Model Checking Embedded Systems with PROMELA *

Óscar R. Ribeiro[1], João M. Fernandes[1], Luís F. Pinto[2]
[1]Dept. Informática, [2]Dept. Matemática
Universidade do Minho, Braga, Portugal
oscar.rafael@di.uminho.pt

Abstract

The design process for embedded systems can benefit from the usage of formal methods, if some properties of the systems are checked, before design and implementation decisions are accomplished. This paper presents a model checking approach using the Spin tool, to verify some important properties of embedded systems, namely liveness, deadlock-freedom, and structural conflicts among transitions. The systems are modelled with a variant of Petri Nets, called SIPN (Synchronous and Interpreted Petri Nets), and this paper discusses how SIPN models should be specified with the PROMELA language (input format for the Spin model checker). The approach is exemplified with a case study.

1. Introduction

Embedded systems are a special type of computer based software systems that present, among others, the following characteristics: reactive, concurrent, safety-critical [4]. For developing these systems, the design process should use formal methods, before final design and implementation decisions are taken.

Concurrency is considered "the" essential feature of reactive systems [15], a class of systems in which embedded systems are included. Development of embedded software requires meta-models (models of computation) that explicitly support concurrency. A concurrent software system is a collection of sequential processes which in abstract are executed in parallel [22, 3, 13, 11], 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) [20] constitute meta-model for expressing concurrent systems, due to their extensive body of results, both theoretical and practical. For developing embedded systems, the application of PNs for modeling, simulation, verification, and synthesis purposes is covered in the literature. The SIPN meta-model was obtained by the enrichment of a safe PN 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. Synchronous circuits represent the largest portion of circuit designs and the state of the art in synthesizing synchronous systems is more advanced and stable than the corresponding one for asynchronous circuits [16].

Model checking [2] is a verification technique that is based on the idea of exhaustively exploring of the reachable state space of a system. The model checker Spin [7, 8] is a verification system, which accepts a specification language called PROMELA (a Process Meta Language) [6, 8, 21]. 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. Testing can only indicate errors and never their absence. It is quite useful, but verification is the only way to guarantee that a system is free of errors, an essential condition for safety-critical systems.

This work presents a model checking approach, using the Spin tool, to verify important properties of embedded system such as liveness, deadlock-freedom, and the absence of structural conflicts among transitions. It discusses in detail how SIPN models of embedded

* This work was partially supported by projects METHODES (contract ref. POSI/2001/CHS/37334) and STACOS (contract ref. POSI/CHS/48875/2002)

systems should be specified with the PROMELA language. It is also presented how the properties to be verified are expressed with Temporal Logic (TL) formulas.

In section 2, we present a formalization of the SIPN meta-model and some of its properties, and in section 3, we discuss how to represent SIPN models with PROMELA language. In section 4 the reactor case study is presented, namely the usage of the model checking approach.

2. SIPN Meta-model

In this section, we introduce the meta-model of SIPNs. We follow the definitions of net and place/transition-systems introduced in [20]. We use place/transition-systems where each place has a unitary capacity, and the weight of each arc is also unitary. This meta-model has been enriched in three different ways. Firstly, two new types of arcs are allowed: enabling arcs (also known as read arcs, test arcs, or positive context arcs), and inhibitor arcs (also negative context arcs) [23, 17, 12, 9]. Secondly, each transition has now an associated guard which is a propositional formula where variables represent input signals of the modeled system, i.e., guards over transitions are formulas containing external variables, which can influence the enabling of transitions. Another difference to the basic meta-model of PN is the presence of synchronism, that affects the usual notion of accessibility, using simultaneous firing of all enabled transitions.

Definition 2.1 *The **structure of a synchronous and interpreted Petri net** (written structure of a SIPN) is a tuple $N = (P, T, F, E, I, G)$ such that:*

1. *(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.*
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. *$G : T \rightarrow PROP$ 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 $\{0, 1\}$.* ■

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

Definition 2.2 *Let N be a structure of a SIPN. For each $t \in T_N$, ${}^\bullet t = \{p \mid p \, \mathrm{F_N} \, t\}$ is called the **preset** of t, $t^\bullet = \{p \mid t \, \mathrm{F_N} \, p\}$ is called the **postset** of t, ${}^\triangleright t = \{p \mid p \, \mathrm{E_N} \, t\}$, and ${}^\circ t = \{p \mid p \, \mathrm{I_N} \, t\}$.*

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 a net is influenced by the interpretation of the variables appearing in its guards (external events). Thus, to simulate the behaviour of a net, 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 SIPNs.

Definition 2.3 *Let N be the structure of a SIPN, M a marking to N, $v \in \mathcal{V}_N$ and $t \in T_N$. The transition t is **enabled for** M **and** v, written as* $\texttt{enabled}(t, M, v)$, *if*

1. $\forall_{p \in {}^\bullet t} \, M(p) = 1$;
2. $\forall_{p \in t^\bullet} \, M(p) = 0$;
3. $\forall_{p \in {}^\triangleright t} \, M(p) = 1$;
4. $\forall_{p \in {}^\circ t} \, M(p) = 0$;
5. *the interpretation of the guard associated with t using the valuation v is true.*

The set of all enabled transitions for M and v is denoted by $T_{\texttt{enabled}(M,v)}$.

*The transition t is **place enabled for** M, written as* $\texttt{p–enabled}(t, M)$, *when the first four of 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 enabled, denoted as $enalbed(A)$, if there exists a marking M and a valuation v, such that all transitions in A are enabled for M and v. The set A is p–enabled (*$\texttt{p–enabled}(A)$*) if there exists a marking M, such that all transitions in A are p–enabled.* ■

Usually, the firing in a PN is defined as the firing of one, and only one, enabled transition at each time [20]. Thus, from the same marking, we can get different next markings depending on the selected enabled transition. We use a generalization of the traditional firing notion, where the next marking is obtained through the simultaneous firing of all the enabled transitions with the actual marking and valuation. There are several papers, that discuss simultaneous firing of sets of transitions (usually called steps) [18].

Definition 2.4 *Let N be the structure of a SIPN, 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 $t \in T_N$, enabled for M and v, written $\mathtt{M}[\rangle_v\mathtt{M}'$, is defined as:*

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

In other words, the input places of all enabled transition become empty, one token is added to the output places of all enabled transition 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 2.5 *Let N be a SIPN, 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 $\mathtt{enabled}(t, M_i, v_i)$.* ■

Definition 2.6 *A pair $\mathbf{N} = (N, M_0)$ where M_0 is the* ***initial*** *marking to N, is called an* ***SIPN***. ■

Let us now consider the formulation of the liveness property in the SIPN meta-model. It is important to retain that often the verification of liveness is very expensive and sometimes even impracticable. To overcome this difficulty, we follow some of the liveness levels proposed in [19].

Definition 2.7 *Let $\mathbf{N} = (N, M_0)$ be a SIPN and $t \in T_N$. The transition t is:*

dead (*$\mathbf{L_0}$*-Live) *when t can never be fired for any marking accessible from M_0, i.e.*

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

$\mathbf{L_1}$-Live *if exists at least an accessible marking from M_0, for which t can be fired, i.e.*

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

$\mathbf{L_4}$-Live *(or live) if t is L_1-Live for all markings accessible from M_0, i.e.*

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

The SIPN **N** *is said to be L_k-Live if every transition in the net N is L_k-Live, where $k = 0, 1, 2, 3, 4$.*

As a matter of fact, liveness of levels L_2 and L_3 means that, for each transition $t \in T_N$: given any natural k, t can be fired at least k times in some firing sequence; and t appears infinitely, often in some firing sequence. ■

In the next sections we define a methodology to check if a given SIPN verifies the above properties.

3. Checking SIPN Models

Verification is the process of analysing a system for some desired properties. There are two widely used approaches to verification: theorem proving and model checking.

In the theorem proving approach the system and its desired properties are expressed as formulas in some appropriate logic. Such logic is given by a formal system, which defines a set of axioms and a set of inference rules. Theorem proving is then the process of finding a proof of a property from the axioms of the system. Model checking is a technique based on the creation of a finite model of a system and checking that a property is valid in the state-space. In that technique it is important to use algorithms and data structures that permit large search spaces.

In this section we discuss how to represent SIPN models with the PROMELA language, to verify some of their properties. A PROMELA model 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 SIPN the corresponding PROMELA model has three parts:

- the definition of the *p–enabled* condition for each transition;
- the definition of the fire condition for each transition;
- the *do-loop* in the *init* function.

The first two parts are simple to obtain, because they depend only upon the structure of the SIPN.

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 $p{-}enabled$ (recall that set of transitions is $p{-}enabled$ if there exists a potential marking place enabling all transitions in it). Notice that subsets of this sets are still sets of $p{-}enabled$ transitions, and thus, we need only to consider the maximal subsets of $p{-}enabled$ transitions (independently of their guards' valuation). Let us denote the set of such maximal subsets by $PEMAXS$.

Secondly, for each $MT \in PEMAXS$, we calculate its maximal subsets of enabled 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 enabled to fire simultaneously, given a marking it may only $p{-}enabled$ some of transitions in T'. Thus when calculating the guards of the *do-loop* we must consider the firing of all its subsets. (Observe that if a marking place enables all transitions in T', we still have to consider potential situations where only some of the guards of T' transitions are being validated). 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 $p{-}enabled$ and t_k is not $p{-}enabled$.

The corresponding guard in the *do-loop* is given by the conjunction $C_1 \wedge C_2$. Condition C_1 is simply the conjunction of $p{-}enabled$ 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 $\bigcup_{A \in PEMAXS \wedge A \supset (T' \setminus \{t_k\})} A \setminus (T' \setminus \{t_k\})$.

Secondly, we calculate the subset T''' of T'' consisting of a transitions not validated by none of the valuations which validate the transitions in $T' \setminus \{t_k\}$. Condition C_2 is then the conjunction of the negations of the $p{-}enabled$ conditions relative to transitions in T'''.

Doing this to all subset T'' of MT, and all MT in $PEMAXS$, we obtain a *do-loop* which models the structure of the SIPN, where the non-deterministic choices correspond to the choice of a valuation, for a given marking.

The enabled conditions of the transitions are the basic elements used to specify the behavioural properties of the SIPNs.

In the PROMELA model, there is no explicit representation of the transitions' guards. Thus, the enabled condition, for a transition t, is the disjunction of all guards in the *do-loop*, in which the $p{-}enabled$ condition for t occurs. Notice that the enabled condition of two or more transitions is not the conjunction of the corresponding enabled conditions to that transitions, but the disjunction of the guards in the *do-loop*, in which $p{-}enabled$ conditions for all considered transition occur.

We have written an application (in Haskell [10]) to generate the PROMELA model from the corresponding SIPN description.

In the next section we show some results from the use of this application in the considered case study.

4. Case Study: The Reactor

In order to illustrate how to model check SIPNs with PROMELA (and the Spin tool), using the ideas in the previous sections, we consider a case study.

The example, used in [1, 14], is a reactor that controls the filling of a tank (see figure 1).

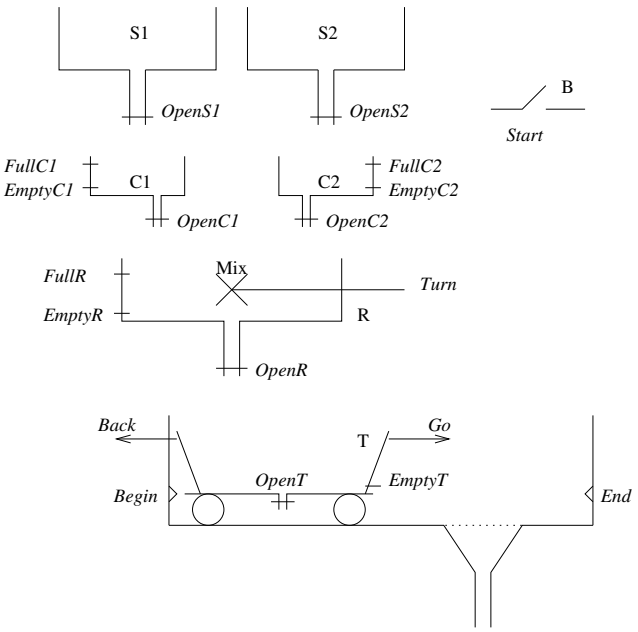

Figure 1. The environment of the reactor

4.1. First Version

The behaviour of the reactor is modelled by the SIPN in figure 2.

The input signals to transitions (e.g. *FullC1*, *End*) are identified with the values of the guards for transitions, assuming also that a transition guard is always true if there is no input signal to it (e.g. t_4). The

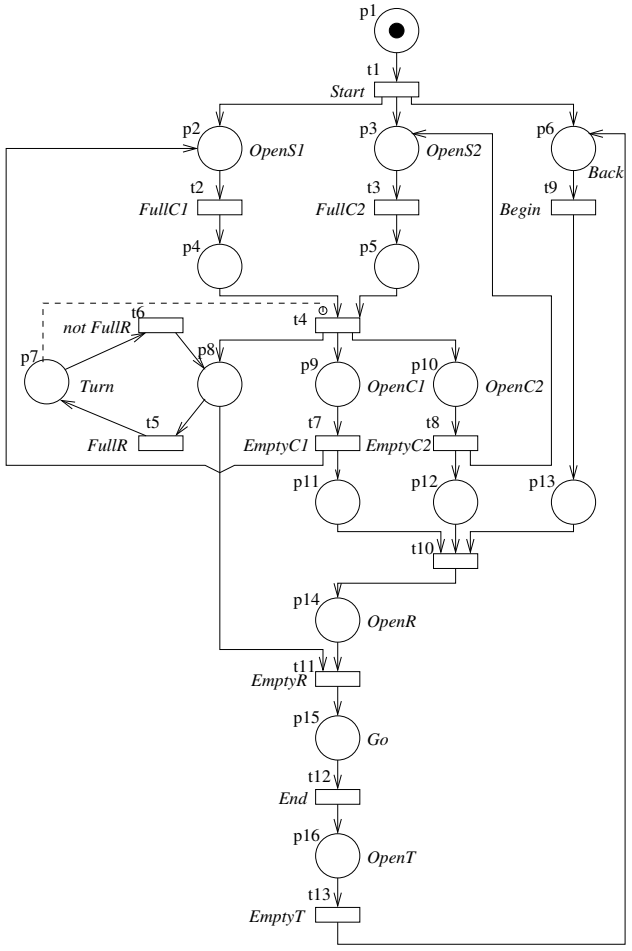

Figure 2. An SIPN for describing the dynamic behaviour of the reactor

```
t1  = Trans "t1" [Place "p1"] [] []
          (Var "Start")
          [Place "p2", Place "p3", Place "p6"]
t2  = Trans "t2" [Place "p2"] [] []
          (Var "FullC1")
          [Place "p4"]
t3  = Trans "t3" [Place "p3"] [] []
          (Var "FullC2")
          [Place "p5"]
t4  = Trans "t4" [Place "p4", Place "p5"]
          [] [Place "p7"]
          (Var "")
          [Place "p8",Place "p9",Place "p10"]
t5  = Trans "t5" [Place "p8"] [] []
          (Var "FullR")
          [Place "p7"]
t6  = Trans "t6" [Place "p7"] [] []
          (Neg (Var "FullR"))
          [Place "p8"]
t7  = Trans "t7" [Place "p9"] [] []
          (Var "EmptyC1")
          [Place "p2",Place "p11"]
t8  = Trans "t8" [Place "p10"] [] []
          (Var "EmptyC2")
          [Place "p3",Place "p12"]
t9  = Trans "t9" [Place "p6"] [] []
          (Var "Begin")
          [Place "p13"]
t10 = Trans "t10" [Place "p11",Place "p12",
                     Place "p13"] [] []
          (Var "")
          [Place "p14"]
t11 = Trans "t11" [Place "p8",
                     Place "p14"] [] []
          (Var "EmptyR")
          [Place "p15"]
t12 = Trans "t12" [Place "p15"] [] []
          (Var "End")
          [Place "p16"]
t13 = Trans "t13" [Place "p16"] [] []
          (Var "EmptyT")
          [Place "p6"]

reactor        =[t1, t2, t3, t4, t5, t6, t7,
                   t8, t9, t10, t11, t12, t13]
reactor_context=
 [Or (Neg (Var "FullC1" ))
     (Neg (Var "EmptyC1" )),
  Or (Neg (Var "FullC2" ))
     (Neg (Var "EmptyC2" )),
  Or (Neg (Var "FullR"  ))
     (Neg (Var "EmptyR"  )),
  Or (Neg (Var "Begin"  ))
     (Neg (Var "End"      ))]
reactor_m0      = [Place "p1"]
sreactor        = (reactor, reactor_m0,
                     reactor_context)
```

Figure 3. Haskell representation of the reactor's SIPN

guards are complemented with a context to be considered when interpreting the variables occurring in the guards. For example the variables $FullR$ and $EmptyR$ could never be both true at the same time, thus the formula $\neg\,(FullR \wedge EmptyR)$ must be included in the context.

In our Haskell specification an SIPN is a triple with an initial marking, a set of formulas (context) and a set of transitions. Each transition has a guard, and the sets of pre-places, post-places and the places linked through enabling and inhibitor arcs. In figure 3 we have the Haskell representation of the reactor's SIPN.

Using the application which implements the concepts presented in the previous section, we obtain the PROMELA model corresponding to the SIPN shown in figure 2. A fragment of this PROMELA model is presented in figure 4. We show the lines of the PROMELA model that are linked to the simultaneous firing of tran-

```
1  #define pen_t4 (p4&&p5&&!p7&&!p8&&!p9&&!p10)
2  #define fire_t4 p4=0;p5=0; p8=1; p9=1; p10=1;
3  #define pen_t10 (p11&&p12&&p13&&!p14)
4  #define fire_t10 p11=0; p12=0; p13=0; p14=1;
5  ...
6  #define enabled_t1_t7 \
7   (pen_t1&&pen_t5&&pen_t7&&pen_t8&&pen_t12)||
8   (pen_t1&&pen_t5&&pen_t7&& pen_t12)
9   ...
10  ||(pen_t1&&pen_t7&&((!pen_t6)||(!pen_t5)))
11 do
12 ...
13 :: pen_t1 && pen_t4 && pen_t10 && pen_t12 ->
14  atomic{fire_t1; fire_t4; fire_t10; fire_t12;}
15 :: pen_t4 && pen_t10 && pen_t12 ->
16  atomic{fire_t4; fire_t10; fire_t12; }
17 :: pen_t1 && pen_t4 && pen_t10 && pen_t13 ->
18  atomic{fire_t1; fire_t4; fire_t10; fire_t13;}
19 :: pen_t4 && pen_t10 && pen_t13 ->
20  atomic{fire_t4; fire_t10; fire_t13; }
21 :: pen_t1 && pen_t4 && pen_t10 ->
22  atomic{fire_t1; fire_t4; fire_t10; }
23 :: pen_t4 && pen_t10 ->
24  atomic{fire_t4; fire_t10; }
25 ...
26 od
```

Figure 4. Fragment of the PROMELA model to reactor's SIPN

sitions t_4 and t_{10} and the corresponding *p–enabled* and fire conditions. We used the PROMELA model in the graphical interface to Spin, called Xspin [8].

We can simulate the behaviour of the reactor. Using the interactive mode of simulation we can select a given line of the *do-loop* we want to execute, that corresponds to the subset of transitions we want to fire simultaneously. Implicitly we are selecting a valuation to the variables.

To verify some properties, we used the TL editor of the Xspin.

Next we present how we verified the properties of the reactor. Structural conflicts among transitions happen when two or more transitions share a place in their presets, or postsets. In this SIPN we have five structural conflicts as follows:

1. place $p8$ is an output place of transitions t_4 and t_6;
2. place $p8$ is an input place of transitions t_5 and t_{11};
3. place $p2$ is an output place of transitions t_1 and t_7;
4. place $p3$ is an output place of transitions t_1 and t_8;
5. place $p6$ is an output place of transitions t_1 and t_{13}.

In the PROMELA model, the enabled conditions for the sets of transitions in structural conflict are defined with a `#define` statement. To verify if those structural conflicts give rise to undesired behavioural situations, we must check if the transitions in conflict are never enabled at the same time. For example, the conflict between t_1 and t_7 can be specified by the temporal formula $\Box\neg enabled\_t1\_t7$. Part of the definition of $enabled\_t1\_t7$ is presented in figure 4, in lines 6 to 10. The verification result of this formula is *valid*, thus the transitions never fire simultaneously.

Another important property to verify for the reactor SIPN is the liveness, which ensures that all the transitions are live. When trying to verify the liveness of transition t_4 (written in TL as $\Box\Diamond enabled\_t4$), we obtain the result *not valid*. Running the suggested guided simulation, we see that the SIPN has a cycle, because transitions t_5 and t_6 could fire infinitely. We could introduce fairness on the PROMELA model to permit the fair firing of transitions, but the SIPN meta-model does not take into account this possibility.

4.2. Second Version

To solve the problem detected in the first version of the SIPN, we must remove the cycles. The modified SIPN is presented in figure 5.

The properties successfully verified for the previous SIPN are still valid in this one. About the liveness properties, we studied the following:

- the transition t_4 is $L_1 - Live$: $\Diamond enabled\_t4$;
- the transition t_4 is $L_4 - Live$: $\Box\Diamond enabled\_t4$;
- the transitions t_4 and t_{10} are both $L_4 - Live$: $\Box(\Diamond enabled\_t4) \wedge \Box(\Diamond enabled\_t10)$.

All these formulas have the *valid* result for the verification.

While the stirrer (Mix) in the tank R is turning, the two tanks C1 and C2 do not open their gates. This property can be specified in TL by the formula: $\Box(p7 \rightarrow \neg\, enabled\_t4)$. The result of verifying this formula is *valid*.

5. Conclusions

In this paper, a model checking approach using PROMELA and the Spin tool is presented. This approach allows important properties of embedded system, 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.

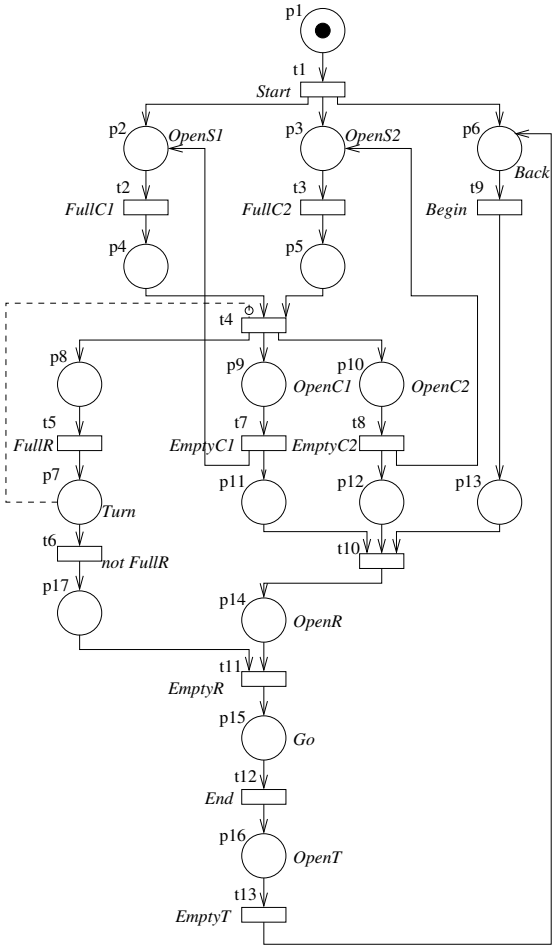

Figure 5. A modified SIPN (without cycles) for describing the dynamic behaviour of the reactor

In the proposed approach, the behavior of the embedded systems are modelled with a variant of Petri Nets, called SIPN. In relation to traditional PNs, SIPNs 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 SIPN models with PROMELA is not trivial, because the associated Spin tool assumes the existence of a finite-state system where only one transition fires at each instant. Therefore, the paper discusses in some detail how SIPN models should be specified with the PROMELA language, so that it is possible to model check them.

The approach is exemplified with a case study, the reactor. The example shows that with the Spin tool, the safety and liveness properties of a system can be verified, before that system is implemented. The paper also presents how the properties to be verified are expressed with TL formulas.

References

[1] M. Adamski. Direct Implementation of Petri Net Specification. In *7th Int. Conf. on Control Systems and Computer Science*, pages 74–85, 1987.

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

[3] L. A. Cortés, P. Eles, and Z. Peng. Verification of embedded systems using a petri net based representation. In *13th Int. Symp. on System Synthesis*, pages 149–55. IEEE CS Press, September 2000.

[4] S. Edwards, L. Lavagno, E. A. Lee, and A. Sangiovanni-Vincentelli. Design of Embedded Systems: Formal Models, Validation, and Synthesis. *Proceedings of the IEEE*, 85(3):366–90, 1997.

[5] J. M. Fernandes, M. Adamski, and A. J. Proença. VHDL Generation from Hierarchical Petri Net Specifications of Parallel Controller. *IEE Proceedings: Computers and Digital Techniques*, 144(2):127–37, Mar. 1997.

[6] G. J. Holzmann. *Design and Validation of Computer Protocols*. Prentice Hall, New Jersey, 1991.

[7] G. J. Holzmann. The Model Checker Spin. *IEEE Trans. on Software Engineering*, 23(5):279–95, May 1997.

[8] G. J. Holzmann. *The Spin Model Checker: Primier and Reference Manual*. Addison-Wesley, September 2003.

[9] R. Janicki and M. Koutny. On Causality Semantics of Nets with Priorities. *Fundamenta Informaticae*, 38(3):223–55, 1999.

[10] S. P. Jones. *Haskell 98 Language and Libraries*. Cambridge University Press, Apr. 2003.

[11] J. B. Jørgensen and S. Christensen. Executable Design Models for a Pervasive Healthcare Middleware System. In *5th Int. Conf. on the Unified Modeling Language (UML 2002)*, volume 2460 of *LNCS*, pages 140–9. Springer, Oct. 2002.

[12] J. Kleijn and M. Koutny. Process Semantics of P/T-Nets with Inhibitor Arcs. In *21st Int. Conf. on Application and Theory of Petri Nets (ICATPN 2000)*, volume 1825 of *LNCS*, pages 261–81. Springer-Verlag, June 2000.

[13] R. J. Machado and J. M. Fernandes. A Petri Net Meta-Model to Develop Software Components for Embedded Systems. In *2nd IEEE Int. Conf. on Application of Concurrency to System Design (ACSD'01)*, pages 18–22. IEEE CS Press, June 2001.

[14] R. J. Machado, J. M. Fernandes, and A. J. Proença. Specification of Industrial Digital Controllers with Object-Oriented Petri Nets. In *IEEE Int. Symp. on Industrial Electronics (ISIE '97)*, volume 1, pages 78–83, July 1997.

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

[16] G. D. Micheli. *Synthesis and Optimization of Digital Circuits.* McGraw-Hill, Inc., 1994.

[17] U. Montanari and F. Rossi. Contextual Nets. *Acta Informatica*, 32(6):545–96, 1995.

[18] M. Mukund. Petri Nets and Step Transition Systems. *International Journal of Foundations of Computer Science*, 3(4):443–78, 1992.

[19] T. Murata. Petri Nets: Properties, Analysis and Applications. In *Procedings of the IEEE*, pages 541–80, April 1989.

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

[21] T. C. Ruys. *Towards Effective Model Checking.* PhD thesis, University of Twente, Department of Computer Science, 2001.

[22] M. Sgroi, L. Lavagno, Y. Watanabe, and A. Sangiovanni-Vincentelli. Synthesis of Embedded Software using Free-Choice Petri Nets. In *36th ACM/IEEE Design Automation Conference (DAC'99)*, pages 805–10. ACM Press, 1999.

[23] W. Vogler. Partial Order Semantics and Read Arcs. In *Mathematical Foundations of Computer Science*, pages 508–17, 1997.