Formal Methods for Systems Engineering Behavior Models

Charlotte Seidner, Olivier Henri Roux

Formal Methods for Systems Engineering Behavior Models

Charlotte Seidner and Olivier H. Roux

***Abstract*—Safety analysis in Systems Engineering (SE) processes, as usually implemented, rarely relies on formal methods such as model checking since such techniques, however powerful and mature, are deemed too complex for efficient use. This paper thus aims at improving the verification practice in SE design: considering the widely-used model of EFFBDs (Enhanced Function Flow Block Diagrams), it formally establishes its syntax and behavioral semantics. It also proposes a structural translation of EFFBDs to transition time Petri nets (TPNs); this translation is then proved to preserve the behavioral semantics (i.e. timed bisimilarity). After proving results on the boundedness of the resulting TPNs, it was possible to extend a number of fundamental properties (such as the decidability of liveness, state-access, etc.) from bounded TPNs to so-called *bounded EFFBDs* . Finally, these results led to both implementing and integrating a formal verification tool within a development platform for system design for defense applications and in which the underlying complexity is totally concealed from the end-user.**

***Index Terms*—Systems Engineering, time Petri nets, embedded system design, formal verification, timed bisimulation.**

I. INTRODUCTION

SYSTEMS Engineering (SE) is defined by the INCOSE[1] as an "interdisciplinary approach" to perform the "realization of successful systems", from the definition of "customer needs and required functionality early in the development cycle" to "design synthesis and validation" [1]–[3]. Application fields, and particularly in embedded system design, are quite numerous: defense, aerospace engineering, road or railroad transport, computer science, etc. However, the development of ever larger and more complex systems has made safety and dependability assessment most essential. Verification and validation processes are then used to assess some (temporal) properties, which are usually classified into safety *("something bad will never occur")* and liveness *("something good will eventually happen")* properties ([4]).

To assess the system safety, a common practice consists in performing simulations on a behavioral model of the system. However, this analysis cannot be exhaustive, even on "reasonably-sized" systems and carries the risk of missing potential safety-critical situations. On the other hand, model checking techniques, where specifications to be respected are formally expressed and confronted to a formal model of the system, may address the simulation method shortcomings. In addition, when considering high-level models usually handled in SE processes and rather than develop specific methods (along with a complex body of theory), it appears more efficient first to supply them with a behavioral semantics, then to translate them into formal lower-level models, such as Petri nets, on which model checking techniques and results are well established. As the translation is designed to preserve the behavior and properties to be checked, the method benefits from the powerful tools and results obtained on the lower-level model to assess high-level, temporal properties.

This work focuses on the use of *Enhanced Function Flow Block Diagrams* (EFFBDs, [5]), a graphical formalism widely used in SE projects developed by major companies such as the NASA, BOEING or AIRBUS [6]. Although no formal semantics was ever established for this model, it has shown to be consistent and mature over the last decades and its implementation in various design and modeling tools provides *de facto* a consistent semantics. Concerning the lower-level model, time Petri nets (TPNs, [7]) appeared well-suited to the EFFBD structure and the use of a model checker such as ROMÉO [8] an adequate tool for the identified needs [9]. Finally, TPN (as well as timed automaton or TA, [10]) semantics is expressed as a time transition system (TTS, [11]). It is thus pertinent to describe EFFBD semantics as another TTS and then to define a translation from EFFBD to TPN (or TA) that preserves the behavioral semantics[2].

A. Related works

Over the last decades, a large number of research efforts have focused on safety and dependability methods. In the software engineering field, in particular, it led to numerous works in safety and dependability analysis for UML-based system design (see for instance [12] and [13]).

In SE (which application field is even broader than software engineering), similar projects have been emerging during the last few years. Amongst the modeling tools used or developed within the SE community, the *Systems Modeling Language* (SysML, [14]) should be mentioned as the result of the Object Management Group (OMG) and the INCOSE joint initiative[3]. SysML is also widely based on a subset of UML. However, the well-known issue of the lack of precise semantics in UML has not yet been solved in SysML [15].

Recently, the *Architecture Analysis and Design Language* (AADL, [16]) has gained importance as a powerful modeling tool by allowing a number of formal analyses [17]. However, it appears that the language still suffers from semantical imprecisions. For instance, whereas some dynamical aspects

[1] INternational COuncil on Systems Engineering.

[2] It would actually be easier to express EFFBD semantics as a TA or a TPN as the expressive power of both these models is greater. However, TA and TPN expressivities (w.r.t. bisimulation) are not comparable. Therefore, there is no translation from "general" (i.e. potentially unbounded) TPN into TA (and conversely). As a consequence, a semantics given with one of these models would be difficult to exploit with the other one.

[3] The specification is actually inspired by the EFFBD formalism.

in AADL are described with hybrid automata, others are not covered, thus allowing different interpretations. Moreover, AADL still lacks the maturity of models such as EFFBDs which go back, in their simplest form, to the 1950's.

Lastly, the author of [18] proposed a semi-formal semantics for FFBDs (a subset of EFFBDs) and a translation to Petri nets (PNs) preserving the "causal chain representation" i.e. a non temporal bisimilarity. However, no formal proof is given for the translation; in addition, FFBDs are strictly less expressive than EFFBDs as they cannot represent data flows nor resource usage.

B. Contribution – outline of the paper

This paper formally establishes EFFBDs syntax and behavioral semantics. To the authors' knowledge, this formalization was never proposed so far. It also proposes a structural translation of EFFBDs to transition time Petri nets (TPNs); this translation is proved to preserve behavioral semantics (i.e. timed bisimilarity). After proving results on resulting TPNs boundedness, it was possible to extend a number of fundamental properties (such as the decidability of liveness, state-access, etc.) from bounded TPNs to the so-called bounded EFFBDs. These results eventually led to both the implementation and integration of an operational formal verification tool within a development platform, used in systems design for defense applications.

Section II gives an informal presentation of the EFFBD formalism, whereas Section III proposes a formal definition of the formalism including its semantics. Due to the formalism richness, however, parts of the description are shown in Appendix A. Section IV briefly introduces the TPNs and presents the patterns used to translate EFFBDs into TPNs. Section V establishes some properties of the translation and particularly the method correctness. Section VI gives an insight of the tools developed in application of these results and Section VII concludes the paper by presenting further research works.

II. An informal presentation of the EFFBDs

In order to provide an efficient specification of both functional and data control, systems engineers often use relatively simple graphical representations such as EFFBDs. These diagrams provide the designer with an easy framework to describe the behavior of complex, distributed, hierarchical, concurrent and communicating systems. EFFBDs describe the functions performed by the system and the order in which they are to be executed. This order is specified through the functions dynamic parameters (i.e. their *execution duration*), control environment *(control constructs)* and data (or items) environment.

EFFBDs offer a large range of control constructs such as parallel branches, loops, selection branches, etc[4]. This section provides an informal presentation of available control structures and item controls. Fig. 1 shows an example of an EFFBD; the diagram does not correspond to an actual system but rather illustrates the main features of the EFFBD formalism.

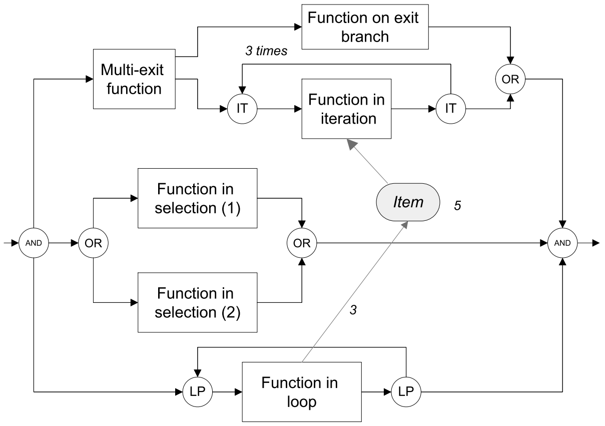

Fig. 1. Example of an EFFBD *(adapted from [6])*

The description given in this section is largely inspired by the implementation realized in the Core® software tool, a system design platform developed by Vitech Corp.[5]

*A. Parallel structures (*AND *nodes)*

A *parallel structure* consists of two AND nodes and n parallel branches ($n \geq 2$). After the structure has been entered, the first construct of every branch is enabled. The structure is exited as soon as the last construct on every branch has been completely executed; this rule may induce some synchronization-waiting states.

*B. Selection structures (*OR *nodes)*

A *selection structure* consists of two OR nodes and n select branches ($n \geq 2$). When the structure is entered, one of the branches is selected and its first construct is enabled. The structure is exited as soon as the last structure of the chosen branch has been completely executed.

The *selection process*, i.e. the set of rules determining the branch choice, takes various forms in the different implementations such as *selection probabilities* or *internal scripts*, all of which are not part of the EFFBD formalism. To make the study simpler, this paper considers that every branch can be selected (no information on the probabilities is given).

*C. Iteration structures (*IT *nodes)*

An *iteration structure* consists of two IT nodes surrounding an iterated branch, an internal counter and a maximal iteration value i_{max} ($i_{max} \geq 2$). Entering the iteration enables the first construct of the iterated branch and initializes the counter to 1. When the last construct of the branch is exited, two behaviors are then possible:

- if the counter value is strictly less than i_{max}, the first construct of the iterated branch is enabled again and the counter incremented;
- else, the structure is exited.

[4] Only *well-formed* diagrams are valid which implies that constructs are exited in the reverse order they were entered.

[5] http://www.vitechcorp.com

D. Loop structures (LP nodes)

A *loop construct* consists of two LP nodes surrounding a loop branch. Entering the loop enables the first construct of the loop branch; when the last construct of the branch is exited, the control returns to the loop opening. The behavior thus defined is infinite.

E. Functions and items

Function constructs (represented by rectangular nodes) are the system functions containers and thus the model core. Function constructs can be *single-exit* or *multi-exit*, in which case they have n exit branches ($n \geq 2$) converging on a closing OR node. As multi-exit functions are equivalent to single-exit functions followed by a selection structure, they shall not be discussed hereunder.

A function can start its execution if and only if it is both *enabled* (by the control environment) and *triggered* (by the item environment). Functions wait for all their input items to be available in proper quantity before consuming them and beginning their execution. Execution durations are described by probability laws; however, for the sake of simplicity, it is considered here that execution durations belong to a time interval $[\alpha, \beta]$. Taking more complex probability laws into consideration would induce the resort to stochastic models such as Generalized Stochastic Petri Nets, which is beyond the scope of this paper. When the execution has been completed, the (potential) output items are produced and the function construct is exited.

F. Other structures

EFFBDs support a few additional structures, such as *sub-scenarios* or *termination* structures. The former can be brought down to a simplifying and compact model design but add no expressivity to the model and shall therefore not be discussed hereunder.

Termination structures help modeling the early termination of parallel branches or the forced exit from a loop construct; they do add to the expressivity of the model but, since this is preliminary work, this feature shall not be considered henceforward[6]. However, these structure types are both extensively presented in [19].

III. Formal presentation of the EFFBDs

This section presents EFFBDs syntax and behavioral semantics. To the authors' knowledge, this semantics has never been formally established so far. Only a simplified version of the formalism is presented here; a complete version can be found in [19]. Most of the formalization presented here (as well as in the following section) were designed so as to provide a straightforward proof of the equivalence between an EFFBD and its corresponding TPN.

A. Notations and definitions

The notations adopted in this paper are the following:

- $\mathbb{N}$, $\mathbb{Z}$ and $\mathbb{R}_{\geq 0}$ are respectively the sets of natural integers, integers and positive real numbers; $\mathbb{Z}^*$ is the set $\mathbb{Z} \setminus 0$;
- the usual operators $+, -, <, \leq, \geq, >, =$ are extended (element-wise) to vectors of A^n with $n \in \mathbb{N}$ and $A = \mathbb{N}, \mathbb{Z}, \mathbb{R}_{\geq 0}$;
- given two sets A and B, B^A is the set of applications from A to B;
- $\setminus$ denotes the set minus operator;
- $\overline{0}$ is the null vector and $\emptyset$ the empty set.

Unless otherwise specified, opening brackets stand for "and" conditions.

The behavioral semantics considered in the following sections are described as TTS, the definition of which is recalled below. TTS are in fact usual transition systems with two types of labels (discrete labels for modeling events and positive real labels for time elapsing).

Definition III.1 (Timed Transition System [11])**.** *A* timed transition system *over the set of actions Σ is a tuple $(Q, Q_0, \Sigma, \rightarrow)$ where Q is a set of states, $q_0 \in Q$ the initial state, Σ a finite set of actions disjoint from $\mathbb{R}_{\geq 0}$ and $\rightarrow \subseteq Q \times (\Sigma \cup \mathbb{R}_{\geq 0}) \times Q$ a set of edges*[7]*.*

The definition of the strong-timed bisimulation which shall be needed in Section V-C, is recalled below.

Definition III.2 (Strong-timed bisimulation [20])**.** *Let $S_1 = (Q_1, q_0^1, \Sigma, \rightarrow_1)$ and $S_2 = (Q_2, q_0^2, \Sigma, \rightarrow_2)$ be two TTS and $\sim$ a binary relation*[8] *over $Q_1 \times Q_2$. $\sim$ is a* strong-timed bisimulation *between S_1 and S_2 if:*

R_0 $q_0^1 \sim q_0^2$

R_d $\forall q_1, q_1', q_2, d \in \mathbb{R}_{\geq 0}$ s.t. $q_1 \xrightarrow{d}_1 q_1'$ and $q_1 \sim q_2$, $\exists q_2'$ s.t. $q_2 \xrightarrow{d}_2 q_2'$ and $q_1' \sim q_2'$

R_d' $\forall q_2, q_2', q_1, d \in \mathbb{R}_{\geq 0}$ s.t. $q_2 \xrightarrow{d}_2 q_2'$ and $q_1 \sim q_2$, $\exists q_1'$ s.t. $q_1 \xrightarrow{d}_1 q_1'$ and $q_1' \sim q_2'$

R_σ $\forall q_1, q_1', q_2, \sigma \in \Sigma$ s.t. $q_1 \xrightarrow{\sigma}_1 q_1'$ and $q_1 \sim q_2$, $\exists q_2'$ s.t. $q_2 \xrightarrow{\sigma}_2 q_2'$ and $q_1' \sim q_2'$

R_σ' $\forall q_2, q_2', q_1, \sigma \in \Sigma$ s.t. $q_2 \xrightarrow{\sigma}_2 q_2'$ and $q_1 \sim q_2$, $\exists q_1'$ s.t. $q_1 \xrightarrow{\sigma}_1 q_1'$ and $q_1' \sim q_2'$

B. Syntax of an EFFBD

The syntax of both untimed and timed EFFBDs is given hereunder.

Definition III.3 (Untimed EFFBD)**.** *An* untimed EFFBD *is a tuple $\mathcal{E}_U = (\mathcal{N}, \mathcal{I}, \mathcal{A}, count, n_0, I_0)$ where:*

- *$\mathcal{N}$ is a finite, non-empty set of* nodes *defined as the union of the following subsets:*
  - *AND_{in} and AND_{out}: the set of opening and closing* AND *nodes;*
  - *OR_{in} and OR_{out}: the set of opening and closing* OR *nodes;*

[6] The translation of such structures in TPNs actually involve *reset* arcs.

[7] $(q, \bullet, q') \in\rightarrow$ is also written $q \xrightarrow{\bullet} q'$.

[8] $(q_1, q_2) \in\sim$ is written $q_1 \sim q_2$.

  - *IT_{in} and IT_{out}: the set of opening and closing* IT *nodes;*
  - *LP_{in} and LP_{out}: the set of opening and closing* LP *nodes;*
  - *FC: the set of function constructs.*
- *$\mathcal{I}$ is a finite set of* items*;*
- *$\mathcal{A}$ is a finite, non-empty set of* control *and* item arcs*: $\mathcal{A} = \mathcal{A}_C \cup \mathcal{A}_I$ where:*
  - *$\mathcal{A}_C \subset \mathcal{N} \times \mathcal{N}$ is the set of control arcs[9];*
  - *$\mathcal{A}_I \subseteq FC \times \mathcal{I} \times \mathbb{Z}^*$ is the set of item arcs;*
- *$count : IT_{in} \rightarrow \mathbb{N} \setminus \{0, 1\}$ is the* counter function *giving for each iteration construct the maximum number of iterations;*
- *$n_0 \in \mathcal{N}$ is the* initial node*;*
- *$I_0 \in \mathbb{N}^{\mathcal{I}}$ gives the* initial item amounts.

Let $(f, item, k)$ be an element of $\mathcal{A}_I$; if $k < 0$, $item$ is consumed by f in the quantity k and if $k > 0$, $item$ is produced by f in the quantity k.

Definition III.4 (Timed EFFBD)**.** *A* timed EFFBD *is a tuple $\mathcal{E}_T = (\mathcal{E}_U, \alpha, \beta)$ where:*

- *$\mathcal{E}_U$ is an untimed EFFBD;*
- *$\alpha \in \mathbb{N}^{FC}$ and $\beta \in (\mathbb{N} \cup \{\infty\})^{FC}$ are respectively the* lower bound *and the* upper bound *of the* function execution duration mappings *($\alpha \leq \beta$).*

C. Additional definitions

Some additional definitions are provided below in order to make the subsequent semantic expressions simpler.

Definition III.5 (Predecessors and successors of a node)**.** *Let $\mathcal{N}$ be a set of nodes and $\mathcal{A}_C$ a set of control arcs. The* predecessors *of a node $n \in \mathcal{N}$ form the set $Pre(n)$ defined by:*

$$Pre(n) = \{n' \in \mathcal{N} \mid (n', n) \in \mathcal{A}_C\}$$

The successors *of a node $n \in \mathcal{N}$ form the set $Post(n)$ defined by:*

$$Post(n) = \{n' \in \mathcal{N} \mid (n, n') \in \mathcal{A}_C\}$$

These definitions are extended to describe the *content* of constructs, so as to formalize the notion of well-formed EFFBDs.

Definition III.6 (Extended predecessors)**.** *The operator Pre is extended to $\widehat{pre}(E)$ where E is a set: $\widehat{pre}(E)$ is defined by $\widehat{pre}(E) = \bigcup_{e \in E}\{Pre(e)\}$*

The operator $\overrightarrow{pre}_i$ (for $i \geq 0$) is defined for the elements x and y by $\overrightarrow{pre}_i(x, y) =$
$$\begin{cases} \{x\} \setminus y \text{ if } i = 0 \\ \overrightarrow{pre}_{i-1}(x, y) \cup (\widehat{pre}(\overrightarrow{pre}_{i-1}(x, y))) \setminus y \text{ otherwise} \end{cases}$$

$\overrightarrow{pre}_*(x, y)$ denotes the fixed point : $\overrightarrow{pre}_*(x, y) = \overrightarrow{pre}_j(x, y)$ s.t. $\overrightarrow{pre}_j(x, y) = \overrightarrow{pre}_{j+1}(x, y)$

$\widehat{pre}(E)$ is the set of the predecessors of E obtained in one step; $\overrightarrow{pre}_i(x, y)$ is the set of the predecessors of x in i steps, excluding y and its predecessors. Finally, $\overrightarrow{pre}_*(x, y)$ is the set of the nodes between y and x, in a well-formed model.

The extended successor operator $\overrightarrow{post}_*$ is likewise defined by replacing Pre by $Post$ in previous definitions.

The definition of a well-formed EFFBD ensues:

Definition III.7 (Well-formed EFFBDs)**.** *An EFFBD is* well-formed *iff for all opening nodes $n \in AND_{in} \cup OR_{in} \cup IT_{in} \cup LP_{in}$, there is exactly one closing node of the same type $\overline{n}$ such that:*

- *(Atomicity) $\overrightarrow{post}_*(n, \overline{n}) \setminus n = \overrightarrow{pre}_*(\overline{n}, n) \setminus \overline{n}$. This set is denoted $atom(n, \overline{n})$ and represents the nodes contained in the structure delimited by n and $\overline{n}$[10];*
- *(Imbrication) $\forall e \in atom(n, \overline{n})$, e being an opening node, there is exactly one closing node of the same nature $\overline{e}$ such that $atom(e, \overline{e}) \subset atom(n, \overline{n})$ i.e. $\overline{e} \in atom(n, \overline{n})$;*
- *(Continuity) $\forall e \in Post(n)$, $\exists e' \in Pre(\overline{n})$ s.t. $e' \in \overrightarrow{post}_*(e, \overline{n})$ and $\forall e \in Pre(\overline{n})$, $\exists e' \in Post(n)$ s.t. $e' \in \overrightarrow{pre}_*(\overline{n}, e)$*

In a well-formed EFFBD, a *branch* $\langle n, e, \overline{n} \rangle$ comprised between nodes n and $\overline{n}$ and beginning with node e ($e \in Post(n)$) is then the set $\overrightarrow{post}_*(e, \overline{n})$. In addition, the set of loop structures (i.e. the nodes couples surrounding a loop branch) is denoted LP; the set of iteration structures is denoted IT.

Finally, the item consuming relation is defined as follows.

Definition III.8 (Consuming relations)**.** *Let FC be a set of function constructs, $\mathcal{I}$ a set of items and $\mathcal{A}_I$ a set of item arcs. The* consuming relation *$Cons : FC \rightarrow \mathbb{N}^{\mathcal{I}}$ is defined for all $f \in FC$ and $A \in \mathcal{I}$ by:*

$$Cons(f)[A] = \begin{cases} |k| \text{ if } \exists\, k < 0 \text{ s.t. } (f, A, k) \in \mathcal{A}_I \\ 0 \text{ otherwise} \end{cases}$$

The producing relation $Prod : FC \rightarrow \mathbb{N}^{\mathcal{I}}$ is similarly defined for $k > 0$.

D. Semantics of an EFFBD

The behavioral semantics of untimed EFFBDs is defined as a *transition system* (TS). The TS *states* are triplets that represent the node activity, the iteration counters and the item levels[11]. The activity $A(n)$ of a node n takes its value in the set $\mathbb{A} = \{inactive, enabled, executing, executed\}$ if $n \in FC$ and in the set $\mathbb{A}^* = \{inactive, enabled, executed\}$ otherwise. The activity $executed$ denotes the fact that a node preceding an AND_{out} node has completed its own execution but is waiting for the other branches to complete their execution. As functions must be in the $executing$ state before becoming $executed$ or $inactive$, they are the only nodes that cannot directly transit from $enabled$ to $executed$. In addition, in the case of closing IT and LP nodes, control always comes back to the corresponding opening node, even if (in the iteration case) the proper number of iterations was reached, hence an additional constraint on these nodes.

[9] Note that $\forall n \in \mathcal{N}, (n, n) \notin \mathcal{A}_C$.

[10] If $\overrightarrow{post}_*(n, \overline{n}) \setminus n \neq \overrightarrow{pre}_*(\overline{n}, n) \setminus \overline{n}$, then $atom(n, \overline{n}) = \emptyset$ by convention.

[11] They either represent the current quantity of a resource or the number of times some data was produced without being consumed.

The value of the counter relative to an iteration is defined as the number of times the first construct on the iterated branch was enabled.

In order to make reading easier, the semantic rules are decomposed in propositions $\mathcal{P}_x$, according to the nature of the processed node. Only $\mathcal{P}_{gen}$,$\mathcal{P}_F$ and $\mathcal{P}'_F$ are provided here; the other rules are given in Appendix A. The former describes the generic rule *"each successor of the processed node is enabled; other nodes, counters and item levels are not affected"*. $\mathcal{P}_F$ describes the function execution start (including input item consuming) and $\mathcal{P}'_F$ describes the function execution end (including output item producing).

Definition III.9 (Semantics of an untimed EFFBD)**.** *The semantics of an untimed EFFBD $\mathcal{E}_U = (\mathcal{N}, \mathcal{I}, \mathcal{A}, count, n_0, I_0)$ is a tuple $||\mathcal{E}_U|| = (S, s_0, \mathcal{N}, \rightarrow)$ where:*

- *$S \subseteq \mathbb{A}^{\mathcal{N}} \times \mathbb{N}^{IT_{in}} \times \mathbb{N}^{\mathcal{I}}$ is the set of the states reachable by iteratively applying $\rightarrow$ from s_0;*
- *$s_0 = (A_0, \overline{0}, I_0)$ is the* initial state*, where $A_0(n_0) = enabled$ and $A_0(n) = inactive$ for $n \neq n_0$;*
- *$\rightarrow \subseteq S \times \mathcal{N} \times S$ is the (discrete)* transition relation*, defined $\forall n \in \mathcal{N}$ by:*

$$(A, C, I) \xrightarrow{n} (A', C', I') \Leftrightarrow$$
$$\begin{cases} A'(n) = NextAct \\ \begin{cases} (A(n) = enabled \wedge PreCondition) \\ \text{or } (n \in FC \wedge A(n) = executing \wedge \mathcal{P}'_F) \end{cases} \end{cases}$$

with:

$$NextAct = \begin{cases} executed \text{ if } \begin{cases} n \notin IT_{out} \cup LP_{out} \\ Post(n) \cap AND_{out} \neq \emptyset \\ A(n) = enabled \Rightarrow n \notin FC \end{cases} \\ executing \text{ if } (n \in FC \wedge A(n) = enabled) \\ inactive \text{ otherwise} \end{cases}$$

$$PreCondition \begin{cases} n \in AND_{in} \cup OR_{out} \cup LP_{in} \Rightarrow \mathcal{P}_{gen} \\ n \in AND_{out} \Rightarrow \mathcal{P}_{A_o} \\ n \in OR_{in} \Rightarrow \mathcal{P}_{O_i} \\ (n \in IT_{in} \wedge C(n) < count(n)) \Rightarrow \mathcal{P}_{I_i} \\ (n \in IT_{in} \wedge C(n) = count(n)) \Rightarrow \mathcal{P}'_{I_i} \\ n \in IT_{out} \Rightarrow \mathcal{P}_{I_o} \\ n \in LP_{out} \Rightarrow \mathcal{P}_{L_o} \\ n \in FC \Rightarrow \mathcal{P}_F \end{cases}$$

$NextAct$ thus describes the resulting activity of the processed node (most of the time, it becomes inactive). Please note the set of conditions for the $executed$ cases, as explained above. $PreCondition$ describes the conditions to fulfill in order to process the node; it is a set of mutually exclusive propositions. Defining $\mathcal{N}^n$ as the set $\mathcal{N} \setminus \{n\}$, the definition of propositions $\mathcal{P}_{gen}$, $\mathcal{P}_F$ and $\mathcal{P}'_F$ are:

$$\mathcal{P}_{gen} \begin{cases} \forall n' \in \mathcal{N}^n, A'(n') = \begin{cases} enabled \text{ if } n' \in Post(n) \\ A(n') \text{ otherwise} \end{cases} \\ C' = C, I' = I \end{cases}$$

$$\mathcal{P}_F \begin{cases} \forall n' \in \mathcal{N}^n, A'(n') = A(n') \\ C' = C \\ (I \geq Cons(n)) \wedge (I' = I - Cons(n)) \end{cases}$$

$$\mathcal{P}'_F \begin{cases} \forall n' \in \mathcal{N}^n, A'(n') = \begin{cases} enabled \text{ if } n' \in Post(n) \\ A(n') \text{ otherwise} \end{cases} \\ C' = C \\ I' = I + Prod(n) \end{cases}$$

The semantics of a timed EFFBD is defined as a TTS. A *valuation* is a mapping $\nu \in (\mathbb{R}_{\geq 0})^{FC}$ such that $\forall f \in FC$, $\nu(f)$ is the time elapsed since f started its execution. $\nu(f)$ is only meaningful if the function f is in execution. The TTS states are 4-tuples representing node activity, iteration counters, item levels and valuations.

Definition III.10 (Semantics of a timed EFFBD)**.** *The semantics of a timed EFFBD $\mathcal{E}_T = (\mathcal{N}, \mathcal{I}, \mathcal{A}, count, n_0, I_0, \alpha, \beta)$ is a tuple $||\mathcal{E}_T|| = (S, s_0, \mathcal{N}, \rightarrow)$ where:*

- *$S \subseteq \mathbb{A}^{\mathcal{N}} \times \mathbb{N}^{IT_{in}} \times \mathbb{N}^{\mathcal{I}} \times (\mathbb{R}_{\geq 0})^{FC}$ is the set of the states reachable by iteratively applying $\rightarrow$ from s_0;*
- *$s_0 = (A_0, \overline{0}, I_0, \overline{0})$, A_0 being defined as above;*
- *$\rightarrow \subseteq S \times (\mathcal{N} \cup \mathbb{R}_{\geq 0}) \times S$ is the* transition relation *including a discrete transition relation and a continuous transition relation.*
  - *The* continuous transition relation *is defined $\forall d \in \mathbb{R}_{\geq 0}$ by:*

    $$(A, C, I, \nu) \xrightarrow{d} (A, C, I, \nu') \Leftrightarrow$$
    $$\begin{cases} \forall n \notin FC, A(n) \neq enabled \\ \forall n \in FC, A(n) = enabled \Rightarrow I < Cons(n) \\ \forall n \in FC, A(n) = executing \Rightarrow \nu(n) + d \leq \beta(n) \\ \nu' = \nu + d \end{cases}$$
  - *The* discrete transition relation *is defined $\forall n \in \mathcal{N}$ by:*

    $$(A, C, I, \nu) \xrightarrow{n} (A', C', I', \nu') \Leftrightarrow$$
    $$\begin{cases} A'(n) = NextAct \text{ as defined above} \\ \begin{cases} (A(n) = enabled \wedge PreCondition^T) \\ \text{or } (n \in FC \wedge A(n) = executing \wedge \mathcal{P}'^T_F) \end{cases} \end{cases}$$

with:

$$PreCondition^T \begin{cases} n \in AND_{in} \cup OR_{out} \cup LP_{in} \Rightarrow \mathcal{P}^T_{gen} \\ n \in AND_{out} \Rightarrow \mathcal{P}^T_{A_o} \\ n \in OR_{in} \Rightarrow \mathcal{P}^T_{O_i} \\ (n \in IT_{in} \wedge C(n) < count(n)) \Rightarrow \mathcal{P}^T_{I_i} \\ (n \in IT_{in} \wedge C(n) = count(n)) \Rightarrow \mathcal{P}'^T_{I_i} \\ n \in IT_{out} \Rightarrow \mathcal{P}^T_{I_o} \\ n \in LP_{out} \Rightarrow \mathcal{P}^T_{L_o} \\ n \in FC \Rightarrow \mathcal{P}^T_F \end{cases}$$

The definition of the $\mathcal{P}_x^T$ conditions are:

$$\mathcal{P}_F^T \begin{cases} \mathcal{P}_F \text{ as defined above} \\ \forall f \in FC \begin{cases} \nu'(f) = 0 \text{ if } f = n \\ \nu'(f) = \nu(f) \text{ otherwise} \end{cases} \end{cases}$$

$$\mathcal{P}_F'^T \begin{cases} \mathcal{P}_F' \text{ as defined above} \\ \alpha(n) \leq \nu(n) \leq \beta(n) \end{cases}$$

The other conditions are the same as in the untimed case, though with addition of constraint $\nu' = \nu$. The definition of the continuous transition relation imposes that time cannot elapse until the system has reached a “stable state” (i.e. no instantaneous transition can be taken).

IV. Translation of an EFFBD into a TPN

This section briefly presents the TPN formalism and the structural translation of an EFFBD into a TPN. To make reading easier, most translation patterns are given in Appendix B.

A. Time Petri nets

Transition time Petri nets (which will simply be known as TPN hereunder) form a timed extension of classical Petri nets, where transitions are taken (or *fired*) within a given time interval [7]. The semantics described here corresponds to the single-server, intermediate case (see for instance [21] for a discussion on the different TPN semantics).

Definition IV.1 (Labeled time Petri nets)**.** *A labeled* time Petri net *over a set of actions* Σ *is a tuple* $\mathcal{T} = (P, T, ^\bullet(.), (.)^\bullet, M_0, a, b, \Sigma, \Lambda)$ *where:*

- $P = \{p_1, ..., p_m\}$ *is a finite, non empty set of* places*;*
- $T = \{t_1, ..., t_n\}$ *is a finite, non empty set of* transitions*;*
- $^\bullet(.) \in (\mathbb{N}^P)^T$ *is the* backward incidence function*;*
- $(.)^\bullet \in (\mathbb{N}^P)^T$ *is the* forward incidence function*;*
- $M_0 \in \mathbb{N}^P$ *is the* initial marking *of the net;*
- $a \in (\mathbb{N})^T$ *and* $b \in (\mathbb{N} \cup \{\infty\})^T$ *are functions giving for each transition respectively its* earliest *and* latest firing times *($a \leq b$);*
- $\Lambda : T \to A$ *is the* labeling function.

A *marking* M of the net is an element of $\mathbb{N}^P$ such that $\forall p \in P$, $M(p)$ is the number of tokens in place p. A transition t is said to be *enabled* by the marking M if $M \geq^\bullet (t)$; it is denoted by $t \in enabled(M)$. A transition t is said to be *firable* when it has been enabled for at least $a(t)$ time units. A transition t_k is said to be *newly enabled* by firing transition t_i from the marking M, which is denoted by $\uparrow enabled(t_k, M, t_i)$, if the transition is enabled by the new marking $M -^\bullet(t_i) + (t_i)^\bullet$ but was not by $M -^\bullet(t_i)$. Formally,

$$\uparrow enabled(t_k, M, t_i) = ((t_k = t_i) \vee (^\bullet(t_k) > M -^\bullet (t_i))) \\ \wedge(^\bullet(t_k) \leq M -^\bullet (t_i) + (t_i)^\bullet)$$

TPNs semantics is defined as a TTS: states are the association of both a marking M and a vector of *clock valuations* v, $v(t)$ representing the time elapsed since transition t was enabled.

Definition IV.2 (Semantics of a TPN)**.** *The semantics of a labeled TPN* $\mathcal{T}$ *is defined as a TTS* $\|\mathcal{T}\| = (Q, q_0, T, \to)$ *such that:*

- $Q \subseteq \mathbb{N}^P \times (\mathbb{R}_{\geq 0})^T$ *is the set of the states reachable by iteratively applying* $\to$ *from* q_0*;*
- $q_0 = (M_0, \overline{0})$*;*
- $\to \subseteq Q \times \{T \cup \mathbb{R}_{\geq 0}\} \times Q$ *is the* transition relation *including a discrete transition relation and a continuous transition relation.*
  - *The* discrete transition relation *is defined by:*

$$\forall t \in T \quad (M, v) \xrightarrow{\Lambda(t)} (M', v') \Leftrightarrow \begin{cases} t \in enabled(M) \\ a(t) \leq v(t) \leq b(t) \\ M' = M -^\bullet (t) + (t)^\bullet \\ \forall t_k \in T, v'(t_k) = \begin{cases} 0 \textit{ if } \uparrow enabled(t_k, M, t) \\ v(t_k) \textit{ otherwise} \end{cases} \end{cases}$$

  - *The* continuous transition relation *is defined by:*

$$\forall d \in \mathbb{R}_{\geq 0} \quad (M, v) \xrightarrow{d} (M, v') \Leftrightarrow \begin{cases} v' = v + d \\ \forall t_k \in T, t_k \in enabled(M) \Rightarrow v'(t_k) \leq b(t_k) \end{cases}$$

B. Translation patterns

The approach proposed in this work is to perform a structural translation by using elementary TPN patterns for each type of node and for each item. A superscript is added to the place and transition labels to distinguish the elementary TPNs. The complete TPN is then simply obtained by connecting patterns together. It should be noted that untimed EFFBDs are simply translated by Petri nets and timed EFFBDs by TPNs

1) Item patterns: An item A is simply translated by a single place $pItem^A$ with the initial marking $I_0(A)$.

2) Control patterns: each pattern encoding a node n is a labeled TPN $\mathcal{T}^n = (P^n, T^n, ^\bullet(.)^n, (.)^{\bullet n}, M_0^n, a^n, b^n, n, \Lambda)$. To make reading easier, only one pattern has been provided here: Fig. 2 gives the EFFBD representation of a function F consuming items A_1 to A_n and producing items B_1 to B_m and of its corresponding TPN pattern. Dashed elements correspond to the item patterns and therefore are not part of the pattern. However, it is considered here that item places belong to the function patterns, so as to simplify the expression of the composition operator, given below.

The other node patterns are provided in Appendix B. Each pattern begins with one entry place (m for an AND_{out} node with m predecessors) and ends with one transition (m for an OR_{in} node, none for an LP_{out} node).

The initial marking M_0^n depends on the nature of the node; it also depends whether $n = n_0$ or not. Let $pEntry^n$ be the entry place of the pattern associated with node n. $M_0^n[pEntry^n]$ is 1 if $n = n_0$ and 0 otherwise[12]. In addition, if $n \in IT_{in}$, then $M_0^n[pNoMore^n] = count(n)$ where $pNoMore^n$ is the place in the pattern ensuring that the iteration is performed at

[12] If $n \in AND_{out}$, then $n \neq n_0$ and all $pSynch^{n',n}$ synchronization places are initially empty.

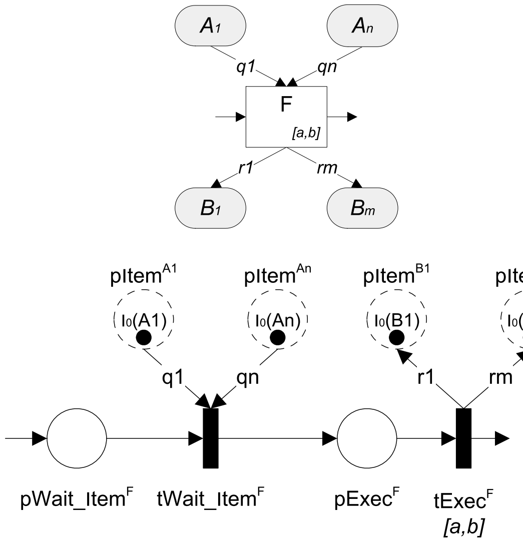

Fig. 2. Pattern of a function construct F

most $count(n)$ times. Item places initial markings are set as described above. All other places are initially empty.

Functions a^n and b^n also depend on the nature of n:

- if $n \notin FC$: $\forall t \in T^n$, $a^n(t) = b^n(t) = 0$;
- if $n \in FC$: $a^n(tExec^n) = \alpha(n)$ and $b^n(tExec^n) = \beta(n)$; all remaining transition intervals are set to $[0, 0]$.

The labeling function is defined as: $\forall t \in T^n, \Lambda(t) = n$

3) Building the complete net: once every item and node pattern has been created, partial nets are connected with additional arcs to obtain the final net $\mathcal{T} = (P, T, {}^\bullet(.), (.)^\bullet, M_0, \alpha, \beta, \mathcal{N}, \Lambda)$ where:

- $P = (\bigcup_{n\in\mathcal{N}} P^n) \cup (\bigcup_{A\in\mathcal{I}} pItem^A)$;
- $T = (\bigcup_{n\in\mathcal{N}} T^n)$;
- $\forall t \in T, {}^\bullet(t) = {}^\bullet(.)^n$ where $n \in \mathcal{N}$ is such that $t \in T^n$;
- $\forall t \in T, \forall p \in P, (t)^\bullet[p]$, the weight of the arc linking t to p in $\mathcal{T}$ is defined as:

$$\begin{cases} 1 \text{ if } t = tExit^n \wedge p = pEntry^{n'} \\ \quad \wedge (n, n') \in \mathcal{A}_C \wedge (n', n) \notin IT \cup LP \\ (.)^{\bullet n}[p] \text{ if } p \in P^n \\ 0 \text{ otherwise} \end{cases}$$

  where:
  - n (resp. n') is such that $t \in T^n$ (resp. $p \in P^{n'}$);
  - $tExit^n$ is the exit transition of $\mathcal{T}^n$[13];
  - $pEntry^{n'}$ is the entry place of $\mathcal{T}^{n'}$[14].
- $\forall p \in P, M_0[p] = M_0^n[p]$ where $n \in \mathcal{N} \cup \mathcal{I}$ is such that $p \in P^n$;
- α and β are built as ${}^\bullet(.)$.

An example illustrating the translation process is provided in Appendix C.

V. Properties

This section presents some properties obtained from the resulting TPNs. In addition, after proving the behavioral equivalence between both formalisms, some interesting results on TPNs are applied to EFFBDs.

A. Non re-entrance of the EFFBDs

A fundamental result on well-formed EFFBDs is provided below.

Proposition 1 (Non re-entrance of the EFFBDs)**.** *A well-formed EFFBD is not* reentrant *i.e. each node and structure must be exited before being enabled again.*

Proof: As there is only one initial node in the EFFBD, only opening AND nodes can create two (or more) independent control flows. However, under the assumption of having a well-formed EFFBD, these flows cannot converge to any node but the corresponding closing AND node. Therefore, no node can be enabled while still in execution. ■

In the following, EFFBDs are supposed to be well-formed.

B. Boundedness

A number of powerful results have been proved for *bounded* TPNs (i.e. for which the marking of any place stays finite). Likewise, this section provides a few results on a sub-class[15] of EFFBDs so-called bounded EFFBDs, the definition of which is given below.

Definition V.1 (Bounded EFFBD)**.** *Let $\mathcal{E}$ be an EFFBD and $(S, s_0, \mathcal{N}, \rightarrow)$ its semantics. $\mathcal{E}$ is* bounded *iff:*

$$\forall (A, C, I, \nu) \in S, \forall item \in \mathcal{I}, \exists k \in \mathbb{N} \quad I(item) \leq k$$

Two sufficient conditions to ensure the boundedness of an EFFBD are given hereunder. The proofs are trivial enough to be omitted (they rely on the fact that the only potentially unbounded places correspond to items and that the only "infinite behavior" is caused by loops).

Proposition 2. *An EFFBD that contains no item is bounded.*

Proposition 3. *An EFFBD in which no loop construct contains item producing functions is bounded.*

The example presented in Appendix C further illustrates EFFBD- and TPN-boundedness. The consequence on the boundedness of the resulting TPN is immediate.

Proposition 4. *Let $\mathcal{E}$ be a bounded EFFBD and m defined as:*

$$m = \max(\max_{iter \in IT_{in}} count(iter), \max_{item \in \mathcal{I}} I(item))$$

*Let $\mathcal{T}$ be the TPN obtained from $\mathcal{E}$ and $(Q, q_0, T, \rightarrow)$ its semantics. $\mathcal{T}$ is $m-$*bounded*:*

$$\forall (M, v) \in Q, \forall p \in P \quad M(p) \leq m$$

[13] For OR_{in} nodes, the transition is in fact $tSelect^{k,n}$ where k is the branch number containing n'.

[14] For AND_{out} nodes, the place is in fact $tSynch^{k',n}$ where k' is the branch number containing n'.

[15] Most of the models designed in SE are actually bounded; therefore, the restriction is not too limiting.

Proof: As there is only one initial node, each node pattern is, by construction, 1-bounded (or *safe*) except for ${IT_{in}}^n$ patterns which are k-bounded with $k = count(n)$. In addition, as the EFFBD is bounded, all $pItem^X$ places in $\mathcal{T}$ keep a finite marking. Therefore, $\mathcal{T}$ is bounded. ■

C. Strong timed bisimulation

A binary relation $\sim$ is defined over the behavior of EFFBD models and the corresponding TPNs.

Let $\mathcal{E} = (\mathcal{N}, \mathcal{I}, \mathcal{A}, count, n_0, I_0, \alpha, \beta)$ be an EFFBD, $\mathcal{T}_\mathcal{E} = (P, T, {}^\bullet(.), (.)^\bullet, M_0, a, b, \mathcal{N}, \Lambda)$ the labeled TPN obtained by the translation of $\mathcal{E}$. Let $||\mathcal{E}|| = (S, s_0, \mathcal{N}, \rightarrow_\mathcal{E})$ and $\|\mathcal{T}_\mathcal{E}\| = (Q, q_0, \mathcal{N}, \rightarrow_\mathcal{T})$ be their respective semantics. The binary relation $\sim \subseteq S \times Q$ is defined as follows:

$$\forall s = (A, C, I, \nu) \in S, \forall q = (M, v) \in Q, s \sim q \Leftrightarrow$$
$$\begin{cases} Activity_{en} \wedge Activity_{ex'ing} \wedge Activity_{ex'ed} \\ Counter \wedge Item \wedge Valuation \end{cases}$$

with:

$$Activity_{en} : \forall n \in \mathcal{N} : A(n) = enabled \Leftrightarrow$$
$$\begin{cases} M(pEntry)^n = 1 \text{ if } n \notin AND_{out} \\ \sum_{n' \in Pre(n)} M(pSynch^{n',n}) \geq 1 \text{ else} \end{cases}$$

$$Activity_{ex'ing} : \forall f \in FC : A(f) = executing \Leftrightarrow M(pExec^f) = 1$$

$$Activity_{ex'ed} : \forall n \in \mathcal{N} \text{ s.t. } \exists \alpha_o \in AND_{out} \cap Post(n) : A(n) = executed \Leftrightarrow M(pSynch^{n,\alpha_o}) = 1$$

$$Counter : \forall it \in IT_{in}, \forall k \in \mathbb{N} : C(it) = k \Leftrightarrow M(pNoMore^{it}) = count(it) - k$$

$$Item : \forall A \in \mathcal{I}, \forall k \in \mathbb{N} : I(A) = k \Leftrightarrow M(pItem^A) = k$$

$$Valuation : \forall f \in FC, \forall x \in \mathbb{R}_{\geq 0} : \nu(f) = x \Leftrightarrow v(tExec^f) = x$$

Proposition 5. *The relation $\sim$ is a strong-timed bisimulation relation.*

The proof is given in Appendix D.

D. Additional results

This section recalls theorems about TPNs and describes their extension to EFFBDs.

Theorem 6 (Decidability of the k-boundedness)**.** *For any TPN $\mathcal{T}$ with the semantics $(Q, q_0, T, \rightarrow)$ and a given $k \in \mathbb{N}$, the following problem is decidable [22]:*

$$\forall (M, v) \in Q, \quad \forall p \in P : \quad M(p) \leq k$$

Corollary 1. *For any EFFBD $\mathcal{E}$ with the semantics $(S, s_0, \mathcal{N}, \rightarrow)$ and a given $k \in \mathbb{N}$, the following problem is decidable:*

$$\forall (A, C, I, \nu) \in S, \forall A \in \mathcal{I}, I(A) \leq k$$

Proof: Using proposition 5 and theorem 6, the proof is immediate. ■

As a result, it is always possible to check whether the item level of any EFFBD stays under a limit specified by the system designer, which is particularly useful when assessing the size of a system in the course of the design process.

Theorem 7. *For any bounded TPN $\mathcal{T}$ with the semantics $(Q, q_0, T, \rightarrow)$, the following problems are decidable [22]:*

- *Accessibility of a marking:* "given a marking M, is there a state $(M', v') \in Q$ such that $M = M'$?"
- *Accessibility of a state:* "given a state $q = (M, v)$, $q \in Q$?"

Corollary 2. *For any bounded EFFBD $\mathcal{E}$ with the semantics $(S, s_0, \mathcal{N}, \rightarrow)$ the following problems are decidable:*

- *Accessibility of an activity state:* "given a node n, can n be enabled ?"
- *Accessibility of a state:* "given a state s, $s \in S$?"

As mentioned in the introduction of this paper, the final purpose of this work is to assess the safety of the designed systems. It is therefore necessary to express sometimes complex properties such as *"Upon the reception of an alarm, the system always reacts in an appropriate way in less than 5 time units."*

In that respect, a temporal logic such as the Time Computation Tree Logic (TCTL [23]) is particularly well-suited to express these specifications. Due to a lack of space, its semantics will not be described here; however, it should be noted that the model checking of TCTL has been proved decidable on bounded TPN [24]. Moreover, a subset of TCTL, named TPN-TCTL, has been described by the author of [25]: informally speaking, the formula atomic propositions are expressed in terms of linear inequalities on the marking of the TPN. TPN-TCTL has also been proved as decidable on bounded TPN. In addition, a TPN-TCTL model checker was implemented in ROMÉO[16], a TPN-analysis software tool developed by the authors of [8]. As a result, if a property over a bounded EFFBD can be translated into a TPN-TCTL formula over the corresponding TPN, then it is also decidable. This result shall be discussed in the next section.

VI. APPLICATIONS

The translation method proposed in section IV has been fully implemented and embedded in KIMONO, an operational systems engineering development platform designed as a series of ECLIPSE plug-ins [9]. KIMONO was specifically developed for a French Department of Defense branch, partly by SODIUS and the IRCCYN research laboratory, with the goal, amongst others, of providing the system engineer with efficient tools to assess the system safety during the design process[17].

For this purpose, and in addition to the transition module mentioned above, a deep-analysis module was also developed and implemented. This module:

[16] http://romeo.rts-software.org/

[17] KIMONO being partially protected by confidentiality clauses, no official project page is available.

1) creates a safety property P expressed in natural language over the system functions;
2) transforms this high-level property in a TPN-TCTL formula F over the resulting TPN;
3) checks the formula by using ROMÉO model checker and adapted on-the-fly algorithms;
4) returns the truth value of F and, if applicable, a "witness" of the formula computed by ROMÉO (i.e. a sequence of transition firing);
5) returns the truth value of P and a witness of the property in terms of a *sequence of functions*.

In addition, the module allows the EFFBD simulation through the TPN simulation by using ROMÉO simulation engine.

For the time being, a limited set of property classes are proposed, such as *"the system always reaches its final state"* or *"executing the function F always leads to the execution of function G in less than x time units"*. The translation from P to F, as well as the correctness of the method, shall be discussed in a subsequent paper. A practical example, including the property classes currently investigated, is presented in [26].

It should eventually be noted that steps 2 to 4 are totally concealed to the user. A major concern throughout this study was indeed to develop an *efficient* and *usable* tool: therefore, the use of ROMÉO and, more generally the resort to TPN is hidden in a "black box". As a conclusion, this module, combined with the simulation capability, can be used to point out the modeling weaknesses (such as deadlocks) at an early stage of the design, therefore providing valuable help to the system architect.

VII. CONCLUSION

This paper has proposed a formal description and behavioral semantics for a modeling language widely used in SE processes although, to the authors' knowledge, never formally established. This first step led to the definition of a transformation from EFFBD to TPN, proved as preserving the behavior of the high-level model. As a result, a number of fundamental properties, inherited from the research works carried on TPN were applied to EFFBDs. In turn, this work hinted at the possibility and benefit of performing safety assessment on models designed by a typical systems engineer *via* model checking techniques, in a completely transparent way. Finally, the paper gave a short glance on the tools developed in application of those results.

Further work will focus on proving the correctness of the translation of high-level properties to TPN-TCTL formulas, and on the study of the method algorithms complexity. In addition, it has been planned to extend KIMONO features by providing the system designer with a complementary tool offering safety and dependability-inspired design patterns based on the results given by the developed analysis module.

APPENDIX A
SEMANTICS RULES

This appendix provides the $\mathcal{P}_x$ propositions describing the conditions the system state must fulfill to process nodes in AND_{out}, OR_{in}, IT_{in} (distinguishing whether the iteration maximum number has been reached or not) and LP_{out}.

$$\mathcal{P}_{A_o}\begin{cases}\forall n' \in \mathcal{N}^n \begin{cases} A(n') = \mathit{executed} \wedge A'(n') = \mathit{inactive} \\ \quad \text{if } n' \in \mathit{Pre}(n) \\ A'(n') = \mathit{enabled} \text{ if } n' \in \mathit{Post}(n) \\ A'(n') = A(n') \text{ otherwise}\end{cases} \\ C' = C, I' = I\end{cases}$$

$$\mathcal{P}_{O_i}\begin{cases}\exists\ n_{Sel} \in \mathit{Post}(n) \begin{cases} A'(n_{Sel}) = \mathit{enabled} \\ \forall n_{NoSel} \in \mathit{Post}(n) \setminus \{n_{Sel}\} \\ \quad A'(n_{NoSel}) = \mathit{inactive} \\ \forall n' \in \mathcal{N}^n \setminus \mathit{Post}(n), \\ \quad A'(n') = A(n')\end{cases} \\ C' = C, I' = I\end{cases}$$

$$\mathcal{P}_{I_i}\begin{cases}\forall n' \in \mathcal{N}^n, A'(n') = \begin{cases}\mathit{enabled} \text{ if } n' \in \mathit{Post}(n) \\ A(n') \text{ otherwise}\end{cases} \\ \forall it \in IT_{in} \begin{cases} C'(it) = C(it) + 1 \text{if } it = n \\ C'(it) = C(it) \text{ otherwise}\end{cases} \\ I' = I\end{cases}$$

$$\mathcal{P}'_{I_i}\begin{cases}\forall n' \in \mathcal{N}^n, A'(n') = \begin{cases}\mathit{enabled} \text{ if } n' \in \mathit{Post}(i_o) \\ \quad \text{where } (n, i_o) \in IT \\ \mathit{executed} \text{ if } (n' = i_o) \\ \quad \wedge \mathit{Post}(i_o) \cap AND_{out} \neq \emptyset \\ A(n') \text{ otherwise}\end{cases} \\ \forall it \in IT_{in} \begin{cases} C'(it) = 0 \text{ if } it = n \\ C'(it) = C(it) \text{ otherwise}\end{cases} \\ I' = I\end{cases}$$

$$\mathcal{P}_{I_o}\begin{cases}\forall n' \in \mathcal{N}^n, A'(n') = \begin{cases}\mathit{enabled} \text{ if } (n', n) \in IT \\ A(n') \text{ otherwise}\end{cases} \\ C' = C, I' = I\end{cases}$$

$$\mathcal{P}_{L_o}\begin{cases}\forall n' \in \mathcal{N}^n, A'(n') = \begin{cases}\mathit{enabled} \text{ if } (n', n) \in LP \\ A(n') \text{ otherwise}\end{cases} \\ C' = C, I' = I\end{cases}$$

APPENDIX B
TRANSLATION PATTERNS

Fig. 3 to 6 show for each construct its EFFBD representation and its corresponding patterns. Since most constructs contain two nodes, patterns are in fact composed of two elementary TPNs. In addition, when the interval associated with a transition is $[0, 0]$, it has been omitted on the graphical representation of the TPN. Likewise, when the weight of an arc is 1, it is not shown on the arrow.

Fig. 3 gives the pattern of a parallel structure. Note the $pSynch^{x,O}$ places: they perform the synchronization ending the parallel structure.

Fig. 5 gives the pattern of an iteration structure. Note the additional places $pNoLess^I$ and $pNoMore^I$; the former ensures that the iterate branch is taken *at least* $count(I) = i_M$ times while the latter ensures it is taken *at most* i_M times. The

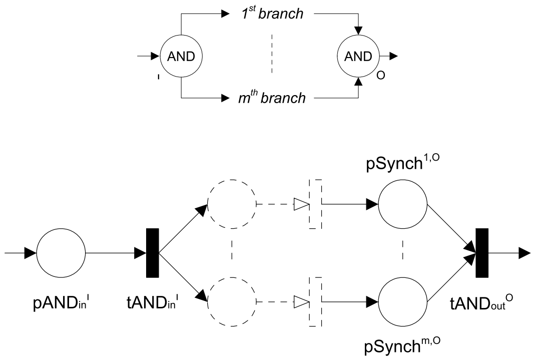

Fig. 3. Patterns of nodes $I \in AND_{in}$ and $O \in AND_{out}$

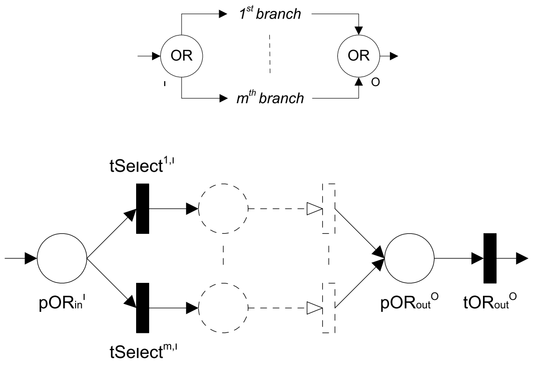

Fig. 4. Patterns of nodes $I \in OR_{in}$ and $O \in OR_{out}$

additional arc between pIT^I_{in} and tIT^I_{exit} enforces a complete reset of the pattern at the exit.

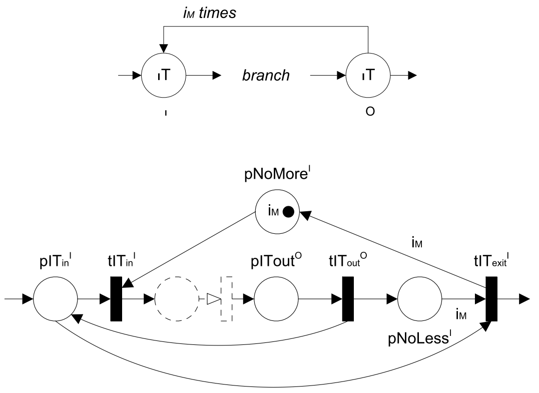

Fig. 5. Patterns of nodes $I \in IT_{in}$ and $O \in IT_{out}$

Appendix C
Example of a bounded EFFBD

The EFFBD represented Fig. 7 models a (very basic) bounded buffer. Task `Write` (or W) writes some data in an initially empty bounded buffer while task `Read` (or R) reads and erases previous data. Items `BufferIn` (BI) and `BufferOut` (BO) model the space occupied or left in the buffer; the initial amount of BI is 0 and BO initial amount is the buffer size (set to 3 in this example). Function durations respectively belong to in the intervals $[\alpha^W, \beta^W]$ and $[\alpha^R, \beta^R]$.

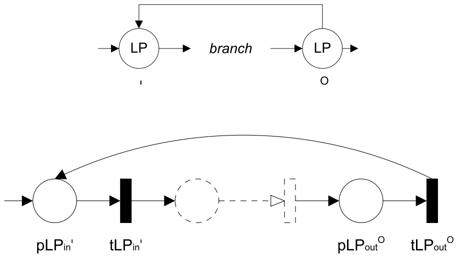

Fig. 6. Patterns of nodes $I \in LP_{in}$ and $O \in LP_{out}$

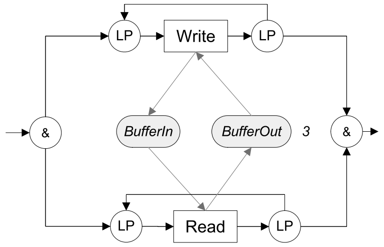

Fig. 7. Example of a bounded EFFBD

To write data in a full buffer, W must wait for R to free at least on space; conversely, R cannot access an empty buffer and must wait for W to provide at least one data piece. This EFFBD only provides a coarse modeling as, for instance, no monitoring task is defined[18].

The translation into a TPN is given Fig. 8. As no monitoring task was defined to stop the system, both loops are infinite and therefore, the closing AND pattern is useless.

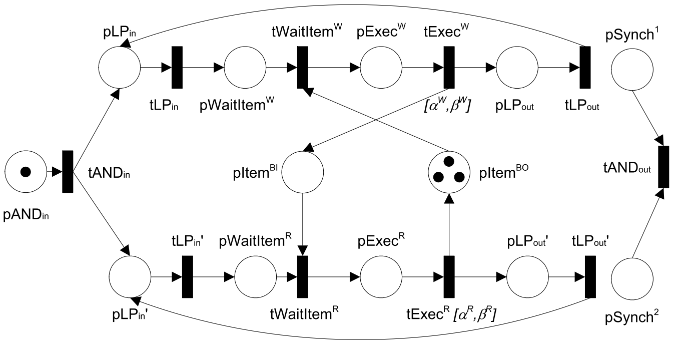

Fig. 8. Translation of the EFFBD of Fig. 7

Although this model does not respect the sufficient conditions given in Section V-B, both EFFBD and TPN are trivially bounded.

Appendix D
Proof of proposition 5

Proof: only well-formed EFFBDs are considered here. The proof mechanism is based on a structural induction: the bisimulation is proved, point by point, on every possible transition (here, only the AND_{in} case is given; the proof for other nodes follows the same arguments and is left to the reader).

[18] It would actually be modeled by a termination structure.

Due to the atomicity, imbrication and continuity properties of the well-formed EFFBD, these elementary bisimulation results are straightforwardly propagated to any combination.

Let $\mathcal{E} = (\mathcal{N}, \mathcal{I}, \mathcal{A}, count, n_0, I_0, \alpha, \beta)$ be an EFFBD and $\mathcal{T}_{\mathcal{E}} = (P, T, {}^{\bullet}(.), (.)^{\bullet}, M_0, a, b, \mathcal{N}, \Lambda)$ the TPN obtained by the translation of $\mathcal{E}$. Let $||\mathcal{E}|| = (S, s_0, \mathcal{N}, \rightarrow_{\mathcal{E}})$ and $||\mathcal{T}_{\mathcal{E}}|| = (Q, q_0, T, \rightarrow_{\mathcal{T}})$ be their respective semantics and $\sim$ be the binary relation defined in Section V-C.

R_0 Trivially, $s_0 \sim q_0$.

R_d Let $s = (A, C, I, \nu)$, $s' = (A, C, I, \nu') \in S$ and $d \in \mathbb{R}_{>0}$[19] s.t. $s \xrightarrow{d}_{\mathcal{E}} s'$. Let $q = (M, v) \in Q$ s.t. $s \sim q$. It follows:

- from the definition of $||\mathcal{E}||$, $\nu' = \nu + d$ and $\forall f \in FC$ s.t. $A(f) = executing$, $\nu(f) + d \leq b(f)$
- according to the definition of $\sim$ and to the function pattern, $\forall f \in FC$, $\nu(f) = v(tExec^f)$ and $\beta(f) = b(tExec^f)$
- $enabled(M) = \{tExec^f | A(f) = executing\}$ as nodes $n \notin FC$ are not enabled and $\forall f' \in FC$ s.t. $A(f) = enabled$, $I < Cons(f')$ i.e. $tWait\_Item^{f'} \notin enabled(M)$
- as a result, $\forall t \in enabled(M)$, $v(t) + d \leq \beta(t)$

Let $q' = (M, v + d)$. Trivially, $q \xrightarrow{d}_{\mathcal{T}} q'$ and $s' \sim q'$.

R'_d Conversely, let $q = (M, v)$, $q' = (M, v') \in Q$ and $d \in \mathbb{R}_{>0}$ s.t. $q \xrightarrow{d}_{\mathcal{T}} q'$. Let $s = (A, C, I, \nu) \in S$ s.t. $s \sim q$. It follows:

- from the definition of $||\mathcal{T}_{\mathcal{E}}||$, $v' = v + d$ and $\forall t \in enabled(M)$, $v(t) + d \leq b(t)$
- according to the definition of $\sim$ and to the function pattern, $\forall f \in FC$, $\nu(f) = v(tExec^f)$ and $\beta(f) = b(tExec^f)$
- since $d > 0$, only $tExec^f$ transitions are enabled where $f \in FC$ i.e. $\forall n \notin FC$, $A(n) \neq enabled$ and $\forall f' \in FC$, $A(f') = enabled \Rightarrow I < Cons(f')$
- as as result, $\forall f \in FC$, $A(f) = executing \Rightarrow \nu(f) + d \leq \beta(f)$

Let $s' = (A, C, I, \nu + d)$. Trivially, $s \xrightarrow{d}_{\mathcal{E}} s'$ and $s' \sim q'$.

R_σ Let $s = (A, C, I, \nu)$, $s' = (A', C, I, \nu) \in S$ and $\sigma \in AND_{in}$ s.t. $s \xrightarrow{\sigma}_{\mathcal{E}} s'$. Let $q = (M, v) \in Q$ s.t. $s \sim q$. It follows:

- by definition of $||\mathcal{E}||$, $A(\sigma) = enabled$, any node contained in the AND structure is inactive, $A'(\sigma) = inactive$[20] and $\forall n \in Post(\sigma)$, $A'(n) = enabled$
- according to the definition of $\sim$ and to the translation patterns, $M(pANDin^\sigma) = 1$ and therefore $tANDin^\sigma \in enabled(M)$
- $v(tANDin^\sigma) = 0$

Let $q' = (M', v)$ s.t. $M'(pANDin^\sigma) = 0$ and $\forall n \in Post(\sigma)$, $M'(pEntry^n) = 1$ where $pEntry^n$ is the entry place of the TPN $\mathcal{T}^n$. Trivially, $q \xrightarrow{\sigma}_{\mathcal{T}} q'$. According to the translation patterns, the firing interval of any transition newly enabled by M' is $[0, 0]$ and therefore $s' \sim q'$.

R'_σ Conversely, let $q = (M, v)$, $q' = (M, v') \in Q$, $\sigma \in AND_{in}$ s.t. $q \xrightarrow{\sigma}_{\mathcal{T}} q'$. Let $s = (A, C, I, \nu) \in S$ s.t. $s \sim q$. It follows:

- by definition of $||\mathcal{T}_{\mathcal{E}}||$, $M(pANDin^\sigma) = 1$[21] and $\forall p$ s.t. $(tANDin^\sigma)^{\bullet}[p] = 1$, $M'(p) = 1$
- according to the definition of $\sim$ and to the translation patterns, $A(\sigma) = enabled$, $A'(\sigma) = inactive$ and $\forall n \in Post(\sigma)$, $A(n) = enabled$

Let $s' = (A', I, C, \nu)$ s.t. $A'(\sigma) = inactive$ and $\forall n \in Post(\sigma)$, $A'(n) = enabled$. Trivially, $s \xrightarrow{\sigma}_{\mathcal{E}} s'$. Finally, according to the translation patterns, the valuation of any $tExec^f$ transition for $f \in FC$ are not affected by the firing of $tANDin^\sigma$. Therefore $s' \sim q'$.

Since nodes are either in sequence or fully nested, the bisimulation relation is propagated throughout the complete EFFBD. The binary relation $\sim$ is therefore a strong-timed bisimulation. ■

References

[1] *What is Systems Engineering?*, INCOSE, 2004. [Online]. Available: http://www.incose.org

[2] IEEE, “IEEE-1220: Application & management of systems engineering process,” 1994.

[3] INCOSE Technical Board, *Systems Engineering handbook: a “what to” guide for all SE practitioners (version 2a)*. INCOSE, Jun. 2004.

[4] L. Lamport, “Proving the correctness of multiprocess programs,” *IEEE Transactions on Software Engineering*, vol. 3, no. 2, pp. 125–143, 1977.

[5] US Air Force, “MIL-STD-499 Functional Flow Diagrams,” 1968, dI-S-3604/S-126-1.

[6] J. Long, “Relationships between common graphical representations in Systems Engineering,” in *5th International Symposium of the INCOSE*. St. Louis, Missouri: INCOSE, Jul. 1995, updated July 2002.

[7] P. Merlin, “A study of recoverability of computing systems,” Ph.D. dissertation, Dpt. of Computer Science, University of California, Irvine, CA, 1974.

[8] G. Gardey, D. Lime, M. Magnin, and O. H. Roux, “Romeo: A tool for analyzing time Petri nets,” in *Proceedings of the 17th International Conference on Computer-Aided Verification (CAV’05)*, Lecture Notes in Computer Science, 2005.

[9] C. Seidner, J.-P. Lerat, and O. H. Roux, “Usability of formal verification on EFFBD models: Applying Petri nets to Systems Engineering issues,” in *17th International Symposium of the International Council on Systems Engineering (IS2007)*, San Diego, CA, Jun. 2007.

[10] R. Alur and D. L. Dill, “A theory of timed automata,” *Theorical Computer Science*, vol. 126, no. 2, pp. 183–235, 1994.

[11] T. A. Henzinger, Z. Manna, and A. Pnueli, “Timed transition systems,” *Real Time: Theory in Practice*, Lecture Notes in Computer Science, 1992.

[12] A. Bondavalli, M. Dal Cin, D. Latella, I. Majzik, A. Pataricza, and G. Savoia, “Dependability analysis in the early phases of UML based system design,” *Journal of Computer Systems Science and Engineering*, vol. 16, no. 5, pp. 265–275, 2001.

[13] S. Bernardi and J. Merseguer, “A UML profile for dependability analysis of real-time embedded systems,” in *WOSP ’07: Proceedings of the 6^{t}h international Workshop on Software and Performance*, 2007, pp. 115–124.

[14] SysML Finalization Task Force, *OMG Systems Modeling Language (OMG SysML ^{TM}) Specification – Proposed Available Specification*. Object Management Group, Mar. 2007. [Online]. Available: http://www.omg.org/cgi-bin/apps/doc?ptc/07-02-03.pdf

[15] Y. Vandeperren and W. Dehaene, “SysML and Systems Engineering applied to UML-based SoC design,” in *2nd UML-SoC Workshop at 42nd Design Automation Conference*, Jun. 2005.

[16] SAE Aerospace, “Architecture Analysis & Design Language (AADL) – SAE AS 5506,” Nov. 2004. [Online]. Available: http://www.aadl.info

[19] If $d = 0$, the case is trivial.

[20] $A'(\sigma) \neq executed$ in a EFFBD where no branch is empty, which is supposed true here.

[21] $M(pANDin^\sigma) \leq 1$ as shown in Section V-A.

[17] A.-E. Rugina, “System dependability evaluation using AADL (Architecture Analysis and Design Language),” in *Rencontres Jeunes Chercheurs en Informatique Temps-Réel (RJCITR)*, Sep. 2005.

[18] E. Herzog, “An approach to Systems Engineering tools data representation and exchange,” Ph.D. dissertation, Linköpings Universitet, 2004.

[19] C. Seidner and O. H. Roux, “On the formal verification of EFFBD models using a structural translation to time Petri nets,” IRCCyN, Nantes, France, Tech. Rep. RI2007-3 ref. 3695, Oct. 2007. [Online]. Available: http://www.irccyn.ec-nantes.fr/~seidner

[20] D. Park, “Concurrency on automata and infinite sequences,” *Conf. on Theoretical Computer Science*, Lecture Notes in Computer Science, vol. 104, 1981.

[21] B. Bérard, F. Cassez, S. Haddad, D. Lime, and O. H. Roux, “Comparison of different semantics for time Petri nets,” in *Automated Technology for Verification and Analysis (ATVA’05)*, ser. Lecture Notes in Computer Science, vol. 3707. Springer, Oct. 2005.

[22] B. Berthomieu and M. Menasche, “An enumerative approach for analyzing time Petri nets,” in *IFIP Congress Series*, vol. 9, 1983, pp. 41–46.

[23] R. Alur, C. Courcoubetis, and D. L. Dill, “Model-checking for real-time systems,” in 5^{th} *IEEE Symposium on Logic in Computer*, June 1990, pp. 414–425.

[24] F. Cassez and O. H. Roux, “Structural translation from time Petri nets to Timed Automata - Model-checking time Petri nets via Timed Automata,” *The Journal of Systems and Software*, vol. 79, no. 10, pp. 1456–1468, 2006.

[25] G. Gardey, “Contribution à la vérification et au contrôle des systèmes temps réel – Application aux réseaux de Petri temporels et aux automates temporisés,” Ph.D. dissertation, École Centrale de Nantes – Université de Nantes, 2006.

[26] C. Seidner, J.-P. Lerat, and O. H. Roux, “Usability and usefulness of formal verification in a system design process,” in 18^{th} *International Symposium of the INCOSE*. Utrecht, Netherlands: International Council on Systems Engineering, Jun. 2008.