Translating Synchronous Petri Nets into PROMELA for Verifying Behavioural Properties

Óscar R. Ribeiro, João M. Fernandes
Dept. Informática / CCTC, Universidade do Minho
Braga, Portugal
{oscar.rafael, jmf}@di.uminho.pt

***Abstract*— For developing embedded systems, the design process may benefit in some contexts from the usage of formal methods, namely to find critical errors and flaws, before final design and implementation decisions are taken. The Synchronous and Interpreted Petri Net (SIP-net) modelling language is considered in this article to model embedded systems. This model of computation is based on safe Petri nets with guarded transitions and synchronous transitions firing, and also includes enabling and inhibitor arcs. The Spin tool, whose input language is PROMELA, is a verification system based on model checking techniques. This article presents a program to translate SIP-net models into PROMELA code and discusses in detail the adequacy of the created PROMELA specification for verification through model checking techniques.**

I. Introduction

Concurrency is considered one of the essential features of reactive systems [1], a class where embedded systems can be included. Development of this kind of systems requires models of computation that explicitly support concurrency. A concurrent system is a collection of sequential processes that in abstract are executed in parallel, i.e., it is not required that a separate physical processor is used to execute each process. A process is a set of instructions in a programming language which are executed sequentially. Thus, the semantics of concurrent systems is usually based on the notion of a global state, where the instructions of each process define a set of events denoting transitions between states. However, in a concurrent system an event affects and is affected by a limited number of other events (event scope). Events with disjoint scopes are free to occur independently.

Petri Nets (PNs) [2] constitute a suitable model of computation for expressing concurrent systems, due to their extensive body of results, both theoretical and practical. PNs have shown to be a powerful technique to specify and model the behaviour of systems, where concurrency, resource sharing, and synchronisation are important issues to take into account. In particular, PNs have already proven to be a suitable language for embedded systems [3], [4].

Synchronous and Interpreted Petri Nets (SIP-nets) were obtained by the enrichment of safe PNs with guarded transitions, synchronous firing, and also enabling and inhibitor arcs [5]. With these characteristics, the models that can be obtained are easier to synthesize [6]. In fact, synchronous circuits represent the largest portion of circuit designs and the state of the art in synthesising synchronous systems is more advanced and stable than the corresponding one for asynchronous circuits [7].

Work supported by Fundação para a Ciência e a Tecnologia (FCT) under grant SFRH/BD/19718/2004

When a formal model of a given system is created, it can be analysed with respect to some desired properties, thus allowing the detection of design errors prior to the system implementation. In general, the major weakness of PNs is the complexity problem; PN-based models tend to become too large for analysis even for a small real system [8]. In [5] reachability graph analysis of the SIP-net description is used to investigate the properties of the modeled system.

Model checking [9] is a verification technique that is based on the idea of exhaustively exploring the reachable state space of a system. The model checker Spin [10], [11] is a verification system, which accepts a specification language called PROMELA (Process Meta Language) [11], [12], [13]. Spin has two main modes of operation: simulation and verification. Verification requires exhaustive search, whereas simulation does not and thus can deal with bigger state spaces. Simulation is a testing technique that can only indicate errors and never their absence. It is quite useful in practical terms, but in some situations verification is the only solution, especially whenever one needs to formally guarantee that a system is free of errors, an essential condition for safety-critical systems. Spin uses the linear temporal logic (TL) to specify the properties to be verified. We choice Spin tool over other model checkers because we have done some previous works on the formalization of SIP-net models' properties in TL.

The main motivation of this work is to study how SIP-net models can be verified using the Spin tool. We present a model checking approach, that uses the Spin tool to verify critical properties of embedded systems such as liveness, deadlock-freedom, and the absence of structural and behavioural conflicts among transitions.

The suggested design flow of our approach is presented in fig. 1. As tool support, we have created a computer application, written in the Haskell language, to automatically translate SIP-net models into corresponding PROMELA specification. With the assistance of the Spin tool, the generated PROMELA specification can be either simulated or verified with respect to some TL properties. This is possible, since the computer application can generate two different types of PROMELA specifications, one more adequate to be simulated and the other more adequate to the verification of some properties.

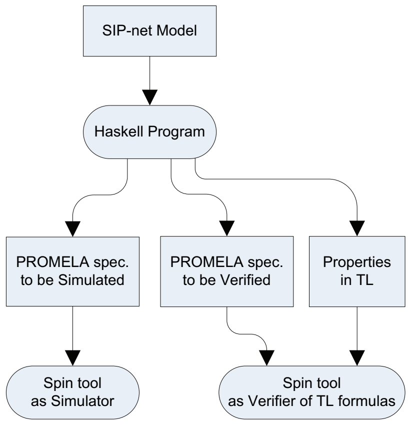
SIP-net Model
Haskell Program
PROMELA spec. to be Simulated
PROMELA spec. to be Verified
Properties in TL
Spin tool as Simulator
Spin tool as Verifier of TL formulas

Fig. 1. Design flow of the approach.

We already reported in [14] the application of this approach in a case study. This article focus on the presentation of the issues related to the translation process (from SIP-nets models to PROMELA specifications) and its tool support, using a small, but illustrative, example.

This article is structured as follows. Sect. 2 describes the SIP-net modelling language and some of its properties. In sect. 3 we show how some small examples of SIP-net models can be translated into PROMELA specifications. In sect. 4 some general rules to obtain a PROMELA specification for a given SIP-net model are defined. In sect. 5 we present the computer application, written in Haskell, that supports the automatic translation from SIP-net models to PROMELA specifications. Sect. 6 shows how the properties of SIP-net models can be described in terms of temporal logic. We conclude the article with its conclusions, and some directions to the future work.

II. SIP-net Modelling Language

In this section we introduce the SIP-net modelling language, showing how it was obtained from generic PN modelling languages. We present the structural apparatus to create a SIP-net model, the dynamics of associated to an SIP-net model, and some important properties of SIP-net models.

A. Structure of SIP-net models

The definition of a generic PN, of type place/transition system, presented in [2] is the following.

Definition 2.1: A 6-tuple $N = (P, T, F, C, M_0, W)$ is a **Petri net** if and only if

- (P, T, F) is a **basic net**, i.e. P, T are disjoint finite sets and $F \subseteq (P \times T) \cup (T \times P)$ is a binary relation called **flow relation**, whose elements are called **arcs**. The elements of P and T are called **places** and **transitions**, respectively;
- $C\colon P \to \mathbb{N} \cup \{\omega\}$ gives the **capacity** (ω represents the infinite capacity) for each set;
- $W\colon F \to \mathbb{N} \setminus \{0\}$ gives one **weight** for each arc;
- $M_0\colon P \to \mathbb{N} \cup \{\omega\}$ is the **initial marking** for net, respecting the capacities, i.e., $\forall_{p \in P}\ M(p) \leq C(p)$.

■

The SIP-net modelling language has been enriched with respect to the generic PN modelling language in three different ways:

- two new types of arcs are allowed: enabling arcs (also known as read arcs, test arcs, or positive context arcs), and inhibitor arcs (also designated negative context arcs) [15], [16];
- transitions can have associated guards, which are propositional formulas where variables represent input signals of the modeled system, i.e., guards over transitions are formulas containing external variables, which may affect the enabling of transitions;
- transitions firing are synchronized with the active edge of a (global) clock.

Definition 2.2: The **structure of a synchronous and interpreted Petri net** (written structure of an SIP-net model) is a tuple $N = (P, T, F, E, I, G)$ such that:

1) (P, T, F) is a **basic net**;
2) $E, I \subseteq P \times T$ are sets of enabling and inhibitor arcs respectively; the sets E, I and F are expected to be all disjoint;
3) $G : T \to PROP$ is a mapping associating a propositional formula to each transition.

Often P, T, F, E, I, G are denoted by P_N, T_N, F_N, E_N, I_N, G_N respectively. A **marking** to N is a mapping from P_N into the set $\{0, 1\}$. ■

Graphically, the places and transitions are represented by circles and rectangles, respectively. The flow relation elements are represented by arrows.

The control part of embedded systems has input and output signals. Although the output signals are represented in the SIP-net model, associated with places, we do not consider them in the formalization of the SIP-net model. When analysing the behaviour of an SIP-net model, it is not necessary to consider the output signals, because we assume that their influence on the behaviour of the SIP-net model, if existent, is reflected by changes in the input signals.

Inhibitor arcs can be used to model a priority between processes. An inhibitor arc, represented by a dashed line with a circle, connects a place to a transition and disables the transition when the place is marked. Consider, for example, a system in which two processes access a shared resource. A conflict arises when both processes apply for the resource. To solve this problem, an inhibitor arc connecting the resource (a place) to one of the processes can be introduced.

Enabling arcs can be used to model a synchronization between two processes. An enabling arc, represented by a dashed line with an arrow, connects a place to a transition and enables the transition when the place is marked. Nevertheless, when the transition fires, no token is removed from the place connected to the transition through an enabling arc.

It is useful for each transition to determine the set of all places connected to it through a normal, inhibitor or enabled arc.

Definition 2.3: Let N be a structure of an SIP-net model. For each $t \in T_N$:

1) $^\bullet t = \{p \mid p \, \mathrm{F_N} \, t\}$ is called the **preset** of t;
2) $t^\bullet = \{p \mid t \, \mathrm{F_N} \, p\}$ is called the **postset** of t;
3) $^\triangleright t = \{p \mid p \, \mathrm{E_N} \, t\}$ is the set of places linked with t through an enabling arc;
4) $^\circ t = \{p \mid p \, \mathrm{I_N} \, t\}$ is the set of places linked with t through an inhibitor arc.

For $T' \subseteq T_N$, let $^\bullet T' = \bigcup_{t \in T'} {}^\bullet t$, $T'^\bullet = \bigcup_{t \in T'} t^\bullet$, $^\triangleright T' = \bigcup_{t \in T'} {}^\triangleright t$, and $^\circ T' = \bigcup_{t \in T'} {}^\circ t$. ■

As shown later, the behaviour of an SIP-net model is affected by the interpretation of the variables appearing in the guards of its transitions (external events). Thus, to simulate the behaviour of an SIP-net model, every possible valuation of the variables in the guards of its transitions must be considered. In rigour, we take a valuation of a net N to be a mapping from the set of all the variables occurring in the guards of N into the two-valued set $\{0, 1\}$. The set of all valuations to N is denoted by $\mathcal{V}_N$. In the next definitions we consider only conflict-free SIP-net models.

B. Simultaneous firing in SIP-net models

Now, we attend to the dynamics associated to an SIP-net model.

Definition 2.4: Let N be the structure of an SIP-net, M a marking to N, $v \in \mathcal{V}_N$ and $t \in T_N$. The transition t is **ready** for M and v, $\mathtt{ready}(\mathtt{t}, \mathtt{M}, v)$, if

1) Each input place to t has one token, i.e., $\forall_{p \in {}^\bullet t} M(p) = 1$;
2) Each output place to t have no tokens, i.e., $\forall_{p \in t^\bullet} M(p) = 0$;
3) Each place connected to t with an enabling arc has one token, i.e., $\forall_{p \in {}^\triangleright t} M(p) = 1$;
4) Each place connected to t with an inhibitor arc has no tokens, i.e., $\forall_{p \in {}^\circ t} M(p) = 0$;
5) The interpretation of the guard associated with t using the valuation v is true.

The set of all ready transitions for M and v is denoted by $T_{\mathtt{ready}(\mathtt{M}, v)}$.
The transition t is **enabled** for M, written as $\mathtt{enabled}(\mathtt{t}, \mathtt{M})$, when the first four previous conditions, which are related only with the places linked to the transitions, are verified. A set $A \subseteq T_N$ is said to be **ready**, denoted as $\mathtt{ready}(A)$, if there exists a marking M and a valuation v, such that all transitions in A are ready for M and v. The set A is **enabled**, written as $\mathtt{enabled}(A)$, if there exists a marking M, such that all transitions in A are enabled. ■

Although the condition 2) of definition 2.4 is not very often used, it is a way to guarantee the boundedness of net, because a net describing a control unit should be safe.

Usually, the firing in a classical PN is defined as the firing of one, and only one, ready transition at each time [2]. So, there is no simultaneous firings of transitions. The SIP-net token game differs in several ways from the standard one for PNs: instead of just one transition firing at a time, all transitions that are ready to fire must do so; firing of a transition is blocked if any place in its postset is non-empty. The use of simultaneous firing of sets of transitions (usually called steps) is discussed in [17].

Definition 2.5: Let N be the structure of an SIP-net model, M a marking to N and $v \in \mathcal{V}_N$. The marking M' to N, obtained from M with the valuation v through the **simultaneous firing** of all $t \in T_N$, ready for M and v, written $\mathtt{M}[\rangle_v \mathtt{M}'$, is defined as:

$$\forall_{p \in P_N} M'(p) = \begin{cases} 0 & if\ p \in {}^\bullet T_{\mathtt{ready}(\mathtt{M}, v)} \\ 1 & if\ p \in T_{\mathtt{ready}(\mathtt{M}, v)}{}^\bullet \\ M(p) & otherwise. \end{cases}$$

In other words, the input places of all enabled transitions become empty, one token is added to the output places of all enabled transitions, and the other places are left unchanged. ■

We can consider the above definition as a mathematical relation between two markings and one valuation. Thus the reflexive and transitive closure is defined as follow.

Definition 2.6: Let N be the structure of an SIP-net, and M a marking to N. A marking M' to N is **accessible** from M, written $\mathtt{M}[*\rangle\mathtt{M}'$, if:

1) $M = M'$, or
2) $\exists_{M''} \quad \mathtt{M}[*\rangle\mathtt{M}'' \wedge (\exists_{v \in \mathcal{V}_N} \quad \mathtt{M}''[\rangle_v \mathtt{M}')$.

The set of all markings to N accessible from M is denoted by $[\mathtt{M}\rangle$.
A **firing sequence** is a sequence with the form $\mathtt{M}_0[\rangle_{v_0}\mathtt{M}_1[\rangle_{v_1}\ldots[\rangle_{v_{k-1}}\mathtt{M}_k[\rangle_{v_k}\ldots$ where $k \in \mathbb{N}$, for all i, M_i is a marking to N, and $v_{i-1} \in \mathcal{V}_N$. By definition of simultaneous firing we can observe that a transition $t \in T_N$ can be fired in a firing sequence if there exists a natural i, such that $\mathtt{ready}(\mathtt{t}, \mathtt{M_i}, v_\mathtt{i})$. ■

In order to have the behaviour of an SIP-net model completely defined it must be formed by a structure of an SIP-net plus a marking to this structure.

Definition 2.7: A pair $\mathbf{N} = (N, M_0)$ where M_0 is the initial marking to N, is called an **SIP-net**. ■

C. Some Properties of SIP-net models

Since transitions in SIP-net models fire synchronously, one may observe some conflicts amongst transitions, which are not present in other models of computation based on PNs.

Definition 2.8: Let $\mathbf{N} = (N, M_0)$ be an SIP-net, and $t_1, t_2 \in T$. The transitions t_1 and t_2 are in **structural conflict** if transitions t_1 and t_2 have a common pre-place (post-place), i.e., $^\bullet t_1 \cap {}^\bullet t_2 \neq \emptyset$ ($t_1{}^\bullet \cap t_2{}^\bullet \neq \emptyset$).

If the transitions t_1, t_2 are in structural conflict, they are also in **behavioural conflict** if there exist an accessible marking M and a valuation v that enable both transitions. ■

Let us now consider the formulation of the liveness property in the SIP-net modelling language. Often, the verification of liveness is very expensive and sometimes even impracticable. To overcome this difficulty, we follow the liveness levels proposed in [8], some of which reduce the cost of verification.

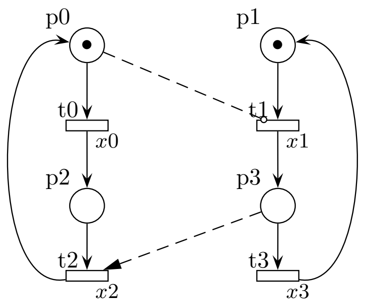

Fig. 2. An example of an SIP-net model.

Definition 2.9: Let $\mathbf{N} = (N, M_0)$ be an SIP-net and $t \in T_N$. The transition t is **L0-live (or dead)** when it can never be fired for any marking accessible from M_0:

$$\forall_{M' \in [\mathtt{M_0}\rangle} \quad \forall_{\mathtt{v} \in \mathcal{V}_R} \quad \neg\ \mathtt{ready}\,(\mathtt{t}, \mathtt{M'}, v);$$

The transition t is **L1-live**, if there exists at least an accessible marking from M_0, for which t can be fired:

$$\exists_{M' \in [\mathtt{M_0}\rangle} \quad \exists_{\mathtt{v} \in \mathcal{V}_R} \quad \mathtt{ready}\,(\mathtt{t}, \mathtt{M'}, v);$$

The transition t is **L4-live (or live)**, if it is L1-live for all markings accessible from M_0:

$$\forall_{M \in [\mathtt{M_0}\rangle} \quad \mathtt{L_1\text{-}Live}\,(\mathtt{N}, \mathtt{M}, \mathtt{t}).$$

The SIP-net $\mathbf{N}$ is said to be **Lk-live**, if every transition in the SIP-net N is **Lk-live**, where $k = 0, 1, 4$. ■

We are not considering in this work the liveness levels L_2 and L_3, which means respectively for each transition t: given any natural m, t can be fired at least m times in some firing sequence; and t appears infinitely, often in some firing sequence.

III. An Example on Specifying SIP-net Models with PROMELA

In this section, we present a specification in the PROMELA language for the illustrative SIP-net model represented in fig. 2, which is based on the relation, through enabling and inhibitor arcs, between two similar SIP-nets.

Formally, this SIP-net model is defined by the tuple (N, M_0):

- $N = (P, T, F, E, I, G)$ is the structure of the SIP-net model, where: $P = \{p_0, p_1, p_2, p_3\}$, $T = \{t_0, t_1, t_2, t_3\}$, $F = \{(p_0, t_0), (p_1, t_1), (t_0, p_2), (t_1, p_3), (p_2, t_2), (p_3, t_3), (t_2, p_0), (t_3, p_1)\}$, $E = \{(p_3, t_2)\}$, $I = \{(p_0, t_1)\}$ and $G = \{(t_0, x0), (t_1, x1), (t_2, x2), (t_3, x3)\}$;
- $M_0 = \{(p_0, 1)(p_1, 1)(p_2, 0)(p_3, 0)\}$ is the initial marking.

The specification of this SIP-net model in PROMELA uses the following guidelines:

- Each place is represented by a Boolean value and an array of Booleans, with length equal to the number of places, is used to represent the set of places;
- Each variable occurring in the guards is represented as a Boolean variable and the set of variables is represented by an array of Booleans with length equal to the number of variables.

```
1  #define M 4
2  #define Nvar 4
3  #define rd_t0 (p[0] && !p[2] && v[0])
4  #define rd_t1 (p[1] && !p[3] && !p[0] && v[1])
5  #define rd_t2 (p[2] && !p[0] &&  p[3] && v[2])
6  #define rd_t3 (p[3] && !p[1] && !p[2] && v[2])
7  #define fire_t0 p[0] = 0; p[2] = 1;
8  #define fire_t1 p[1] = 0; p[3] = 1;
9  #define fire_t2 p[2] = 0; p[0] = 1;
10 #define fire_t3 p[3] = 0; p[1] = 1;
11 bool p[M];
12 bool v[Nvar];
```

Fig. 3. Definitions and declarations in PROMELA.

With these guidelines, we present two approaches to obtain a PROMELA specification from an SIP-net model. These approaches differs on the way they treat with external variables occurring in the guards of SIP-net model. The first one uses an explicit representation of external variables. The second one does not use an explicit representation of variables, instead the semantics of variables are implicitly represented in the options of PROMELA specification, that is the second of the above guidelines will be not used.

A. Using Explicit Representation of External Variables

The SIP-net model has four places (p_0, p_1, p_2, p_3) and four different variables in the guards ($x0$, $x1$, $x2$, $x3$). In PROMELA, we write the specification shown in lines 1, 2, 11 and 12 in fig. 3 to declare those places and guards. The array p represents the places ($p[i]$ is the place p_i of the SIP-net model) and the array v represents the set of variables ($v[j]$ is the guard xj of the SIP-net model) The definition of the enabling and the firing conditions for each transition uses arrays p and v. With these representations, ready conditions and firing rules for each transition are defined from lines 3 to 10 in fig. 3, where rd_ti represents the ready condition of transition ti and $fire\_ti$ represents the firing rule of transition ti.

According to the definition of simultaneous firing (def. 2.5), all ready transitions at a given moment must be fired. Therefore, the PROMELA specification must include one firing rule for each of the fifteen possible non-empty combinations of the three transitions.

There is one (and only one) choice in the do-loop for each subset of transitions. The choice condition for a subset T' of transitions is constructed by:

1) the conjunction of the ready conditions of all the transitions in T', and
2) the conjunction of the negation of ready conditions corresponding to each transition not in T'.

The do-loop may present non-determinism. In the above case we only have deterministic choices, because at most only one guard can be made true at a given moment. This means that we can not have two or more guards simultaneously true. Informally, given two guards, there is at least one ready condition of a transition t_i which appears uncomplemented in one of the guard and appears complemented (or negated)

```
1  proctype randomVar(int ivar)
2  {do
3     :: atomic{ v[ivar] = 0 ; }
4     :: atomic{ v[ivar] = 1 ; }
5   od }
6  init{ atomic{
7    run randomVar(0);run randomVar(1);
8    run randomVar(2); run randomVar(3);
9    p[0]=1; p[1]=1; p[2]=0; p[3]=0; /*Initial Marking*/
10   run procParSyn();}}
```

Fig. 4. PROMELA process for specifying the SIP-net model example.

in the another one. Some of the guards are always false. We are not interested in considering the guards with all the ready conditions negated, because there is no action defined to that guard.

The behaviour of an SIP-net model depends on external stimuli through the variables occurring in the guards. In this PROMELA specification, the variables are present on the ready conditions (see lines 3 to 6 in fig. 3).

In an SIP-net model, nothing is known about the behaviour of its environment, more precisely the value of the variables are not modelled by the SIP-net model. Thus, we must consider all the possibilities for the value of each variable. In PROMELA, it can be done as illustrated in fig. 4, where the parameter ivar identifies the variable to be considered. The do-loop gives non-deterministically the value 0 or 1 to the considered variable.

Running one process for each variable allows us to simulate the behaviour of the environment. The main process, for the SIP-net model example, firstly creates four instances of the randomVar process, one for each variable, (see lines 7 and 8 in fig. 4). After that, the places are initialized according to the initial marking. Finally, an instance of the procParSyn process is created. The corresponding PROMELA specification is presented in fig. 4.

To study the appropriateness of the PROMELA specification for the considered example, its behaviour has been simulated with the Spin tool. By running the interactive simulation, we have the possibility to choose one of two different values (0 or 1), for each of the guards $x0$, $x1$, $x2$, and $x3$. We have these options in all points of the simulation, which allow us to control the behaviour of the environment. In the PROMELA specification, both the controller and the environment run concurrently. Thus, we obtain a simulation of the system, whose non-determinism is introduced by external variables.

In this PROMELA specification we are supposing that changing values of variables is part of system's behaviour, but this not the case for the SIP-net model, where the variables are considered elements of the environment.

Let us see what happens when we are using the Spin tool to analyse the obtained PROMELA specification.

When simulating the PROMELA specification, since there is no explicit representation of environment's behaviour, it is possible to change the value of all variables in any point of execution, because as we said before the non-determinism is introduce by the value of each variable. In this way, when changing the variables we are defining the set of transitions to be fired. If none of the guards associated to the enabled transition, for the selected values, are evaluated to true we have a situations that the system we are simulating can not go to one its next states.

When we are simulating this useful, because we are only manipulating the environment and observing what happens to the system.

This solution is not adequate for verification of system's properties, because it is possible to obtain a path execution where there are only the actions changing the value of one variable, which are not actions of the system we are considering. In this way when we try to verify a system property we have not the expected result. For example, when we try to verify on this PROMELA specification the liveness property of transition t_0, which is based on the inspection if for all states (in the system generated by the PROMELA specification) there is at least one next state validating the ready condition for transition t_0. When using the Spin tool to verify this property the obtained result is "not valid". Running the resulting "trail" simulation is based on an execution where the initial value of variable x_0 (false) is never changed, and consequently the transition t_0 never becomes ready, because t_0 has the variable x_0 as its guard.

B. Without Explicit Representation of External Variables

To overcome this problem, which prevents us from analysing the behaviour of the system, we adopt a different solution. The key idea of our approach to generate the PROMELA specification is that the semantics of guards are present, without explicitly representing the variables. This is possible if we guarantee that in each point of the PROMELA specification execution there are all the possible choices corresponding to the different valuations of the variables.

In the SIP-net model of fig. 2, transitions t_0 and t_1 are never enabled for a given marking, because there is a inhibitor arc from place p_0 to transition t_1. When analysing the SIP-model it is not necessary to consider that guards $x0$ and $x1$ are both true.

For a marking which puts a token into the places p_2 and p_3 (notice that this marking is accessible from the initial marking), the transitions t_2 and t_3 are both enabled. The variables $x2$ and $x3$ permit the sequential (non-concurrent) firing of transitions t_2 and t_3, i.e., transition t_2 can be fired without t_3 being fired at the same time (or vice-versa). This happens if $x2$ is true and $x3$ is false (or $x2$ is false and $x3$ is true). We still can fire these two transitions simultaneously if $x2$ and $x3$ are both true. We are not interested in the case that transitions $x2$ and $x3$ are both false.

Now, we can apply these ideas to the PROMELA specification for the considered example. Since in this solution there are no variables in the PROMELA specification, it is necessary to remove the references to the variables in the definition of the ready conditions. These new conditions only include references to places and are therefore called enabled conditions (see def. 2.4). They have the PROMELA definition presented in fig. 5.

```
#define en_t0 (p[0] && !p[2])
#define en_t1 (p[1] && !p[3] && !p[0])
#define en_t2 (p[2] && !p[0] &&  p[3])
#define en_t3 (p[3] && !p[1])
```

Fig. 5. Definition of the enabled conditions.

v_i	$x0$	$x2$	$\neg x0 \wedge \neg x2$	Transitions
v_1	1	1	0	$\{t_0, t_1, t_2\}$
v_2	1	0	0	$\{t_0, t_1\}$
v_3	0	1	0	$\{t_2\}$
v_4	0	0	1	$\{t_3\}$

Fig. 6. Truth values of guards.

The structure of the main process is a do-loop, whose guards are based on the conjunction of the conditions corresponding to the enabled condition of a given subset of transitions.

In the solution that uses variables, when we have a valuation to the variables and a marking, we know that at most only one line of the do-loop has choice condition that holds true.

With one line in the PROMELA specification for each subset of transitions, we guarantee that the subsets of transitions, which are not expected to fire, have a false choice condition associated with them. In the solution without references to variables, we can not put one line for each subset of transitions. We select only the subsets of transitions, whose corresponding guards can be simultaneously true. For each $A \subseteq T_N$, we write $G_N(A)$ to denote $\bigcup_{t \in A}\{X : (t, X) \in G_N\}$. To write the PROMELA specification for the SIP-net model example, we select all the consistent sets in $\{G(A) | \emptyset \subset A \subseteq T_N\}$.

With respect to the SIP-net model in fig. 2, the sets of the variables in the guards are disjoint and guards could have the value true or false, so all the sets are consistent.

We most consider only the subsets of transitions which elements could be enabled for a given marking, otherwise we are generating a correct PROMELA specification but some of choices are "dead" code and so they could be removed.

We now change the guards of the transitions of the SIP-net model, such that: $G(t_1) = x0$, $G(t_3) = \neg x0 \wedge \neg x2$.

The variables occurring in the guards are $x0$ and $x2$. The variable $x0$ occurs in the guards of transitions t_0, t_1 and t_3, and the variable $x2$ occurs in the guards of transitions t_2 and t_3. In fig. 6, for each combination of the values for $x0$ and $x2$, the Boolean values of the guards are presented. The last column shows the set of transitions whose guards are validated by the valuation (v_i) in the row.

Although the guards of the transitions t_0, t_1 and t_2 are consistent, none of the transitions is simultaneously enabled with the other two transitions, due to the restrictions imposed by the structure of the SIP-net model. Thus, by the valuation v_1 in fig. 6, the transitions t_0, t_1, and t_2 can be fired alone. There exists a marking enabling transitions t_2 and t_3, but as we can state in fig. 6 there is no valuation validating the ready condition for this two transitions. Thus the set of transitions $\{t_2, t_3\}$ is not considered in the do-loop of PROMELA specification.

IV. Rules to Specify SIP-net Models with PROMELA

Based on the ideas expressed in the previous sections, we define more general rules in this section. A PROMELA specification is constructed from three basic types of objects: processes, data objects, and messages.

The principal process is called init. Many of PROMELA notational conventions derive from the C language, including declaration and initialization of variables. The do-loop statement gives a cyclic non-deterministic choice of one guard, and each guard has actions associated with it.

For a given SIP-net model, we define that the corresponding PROMELA specification has three parts: (1) the definition of the enabled condition for each transition, (2) the definition of the firing condition for each transition, and (3) the do-loop in the init function.

The first two parts are straightforward to obtain, because they only depend upon the structure of the SIP-net model. The do-loop is harder to obtain, because it must include all the possible subsets of transitions that may fire simultaneously. To define the alternative choices of the do-loop, some calculations must be performed based on the guards of each transition.

Firstly, we calculate the subsets of transitions which may be simultaneously enabled. Notice that subsets of this set are still sets of enabled transitions, and thus, we need only to consider the maximal subsets of enabled transitions, independently of the valuation of their guards. Let us denote the set of such maximal subsets by $PEMAXS$.

Secondly, for each $MT \in PEMAXS$, we calculate its maximal subsets of ready transitions, i.e., we consider the maximal subsets of MT whose guards can be made true under the same valuation. Let $T' = \{t_1, \ldots, t_k\}$ be such a subset of MT. Although all the transitions in T' could be ready to fire simultaneously, given a marking it may enable only a subset of the transitions in T'. Thus when calculating the guards of the do-loop, we must consider the firing of all its subsets. Without loss of generality, we study what happens to the subset $T' \setminus \{t_k\}$. There exists a marking for which all transitions in $T' \setminus \{t_k\}$ are enabled and t_k is not enabled.

The guard corresponding to the set $T' \setminus \{t_k\}$ in the do-loop is given by the conjunction $C_1 \wedge C_2$. Condition C_1 is simply the conjunction of the enabled conditions of the transitions in $T' \setminus \{t_k\}$. For condition C_2, firstly we calculate the set T'' of transitions which contains the transition t_k and the transitions in the set $\bigcup_{A \in PEMAXS \wedge A \supset (T' \setminus \{t_k\})} \quad A \setminus (T' \setminus \{t_k\})$.

Secondly, we calculate the subset T''' of T'' consisting of the transitions not validated by any of the valuations which validate the transitions in $T' \setminus \{t_k\}$. Condition C_2 is then the conjunction of the negations of the enabled conditions with respect to transitions in T'''.

Performing the previous calculations to all subset T'' of MT, and all MT in $PEMAXS$, we obtain a do-loop modelling the structure of the SIP-net model, where the non-deterministic choices correspond to the choice of a valuation, for a given marking.

```
type SIPnet      = (Structure, Marking, Context)
type Structure = [Trans]
type Marking   = [Place] -- marked places
type Context   = [Guard]
data Trans     = Trans TransId [Place]
                    [Place] [Place] Guard [Place]
data Place      = Place PlaceId
type Guard      = Formula
type TransId    = Id
type PlaceId    = Id
type Id = String
```

Fig. 7. Haskell data types to represent SIP-net models.

```
parsyn :: SIPnet
parsyn = [
 Trans "t0" [Place "p0"] [] [] (Var "x0") [Place "p2"],
 Trans "t1" [Place "p1"] [] [Palce "p0"]
                             (Var "x1") [Place "p3"],
 Trans "t2" [Place "p2"] [Place "p3"] []
                             (Var "x2") [Place "p0"],
 Trans "t3" [Place "p3"] [] [] (Var "x3") [Place "p1"] ]
prsm30 = [Place "p0",Place "p1"]
sparsyn= (parsyn,prsm30,[])
```

Fig. 8. Haskell specification of the SIP-net model example.

The ready conditions are the basic elements used to specify the behavioural properties of the SIP-net models. In the PROMELA specification, there is no explicit representation of the transitions' guards. Thus, the ready condition, for a transition t, is the disjunction of all guards in the do-loop, in which the enabled condition for t occurs. Notice that the ready condition of two or more transitions is not the conjunction of the corresponding ready conditions of those transitions, but the disjunction of the guards in the do-loop, in which enabled conditions for all considered transitions occur.

V. The Computer Application

In this section, the computer application that was created to generate the PROMELA specification for a given SIP-net model is presented. The aplication was written in Haskell.

A. Constructing the Application

In order to describe an SIP-net model in Haskell we use the data type definition presented in fig. 7, that is we represent an SIP-net model in Haskell as a triple with the structure of the net, a marking and a context.

A marking of a net is represented in Haskell as the set of places which have one token. In Haskell a set is represented as an Haskell list. Notice that SIP-net models are safe nets, so a place has at most one token. This allows the use of a set to represent the marked places. A context is a set of guards, which is in Haskell a formula of propositional logic.

The structure of an SIP-net model is a set of transitions. Each transition has a label of identification (TransId), the set of pre-places, the set of the places linked through enabling arcs, the set of the places linked through inhibitor arcs, a guard, and the set of post-places, respectively. For example, the SIP-net model illustrated in fig. 2 has the Haskell representation presented in fig. 8.

Next we present functions included in the tool to implement the calculations described in the previous section. We also show its usage in the Haskell representation in fig. 8. The function enabled has the following signature:

```
enabled :: SIPnet -> [[Trans]]
```

Given an SIP-net model, the program calculates the maximal subsets of enabled transitions, which is denoted by $PEMAXS$ in the previous section. Applying this function to the SIP-net model, we obtain the following result:

```
*Ex_parsyn> enabled sparsyn
[[t1],[t2,t3],[t0,t3]]
```

For each subset of $PEMAXS$ we calculate the maximal subsets whose transitions could be simultaneously ready.

Given an SIP-net model and the set of enabled transitions, it is calculated the set of pairs, such that the first component has the transitions that can be ready. This ready condition is determined by the no satisfaction of the enabled condition of transition in each element of the second component.

B. Using the Application

Using the previous results we can generate the PROMELA specification for the SIP-net model. Additionally, we generate the TL conditions to test the potential conflicts in the SIP-net model. These conditions are based on their ready condition, already included in the PROMELA specification.

The toPROMELA function creates a String from an SIP-net model, and has the following signature:

```
toPROMELA :: SIPnet -> String
```

The result of applying toPROMELA function to a SIP-net model is a string with the PROMELA specification corresponding to the SIP-net model, that constitute the input to Spin tool to analyze the properties of the given SIP-net model. For the considered example we have the following PROMELA specification:

```
1  *Ex_parsyn> (putStr.toPROMELA) sparsyn
2  #define en_t0 (p[0] && !p[2])
3  #define en_t1 (p[1] && !p[3] && !p[0])
4  #define en_t2 (p[2] && !p[0] &&  p[3])
5  #define en_t3 (p[3] && !p[1])
6  #define fire_t0 p[0] = 0; p[2] = 1;
7  #define fire_t1 p[1] = 0; p[3] = 1;
8  #define fire_t2 p[2] = 0; p[0] = 1;
9  #define fire_t3 p[3] = 0; p[1] = 1;
10 /* Enabled Conditions for each transition */
11 #define ready_t0 (en_t0 && en_t3) || (en_t0)
12 #define ready_t1 (en_t1)
13 #define ready_t2 (en_t2 && en_t3) || (en_t2)
14 #define ready_t3 (en_t0 && en_t3) || (en_t3)
15 /* Enabled conditions for transitions
16 in a potential conflict. */
17 bool p0, p1, p2, p3;
18 init{ atomic{ p0 = 1; p1 = 1; p2 = 0; p3 = 0;  }
19 do
20   :: en_t0 && en_t3 -> atomic{ fire_t0; fire_t1; }
21   :: en_t2 && en_t3 -> atomic{ fire_t0; fire_t1; }
22   :: en_t0          -> atomic{ fire_t0; }
23   :: en_t1          -> atomic{ fire_t1;}
24   :: en_t2          -> atomic{ fire_t2;}
25   :: en_t3          -> atomic{ fire_t3;}
26 od }
```

In this PROMELA specification, for each transition, we have the definitions of: the enabled conditions (between lines 2 and 5), the fire actions (between lines 6 and 9), and the ready conditions (between lines 10 and 14).

These definitions use the variables representing the places of the net, which are declared in line 17. After that we have the init process. In line 18 the marking values are assigned to the places. The do-loop guards, lines between 21 and 25, represent the different sets of transitions which can fire simultaneously, and the corresponding action to fire the transitions.

The ready condition, for a transition t, is the disjunction of all guards in the do-loop, which have an occurrence of enabled condition for t. This can be seen in the previous example on the definition of ready conditions. The ready condition of two or more transitions is not the conjunction of the corresponding ready conditions to those transitions, but the disjunction of the guards in the do-loop, in which the enabled conditions for all considered transitions occurs. The ready conditions are the basic elements when specifying the behavioural properties of SIP-net models.

To check if a potential conflict among transitions constitute a behavioural conflict we need to evaluate the simultaneous enabling of these transitions, thus we need to have the enabling conditions of transitions in potential conflict. Thus we decide to define only the following conditions:

- the ready conditions for each transition (presented between lines 11 and 14); and
- the ready conditions for each set of transitions in potential conflict (in the example there are no potential conflicts, thus no enabling conditions for more than one transition is defined, but we put commentary on line 15).

C. Verifying SIP-nets Properties

Next, we present the specification and the verification of properties of the SIP-net models in the context of the PROMELA specification, using the TL formulas. The three SIP-net model properties considered here are: (1) behavioural conflicts freedom, (2) dead-lock freedom, and (3) liveness of its transitions.

To check the freedom of behavioural conflicts in SIP-net models, we use the TL formula with the $\Box$ operator and the negation of the ready condition corresponding to the set of transitions that are in conflict. If the transitions in potential conflict are the transitions t_i and t_j, the formula is the following: $\Box\ \neg ready\_ti\_tj$.

According to def. 2.9 we can define in TL two levels of liveness (L_1 and L_4) in the context of the generated PROMELA specification. The operators $\Box$ and $\Diamond$ represent, respectively, the universal, and the existential quantification over the states of the system.

Given an SIP-net model N and $t \in T_N$. The TL formula to be verified in order to prove that: $t\ is\ L_1\text{-}Live$ is checked by the formula $\Diamond\ ready\_t$; and $t\ is\ L_4\text{-}Live$ is checked by the formula $\Box\Diamond\ ready\_t$. In this way, we can study the behavioural conflicts freedom, liveness of system's transitions and also the absence of dead-locks.

VI. Conclusions

In this article, an approach to apply the model checking technique using the Spin tool is presented. This approach allows important properties of embedded systems, such as liveness, deadlock-freedom, and the absence of structural conflicts among transitions, to be verified. The need to verify properties of a computer-based system is of paramount importance, namely when the systems are safety-critical.

In the proposed approach, the behaviour of the embedded systems (i.e., its control part) is modelled with a variant of Petri Nets, called SIP-net. When compared to traditional PN models, SIP-net models present guards associated to transitions, inhibitor and enabling arcs, and synchronous firings of the transitions. Due to this synchronous nature of the firings, the description of the SIP-net models with PROMELA is not trivial, because the Spin tool assumes the existence of a finite-state system where only one transition fires at each instant.

Since the values of the input signals are not known in the modelled system, we must consider all their possible values. In the SIP-net models the input signals are used as variables of guards. In this way, all possible scenarios for values of guards must be considered.

Therefore, this work discusses in some detail how SIP-net models should be specified with the PROMELA language (input format for the Spin model checker), so that some of their behaviour properties are verified. This model checking approach, namely the Haskell program, was already used in a case study [14].

References

[1] Z. Manna and A. Pnueli, *The Temporal Logic of Reactive and Concurrent Systems: Specification*. New York, USA: Springer-Verlag, 1992.

[2] W. Reisig, *Petri Nets - An introduction*, EATCS monographs on theorical computer science ed. Heidelberg, Germany: Springer-Verlag, 1985.

[3] M. A. Adamski, A. Karatkevich, and M. Wegrzyn, Eds., *Design of Embedded Control Systems*. Springer, Berlin, 2005.

[4] A. Yakovlev, L. Gomes, and L. Lavagno, Eds., *Hardware Design and Petri Nets*. Springer, Berlin, 2000.

[5] J. M. Fernandes, M. A. Adamski, and A. J. Proença, "VHDL Generation from Hierarchical Petri Net Specifications of Parallel Controller," *IEE Proceedings: Computers and Digital Techniques*, vol. 144, no. 2, pp. 127–37, Mar. 1997.

[6] M. A. Adamski, "Direct Implementation of Petri Net Specification," in *7th Int. Conf. on Control Systems and Computer Science*, 1987, pp. 74–85.

[7] G. De Micheli, *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, 1994.

[8] T. Murata, "Petri Nets: Properties, Analysis and Applications," in *Procedings of the IEEE*, April 1989, pp. 541–80.

[9] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. MIT Press, 2000.

[10] G. J. Holzmann, "Design and validation of computer protocols," *IEEE Trans. on Software Engineering*, vol. 23, no. 5, pp. 279–95, May 1997.

[11] ——, *The Spin Model Checker: Primier and Reference Manual*. Addison-Wesley, September 2003.

[12] ——, *Design and Validation of Computer Protocols*. New Jersey: Prentice Hall, 1991.

[13] T. C. Ruys, "Towards effective model checking," Ph.D. dissertation, University of Twente, Department of Computer Science, Mar. 2001.

[14] O. R. Ribeiro, J. M. Fernandes, and L. F. Pinto, "Model Checking Embedded Systems with PROMELA," in *12th IEEE Int. Conf. on the Eng. of Computer Based Systems (ECBS 2005)*. IEEE Computer Society Press, 2005, pp. 378–85.

[15] J. L. Peterson, *Petri Net Theory and the Modeling of Systems*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1981.

[16] J. Kleijn and M. Koutny, "Process Semantics of P/T-Nets with Inhibitor Arcs," in *ICATPN*, 2000, pp. 261–81.

[17] M. Mukund, "Petri Nets and Step Transition Systems," *Int. Journal of Foundations of Computer Science*, vol. 3, no. 4, pp. 443–78, 1992.