Supporting the changeability of SIPN-based logic control algorithms by verification and validation

Stéphane Klein, Georg Frey, Jean-Jacques Lesage, Lothar Litz

SUPPORTING THE CHANGEABILITY OF SIPN-BASED LOGIC CONTROL ALGORITHMS BY VERIFICATION AND VALIDATION

Stéphane Klein[1,3], **Georg Frey**[2], **Jean-Jacques Lesage**[3] **and Lothar Litz**[1]

[1] Kaiserslautern University of Technology, Institute of Automatic Control, PO Box 3049,
D-67653 Kaiserslautern, Germany
e-mail: {sklein, litz}@eit.uni-kl.de

[2] Kaiserslautern University of Technology, Juniorprofessorship for Agentbased Automation, PO Box 3049,
D-67653 Kaiserslautern, Germany
e-mail: frey@eit.uni-kl.de

[3]Universitary Laboratory in Automated Production Research (LURPA), Ecole Normale Supérieure de Cachan
61, avenue du Président Wilson, F-94235 Cachan Cedex, France
e-mail: {klein, lesage}@lurpa.ens-cachan.fr

ABSTRACT

In this paper the advantages of verification and validation to support changes of an existing PLC program are shown. The controller is defined using Signal Interpreted Petri Nets (SIPN) and verification and validation are performed using symbolic model-checking. The main focus of this paper is to show the process and the benefits of verification and validation for the reliability of the control algorithm when specified changes are to make. This is clarified by the example of a heating tank controller throughout the text.

1 INTRODUCTION

In industrial applications, PLC programs are often modified to take into account changes of the plant or of the requirements. Using formal methods to develop control algorithms enhances their changeability. Formal methods like verification and validation (V&V), together with appropriate formal algorithm descriptions like SIPN prove the requirements to be fulfilled.

This paper points out how V&V can be used to modify successfully an existing control algorithm. Indeed it will be shown that modifying a correct algorithm often leads to formal and functional requirements no longer to be fulfilled. Using V&V, the errors can be found and solved before the control algorithm is implemented on the PLC.

This paper is organized as follows. In section 2, the SIPN concept is briefly presented and used to design a first control algorithm for a heating tank. Section 3 discusses verification, validation and the model-checking technique. They are applied on the former designed SIPN in section 4. In section 5 the informal specification is extended and the algorithm is redesigned. It is shown how to develop the new algorithm step by step and how to ensure the quality of the resulting algorithm by verification and validation. Finally, section 6 concludes the paper.

2 SIGNAL INTERPRETED PETRI NETS

2.1 Definition

Signal Interpreted Petri Nets (SIPN) are an extension of Condition/Event Petri nets and allow the handling of binary I/O-signals in a well-defined way. They are well suited to design control algorithms for discrete event systems resulting in languages standardized in IEC 61131-3. SIPNs are defined as a 9-tupel

$$SIPN=(P,T,F,M_0,I,O,C,A,\Omega)$$

with the sets P of places, T of transitions, F of arcs and an initial binary marking M_0 like in ordinary C/E Petri net. To become SIPN, the extensions are:

- Transitions are associated to a set C of firing conditions as Boolean functions of the input variables I.
- Places are associated with a mapping A of actions defining a subset of output signals O. Marked places define a subset of output signals
- A Signal Algebra Ω defines how to handle the output subsets of all binary marked places.

For a more formal definition of SIPN, see [1].

2.2 Dynamic behavior

The dynamic Behavior of an SIPN is given by the firing process defined by four rules, see also Figure 1:

1. A transition is enabled, if all its pre-places are marked and firing ensures binary marking of all its post-places.
2. A transition fires immediately, if it is enabled and its firing condition is fulfilled.
3. All transitions that can fire and are not in conflict with other transitions fire simultaneously.
4. The firing process is iterated until a stable marking is reached (i.e. until no transition can fire anymore). Since firing of a transition is supposed to take no time, iterative firing is interpreted as simultaneous, too. For that reason, no changes of input signals may occur during the firing process.

After a new stable marking is reached, the output signals are computed according to the marking and the signal algebra.

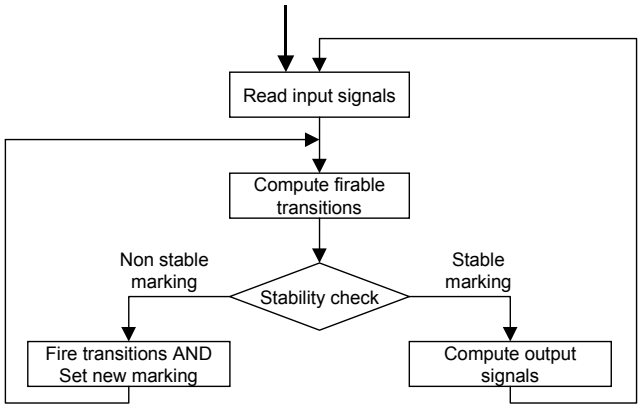

Figure 1. Evolution algorithm of an SIPN.

2.3 Example

As an example for controller design with SIPN, a heating tank is used. A description of the tank is given in Figure 2 and the signals are defined in Table 1 and Table 2.

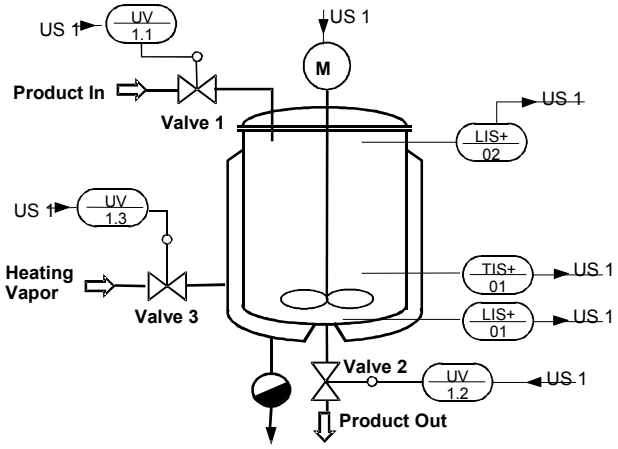

Figure 2. Description of the heating tank

Table 1. Table of input signals

Name	Description
i1	Lower level sensor
i2	Upper level sensor
i3	Temperature sensor
i4	Start button

Table 2. Table of output signals

Name	Description
o1	Valve 1 (filling)
o2	Valve 2 (emptying)
o3	Stirring motor
o4	Valve 3 (Heating)

The informal specification is the description of the client's requirements in natural language. These are:

After pressing the start button the tank is filled. The full tank is heated until the desired temperature is reached. Then the heated tank is emptied. During the whole process the contents are stirred.

2.4 Design of the control algorithm

Based on the informal specification, the control algorithm is designed using SIPN. This algorithm is given in Figure 3.

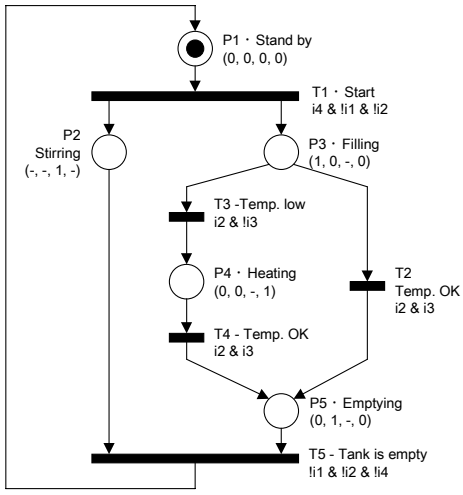

Figure 3. Control algorithm of the heating tank (Version 1)

Starting form the initial place P1, transition T1 can fire if the tank is empty (!i1), not full (!i2) and the start button (i4) is pressed. After this transition has fired, the stirring motor is set ON (o3 = 1) in place P2 and the tank is being filled (o1 = 1) in place P3. Once the tank is full and the temperature of its contents is beyond the desired limit (i3 = 1), transition T2 fires and the tank is emptied (o2 = 1) in place P5. If the temperature of the full tank is below the limit, the contents are heated (o4 = 1) in place P4 before the tank is emptied in P5. When the tank is empty, transition T5 fires and the initial state where al the actions are set to 0 is reached again.

3 Verification and Validation

3.1 Definition

Verification and Validation are often confused. Therefore, let us remind the definitions given by Boehm [2]:

"Verification: am I building the product right ?"

"Validation: am I building the right product ?"

Hence, according to [3], the verification is the proof that the internal semantics of a model is correct without regard to the modeled system whereas the validation determines if the model agrees with the designer's intention.

In [1], four properties for the **formal correctness** of an algorithm are defined. Besides completeness they ensure the deterministic behavior of the control algorithm, which can be defined in an informal way as follows:

Every control algorithm must have deterministically defined dynamics and I/O-behavior. If it had not, its behavior in a given situation would depend on implementation aspects. In detail, this means that in every state of the control-

ler the reaction on possible input signals is defined and a non-contradictory value for each output signal is specified.
Properties like these (verification part) are widely accepted as mandatory and can be found in many works on formal methods in control. They can be automatically formulated for every designed SIPN.
Properties to express that the designed SIPN fulfills the customer's needs are problem-dependent (validation part). Hence, there is no way to formulate them automatically. They must be generated anew for every problem.
Nevertheless, with symbolic model-checking [4] the same method can be used to perform V&V.

3.2 Model-checking

Model-Checking is a technique in which a finite state model of the system is built and the expected properties of the system are checked on this model [5]. The system is modeled as a finite state machine, whose evolution is given in an algorithm and the properties are expressed in a temporal logic. A search procedure (exhaustive state space search) is then used to check whether the expected properties hold on the finite state transition system or not. Figure 4 shows the tasks that have to be done within the model-checking procedure.
In this work, the symbolic model-checker SMV [6] is used to perform V&V.

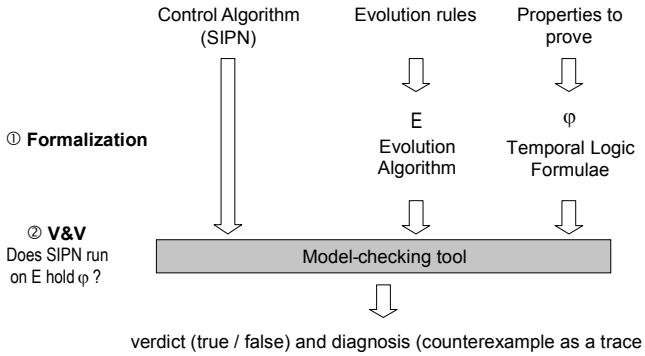

Figure 4. Model-checking process.

3.3 Temporal logic formulae

The properties that are to be checked have to be written in temporal logic (TL) [4]. This is a special logic tailored for statements and reasoning which involve the notion of order in time. In TL, the parameter t (time) is not explicitly present in the formulae. TL offers concepts immediately ready for use. Its operators mimic linguistic constructions (the adverbs "always", "until", the tenses of verbs, etc.) with the result that natural language and their TL formalization are fairly close. The SMV model-checker uses Computation Tree Logic (CTL), which is together with PLTL (Propositional Linear Temporal Logic) the most used TL in model-checking tools.
An application of V&V on SIPN using model-checking with CTL formulae is described in [7].

4 V&V Process by Example

4.1 Specification

To apply V&V on the controller designed in sub-section 2.4, formal properties have to be derived by formalization of all the informal specifications given in sub-sections 2.3 and 3.1. The specification in 2.3 was also used to formalize the control algorithm by SIPN. It may seem to be redundant having two different formalization steps. But two formalizations performed by different methods (SIPN and TL) considerably enhance the reliability. For it is very improbable to make the same mistake twice following the completely different attempts of SIPN and TL.
To be used in V&V with model-checking, the SIPN has to be translated into SMV input code. This translation considers the structure and the evolution of the SIPN, as well. To reproduce the evolution algorithm of an SIPN (Figure 1), a variable eoc (*End of Cycle*) [8] is introduced. It is set to 1 as soon as a stable marking is reached. No more transition can fire in that case without changing of the input values. Hence this variable when remaining false all the time detects infinite loops in the control algorithm, called unstable marking. During V&V eoc is used to check properties involving output signals since these are only set in stable markings. For more details see [9].

4.2 Verification

Verification has to make sure hat the designed SIPN behaves in a deterministic way yielding the properties:

1. The algorithm always produces stable markings whatever happens in the future. In TL, this is
   `AG EF eoc`
2. The output signals are correctly defined, meaning that in every stable marking an output signal oi is set either to 0 or to 1 (XOR).
   `AG ! (eoc & ((set_oi & reset_oi) | (!set_oi & !reset_oi)))`
3. There are no transition conflicts. When there is a choice between several ways to mark or unmark a place, only one of them must be possible:
   `AG ! (T2 & T3)`

These properties ensure that the designed SIPN has a deterministic behavior. Some further properties about the behavior of the SIPN can be added. These are:

4. The algorithm is reversible, i.e. the initial marking can always be reached again:
   `AG EF m0`
5. The SIPN is live, i.e. every part of the controller can be executed again and again:
   `AG ((EF T1) | (EF T2) | (EF T3) | (EF T4) | (EF T5))`
6. The algorithm is deadlock free. This means that the algorithm never gets stuck without any way to evolve:
   `AG EF (T1 | T2 | T3 | T4 | T5)`

Using SMV to perform verification, the six properties were all verified on the given SIPN.

4.3 Validation

The validation has to check whether the designed control algorithm fulfills the functional specifications given in 2.3 in an *informal* way. Formalization means translating them into temporal logic formulae. Let us consider two properties:

Property 1: The contents of the tank remain stirred during the whole process.

First "during the whole process" has to be defined. In this case, it means during the phases where the tank is filled, heated and emptied. Therefore, we can express this property as follows: When the tank is being filled, heated or emptied, the stirring motor has to be turned on. In TL, the following formula results:

AG ((o1 | o2 | o4) -> o3)

Property 2: When the start button is pressed in the initial state the tank must be filled.

Here the "initial state" has to be interpreted in a more technical sense. The tank has also to be empty at the beginning of the cycle. Furthermore, outputs are only computed after a stable marking has been reached. Hence the property can only be checked on stable markings. This results in:

AG (m0 & !i1 & !i2 & i4 & eoc -> AX o1)

After the translation of the SIPN, the validation is performed using SMV and does not show any errors. As the SIPN has been verified and validated, it can now be implemented on a PLC.

5 CHANGEABILITY AND V&V

5.1 General problem

Any changes of the SIPN, e.g. adding places and transitions modify its behavior. Therefore it has to be ensured anew that the modified SIPN is still formally correct and that it fulfills the new set of requirements. Normally, this results in cycles of SIPN changes with a new verification and validation steps until the results all become true.

5.2 Extended specification

To discuss the changeability, the specification given so far is changed by adding hardware and a new requirement. A Stop button (i5) is implemented and the informal specification is extended as follows:

After pressing the stop button, the chemical process has to be stopped. To start again, the start button has to be pressed. The tank is then emptied before a new cycle can begin.

5.3 SIPN Redesign with V&V

SIPN Version 2

To take this new requirement into account, the original SIPN of section 2.4 is modified, see Figure 5. A place has been inserted in which the filling, the heating and the emptying can be stopped. Thus the chemical process is supposed to be stopped.

Before the functional requirements are checked, the SIPN must be declared as formally correct. Like the former designed SIPN, this one has to fulfill the standard expectations formulated in section 4.1. Performing verification on this SIPN does not reveal any errors.

After extending the expectations, new properties have been added. That means that the redesigned control algorithm has to fulfill the properties given in section 4.3 as well as the following one:

Property 3: When the stop button is pressed then the process is stopped. That means that in all the stable markings reached after pressing the stop button, the actuators must be set to 0. In TL it can expressed as:

AG (i5 & eoc -> AX (!o1 & !o2 & !o3 & !o4)

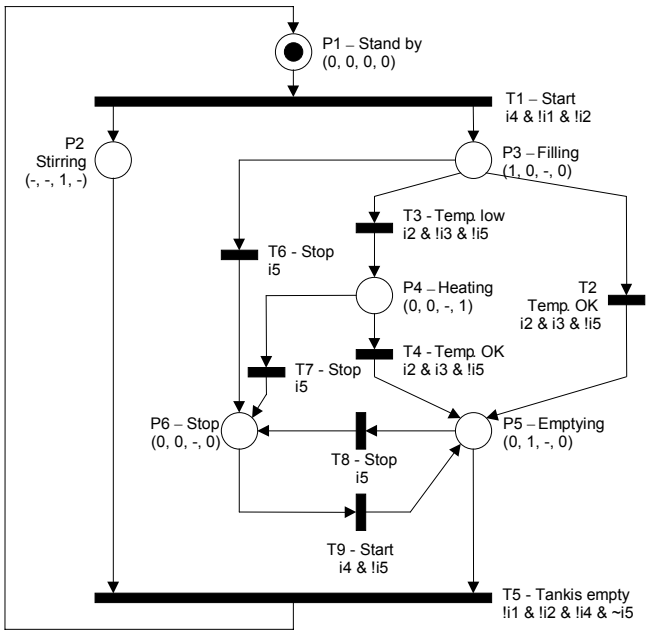

Figure 5. SIPN with extended specification (Version 2)

Performing validation with the newly defined algorithm tells us that the old properties are fulfilled but that the new one is not. Table 3 shows a sequence leading to a state where the property does not hold.

In state 4, i5 and eoc hold but in the next stable state (state 5), o3 still has value 1. When stop is pressed while place P3 is active, the filling is stopped but the stirring still remains. Indeed there is no stop transition leading from P2 to a place where o3 is reset. In this interpretation the chemical process is supposed to be stopped when filling, heating and emptying are stopped. During the formalization of the corresponding TL formula, *The process has to be stopped* has been interpreted differently, "all actions are stopped".

SIPN Version 3

After redefining the requirement more precisely, the second way (all actions stopped) is chosen. Hence the SIPN has to be modified again. In Figure 6, a transition T10 leading from P2 to P6 has been added.

Before looking at functional properties the formal ones have to be checked again. As a result of verification, SMV shows a conflict between transitions T6 and T10 as illustrated in the counter-example given in Table 4.

Table 3. Counter-example for property 3

State	1	2	3	4	5
P1	1	1	0	0	0
P2	0	0	1	1	1
P3	0	0	1	1	0
P4	0	0	0	0	0
P5	0	0	0	0	0
P6	0	0	0	0	1
o1	0	0	1	0	0
o2	0	0	0	0	0
o3	0	0	1	1	1
o4	0	0	0	0	0
i1	0	0	0	0	0
i2	0	0	0	0	0
i3	0	0	0	0	0
i4	0	1	1	1	0
i5	0	1	1	1	0
T1	0	1	0	0	0
T2	0	0	0	0	0
T3	0	0	0	0	0
T4	0	0	0	0	0
T5	0	0	0	0	0
T6	0	0	1	0	0
T7	0	0	0	0	0
T8	0	0	0	0	0
T9	0	0	0	0	0
eoc	0	0	0	1	1

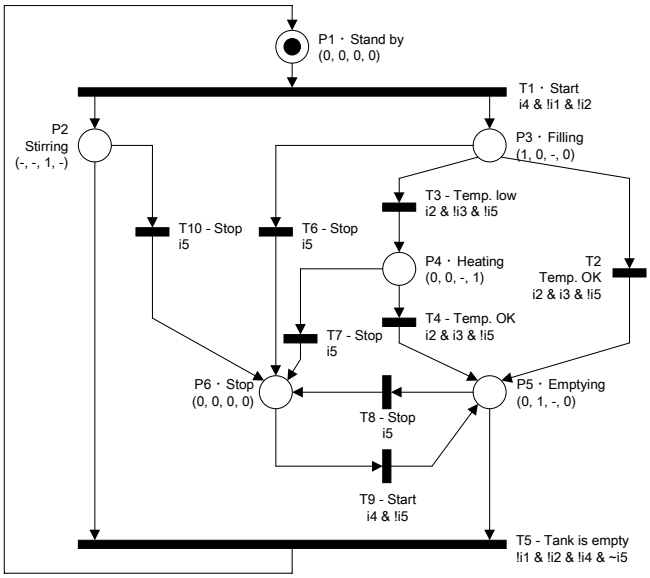

Figure 6. Modified SIPN (Version 3)

When P2 and P3 are marked and the stop button is pressed, we can not decide whether T6 or T10 fires. The designed algorithm is not deterministic. This clearly is a design error.

Table 4. Counter-example for the conflict between T6 and T10

State	1	2	3
P1	1	1	0
P2	0	0	1
P3	0	0	1
P4	0	0	0
P5	0	0	0
P6	0	0	0
i1	0	0	0
i2	0	0	0
i3	0	0	0
i4	0	1	1
i5	0	1	1
T1	0	1	0
T2	0	0	0
T3	0	0	0
T4	0	0	0
T5	0	0	0
T6	0	0	1
T7	0	0	0
T8	0	0	0
T9	0	0	0
T10	0	0	1
eoc	1	0	0

SIPN Version 4

As a solution, a new place P7 where the stirring motor is set to 0 is added, see Figure 7. Again this SIPN has to be verified before we can perform validation. Now, SMV tells us that the SIPN has a deadlock.

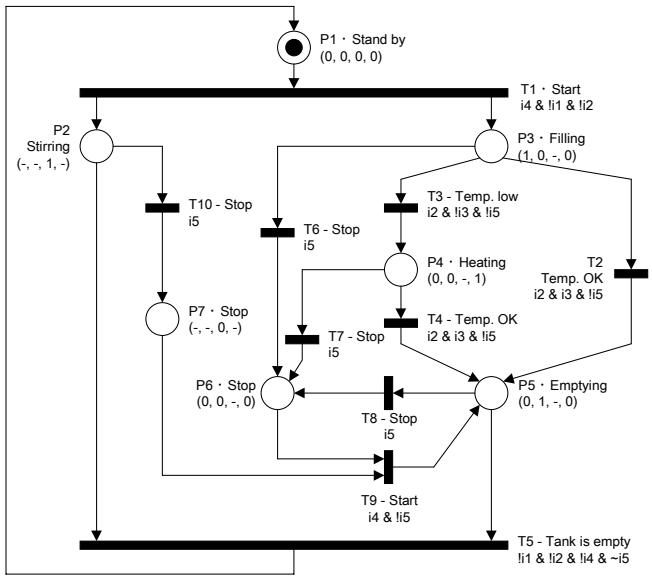

Figure 7. Modified SIPN (Version 4)

Examining the counter-example in Table 5, we see that after stop button has been pressed and the cycle started again, place P5 is marked and the tank is emptied. After P5 has been reached (state 6), only one transition (T8) is enabled. When this transition has fired, only P6 is marked. Since P7 is no more marked, transition T10 is not enabled and the algorithm gets stuck.

Table 5. Counter-example for the deadlock

State	1	2	3	4	5	6	7	8
P1	1	1	0	0	0	0	0	0
P2	0	0	1	0	0	0	0	0
P3	0	0	1	0	0	0	0	0
P4	0	0	0	0	0	0	0	0
P5	0	0	0	0	0	1	1	0
P6	0	0	0	1	1	0	0	1
P7	0	0	0	1	1	0	0	0
i1	0	0	0	0	0	0	0	0
i2	0	0	0	0	0	0	0	0
i3	0	0	0	0	0	0	0	0
i4	0	1	1	1	1	1	0	0
i5	0	1	1	1	0	0	1	1
T1	0	1	0	0	0	0	0	0
T2	0	0	0	0	0	0	0	0
T3	0	0	0	0	0	0	0	0
T4	0	0	0	0	0	0	0	0
T5	0	0	0	0	0	0	0	0
T6	0	0	1	0	0	0	0	0
T7	0	0	0	0	0	0	0	0
T8	0	0	0	0	0	0	1	0
T9	0	0	0	0	1	0	0	0
T10	0	0	1	0	0	0	0	0
eoc	1	0	0	1	0	1	0	1

SIPN Version 5

Analyzing the SIPN, a second way to quit P5 would be firing T5. Unfortunately T5 can not be enabled if place P2 is not marked. Thus starting the cycle again should produce a token in P2. This is done by an arc from T9 to P2 in the new algorithm, see Figure 8.

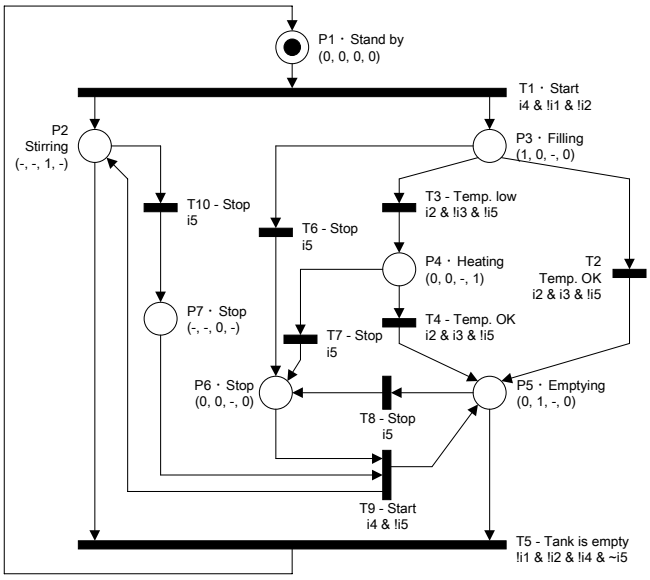

Figure 8. New design of the SIPN (Version 5)

Performing V&V on Version 5 does show no more errors. Thus this SIPN will be implemented on the PLC.

6 CONCLUSION

In this paper the heating tank example is used to show the advantages of formal verification and validation for a control algorithm after changing the informal specification. Starting from a verified and validated SIPN, the initial specification is extended by some hardware and the corresponding behavioral specification. Often, these changes seem to be quite easy. Nevertheless, before the modified algorithm is implemented on a PLC, several cycles of algorithm redesign with verification and validation are necessary.

The chosen example emphasizes the fact that each modification of a formerly correct SIPN may involve errors. An error may cause either formal and functional requirements or one of them not to be fulfilled. Before the functional properties of an algorithm are checked, it should be made sure that it is formally correct, i.e. it behaves in a deterministic way. Verification and validation of the modified SIPN are pursued until the model checker proves all the properties to be true. An important aspect of the formal method described here is related to the redundancy in the two formal specifications of SIPN and Temporal Logic. This redundancy can reveal different kinds of errors, e.g. those due to different interpretations of the requirements or due to a contradictory or incomplete informal specification. As a conclusion, there are no “small” changes in the redesign of an algorithm.

REFERENCES

[1] G. Frey; Design and formal Analysis of Petri Net based Logic Control Algorithms. (ISBN 3-8322-0043-6), Dissertation, University of Kaiserslautern, Shaker Verlag, Aachen, April 2002.

[2] B. W. Boehm; "Guidelines for Verifying and Validating Software Requirements and Design Specifications", P.A. Samet (Editor), Proc. of the EURO IFIP 79, North-Holland Publishing Company, 1979.

[3] J.-M. Roussel, J.-J. Lesage; "Validation and Verification of grafcets using finite state machine", Proc. of IMACS-IEEE "CESA'96", pp. 758-764, Lille (France), 9-12 July 1996.

[4] B. Bérard, M. Bidiot, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci and Ph. Schnoebelen; "Systems and Software Verification, Model-Checking Techniques and Tools", Springer, Berlin, New-York, 2001.

[5] S. Lampérière-Couffin, O. Rossi, J.-M. Roussel, J.-J. Lesage; "Formal Validation of PLC programs: a Survey", Proc. of ECC'99, EUCA (European Union Control Association) - IFAC - IEEE Control Systems Society, paper n°741, Karlsruhe (Germany), 31. August - 3. September 1999.

[6] K. L. Mc Millan; "The SMV system for SMV version 2.5.4". School for Computer Science, Canergie Melon University, Nov. 2000.

[7] X. Weng, L. Litz,: Model checking: towards generating a correct specification for logic controllers. Proc. of the American Control Conference 2002 (ACC2002), Anchorage (USA), pp. 4457-4462, May 8-10, 2002.

[8] G. Canet, S. Couffin, J.-J. Lesage, A. Petit, Ph. Schnoebelen; "Towards the automatic verification of PLC programs written in Instruction List", Proc. of the IEEE Conference on Systems Man and Cybernetics SMC 2000, pp. 2449-2454, Nashville, Tennessee (USA), Oct. 8-11, 2000.

[9] S. Klein, G. Frey and L. Litz; "Designing fault-tolerant controllers using SIPN and model-checking". To be published in the proc. of the 5th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes (SAFEPROCESS 2003), Washington D.C. (USA), June 9-11, 2003.