Formal Synthesis of Real-Time System Models in a MDE Approach

Cédrick Lelionnais, Jérôme Delatour, Matthias Brun, Olivier Henri Roux, Charlotte Seidner

Formal Synthesis of Real-Time System Models in a MDE Approach

Cédrick Lelionnais,
Jérôme Delatour,
and Matthias Brun

ESEO-TRAME
Angers, FRANCE
Email: cedrick.lelionnais@eseo.fr
Email: jerome.delatour@eseo.fr
Email: matthias.brun@eseo.fr

Olivier H. Roux
and Charlotte Seidner

IRCCyN - Université de Nantes
École Centrale de Nantes
Nantes, FRANCE
Email: olivier-h.roux@irccyn.ec-nantes.fr
Email: charlotte.seidner@irccyn.ec-nantes.fr

***Abstract*—The development of real-time embedded systems is quite complex because of the wide range of execution platforms and of the importance of non-functional requirements. Furthermore, Model Driven Engineering is particularly suitable for handling the diversity of implementation targets. Therefore, several real-time embedded systems development suites leverage Model Driven Engineering by automatically generating platform-specific code from high-level design models. Such tools may also take non-functional requirements into account by integrating verification activities. These activities typically rely on the generation of formal models from the same high-level design descriptions used for code generation. However, few tool suites support both code and formal model generation. Furthermore, among these, most overlook real-time operating systems mechanisms. Therefore, both code and formal models generated by these tool suites may not behave as specified in the high-level design descriptions. The present work extends the SExPIsTools code generator tool suite with a support for the generation of formal models. The proposed strategy relies on the composition of formal model fragments described using an extension of the classical Time Petri Nets. This paper presents a formalization of this composition that generically considers the behavior of platforms. As an illustration, we then give the formal model describing the behavior of an application on two different platforms (OSEK/VDX and VxWorks) and check a safety property on both models.**

***Keywords*—Real-time operating systems, Model Driven Engineering, Time Petri Nets, Multi-platform deployment, Formal model.**

I. Introduction

Real-Time Embedded Systems (RTES) increasingly surround us in various domains (aircrafts, cars, cell phones, robotics, etc.). RTES engineers are confronted with the challenge of developing more complex, higher quality systems, with shorter development cycles at lower costs. Model Driven Engineering (MDE) helps engineers to develop tool suites that partially automate the development of RTES. Using model transformations, these tool suites mainly produce either executable code or formal models from high-level design descriptions of RTES.

Some of these tool suites have both code and formal models generation processes. However, the mechanisms of Real-Time Operating Systems (i.e., executable software platforms supporting real-time applications, RTOS) are often ignored by these generation processes. As a result, generated code and generated formal models may not behave as specified in the high-level design description. Consequently, verification and validation activities applied on the RTES development could provide erroneous results. For instance, the detection of malfunctions (e.g., wrong treatments of critical data, or bad scheduling of real-time multitasking applications) could be compromised.

Nevertheless, among these tool suites, some consider real-time aspects in their generation processes. They thus take the deployment of real-time applications on RTOS (i.e., mapping of application concepts to execution platform services in order to execute them) into account. However, none of them satisfies the four criteria given below:

- **Portability** of real-time applications to adapt to the RTOS heterogeneity;
- **Reusability** of generation processes for a rapid migration of these applications in a multi-platform deployment case;
- **Maintainability** of RTES to help all stakeholders in their interventions;
- **Correctness** of generation processes in ordrer to have confidence in RTES development.

This work is part of an overall strategy of RTES development using a MDE approach. This strategy is supported by a tool suite called SExPIsTools (for Software Execution Platform Inside Tools). In order to satisfy the criteria previously given, SExPIsTools relies on the following approach:

- **considering any RTOS** as parameter of generation processes to achieve multi-platform deployment;
- **writing more generic transformation rules** to be independent from the considered RTOS;
- **separating domain concerns** (i.e., application deployment choice, RTOS consideration, transformation rules and verification and validation activities) to clarify interventions of each domain specialist;
- **formalizing transformation rules** to increase the correctness of generation processes.

In the present paper, we focus on the latter point. We need to construct RTES formal models, i.e., models of the whole system including the RTOS. For this purpose, our approach relies on a single transformation that does not depend on any specific RTOS. This transformation composes multiple formal model fragments independently of the target RTOS. Each of these fragments represents a part of the formal model that captures the behavior of the RTES.

As a basis of the construction, we use roles to generically identify connection points. These roles are used as a glue between the formal fragments. As a consequence, a new definition is presented to represent these formal fragments based on roles. The class of models we use is Time Petri Nets (TPN). The generic construction is then formalized as a basis of the transformation rules. Finally, an application example is proposed to illustrate a multi-platform deployment case, where two RTOS are considered. As a result of this experimentation, a scheduling safety property is verified on both resulting models.

This paper extends our previous work [1] in which the fundamental rules of composition have been presented. An extension of these rules is given here to both 1) compose formal models of multitasking deployed applications with priority policy, and 2) provide a first validation of the generic construction.

Section II presents multi-platform deployment related works within a MDE approach [2]. A description of SEx-PIsTools is then given in Section III. Section IV gives the formal definition of the TPN fragments composition operator, which is based on roles. The application of this operator to the construction of whole RTES formal models is then described in Section V. Application examples are presented in Section VI. The benefits and limits of this approach are discussed in Section VII. Finally, we conclude in Section VIII.

II. Related Works

The following sub-sections present existing tool suites related to the multi-platform deployment problem.

Firstly, some code generators are introduced. Formal model generation tool suites are then presented, some of which are also capable of generating code. Finally, we will take a stand on the adopted approach.

A. Code generator tool suites

In order to promote the reuse of deployment tools within a single code generator, the genericity of processes has been at the heart of concerns. For instance, TransPI [3] relies on a two-step approach to generate specific code. A first phase considers a generic behavioral representation of the RTOS API (Application Programming Interface) in accordance with POSIX standard. In a second phase, the deployment is refined by configuring the process rules with the API of the targeted RTOS. However, the configuration of new rules does not fully satisfy the reuse of such a process since the tool must be modified.

A similar experimentation [4] improves reuse by specifying the RTOS concerned by the deployment without modification of the process. This orientation has been thought with the aim of porting real-time applications. This strategy relies on the transcription of code snippets by configuration of functions with RTOS information. Those information come from components of the targeted RTOS whose architecture was previously modeled by the generic modeling language AADL, which is dedicated to the real-time domain. The flexibility of this process meets the heterogeneous requirements of platforms.

Another approach [5] also contributes to the multi-platform deployment problem. This fills the behavioral gap in the SRM package of the MARTE UML profile [6]. Before dealing with the RTOS behavior, a transformation process was developped [7] to generate a deployed application model by considering the targeted RTOS structure described with SRM. Descriptions of executable concepts (i.e., resources and services of the API) of the targeted platform required by the application are instantiated through the deployment process. This instantiation is completed by a refinement of descriptions depending on both application data and location of elements playing a generic role (e.g., a task priority or a counting semaphore capacity) within the considered platform. The integration of the behavioral aspect is also based on this notion of role. Code snippets are assigned to execution services (e.g., a task creation or a semaphore taking) in accordance with Java or C++/POSIX implementations.

The main interest of the two last contributions is the independence of specialists during their interventions regarding the tool suite. This criterion guarantees the quality of maintainability, which is added to the already mentioned reusability and portability. Despite this, these contributions present a major drawback that is inherent to all code generators. The formalization is indeed absent from the addressed processes. This weakness prevents specialists from applying verification activities on deployed applications.

B. Formal model generation tool suites

The tool suites presented in this section encourage the behavioral formalization of RTOS.

1) Without code generator: An approach [8] has recently launched a formal synthesis for composing behavioral models of RTES. In order to achieve this, both application and platform are modeled with adequate modeling language. With the help of an Algebra of Communicating and Shared Resources (ACSR), behavior of the targeted platform is formalized. Behavior of the application is described by a Timed and Resource-oriented Statechart (TRoS) including both time annotations and resource constraints. A model of the deployed application is then composed with the obtained formal models and used for analysis. This synthesis provides a detailed design of RTES by formalizing their implementation with a complementary manner. Unfortunately, this process is complex to use, which forces stakeholders to have a good knowledge of formalization tools. Moreover, the composition requires a strong dependency of the application with respect to the platform.

Metropolis [9] supports both design and analysis of heterogeneous embedded systems on the basis of the platform-based methodology. The behavioral representation is illustrated by entities such as concurrent and communication activities. In addition, this environment offers the possibility to use formal languages in accordance with the LTL logic for verifying both

functional and non-functional properties once the deployment reached by mapping of the system components with the platform entities is described. Similarly, GME [10] includes use constraints on the representation of executable concepts thanks to the Platform Modeling Language (PML). This consideration mixed with the integration of formal languages to describe the behavior of platforms provides a deeper design of embedded systems. However, the genericity of modeling languages used for describing the platforms does not facilitate the treatment of particular domains such as real-time. Furthermore, transformation rules are not entirely clear. This leads to a less meaningful separation of domain skills and consequently to a maintainability decrease.

2) With code generator: More specifically to RTES, Ptolemy project [11] is further adapted for describing execution models. The definition of those models relies on the actor-oriented design and revolves around mechanisms of concurrency and communication implemented between RTES components. The behavior represented within those models is translatable into several execution semantics. This benefit offers a code-formal model duality of deployed applications. This approach thus achieves a wide coverage of software execution platforms. Nevertheless, the concepts of structural representation are only intended for the modeling of applications with this approach, and not for RTES themselves. Mechanisms such as RTES components synchronization must therefore be simulated with other verification tools. In spite of a very high completeness, the RTES maintainability with Ptolemy is once again called into question.

C. Positioning

In order to highlight both advantages and drawbacks of the works presented above, Table I classifies the tool suites according to their uses. Depending on whether a given tool suite addresses only code generation, only formal models generation, or both it will be in the first, second, or third column respectively. The first row is for tool suites that are not adapted to RTOS, while the second row is for tool suites that are adapted to RTOS.

TABLE I: Tool suites comparison

	Code	Formal	Code & Formal
Not adapted to RTOS		Metropolis [9] GME [10]	
Adapted to RTOS	TransPI[1] [3] snippets+AADL [4] SRM [5]	ACSR+TRoS[2] [8]	Ptolemy[3] [11]

[1] Less suitable for both reusability and maintainability
[2] Less suitable for reusability
[3] Less suitable for maintainability

Ptolemy seems to be the most versatile. Nevertheless, stakeholders distinction does not appear clearly, which does not facilitate maintainability.

Within MDE, alternative approaches were compared [12] to meet these requirements. The adopted strategy offers the possibility to capitalize most RTOS descriptions for multi-platform deployment in a generic way.

In conjunction with this objective, SExPIsTools integrates the Real-Time Embedded Platform Modeling Language (RTEPML) [13]. RTEPML was developed with the aim of representing executable platform concepts dedicated to the real-time domain. To further detail their representation, RTEPML [14] has been enriched to describe their behavior in TPN. However, the generation process used to take into account these behavioral descriptions needs to be formalized, which was started in [1], and is continued in the work presented in this paper (see Sections IV and V).

III. SExPIsTools

This section presents SExPIsTools. Firstly, the modeling language RTEPML used to describe RTOS mechanisms is presented. Then, both deployment and formal models generation processes integrated in SExPIsTools are described. The role notion used as a generic basis of transformation rules is highlighted.

A. Modeling with RTEPML

RTEPML distinguishes the RTOS structural modeling from the behavioral RTOS modeling.

1) Structural description: RTEPML is born from SRM package [7] mentioned in the previous section. SRM allows the description of a large number of RTOS [15] and had identified all concepts and their mechanisms present in RTOS. In SRM, these concepts are called resources (e.g., task, semaphore, etc.). RTEPML keeps the same taxonomy. In Figure 1, a small part of OSEK/VDX RTOS [16] and VxWorks RTOS [17] descriptions in RTEPML are given. The task concept, called schedulable resource in RTEPML, is described for both RTOS.

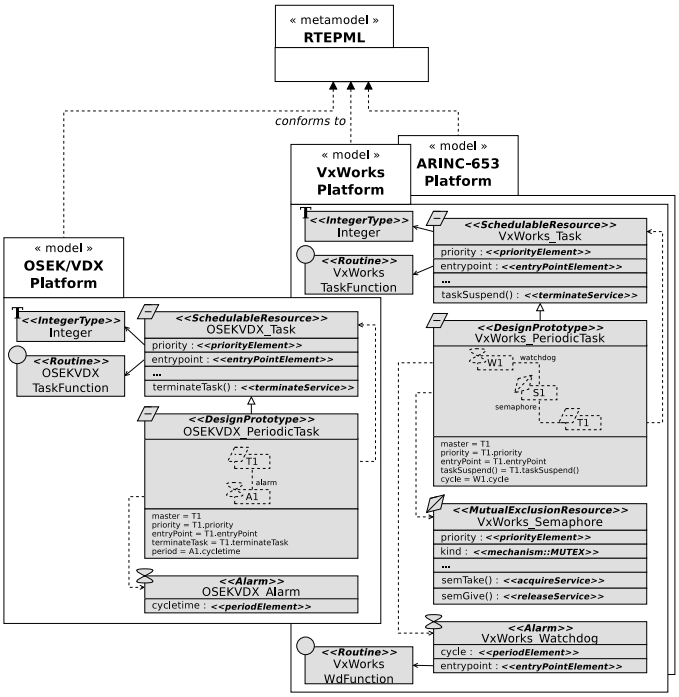


Fig. 1: Structural representation of OSEK/VDX platform

As depicted in Figure 1, thanks to the notion of roles (represented in bold between french quotation marks), we

could specify the priority (an integer) of OSEKVDX_Task or of VxWorks_Task. This priority plays the role of priority element (<<***priorityElement***>>) for both tasks. Roles are thus used to identify both structures and features of each RTOS resource. As another example, on OSEK/VDX model, the <<***terminateService***>> role characterizes the terminateService of OSEKVDX_Task in Figure 1.

Sometimes, certain concepts do inherently not exist on RTOS. As an example, the periodic task concept is missing on both OSEK/VDX and VxWorks platforms. With RTEPML, sets of concepts (i.e., identified with the *DesignPrototype* role in Figure 1) can be composed to translate this kind of concept. Thus, a PeriodicTask concept could be viewed at a composition of a Task and an Alarm for OSEK/VDX. As regards VxWorks, the PeriodicTask concept is differently composed of a Task and a Watchdog synchronizing with a Semaphore.

2) Behavioral description: The behavioral description allows to represent the life cycle of RTOS concepts. Figure 2 extends the representation of OSEK/VDX concepts, given in Figure 1. The <<***behavioralPrototype***>> role leads to the assignation of a behavioral description to each concept, including services.

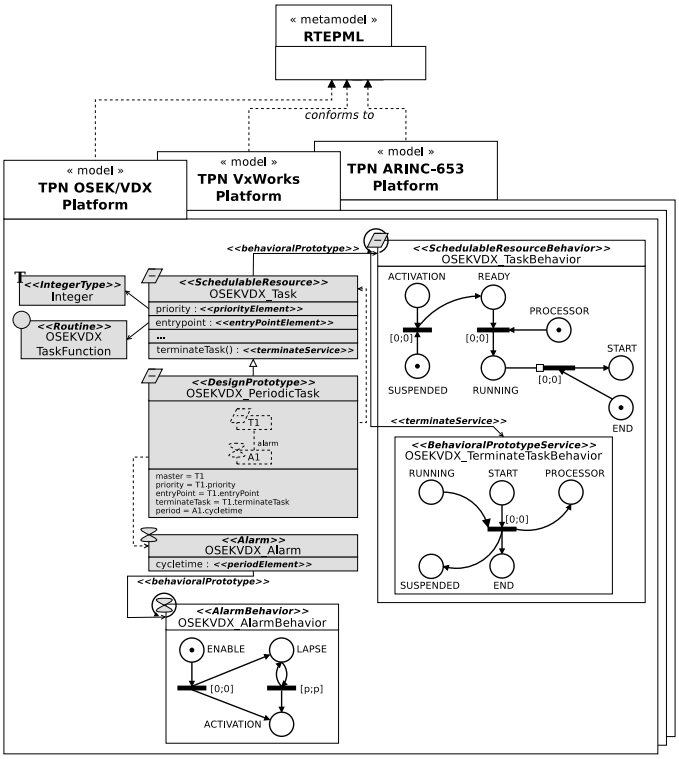


Fig. 2: Behavioral representation of OSEK/VDX platform

Each behavioral description is translated into a Time Petri Net (TPN) [18] [19] whose definition is given Section IV. This class of model is used to describe both synchronism and parallelism, as well as time evolution. Therefore, TPN are well adapted to our concerns.

In the example given in Figure 2, the TPN describing OSEKVDX_AlarmBehavior represents the periodic activation of OSEKVDX_Alarm. Informally, once a token is present in the ENABLE place, making it *marked*, one token is periodically distributed in the ACTIVATION place. Distribution of tokens is initially achieved once the left transition, represented as a black rectangle, is triggered (i.e., when ENABLE is marked and the transition clock has reached 0 time unit). The periodicity is then guaranteed by the right transition triggering (i.e., when LAPSE is marked and the transition clock has reached p time units). With clocks on transitions, we can consequently add as many time constraints as necessary, e.g., on the OSEKVDX_TaskBehavior TPN, a delay between the READY and RUNNING places could represent the required time to start the task execution.

B. Model generation processes within SExPIsTools

SExPIsTools is designed for multi-platform deployment. Both code and formal model generations are performed in two steps. Figure 3 depicts these two steps.

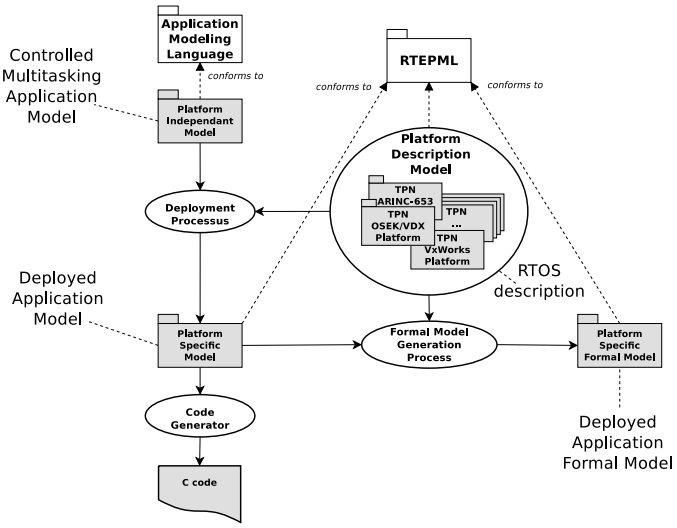


Fig. 3: Multi-platform deployment process within SExPIsTools

The first one concerns the deployment. This is common to both generations, which avoids to deploy twice. The application is deployed on a specific RTOS [12] [13]. The considered RTOS is given as a parameter of the deployment process. Transformation rules of the process are defined independently from the targeted RTOS. This independence is possible thanks to roles previously highlighted. The deployment is performed by mapping each application concept with its execution on the targeted RTOS. The mapping consists in locating the role of the executable concept corresponding to each application concept through the RTOS model. Once a correspondance is established, the structure of the located executable concept is *instanciated*, i.e., duplicated. Each instance is afterwards enriched by specifying its features with the help of the appropriate roles. Features specification emanates from application concepts information. All these specified instances finally constitute the model of the deployed application (i.e., the platform specific model of Figure 3).

The following step concerns either the code generation or the formal model generation. These both processes take in input the same generated deployed application model.

The code generator being not our focus in this paper, we only present the formalization of the deployed application model. Similarly to the deployment, the generation process of deployed application formal models instanciates TPN behavioral descriptions [14]. The location of each TPN is carried out from the structural instances of the deployed application model. Indeed, knowing the source executable concept of each structural instance, the corresponding TPN fragment is located with the <<***behavioralPrototype***>> role. Each corresponding TPN is thus duplicated giving a TPN behavioral instance (or each TPN fragment).

The generation of these TPN fragments engages their composition to constitute a global formal model of the deployed application. The elements serving as connection points must be located through the set of these TPN fragments. Similarly to the structural part, these elements are also located with roles. As instantiation rules, composition rules are based on these roles. In the interest of consistency, we have formalized the sequence of composition rules. This sequence is ordered to avoid any ambiguity in the formal model construction. The TPN fragments are therefore categorized according to RTOS concepts generalized in RTEPML:

- **Concurrent resources:** tasks, interruptions, alarms, etc.
- **Interaction resources:** semaphores, message queues, shared data, events, etc.
- **Routines:** application treatment including services called within the application. This treatment is only represented by the execution time.

The following Algorithm 1 informally describes the sequence of composition rules. We admit here that TPN fragments were already instanciated.

Each composition rule is labelled from a) to d) in comments through this algorithm. Firstly, a) each routine instance is composed of all its called service instances. Then, each concurrent resource must be composed with its execution routine. The execution routine is called the entry point of the concurrent resource. Each entry point is located with the <<***entryPointElement***>> role (see Figure 2). As a consequence, b) each concurrent resource instance is composed with its entry point. The next composition c) concerns all concurrent resource instances in order to put them in concurrency. As a final step, d) interaction resource instances are composed with the set of composed concurrent resource instances so that these latter interact with some of them.

This order will be respected in Section V in which these rules will be formalized. Next, in Section IV, the composition operator of TPN based on roles is defined to formally express these rules afterwards.

Algorithm 1 *Composition rules*

Input:
- $I_S = \{{I_S}^{R_1}, {I_S}^{R_2}, \ldots, {I_S}^{R_l}\}$; // The service calls behavioral instances with $\forall j \in [1, l]$, ${I_S}^{R_j} \subseteq I_S$ and l the number of routines to compose
- $I_C = \{i_{C1}, i_{C2}, \ldots, i_{Cm}\}$; // m concurrent resources behavioral instances
- $I_I = \{i_{I1}, i_{I2}, \ldots, i_{In}\}$; // n interaction resources behavioral instances

Output:
- M // The composed deployed application behavioral model

1: **for** $j = 1$ **to** l **do**
2: // a) Each routine behavioral instance is composed
3: $i_{Rj} \leftarrow ruleComposeRoutine({I_S}^{R_j})$
4: **end for**
5: **for** $k = 1$ **to** m **do**
6: **for all** j such that $1 \leq j \leq l$ **do**
7: **if** $\exists i_{Rj}$ such that i_{Rj} is the entrypoint of i_{Ck} **then**
8: // b) Each entry point is composed
9: $i_{EPk} \leftarrow ruleComposeEntryPoint(i_{Ck}, i_{Rj})$
10: **else**
11: $i_{EPk} \leftarrow i_{Ck}$
12: **end if**
13: **end for**
14: **end for**
15: **for all** k such that $1 \leq k \leq m$ **do**
16: $I_{EP} \leftarrow \{i_{EP1}\} \cup \cdots \cup \{i_{EPk}\} \cup \cdots \cup \{i_{EPm}\}$
17: **end for**
18: // c) All concurrent resources are composed
19: $i_{CR} \leftarrow ruleComposeConcurrentResources(I_{EP})$
20: // d) All interaction resources are composed
21: $i_{IR} \leftarrow ruleComposeInteractionResources(i_{CR}, I_I)$
22: $M \leftarrow i_{IR}$

IV. TPN Composition based on Roles

In order to define the composition of TPN fragments, roles are added to the TPN modeling. These roles are therefore assigned to places. The interest of such a method is to merge places [20] [21], which are the connection points of the deployed system that must be generated in TPN.

In this section, TPN with roles are firstly defined. The definition of the instantiation of TPN with roles is then given. Finally, the composition of TPN is highlighted through a synchronization formalism based on roles.

A. Formal definition of TPN with roles

TPN are a timed extension of classical Petri nets [22] in which an implicit *clock* and an explicit *time interval* are associated with each transition of the net. Informally, the clock measures the time since the transition has been (continuously) enabled, whereas the interval is interpreted as a *firing* condition: the transition, once enabled, may be fired only if the value (or *valuation*) of its clock belongs to the time interval.

In the following, $\mathbb{N}$ denotes the set of natural numbers, $\mathbb{R}_{\geq 0}$ the set of non-negative real numbers, $\emptyset$ is the empty set and $\mathbf{0}$ is the null vector.

Definition 1 (TPN): A TPN $\mathcal{T}$ is a tuple $\langle P, T, \mathsf{Pre}, \mathsf{Post}, m_0, I_s \rangle$ where:

- P is a finite, non-empty set of *places*;
- T is a finite, non-empty set of *transitions*;
- $\mathsf{Pre} : P \times T \rightarrow \mathbb{N}$ is the *backward incidence function*;

- $\mathsf{Post} : P \times T \to \mathbb{N}$ is the *forward incidence function*;
- m_0 is the *initial marking* of the net;
- $I_s : T \to \mathbb{N} \times (\mathbb{N} \cup \{+\infty\})$ assigns a static *time interval* to each transition.

A *marking* of the net $\mathcal{T}$ is an application from P to $\mathbb{N}$ giving for each place of the net the number of tokens it contains. A transition $t \in T$ is *enabled* by a marking m, which is denoted by $t \in enabled(m)$, if all of its input places contain "enough" tokens; more formally, $enabled(m) = \{\, t \in T \mid \forall p \in P,\ m(p) \geq \mathsf{Pre}(p,t) \,\}$. A transition $t \in T$ is *newly enabled* by the firing of transition t_f from the marking m, which is denoted by $t \in \uparrow enabled(m, t_f)$, if it is enabled by the final marking m_f defined by $\forall p \in P,\ m_f(p) = m(p) - \mathsf{Pre}(p,t_f) + \mathsf{Post}(p,t_f)$ but not by the intermediate marking m_i defined by $\forall p \in P,\ m_i(p) = m(p) - \mathsf{Pre}(p,t_f)$. More formally, $\uparrow enabled(m,t_f) = enabled(m_f) \bigcap ((T \setminus enabled(m_i)) \cup \{t_f\})$.

Finally, for any interval I_s, we denote by $I_s{}^{\downarrow}$ the smallest left-closed interval with lower bound 0 that contains I_s. For each transition tr there is an associated clock x_{tr}. We consider valuations on the set of clocks $\{\, x_{tr} \mid tr \in T \,\}$ and we will slightly abuse the notations by writing $v(tr)$ instead of $v(x_{tr})$ to denote the valuation of the clock associated with transition tr.

The operational semantics of a TPN can be formally described as a time transition system; as it is a special case of the semantics of TPN with read and inhibitor arcs (given in Def. 3, we will omit it here for the sake of clarity.

In order to model such behaviors as conditional executions and preemption mechanisms, TPN have been extended with *read arcs* (represented in the following with a white square instead of a regular arrow) and *inhibitor arcs* (represented with a white circle). It should be noted that these arcs only impact the enabling rules of the net but not the marking obtained by firing a transition: read arcs test the presence of tokens in places without consuming them, whereas an inhibitor arc is used to stop the elapsing of time on a transition as long as there is a certain number of tokens in the place.

Definition 2 (TPN with read/inhibitor arcs): A TPN with read and inhibitor arcs (RI_TPN) is a tuple $\mathcal{T}_{RI} = \langle \mathcal{T}, \mathsf{Read}, \mathsf{Inh} \rangle$ where:

- $\mathcal{T} = \langle P, T, \mathsf{Pre}, \mathsf{Post}, m_0, I_s \rangle$ is a TPN,
- $\mathsf{Read} : P \times T \to \mathbb{N}$ is the *read function*;
- $\mathsf{Inh} : P \times T \to \mathbb{N} \cup \{+\infty\}$ is the *inhibition function*[1].

Informally, a transition is enabled if there are "enough tokens" in the places linked by either input arcs or read arcs *and* if there are "not too many tokens" in the places linked by inhibitor arcs. More formally, the definition of the set of transitions enabled by a marking m is updated as follows:

$$enabled(m) = \{t \in T \mid \forall p \in P, \mathsf{Inh}(p,t) > m(p) \geq \max(\mathsf{Pre}(p,t), \mathsf{Read}(p,t))\}$$

[1]If no inhibitor arcs links a transition t to a place p, then $\mathsf{Inh}(p,t) = +\infty$.

The definition of the set of transitions newly enabled from a marking m by the firing of a transition t_f is similarly updated.

Definition 3 (Semantics of the RI_TPN): The operational semantics of the RI_TPN with read and inhibitor arcs $\mathcal{T}_{RI}$ defined above is given by the time transition system $\mathcal{S} = (Q, q_0 \to)$ such that:

- $Q = \mathbb{N}^P \times \mathbb{R}^T_{\geq 0}$;
- $q_0 = (m_0, \mathbf{0})$;
- $\to \in Q \times (T \cup \mathbb{R}_{\geq 0}) \times Q$ is the *transition relation* and is composed of:
  - the *discrete transition transition*, defined $\forall t_f \in T$ by $(m,v) \xrightarrow{t_f} (m',v')$ iff:
    - $(t_f \in enabled(m)$;
    - $v(t_f) \in I_s(t_f)$;
    - $\forall p \in P,\ m'(p) = m(p) - \mathsf{Pre}(p,t_f) + \mathsf{Post}(p,t_f)$;
    - $\forall t \in T,\ v'(t) = \begin{cases} 0 \textbf{ if } t \in \uparrow enabled(m,t_f) \\ v(t) \textbf{ otherwise} \end{cases}$;
  - the *discrete transition transition*, defined $\forall d \in \mathbb{R}_{\geq 0}$ by $(m,v) \xrightarrow{d} (m,v')$ iff $\forall t \in enabled(m)$, $\forall \delta \in]0;d]$, $(v(t) + \delta) \in I_s{}^{\downarrow}(t)$.

Definition 4 (RI_TPN with roles): A RI_TPN with roles is a tuple $\mathcal{N} = \langle \mathcal{T}_{\mathcal{RI}}, R, \lambda \rangle$ where:

- $\mathcal{T}_{\mathcal{RI}}$ is a RI_TPN,
- R is a finite set of roles,
- $\lambda : P \to R \cup \{\perp\}$ is the function assigning a role to a place and $\perp$ denoting that no role is assigned to a place. Hereafter, some notations and properties of this function are enumerated:
  1) $P_\lambda = \{p \in P \mid \lambda(p) \neq \perp\}$ is the set of places with role.
  2) $\lambda_{\setminus P_\lambda} : P_\lambda \to R$ is an injective function;
  3) $\lambda^{-1} : R \cup \{\perp\} \to P \cup \{\emptyset\}$ such that $\begin{cases} \forall r \in R,\ \lambda^{-1}(r) = \begin{cases} p \text{ if } \lambda(p) = r \\ \emptyset \text{ otherwise} \end{cases} \\ \lambda^{-1}(\perp) = \emptyset \end{cases}$

The operational semantics of the RI_TPN with roles $\mathcal{N} = \langle \mathcal{T}_{\mathcal{RI}}, R, \lambda \rangle$ is the same as that of RI_TPN. Indeed, the use of roles within the definition of RI_TPN does not impact its semantics.

B. Instantiation of RI_TPN with roles

As seen previously, all RI_TPN fragments are instantiated before being composed. In order to distinguish the fragments to compose, atomic elements such as roles, places and transitions must be identified according to the instances names, but also according to referenced instances names.

Indeed, referenced instances emerge when instances are service calls. Each service call refers to a resource instance. As an example, a task activation service refers to a task. The two concepts are distinguished because this has an impact during the composition between a service call instance and

its referenced resource instance. For this reason, the renaming of a role and a renaming of places and transitions are distinctly separated. This distinction is made with the following instantiation operator.

Let $\mathcal{N} = \langle P, T, \mathsf{Pre}, \mathsf{Post}, m_0, I_s, \mathsf{Read}, \mathsf{Inh}, R, \lambda \rangle$ be the RI_TPN with roles to instantiate. The following labels ins and ref respectively gives the names of the instance and the referenced instance. If the instance is a resource, there is no referenced instance with $ref = ins$. The global renaming function $\rightharpoonup$ is a bijective function from Set to Set' where $Set \in \{P, T, R\}$.

Definition 5 (Instantiation of RI_TPN with roles): The instantiation of $\mathcal{N}$ denoted by $\mathcal{N}_{ins} = Ins(\mathcal{N}, ins, ref) = \langle P_{ins}, T_{ins}, \mathsf{Pre}_{ins}, \mathsf{Post}_{ins}, m_{0-ins}, I_{s-ins}, \mathsf{Read}_{ins}, \mathsf{Inh}_{ins}, R_{ref}, \lambda_{ins} \rangle$ is defined by:

$$
\begin{array}{rcl}
\mathcal{N}_{ins} & = & Ins(\mathcal{N}, ins, ref) \\
& = & \mathcal{N} \left| \begin{array}{l}
P_{ins} = \{p_{ins} \text{ s.t. } p \in P \text{ and } p \rightharpoonup p_{ins}\}, \\
T_{ins} = \{t_{ins} \text{ s.t. } t \in T \text{ and } t \in T, t \rightharpoonup t_{ins}\}, \\
R_{ref} = \{r_{ref} \text{ s.t. } r \in R \text{ and } r \rightharpoonup r_{ref}\}, \\
\forall p \in P, \forall t \in T, \forall r \in R, \text{ s.t. } p \rightharpoonup p_{ins}, t \rightharpoonup t_{ins}, \\
\text{ and } r \rightharpoonup r_{ref} \text{ we have:} \\
\mathsf{Pre}_{ins}(p_{ins}, t_{ins}) = \mathsf{Pre}(p, t), \\
\mathsf{Post}_{ins}(p_{ins}, t_{ins}) = \mathsf{Post}(p, t), \\
\mathsf{Read}_{ins}(p, t) = \mathsf{Read}_{ins}(p_{ins}, t_{ins}), \\
\mathsf{Inh}_{ins}(p, t) = \mathsf{Inh}_{ins}(p_{ins}, t_{ins}), \\
\lambda_{ins}(p) = r \text{ iff } \lambda_{ins}(p_{ins}) = r_{ref} \\
\lambda_{ins}(p) = \perp \text{ iff } \lambda_{ins}(p_{ins}) = \perp
\end{array} \right.
\end{array}
$$

C. Specific extension of instantiated RI_TPN with roles

Once RI_TPN are instantiated, we have sometimes been faced with the need to extend them according to the application to deploy. For intance, in a real-time system based on a cooperative multitasking application with priorities, eligible low priority tasks are inhibited by eligible high priority tasks when allocating the processor.

We have focused on this case through this paper in order to enrich our previous work [1] in which all tasks had the same priorities. The cooperative multitasking case with priorities is depicted in Figure 4. The RI_TPN $\mathcal{N}_{T1} = Ins(\mathcal{N}, ins, ref)$ is an instance of concurrent resource such that a periodic task where $ref = ins = T1$. In bold, some places with inhibitor arcs, represented with circles, are connected to the $resume_{T1}$ transition to inhibit the state change from $READY_{T1}$ to $RUNNING_{T1}$. The marking of one of the places set $\{READY_{T2}, READY_{T3}, \ldots, READY_{Tx}\}$ carries out this inhibition action and ensures the cooperative scheduling of tasks.

This action being a scheduling specific case, we defined a dedicated operator for adapting instantiated RI_TPN of concurrent resources such that $\mathcal{N}_{T1}$ to a cooperative scheduling context.

Let $\mathcal{N}_{ins} = \langle P_{ins}, T_{ins}, \ldots, \lambda_{ins} \rangle$ be a RI_TPN with roles of a concurrent resource firstly instantiated as previously seen. $\mathcal{N}^{cs}_{ins} = \langle P^{cs}_{ins}, T^{cs}_{ins}, \ldots, \lambda^{cs}_{ins} \rangle$ represents the same instance extended according to a set of n concurrent resources identified by $INS = \{ins_1, ins_2, \ldots, ins_n\}$ with upper priorities.

Definition 6 (Extension of RI_TPN with roles): Cooperative scheduling of concurrent resources: The

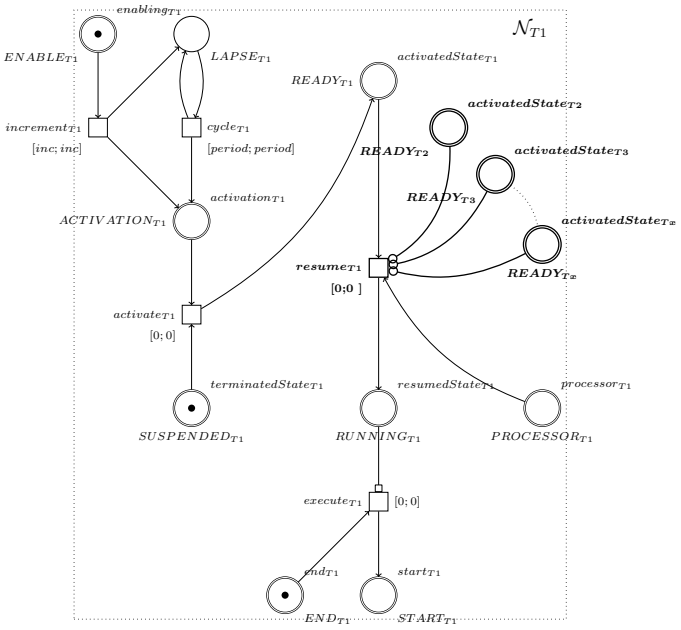


Fig. 4: Specific extension of periodic task in RI_TPN for cooperative multitasking

extension of $\mathcal{N}_{ins}$ in concurrence with n instances adapted to a cooperative scheduling is denoted by:

$$\mathcal{N}^{cs}_{ins} = CoopSched(\mathcal{N}_{ins}, INS)$$

with $\forall t \in T_{ins}$, $\exists \mathsf{Pre}(\lambda^{-1}(activatedState_{ins}), t) \in \mathsf{Pre}_{ins}$ and $\exists \mathsf{Post}(\lambda^{-1}(resumedState_{ins}), t) \in \mathsf{Post}_{ins}$

Formally, this definition gives:

- $R^{cs}_{ins} = R_{ins} \cup R_{INS}$ with $R_{INS} = \bigcup_{\forall i \in [1,n]} \{r_{ins_i}\}$;
- $P^{cs}_{ins} = P_{ins} \cup P_{INS}$ with $P_{INS} = \bigcup_{\forall i \in [1,n]} \{p_{ins_i}\}$;
- $T^{cs}_{ins} = T_{ins}$;
- $\lambda^{cs}_{ins} : P^{cs}_{ins} \rightarrow R^{cs}_{ins}$ is defined by:
  - $\forall p \in P^{cs}_{ins} \setminus P_{INS}$, $\lambda^{cs}_{ins}(p) = \lambda_{ins}(p)$
  - $\forall p \in P_{INS}$ and $\forall i \in [1, n]$, $\lambda^{cs}_{ins}(p) = r_{ins_i}$ with $r_{ins_i} \in R_{INS}$
- $\mathsf{Pre}^{cs}_{ins} : P^{cs}_{ins} \times T^{cs}_{ins} \rightarrow \mathbb{N}$ is defined $\forall p \in P^{cs}_{ins}$ and $\forall t \in T^{cs}_{ins}$ by $\mathsf{Pre}^{cs}_{ins}(p, t) = \mathsf{Pre}_{ins}(p, t)$;
- $\mathsf{Post}^{cs}_{ins} : P^{cs}_{ins} \times T^{cs}_{ins} \rightarrow \mathbb{N}$ is defined $\forall p \in P^{cs}_{ins}$ and $\forall t \in T^{cs}_{ins}$ by $\mathsf{Post}^{cs}_{ins}(p, t) = \mathsf{Post}_{ins}(p, t)$;
- $m_{0^{cs}_{ins}} : P^{cs}_{ins} \rightarrow \mathbb{N}$ is defined $\forall p \in P^{cs}_{ins}$ by $m_{0^{cs}_{ins}}(p) = \begin{cases} m_{0ins}(p) \text{ if } p \in P^{cs}_{ins} \setminus P_{INS} \\ m_{0INS}(p) \text{ if } p \in P_{INS} \end{cases}$ with m_{0INS} is defined by $m_{0INS} : P_{INS} \rightarrow \mathbb{N}$;
- $I_{s^{cs}_{ins}} : T^{cs}_{ins} \rightarrow \mathcal{I}$ is defined $\forall t \in T^{cs}_{ins}$ by $I_{s^{cs}_{ins}}(t) = I_{s_{ins}}(t)$;
- $\mathsf{Read}^{cs}_{ins} : P^{cs}_{ins} \times T^{cs}_{ins} \rightarrow \mathbb{N}$ is defined $\forall p \in P^{cs}_{ins}$ and $\forall t \in T^{cs}_{ins}$ by $\mathsf{Read}^{cs}_{ins}(p, t) = \mathsf{Read}_{ins}(p, t)$;

- $\mathsf{Inh}^{cs}_{ins} : P^{cs}_{ins} \times T^{cs}_{ins} \to \mathbb{N}$ is defined $\forall p \in P^{cs}_{ins}$ and $\forall t \in T^{cs}_{ins}$ by $\mathsf{Inh}^{cs}_{ins}(p,t) = \begin{cases} \mathsf{Inh}_{ins}(p,t) \text{ if } p \in P^{cs}_{ins} \setminus P_{INS} \\ 1 \text{ if } \begin{cases} p \in P_{INS} \\ t \in T^{cs}_{ins}, \\ \exists \mathsf{Pre}(\lambda^{-1}(activatedState_{ins}),t) \in \mathsf{Pre}_{ins} \\ \text{and} \\ \exists \mathsf{Post}(\lambda^{-1}(resumedState_{ins}),t) \in \mathsf{Post}_{ins} \end{cases} \end{cases}$

D. RI_TPN Synchronization based on roles

In order to synchronize some RI_TPN, we must clarify the definition of the composition of RI_TPN, which will be based on roles assigned to places. Let $\mathcal{N}_1, \ldots, \mathcal{N}_n$ be n RI_TPN $\mathcal{N}_i = \langle P_i, T_i, \mathsf{Pre}_i, \mathsf{Post}_i, m_{0_i}, I_{s_i}, Read_i, Inh_i, R_i, \lambda_i \rangle$ with roles such that $\forall k \neq k' \in [1,n] \implies T_k \cap T_{k'} = \emptyset$ and $P_k \cap P_{k'} = \emptyset$. The composition $\mathcal{N} = \langle P, T, \mathsf{Pre}, \mathsf{Post}, m_0, I_s, R, \lambda \rangle$ of the previous RI_TPN with roles will be denoted by $\mathcal{N} = \mathcal{N}_1 || \mathcal{N}_2 || \ldots || \mathcal{N}_n$. Linked to this composition, we define a function leading to the merging of places whose assigned roles will be taken into account in parameters.

The merging function $\hookrightarrow$ is a partial function from $(R_1 \cup \{\bullet\}) \times (R_2 \cup \{\bullet\}) \times \cdots \times (R_n \cup \{\bullet\}) \to P \times R$ where $\bullet$ is a special symbol used when a RI_TPN is not involved in a particular merge of the global system. We then extend the definition of the assigning inverse function with $\lambda^{-1}(\bullet) = \emptyset$

The composition of n RI_TPN with m merging is denoted by

$$\Big(\mathcal{N}_1 || \ldots || \mathcal{N}_n\Big)\Big|_{\substack{(r^1_1, \ldots, r^1_n) \hookrightarrow (p^1, r^1) \\ \ldots \\ (r^m_1, \ldots, r^m_n) \hookrightarrow (p^m, r^m)}}$$

with $\forall i \in [1,n]$, $\forall j \in [1,m]$, $r^j_i \in R_i$, $r^j \in R$ and $p^j \in P$, and $\forall k \in [1,m]$, $k \neq j \Rightarrow r^k_i \neq r^j_i$

We will subsequently use the following notations:

- Let $P^{merged}_i \subseteq P_i$ be the set of places of the net $\mathcal{N}_i$ merged by the composition. Formally $P^{merged}_i = \bigcup\limits_{\forall j \in [1,m]} \{\lambda^{-1}_i(r^j_i)\}$
- Let $P^{\hookrightarrow} \subseteq P$ be the set of places of the net $\mathcal{N}$ obtained by the merging. Formally $P^{\hookrightarrow} = \bigcup\limits_{\forall j \in [1,m]} \{p^j\}$

Definition 7 (Composition of RI_TPN with roles): The composition of the n RI_TPN $\mathcal{N}_i$ with the merging $\hookrightarrow$ denoted by:

$$\mathcal{N} = \Big(\mathcal{N}_1 || \ldots || \mathcal{N}_n\Big)\Big|_{\substack{(r^1_1, \ldots, r^1_n) \hookrightarrow (p^1, r^1) \\ \ldots \\ (r^m_1, \ldots, r^m_n) \hookrightarrow (p^m, r^m)}}$$

is defined by:

- $R = \left(\bigcup\limits_{\forall i \in [1,n]} \left(R_i \setminus \bigcup\limits_{\forall j \in [1,m]} \{r^j_i\}\right) \right) \cup \left(\bigcup\limits_{\forall j \in [1,m]} \{r^j\} \right)$;
- $P = \left(\bigcup\limits_{\forall i \in [1,n]} P_i \setminus P^{merged}_i \right) \cup P^{\hookrightarrow}$;
- $T = \bigcup\limits_{\forall i \in [1,n]} T_i$;
- $\lambda : P \to R$ is defined by:
  - $\forall p \in P \setminus P^{\hookrightarrow}$ meaning that $\exists i$ such that $p \in P_i$ then $\lambda(p) = \lambda_i(p)$
  - $\forall p^j \in P^{\hookrightarrow}$, meaning that p is the result of a merging, $\lambda(p^j) = r^j$
- $\mathsf{Pre} : P \times T \to \mathbb{N}$ is defined $\forall p \in P$ and $\forall t \in T_i \subseteq T$ by $\mathsf{Pre}(p,t) = \begin{cases} \mathsf{Pre}_i(p,t) \text{ if } p \in P \setminus P^{\hookrightarrow} \text{ and } p \in P_i \\ \mathsf{Pre}_i(p',t), \text{ if } \begin{cases} p \in P^{\hookrightarrow} \text{ and } p' \in P_i \\ (\ldots, r^k_i, \ldots) \hookrightarrow (p, \lambda(p)) \\ \lambda_i(p') = r^k_i \end{cases} \\ 0 \text{ otherwise.} \end{cases}$
- $\mathsf{Post} : P \times T \to \mathbb{N}$ is defined $\forall p \in P$ and $\forall t \in T_i \subseteq T$ by $\mathsf{Post}(p,t) = \begin{cases} \mathsf{Post}_i(p,t) \text{ if } p \in P \setminus P^{\hookrightarrow} \text{ and } p \in P_i \\ \mathsf{Post}_i(p',t), \text{ if } \begin{cases} p \in P^{\hookrightarrow} \text{ and } p' \in P_i \\ (\ldots, r^k_i, \ldots) \hookrightarrow (p, \lambda(p)) \\ \lambda_i(p') = r^k_i \end{cases} \\ 0 \text{ otherwise.} \end{cases}$
- $m_0 : P \to \mathbb{N}$ is defined $\forall p \in P$ by: $m_0(p) = \begin{cases} m_{0i}(p) \text{ if } p \in P \setminus P^{\hookrightarrow} \text{ and } p \in P_i \\ \sum\limits_{i=1}^{n} m_{0i}\big(\lambda^{-1}(r^k_i)\big) \text{ if } \begin{cases} p \in P^{\hookrightarrow} \\ (r^k_1, \ldots, r^k_n) \hookrightarrow (p, \lambda(p)) \end{cases} \end{cases}$
- $I_s : T \to \mathcal{I}$ is defined $\forall t \in T$ by: $I_s(t) = I_{s_i}(t)$ **if** $t \in T_i$;
- $\mathsf{Read} : P \times T \to \mathbb{N}$ is defined $\forall p \in P$ and $\forall t \in T_i \subseteq T$ as $\mathsf{Pre}(p,t)$;
- $\mathsf{Inh} : P \times T \to \mathbb{N}$ is defined $\forall p \in P$ and $\forall t \in T_i \subseteq T$ as $\mathsf{Pre}(p,t)$

As an example, $\mathcal{N} = \Big(\mathcal{N}_1 || \mathcal{N}_2 || \mathcal{N}_3\Big)\Big|_{(r_1, r_2, \bullet) \hookrightarrow (p,r)}$ is the parallel composition of the 3 TPN, i.e., $\mathcal{N}_1$, $\mathcal{N}_2$ and $\mathcal{N}_3$, where the place $p_1 \in P_1$ such that $\lambda_1(p_1) = r_1$ and the place $p_2 \in P_2$ such that $\lambda_2(p_2) = r_2$ are merged. The name of the place obtained by this merging in $\mathcal{N}$ is $p \in P$ and its role is $\lambda(p) = r \in R$.

Property 1 (Associativity): The composition of TPN with roles is associative in the following sense:

$$\Big(\mathcal{N}_1 || \mathcal{N}_2 || \mathcal{N}_3\Big)\Big|_{(r_1, r_2, r_3) \hookrightarrow (p,r)} = \Big(\Big(\mathcal{N}_1 || \mathcal{N}_2\Big)\Big|_{(r_1, r_2) \hookrightarrow (p_{12}, r_{12})} || \mathcal{N}_3\Big)\Big|_{(r_{12}, r_3) \hookrightarrow (p,r)} = \Big(\mathcal{N}_1 || \Big(\mathcal{N}_2 || \mathcal{N}_3\Big)\Big|_{(r_2, r_3) \hookrightarrow (p_{23}, r_{23})}\Big)\Big|_{(r_1, r_{23}) \hookrightarrow (p,r)}$$

Property 2 (Commutativity): The composition of TPN with roles is commutative:

$$\Big(\mathcal{N}_1 || \mathcal{N}_2\Big)\Big|_{\substack{(r^1_1, r^1_2) \hookrightarrow (p^1, r^1) \\ \ldots \\ (r^k_1, r^k_2) \hookrightarrow (p^k, r^k)}} = \Big(\mathcal{N}_2 || \mathcal{N}_1\Big)\Big|_{\substack{(r^1_2, r^1_1) \hookrightarrow (p^1, r^1) \\ \ldots \\ (r^k_2, r^k_1) \hookrightarrow (p^k, r^k)}}$$

V. CONSTRUCTION AND ILLUSTRATION

The definitions presented above will help with the formal construction of behavioral models expressed as RI_TPN. This

construction will serve as a basis for the transformation process within the SExPIsTools framework (Figure 3). As described in Algorithm 1, the process consists of four successive composition rules, detailled in the paragraphs below and defined by equations (1) to (4).

A construction example in RI_TPN is provided to illustrate the method. Figure 5 presents some RI_TPN with roles, one per box, instantiated and ready for construction. Every operation details the fragments involved in the composition. The mergeable places are represented in double circle and those ready to be merged are connected by a hook-dotted arc with a letter corresponding to the construction step, i.e., the sequence of rules in Algorithm 1. Finally, roles are indicated above and to the right of places.

The whole model is describing a monoprocessor application $Proc$ with two periodic tasks $T1$ and $T2$ sharing the same semaphore S. A cooperative multitasking is established between $T1$ and $T2$ with a non-preemptive context. $T2$ has a higher priority than $T1$. Each task points to an execution routine composed of three services called in the following order: $Get_k(S); Release_k(S); Terminate_k(Tk)$ with $k \in [1, 2]$.

a) ruleComposeRoutine: The list of services considered in RTEPML is not exhaustive at the moment. The instructions described in RI_TPN are currently activation and termination of task, acquisition and release of semaphore and waiting, notification and inhibition of event.

Let n be the number of call services described following: $\{\mathcal{N}_{S1}, \mathcal{N}_{S2}, \ldots, \mathcal{N}_{Sn}\}$ such that $\forall i \in [1, n], \mathcal{N}_{S_i} = Ins(\mathcal{N}_S, S_i, ref\_S_i)$ with $\mathcal{N}_S$ the RI_TPN describing a service, S_i the instance name and ref_S_i the referenced instance name. The routine construction then implies $n - 1$ compositions, each one having m_j mergings of places with $j \in [1, n-1]$. The construction of a routine instance $\mathcal{N}_R$ is given by (1).

Illustration 1 (see Figure 5): By applying $\mathcal{N}_R$ from (1), $\forall k \in [1, 2]$, $\mathcal{N}_{TkBody}$ is built from RI_TPN $\{\mathcal{N}_{Get_k(S)}, \mathcal{N}_{Release_k(S)}, \mathcal{N}_{Terminate_k(Tk)}\}$. This sequence describes in the order, an acquisition of S, a release of S and a termination of Tk.

b) ruleComposeEntryPoint: Each resource points to a routine described by $\mathcal{N}_R$ previously formed. Consequently, $\mathcal{N}_R$ is composed with $\mathcal{N}_{C\tau} = Ins(\mathcal{N}_C, C_\tau, C_\tau)$ where $\mathcal{N}_C$ is the RI_TPN describing a concurrent resource and C_τ is the label indexed to identify each instance. The construction $\mathcal{N}_{EP}$ of a concurrent resource instance with its executable body is given by (2) for m mergings (we admit here that specific extensions of $\mathcal{N}_{C\tau}$ have already been applied for the needs of the application in this equation).

Illustration 2 (see Figure 5): By applying $\mathcal{N}_{EP}$ from (2), $\forall \phi \in [1, 2]$, $\mathcal{N}_{Task\phi\_withBody}$ is built composing $\mathcal{N}_{T\phi}$ with its entry point $\mathcal{N}_{T\phi Body}$. Prior to each composition, $\mathcal{N}_{T1}$ has been extended since this task has the lowest priority. This extension has thus been achieved by the $CoopSched(\mathcal{N}_{T1}, \{T2\})$ operation.

c) ruleComposeConcurrentResources: At this stage, concurrent resources must be linked together with the aim of being scheduled by the same processor.

Let q_C be the number of concurrent resources with their composed executable bodies such that $\forall i_C \in [1, q_C]$, each resource is described by $\mathcal{N}_{EP_{i_C}}$ in accordance with $\mathcal{N}_{EP}$ previously formed. The construction then implies $q_C - 1$ compositions, each one having m_{j_C} mergings with $j_C \in [1, q_C - 1]$. The construction of $\mathcal{N}_{CR}$ is given by (3).

Illustration 3 (see Figure 5): By applying $\mathcal{N}_{CR}$ from (3), $\mathcal{N}_{DeployedApplication_{CR}}$ is firstly composed of $\mathcal{N}_{T1\_withBody}$ and $\mathcal{N}_{T2\_withBody}$.

d) ruleComposeInteractionResources: Note that the processor is also a shared resource. It will therefore be considered as an interaction resource.

Let q_I be the number of interaction resources considered such that $\forall i_I \in [1, q_I]$, each resource is described by $\mathcal{N}_{I_{i_I}} = Ins(\mathcal{N}_I, I_{i_I}, I_{i_I})$ with $\mathcal{N}_I$ the TPN describing an interaction resource. Each interaction resource is composed with $\mathcal{N}_{CR}$ previously formed. The global construction then implies q_I compositions, each one having m_{j_I} mergings with $j_I \in [1, q_I]$. The global composition $\mathcal{N}_{IR}$ is given by (4).

Illustration 4 (see Figure 5): By applying $\mathcal{N}_{IR}$ from (4), $\mathcal{N}_{DeployedApplication}$ is finalized by composing $\mathcal{N}_{DeployedApplication_{CR}}$, $\mathcal{N}_S$ and $\mathcal{N}_{Proc}$.

VI. Experimentation

We illustrate the use of the formal model generation process on a case study. This case study is adapted from a schedulability case [23] in the context of cooperative multitasking.

A. Case study description

We consider an application with three concurrent real-time activities implemented as three real-time schedulable tasks $T1$, $T2$ and $T3$. The concurrency of these tasks emanates from a cooperative multitasking scheduler (based on a non-preemptive priority policy). Here are their characteristics:

- $T1$ is periodic with period $P1 = a$ with $a \in [0, +\infty[$ and has an execution time $C1 \in [10, 20]$.
- $T2$ is sporadic with only a minimal delay of $P2 = 2a$ time units between two activations. The execution time of $T2$ is $C2 \in [18, 28]$.
- Finally, $T3$ is periodic with period $P3 = 3a$ time units and has an execution time $C3 \in [20, 28]$.

These three tasks are defined with the following priority order: $T1 > T2 > T3$. Period a of $T1$ is a parameter determining the limit condition of schedulability of the tasks.

B. Purpose

The formal model generation process will be applied for two different RTOS. The two chosen RTOS are those used to present RTEPML in Section III: OSEK/VDX [16] and VxWorks [17]. Both are used in the industrial sector, have different API and behave differently. Roméo [24], the model-checking tool developed within our team is used to check the generated formal models.

$$\mathcal{N}_R = \Bigg(\Big(\big(\mathcal{N}_{S_1}||\mathcal{N}_{S_2}\big)\Big|_{\substack{(end_{ref\_S_1}, start_{ref\_S_2}) \hookrightarrow (S_{S_1 \to S_2}, \perp) \\ (r^2_{S_1}, r^2_{S_2}) \hookrightarrow (p^2_{S_2}, r^2_{S_2}) \\ \dots \\ (r^{m_1}_{S_1}, r^{m_1}_{S_2}) \hookrightarrow (p^{m_1}_{S_2}, r^{m_1}_{S_2})}} ||\mathcal{N}_{S_3}\Big)\Big|_{\substack{(end_{ref\_S_2}, start_{ref\_S_3}) \hookrightarrow (S_{S_1S_2 \to S_3}, \perp) \\ (r^2_{S_1S_2}, r^2_{S_3}) \hookrightarrow (p^2_{S_3}, r^2_{S_3}) \\ \dots \\ (r^{m_2}_{S_1S_2}, r^{m_2}_{S_3}) \hookrightarrow (p^{m_2}_{S_3}, r^{m_2}_{S_3})}}$$

$$\dots ||\mathcal{N}_{S_n}\Big)\Big|_{\substack{(end_{ref\_S_{n-1}}, start_{ref\_S_n}) \hookrightarrow (S_{S_1S_2\dots S_{n-1} \to S_n}, \perp) \\ (r^2_{S_1S_2\dots S_{n-1}}, r^2_{S_n}) \hookrightarrow (p^2_{S_n}, r^2_{S_n}) \\ \dots \\ (r^{m_{n-1}}_{S_1S_2\dots S_{n-1}}, r^{m_{n-1}}_{S_n}) \hookrightarrow (p^{m_{n-1}}_{S_n}, r^{m_{n-1}}_{S_n})}} \quad (1)$$

with $\forall k \in [1, m_j]$ and $n \geq 2$ if $k \geq 2$ then $r^k_{S_1 \dots S_j} = r^k_{S_{j+1}}$

$$\mathcal{N}_{EP} = \Big(\mathcal{N}_{C_\tau}||\mathcal{N}_R\Big)\Big|_{\substack{(start_{C_\tau}, start_{ref\_S_1}) \hookrightarrow (S, \perp) \\ (end_{C_\tau}, end_{ref\_S_n}) \hookrightarrow (E, \perp) \\ (r^3_{C_\tau}, r^3_R) \hookrightarrow (p^3_{C_\tau}, r^3_R) \\ \dots \\ (r^m_{C_\tau}, r^m_R) \hookrightarrow (p^m_{C_\tau}, r^m_R)}} \quad (2)$$

with $\forall k \in [1, m]$ if $k \geq 3$ then $r^k_{C_\tau} = r^k_R$

$$\mathcal{N}_{CR} = \Bigg(\Big(\mathcal{N}_{EP_1}||\mathcal{N}_{EP_2}\Big)\Big|_{\substack{(processor_{EP_1}, processor_{EP_2}) \hookrightarrow (P_{EP_1 \to EP_2}, processor_{Proc}) \\ (r^2_{EP_1}, r^2_{EP_2}) \hookrightarrow (p^2_{EP_2}, r^2_{EP_2}) \\ \dots \\ (r^{m_1}_{EP_1}, r^{m_1}_{EP_2}) \hookrightarrow (p^{m_1}_{EP_2}, r^{m_1}_{EP_2})}}$$

$$\dots ||\mathcal{N}_{EP_{q_C}}\Big)\Big|_{\substack{(processor_{Proc}, processor_{EP_{q_C}}) \hookrightarrow (P_{EP_1 \dots EP_{q_C-1} \to EP_{q_C}}, processor_{Proc}) \\ (r^2_{EP_1 \dots EP_{q_C-1}}, r^2_{EP_{q_C}}) \hookrightarrow (p^2_{EP_{q_C}}, r^2_{EP_{q_C}}) \\ \dots \\ (r^{m_{q_C-1}}_{EP_1 \dots EP_{q_C-1}}, r^{m_{q_C-1}}_{EP_{q_C}}) \hookrightarrow (p^{m_{q_C-1}}_{EP_{q_C}}, r^{m_{q_C-1}}_{EP_{q_C}})}} \quad (3)$$

with $\forall k_C \in [1, m_{j_C}]$ and $q_C \geq 2$ if $k_C \geq 2$ then $r^{k_C}_{EP_1 \dots EP_{j_C}} = r^{k_C}_{EP_{j_C+1}}$

$$\mathcal{N}_{IR} = \Bigg(\Big(\mathcal{N}_{CR}||\mathcal{N}_{I_1}\Big)\Big|_{\substack{(r^1_P, r^1_{I_1}) \hookrightarrow (p^1_{I_1}, r^1_{I_1}) \\ \dots \\ (r^{m_1}_P, r^{m_1}_{I_1}) \hookrightarrow (p^{m_1}_{I_1}, r^{m_1}_{I_1})}} \quad \dots ||\mathcal{N}_{I_{q_I}}\Big)\Big|_{\substack{(r^1_{PI_1 \dots I_{q_I-1}}, r^1_{I_{q_I}}) \hookrightarrow (p^1_{I_{q_I}}, r^1_{I_{q_I}}) \\ \dots \\ (r^{m_{q_I}}_{PI_1 \dots I_{q_I-1}}, r^{m_{q_I}}_{I_{q_I}}) \hookrightarrow (p^{m_{q_I}}_{I_{q_I}}, r^{m_{q_I}}_{I_{q_I}})}} \quad (4)$$

with $\forall k_I \in [1, m_{j_I}]$ and $q_I \geq 1$, $r^{k_I}_{PI_{j_I-1}} = r^{k_I}_{I_{j_I}}$

The aim is to verify the limits of the schedulability and the valid values of parameter a (i.e., the period of $T1$). The application is schedulable if each activity always has at most one running instance.

The sufficient condition ensuring that the system is schedulable with a non-preemptive priority policy requires a processor load U such that:

$$U = \sum_{i=1}^{n} (C_i/P_i) \leq 1 \quad (5)$$

with n representing the number of tasks, C_i indicating the worst execution time of each task, and P_i being the period (resp. minimal delay) of each periodic (resp. sporadic) task T_i.

The theoretical expected values (calculated without taking into account the RTOS mechanism) for the a parameter are $a \geq 44$ [25]. We expect that our formal verification on the two deployed application on VxWorks and OSEK/VDX leads to the same result.

C. Formal composition fragment

For the sake of clarity, Figure 6 only shows the behavioral arrangement of task $T3$ ($\mathcal{N}_{T3}$) considering the OSEK/VDX norm (Figure 6(a)) on the left side, and the VxWorks platform (Figure 6(b)) on the right side.

$T3$ has been chosen as an illustration instead of other tasks because it has the lowest priority. Consequently, it presents the most complex case. We can indeed note the presence of inhibition arcs since tasks are scheduled in accordance with a cooperative multitasking non-preemptive priority policy. The $\mathcal{N}_{T3}$ instance has consequently been extended by applying $CoopSched(\mathcal{N}_{T3}, \{T1, T2\})$, for each targeted RTOS.

The body of $T3$ is simplified and contains only one service call to *suspend* (in OSEK/VDX variant) or *pend* (in VxWorks variant).

On both Figure 6(a) and Figure 6(b), roles appear in bold to highlight connection points useful for the composition through the RI_TPN. In a similar manner, the mergeable places connected by a hook-dotted arc are the ones located to compose $T3$ and its body according to equation (2).

The same reasoning is obviously applied to $T1$ and $T2$

before composing them with $T3$ in compliance with our formalization through equations (3) and (4).

D. Application verification

Once the models are composed, they have subsequently been checked using Roméo in order to determine the limit value of a so that the RTES application is schedulable. Given the structure of both nets, the systems are schedulable if, at any time, there is at most one token in each place (the nets are then said to be *safe*). Additionally, Roméo provides the set of values of parameter a for which the property is true. The outcome is given hereafter:

```
———-
Checking property AG[0,inf]bounded(1) on TPN:
"/home/clelionnais/TPN/OSEKVDX_NonPreemptiveApplication.xml"
Waiting for response...
Result:
{a >= 44
}
———-
Checking property AG[0,inf]bounded(1) on TPN:
"/home/clelionnais/TPN/VxWorks_NonPreemptiveApplication.xml"
Waiting for response...
Result:
{a >= 44
}
```

Both results match the theoretical value mentionned earlier. We can thus observe that taking into account RTOS mechanisms does not change the theoretical result in this case. Expected constraints are therefore satisfied.

Other properties could be verified, for which taking RTOS mechanisms into account could have an impact. However, this is beyond the scope of this paper. Alternatively, the same property could also be verified starting from a different design model. For instance, we could attempt to model periodicity with a delay instead of an alarm. In such a case, we would be able to verify that the expected properties are not preserved.

However, the purpose of our present case study is simply to illustrate that we can support different platforms (OSEK/VDX and VxWorks) without changing our formalization rules.

VII. Benefits and Limits

One of the major advantage of SExPIsTools is the multi-platform deployment process. The possibility of capitalizing a large number of RTOS models as a parameter of the process, satisfies both reusability and portability criteria. The role notion presented in this paper encourages us in this way to provide more genericity to our transformation rules.

This role notion also fulfills the maintainability requirements. Composition rules have been written independently of the RTOS modeling. In addition, the formalization of these rules could have been done without dealing with other stakeholders concerns. Our Algorithm 1 has been strengthened, detecting errors (i.e., TPN fragments composition ambiguity within rules). As a result, the correctness of the generation process has been improved.

Furthermore, a deployment on two RTOS with different mechanisms has been achieved to show our strategy. The same safety property of schedulability has been verified on both deployments. This illustrates both genericity and correctness of deployed application model construction in TPN.

Another important point is the behavioral modeling in TPN. This results in the possibility to apply verification activities. Moreover, the verification of time properties such as RTES time constraints is possible.

However, to date, this synthesis is an ongoing sketch of proof. The purpose of such a work is to demonstrate the feasability to develop a versatile tool suite. This experimentation must deal with other aspects by considering:

- **more complex RTOS mechanisms** such as preemption, priority ceiling protocol or special queues of message box;
- **other RTOS descriptions** such as ARINC-653 [26], which presents other concepts (e.g., memory partition);
- **other verifications** such as time constraints;
- **more precisely the application**, so that it is not seen as just an ordered sequence of called services.

VIII. Conclusion

In this paper, we have presented a first formalization of the formal model generation process of our SExPIsTools tool suite. As its name suggests, this process generates, from high-level design descriptions, a formal model of the deployed application on a specific RTOS.

The presented formalization focuses on both instantiation and composition rules of the generation process. Indeed, several formal model fragments describing parts of the RTOS and RTES behaviors need to be instantiated and composed. This results in a verifiable global model of the deployed application. The composition rules are independent of a specific RTOS thanks to the notion of role. This notion is an essential point of our strategy and represents a major benefit compared to other existing approaches.

This formalization leads to the definition of a new class of Petri Net, the Time Petri Net with roles and read/inhibitor arcs. A new operator, compared to our previous work [1], has been defined. It allows to model the cooperative scheduling of non-preemptive tasks. This comes to strengthen the instantiation of RI_TPN behavioral fragments according to a priority policy before composing them.

An example of a composition of an application with two RTOS (OSEK/VDX and VxWorks), taking into account the different behavior of the platform, has been given.

Future prospects are scheduled in order to meet the needs identified in Section VII. We are exploring the possibility of extending the formalization with other model classes such as Scheduling TPN [27], where both cooperative and preemptive scheduling are considered.

REFERENCES

[1] C. Lelionnais, M. Brun, J. Delatour, O. H. Roux, and C. Seidner, "Formal Composition Based on Roles within a Model Driven Engineering Approach," in Advances in System Testing and Validation. Venice, Italy: IARIA, Nov. 2013, pp. 27–32. [Online]. Available: http://hal.archives-ouvertes.fr/hal-00941024 and www.thinkmind.org

[2] J. Miller and J. Mukerji, "Model Driven Architecture (MDA) Guide, version 1.0.1." Tech. Rep., June 2003.

[3] J. C. Maeng, D. Na, Y. Lee, and M. Ryu, "Model-Driven Development of RTOS-Based Embedded Software," in Proceedings of the 21st International Conference on Computer and Information Sciences, ser. ISCIS'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 687–696.

[4] B. Kim, I. Lee, L. T. X. Phan, and O. Sokolsky, "Platform dependent code generation of real-time embedded software," in Proceedings of the 4^{th} ACM/IEEE International Conference on Cyber-Physical Systems, ser. ICCPS '13. New York, NY, USA: ACM, 2013, pp. 246–246.

[5] W. El Hajj Chehade, "Contribution to Multiplatform Deployment of Multitasking Applications by High-Level Execution Services Behavioral Modeling," Ph.D. dissertation, Laboratoire d'Ingénierie Dirigée par les Modèles des Systèmes Temps Réels Embarqués (LISE) - CEA Saclay, 2011.

[6] Object Management Group (OMG), "UML Profile for Modeling and Analysis of Real Time and Embbeded Systems (MARTE), version 1.1." Tech. Rep., June 2011.

[7] F. Thomas, S. Gérard, J. Delatour, and F. Terrier, "Software Real-Time Resource Modeling," in Embedded Systems Specification and Design Languages. Springer, 2008, pp. 169–182.

[8] J. Kim, I. Kang, J.-Y. Choi, I. Lee, and S. Kang, "Formal synthesis of application and platform behaviors of embedded software systems," Software & Systems Modeling, 2013, pp. 1–21.

[9] A. Pinto, "Metropolis Design Guidelines," University of California, Berkeley, USA, Tech. Rep., Nov. 2004.

[10] J. Davis, "GME: The Generic Modeling Environment," in Companion of the 18^{th} Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications, ser. OOPSLA '03. New York, NY, USA: ACM, 2003, pp. 82–83.

[11] E. A. Lee, "Overview of the Ptolemy Project," EECS Department, University of California, Berkeley, Tech. Rep. UCB/ERL M03/25, July 2003.

[12] M. Brun, "Contribution to the Software Execution Platform Integration during an Application Deployment Process," Ph.D. dissertation, École Centrale de Nantes, Nantes, France, Oct. 2010.

[13] M. Brun and J. Delatour, "Contribution on the Software Execution Platform Integration During an Application Deployment Process," in First Topcased Day, Toulouse, France, Feb. 2011.

[14] C. Lelionnais, M. Brun, J. Delatour, O. H. Roux, and C. Seidner, "Formal Behavioral Modeling of Real-Time Operating Systems," in 14^{th} Int. Conf. Ent. Information Systems - Model Driven Development for Information Systems (MDDIS 2012), Wroclaw, Poland, June 2012.

[15] F. Thomas, J. Delatour, F. Terrier, and S. Gerard, "Towards a Framework for Explicit Platform-Based Transformations," in 11^{th} IEEE International Symposium on Object Oriented Real-Time Distributed Computing (ISORC), May 2008, pp. 211–218.

[16] OSEK/VDX Group, "OSEK/VDX Operating System Specification, version 2.2.3," Tech. Rep., Feb. 2005, http://www.osek-vdx.org/.

[17] WindRiver, "VxWORKS Programmer's Guide, version 6.9." Tech. Rep., Feb. 2011.

[18] P. M. Merlin, "A Study of the Recoverability of Computing Systems," Ph.D. dissertation, 1974, aAI7511026.

[19] M. Boyer and O. H. Roux, "On the Compared Expressiveness of Arc, Place and Transition Time Petri Nets," Fundamenta Informaticae, vol. 88, no. 3, 2008, pp. 225–249.

[20] F. Taïani, M. Paludetto, and J. Delatour, "Composing Real-Time Objects: A Case for Petri Nets and Girard's Linear Logic," in Proceedings of the 4^{th} IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, ISORC-2001. IEEE, 2001, pp. 298–305.

[21] F. Peres, B. Berthomieu, and F. Vernadat, "On the Composition of Time Petri Nets," Discrete Event Dynamic Systems, vol. 21, no. 3, Sept. 2011, pp. 395–424.

[22] C. A. Petri, "Kommunikation mit Automaten," Ph.D. dissertation, Institut für Instrumentelle Mathematik, Bonn, 1962.

[23] G. Bucci, A. Fedeli, L. Sassoli, and E. Vicario, "Timed State Space Analysis of Real-Time Preemptive Systems," IEEE Transactions on Software Engineering, vol. 30, no. 2, Feb. 2004, pp. 97–111.

[24] D. Lime, O. H. Roux, C. Seidner, and L. M. Traonouez, "Roméo: A Parametric Model-Checker for Petri Nets with Stopwatches," in 15^{th} International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2009), ser. Lecture Notes in Computer Science, S. Kowalewski and A. Philippou, Eds., vol. 5505. York, United Kingdom: Springer, Mar. 2009, pp. 54–57.

[25] A. Jovanović, D. Lime, and O. H. Roux, "Integer Parameter Synthesis for Timed Automata," in 19^{th} International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2013), ser. Lecture Notes in Computer Science, N. Piterman and S. Smolka, Eds., vol. 7795. Rome, Italy: Springer, Mar. 2013, pp. 401–415.

[26] Airlines Electronic Engineering Committee, "Avionics Application Software Standard Interface, ARINC Specification 653-1," Tech. Rep., Oct. 2003, aeronautical radio INC., Annapolis, Maryland, USA.

[27] D. Lime and O. H. Roux, "Formal Verification of Real-Time Systems with Preemptive Scheduling," Journal of Real-Time Systems, vol. 41, no. 2, 2009, pp. 118–151, copyright Springer.

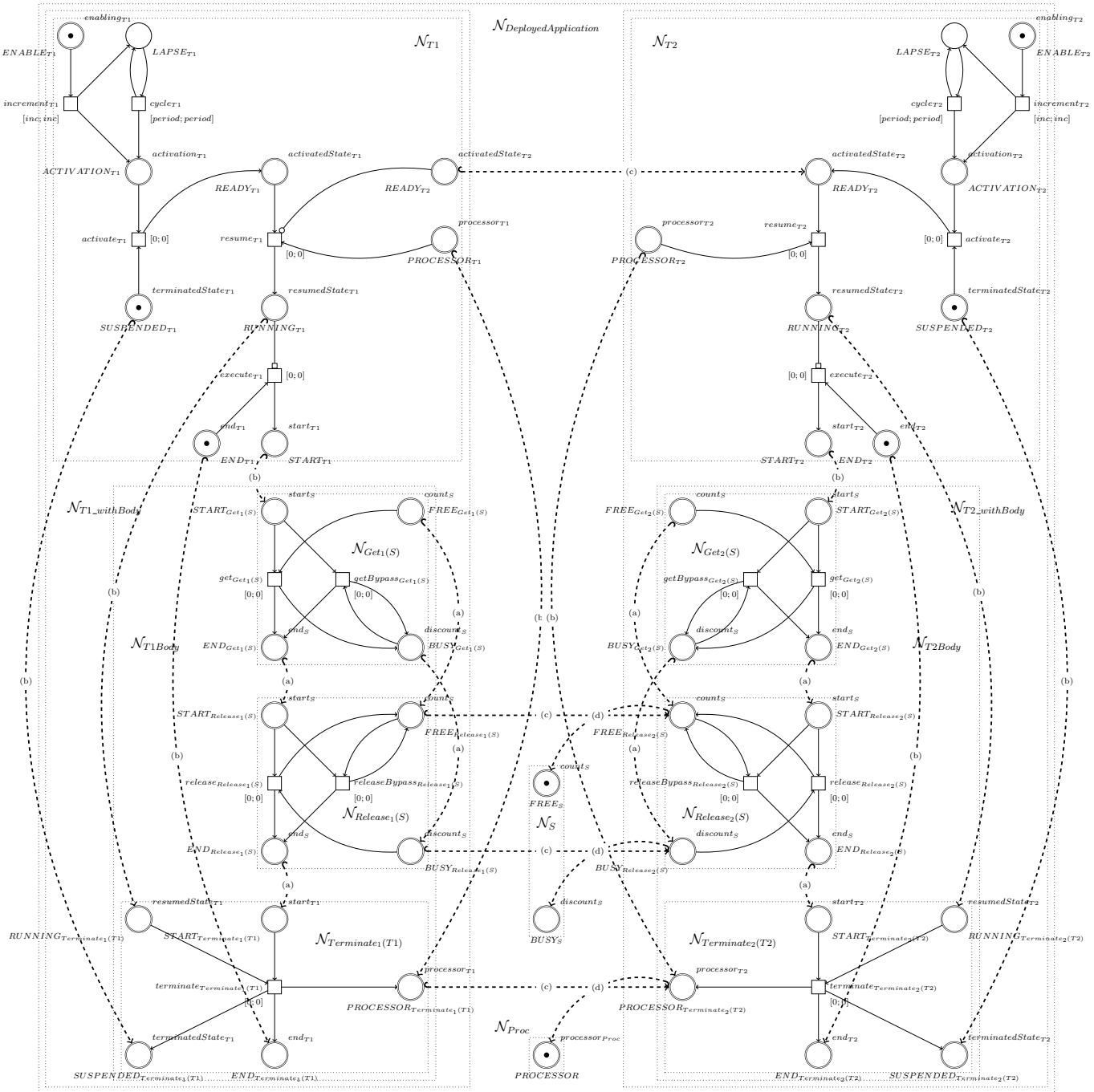
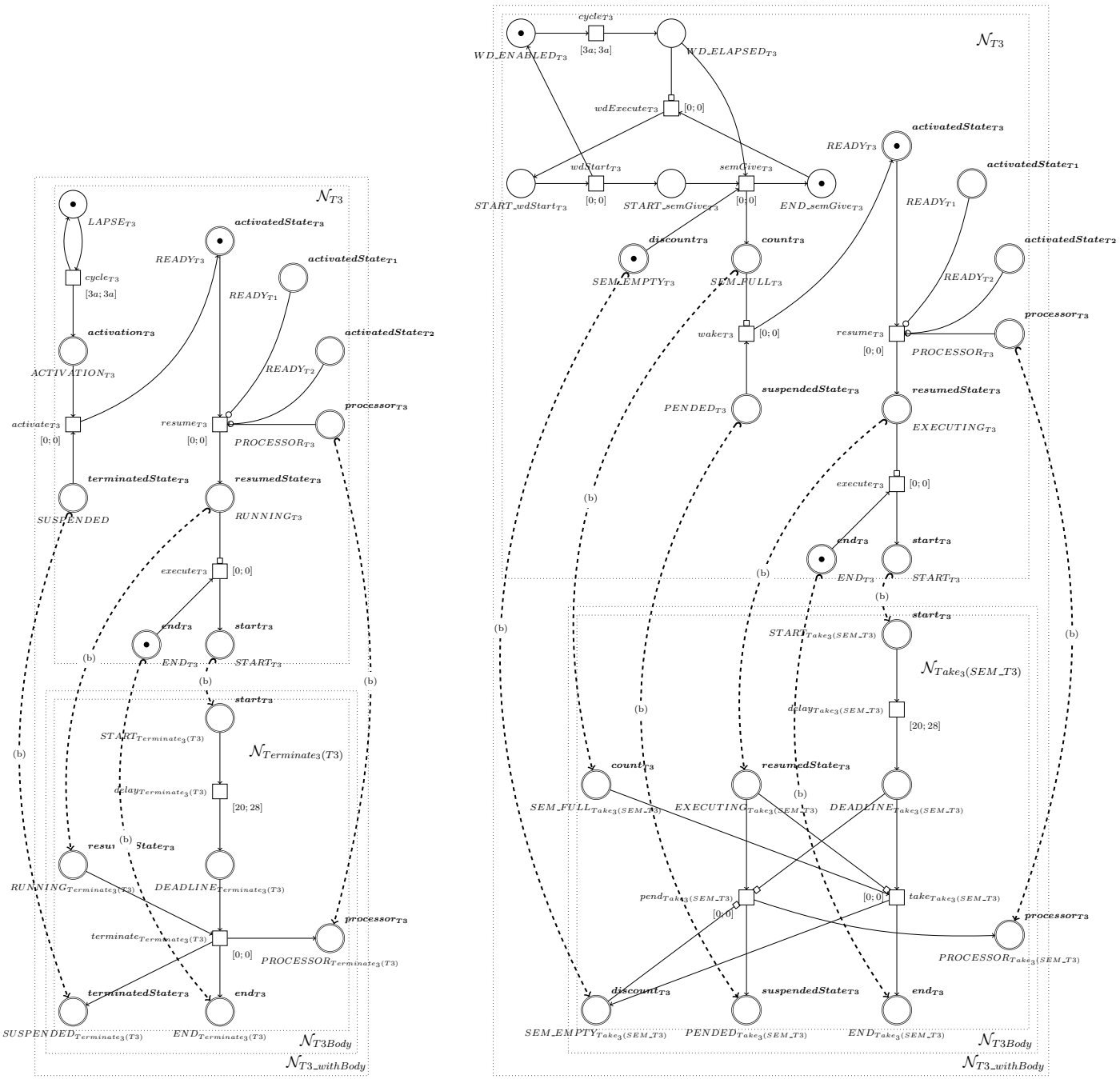


Fig. 5: Deployed application of semaphore sharing composed in RI_TPN



(a) OSEK/VDX (b) VxWorks

Fig. 6: RI_TPN of periodic task implemented on two different RTOS