication with SM | SM |
| LS-IRC_Power | Switching on/off LS and IRC | LS and IRC |
| TVC_Configuration | Management of the TVC configuration | TVC |
| SMU-OU_Commands | Management of messages exchanged by SMU and OU | - |

| **TABLE II** |
| **SW Requirements Analysis: SRS Document. Decomposition of SMU-OU_Commands in sub-capabilities.** |
| **SUB-CAPABILITY** | **DESCRIPTION** | **COLLABORATION** |
| SMU_Commands | Management of SMU commands and OU replies | - |
| TVC_Commands | Management of the TVC configuration parameters | - |
| HCl_Tranmission | Activation of data transmission to IRC, TVC, and LS | - |
| LS-IRC_State | Management of the switched on/off state of IRC and LS | - |
| IHCs_Data | Processing of HCl data | - |
| SM_LocationData | Processing of the SM location data elaborated by IT | - |
| OperationModes | Management of system Operation Modes | - |

Copyright (c) 2011 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.
with deadline of 10 ms. \textit{Tsk1} requires \textit{cpu} with priority level 1; the other tasks require \textit{cpu} with priority level 2. Specifically, minimum inter-release times and deadlines of \textit{Tsk3} and \textit{Tsk4} directly come from User Requirements constraining timeliness of image processing and system management; \textit{Tsk5}, \textit{Tsk6}, and \textit{Tsk7} are modeled as one-shot tasks since communication with IRC, TVC, and LS is activated on demand by \textit{Tsk1}, depending on the HCI/CSCI addressed by the current SMU command; \textit{Tsk1} and \textit{Tsk2} are modeled as periodic tasks to guarantee recurrent control on servo-motors and SMU-OU messages; \textit{Tsk2} period and deadline are chosen equal to \textit{Tsk3} deadline so as to timely actuate SMU commands addressing servo-motors; \textit{Tsk1} period and deadline are selected equal to half \textit{Tsk3} 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. 6 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, \textit{Tsk1} is made of twenty-two chunks, which result from the four capabilities assigned to SC, the sub-capabilities of \textit{SMU-OU\_Commands}, and some branches introduced during refinement of entry-points: chunk \textit{C11} executes entry-point \textit{f11} with an Execution Time constrained within $[0.005, 0.100]$ ms, and it is synchronized with chunk \textit{C23} on semaphore \textit{sem1} to access data that \textit{Tsk1} shares with \textit{Tsk2}.

3) 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 task 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. 7 illustrates the concept with reference to task \textit{Tsk1} of the industrial case study. The task is periodically activated every 10 ms; after activation, it performs the sequence of chunks \textit{C11}, \textit{C12}, \textit{C13}, and \textit{C14}, synchronizing on semaphores \textit{sem1}, \textit{sem2}, \textit{sem3}, and \textit{sem4}, respectively, to access shared memories. Afterwards, different paths are followed depending on \textit{OU} and \textit{EU} configuration parameters.

For reasons of space, UML-MARTE Activity Diagrams of \textit{Tsk2} through \textit{Tsk7}, each composed of a task swimlane, are not shown here.

D. SW Design: semi-formal specification through timelines

In practical applications, UML-MARTE diagrams often tend to explode in complexity, as illustrated by Figs. 6 and 7. 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 a semi-formal specification suitable for SW documentation and a formal specification supporting correctness verification through analysis. In this perspective, the proposed approach exposes similarities with [29], [26], [45], where intermediate artifacts are used as an interface between design-oriented and analysis-oriented models. In the formalism of timelines:

- a task is specified by an open box containing its chunk sequence and decorated with its name and release interval; a down-headed arrow specifies its deadline; and, a double-headed or a single-headed arrow indicates whether the task is recurrent or one-shot, respectively;
- a chunk is specified by a rectangle annotated with its name, Execution Time interval, required resources (e.g. \textit{cpu}), priority level, and entry-point;
- branches and re-joins in a sequence of chunks are specified using diamonds;
- activations of a one-shot task in reaction to an event thrown by a chunk of a different task are specified by a dotted-arrow from the chunk to the activated task;
- binary semaphore operations are specified by decorating chunks with circles annotated with a sequence of operations, each referred to a semaphore name.

Fig. 8 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. 8 replaces the Class Diagram of Fig. 4, the Object Diagram of Fig. 6, the Activity Diagram of Fig. 7, and the remaining Activity Diagrams of \textit{Tsk2} through \textit{Tsk7} (not reported here).

IV. SUPPORTING THE DEVELOPMENT PROCESS THROUGH pTPNs

In this Section, 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 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 [16], [15] extends the model of TPNs [37], [11], [53] with a set of preemtable resources whose availability conditions the progress of timed transitions. According to this, each transition is associated with a firing interval, delimited by a static \textit{Earliest Firing Time} (EFT) and a static \textit{Latest Firing Time}, and may request a set of resources with a priority level. A transition is \textit{enabled} if all its input places contain at least one token: in this case, it is associated with a dynamic \textit{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. For the sake of readability, the process is illustrated by referring to the SC task-set of the industrial case study, discussing derivation of the pTPN shown in Fig. 9 from the timeline depicted in Fig. 8.

Periodic releases of $\text{Tsk}_1$ and $\text{Tsk}_2$ are modeled by transitions with neither input places nor resource requests; therefore, they fire repeatedly with inter-firing times falling within their static firing intervals. According to this, $t_{10}$ and $t_{20}$ model releases of $\text{Tsk}_1$ and $\text{Tsk}_2$, respectively. $\text{Tsk}_3$ and $\text{Tsk}_4$ are specified as sporadic, since they are both activated by an external event not under scheduler control. However, since SW requirements prescribe the activating events to be periodic, then releases of $\text{Tsk}_3$ and $\text{Tsk}_4$ are accounted by transitions $t_{30}$ and $t_{40}$, respectively, with deterministic firing interval of $[20, 20]$ and $[40, 40] \text{ ms}$, respectively. One-shot tasks $\text{Tsk}_5$, $\text{Tsk}_6$, and $\text{Tsk}_7$ are activated in a mutually exclusive manner, thus they are mapped on the same pTPN representation. Their releases are represented by transition $t_{50}$, preconditioned by the output place of the transitions that model the completion of the activating chunks. Its firing interval accounts for the time spent in the elaboration of the activating signal.

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 $\text{C}_{11}$ of $\text{Tsk}_1$. Branches are modeled by immediate transitions preconditioned by the output place of the preceding chunk; conversely, rejoin transition is accounted for making the chunks share the same output place. For instance, transitions $t_{113}$, $t_{146}$, and $t_{157}$ are preconditioned by place $p_{113}$ to represent a branch among the mutually exclusive chunks $\text{C}_{15}$, $\text{C}_{16}$, and $\text{C}_{17}$ of $\text{Tsk}_1$; conversely, transitions $t_{123}$, $t_{128}$, $t_{131}$, $t_{138}$, $t_{145}$, $t_{152}$, $t_{156}$, and $t_{158}$ share the output place $p_{152}$ to account for a rejoin after the completion of the mutually exclusive chunks $\text{C}_{17}$, $\text{C}_{110}$, $\text{C}_{111}$, $\text{C}_{112}$, $\text{C}_{113}$, $\text{C}_{114}$, $\text{C}_{115}$, $\text{C}_{116}$, $\text{C}_{118}$, $\text{C}_{119}$, $\text{C}_{120}$, and $\text{C}_{121}$ of $\text{Tsk}_1$.

According to the priority ceiling emulation protocol [49], low-priority tasks $\text{Tsk}_2$, $\text{Tsk}_3$, ..., and $\text{Tsk}_7$ undergo a priority boost and synchronize on a semaphore in the sections where they access memories shared with the high-priority task $\text{Tsk}_1$. 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 nonpointlike firing interval. For instance, $\text{sem}1$ represents a binary semaphore synchronizing the access to a memory shared between chunks $\text{C}_{11}$ and $\text{C}_{23}$; $t_{27}$ models a priority boost; $t_{11}$ and $t_{28}$ account for $\text{sem}1$ wait operations; $t_{12}$ and $t_{29}$ represent the completion of $\text{C}_{11}$ and $\text{C}_{23}$, respectively; $t_{13}$ and $t_{210}$ model $\text{sem}1$ signal operations; $t_{211}$ accounts for a priority deboost. This differs from [20], where priority boost and semaphore wait operations are represented by immediate transitions, while priority deboost and semaphore 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 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 [16], [15]. 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 ﬁring. 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. 6 and 8. 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. 8. 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. The sequence and the histogram of observed Execution Times are reported in Figs. 10 and 11, respectively. They show that: 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 HClIs/CSCIIs of the system (in Fig. 3, SM, IRC, TVC, LS, and IT) 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 Tsk1 was overtaken by the low-priority task Tsk2. 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 C21 of Tsk2 and chunk C13 of Tsk1) synchronizing on a semaphore that was not explicitly represented in the dynamic architecture. The inconsistency was fixed by adding a semaphore named sem3 to the SC task-set and by repeating formal verification.

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. 9.

We measured inter-release times of periodic task Tsk2, which correspond to inter-firing times of transition t20 in the pTPN of Fig. 9. The n-th inter-release time δn is equal to:

\[ \delta_n = (1 + n) \cdot T + \varepsilon_{n+1} - (n \cdot T + \varepsilon_n) = T + \varepsilon_{n+1} - \varepsilon_n, \]

where \( T = 20 \) ms is Tsk2 period and \( \varepsilon_n \) is the duration that elapses between the time \( n \cdot T \) at which the n-th task job should be released and the n-th time-stamp. We can fairly assume that \( \varepsilon_1, ..., \varepsilon_N \) are independent and identically distributed random variables. If \( \varepsilon_1, ..., \varepsilon_N \) were uniformly distributed over an interval \([0, \gamma]\), then \( \delta \) 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. 12 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, being release time jitters 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.

![Histogram of observed inter-release times of periodic release of Tsk2.](image)

Fig. 12. Histogram of observed inter-release times of periodic release of Tsk2.

Fig. 13 shows the histogram of measured Execution Times of the wait operation performed by Tsk2 on semaphore sem3, which corresponds to transition t22 with firing interval \([0, 0.05]\) ms in the pTPN of Fig. 9. Observed Execution Times fall in the interval \([0.006, 0.050]\) ms, included in the interval \([0, 0.05]\) ms, with a peak on 0.012 ms. As already remarked in Sect. IV-A, the time spent for semaphore operations on the RTOS in use here is not negligible with respect to the order of the Execution Time of entry-points.

![Histogram of observed Execution Times of the wait operation performed by Tsk2 on semaphore sem3.](image)

Fig. 13. Histogram of observed Execution Times of the wait operation performed by Tsk2 on semaphore sem3.

We finally measured Execution Times of entry-point f1.22
of chunk \( C_{122} \) of \( T_{sk1} \), which corresponds to transition \( t_{159} \) with firing interval \([0.05, 0.2] \) ms in the pTPN of Fig. 9. The histogram of observed values plotted in Fig. 14 has a thin spectrum within \([0.080, 0.183] \) ms, contained in the interval \([0.05, 0.2] \) ms, with a peak on 0.089 ms. This actually reflects the absence of data-dependent alternatives in the implementation of the entry-point.

![Histogram of observed Execution Times of entry-point \( \sum f_{122} \) of \( T_{sk1} \).](image)

**V. CONCLUSIONS**

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. 15, with emphasis on development artifacts and iterations. UML-MARTE is conveniently used to manage the documentation process prescribed by MIL-STD-498, providing a semi-formal specification that is practical enough 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 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 eases 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.

**ACKNOWLEDGMENTS**

This work was supported by the SW-Intiative of the FinMecanica Group. The authors would like to thank Paolo Viliani and Simonetta Orsi of Selex Galileo - Firenze for guidance in understanding the practice of industrial development.

**REFERENCES**


Fig. 4. SW Design: SDD document. UML-MARTE Class Diagram of the task-set model.
Fig. 5. SW Design: SDD document. UML-MARTE component diagram of the SC task-set.
Fig. 6. 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., Task1 is made of 22 chunks from C11 up to C122.
Fig. 7. SW Design: SDD document. UML-MARTE Activity Diagram of task \texttt{Tsk1} of SC (times expressed in ms). For the sake of readability, only a few UML-MARTE stereotypes, activity names, shared memories, and guard conditions are shown (up to the first branch).
Fig. 8. SW Design: SDD document. The timeline schema of the SC task-set (times expressed in ms).
Fig. 9. SW Design: SDD document. The pTPN model of the dynamic architecture of the SC (times expressed in ms). To reduce the cluttering, the figure does not show the names of places and transitions that are not mentioned in the text.
Fig. 10. Sequence of observed Execution Times of the logging operation.

Fig. 11. Histogram of observed Execution Times of the logging operation.

Fig. 15. A scheme of the V-Model life cycle [19] tailored according to MIL-STD-498 [52]. With respect to Fig. 1, 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).