CONTROLLER DESIGN FOR AN FMS USING SIGNAL INTERPRETED PETRI NETS AND SFC

Validation of both Descriptions via Model-Checking

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

[1]University of Kaiserslautern, Institute of Automatic Control, PO Box 3049, D-67653 Kaiserslautern, Germany
e-mail: {sklein, weng, frey, litz}@eit.uni-kl.de, Phone.: +49 631 205 4310, Fax: +49 631 205 4462

[2]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

The aim of this paper is to study the validation of both the behavioral model and the implementation program of an automated system. To clarify our approach, we use a small production line whose control algorithm is built using Signal Interpreted Petri Nets, validated using Model-checking techniques and translated into a Sequential Function Charts (SFC) implementation program. This program is then validated anew. Using this example, we show the utility of those two validation steps before the final implementation of the program on a Programmable Logic Controller (PLC).

1 INTRODUCTION

In this paper, a flexible manufacturing system is used to motivate validation of both the behavioral model and the implementation program of a controller. To achieve that, a behavioral model of the controller is designed using Signal Interpreted Petri Nets (SIPN) and validated using Model-Checking. Once the behavioral model is correct, it is translated into an implementation program written in Sequential Function Charts (SFC). This implementation program is validated anew in order to assert that the properties of the controller have been preserved during the translation.

2 WORKING EXAMPLE

2.1 Presentation and requirements

We use the example of a flexible manufacturing system (Figure 1) provided as a benchmark available under [1]. This machining line consists of three stations: a drilling machine, a vertical milling machine (with tool changer), and a horizontal milling machine. Each station has an independent conveyor.

The module contains 15 input signals (11 limit switches and 4 infrared sensors) and 15 output signals (4 reversible motors and 7 non-reversible motors). The input switches and sensors are normally closed contacts.

To design the control algorithm of the manufacturing line, we study the "complex operations" scenario [2] given as a timing chart. Using this scenario, the three machines of the line are working.

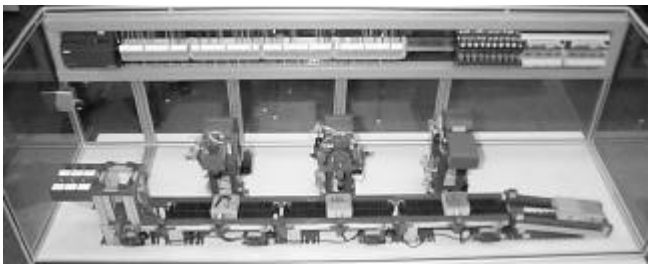

Figure 1. Flexible Manufacturing System.

2.2 Expected behavioral properties

The example was delivered without the properties the controller has to fulfil. Hence properties have been deduced from the timing chart given in [2] using the method developed in [3]. This method allows to find properties by examining under which input conditions an output is set to 0 or set to 1. An easy way to use this method is to collect the variables in a table as shown in Table 1.

	Vertical mill at lower position (I05 = 0)	Vertical mill at upper position (I06 = 0)	Tool in position (I07 = 0)	Part at milling position (I11 = 0)
Vertical Mill Conveyor ON (O04 = 1)				0
Vertical Mill Moving UP (O05 = 1)	0		1	
Vertical Mill Moving DOWN (O06 = 1)		0	1	1
Vertical Mill Motor ON (O07 = 1)			1	1
Tool charger rotates (O10 = 1)	1	0	0	1

Table 1. I/O properties.

For example, the following property can be written:

Prop1: the mill moves up (O*05*=1) if it is not yet at upper position (I*05*=1) and if there is a tool in position (I*07*=0).

The method has been extended in order to write properties that concern only output signals. This extension consists of

Klein, S., Weng, X., Frey, G., Lesage, J.-J. and Litz, L.: Controller Design for an FMS using Signal Interpreted Petri Nets and SFC. Proceedings of the American Control Conference 2002 (ACC2002), Anchorage, Alaska, pp. 4141-4146, May 2002.

scanning which output signals can not be simultaneously true or false. In Table 2, "X" stands for incompatible actions and "N" for needed outputs.

	O04 = 1	O05 = 1	O06 = 1	O07 = 1	O10 = 1
Vertical Mill Conveyor ON (O04 = 1)		x	x		
Vertical Mill Moving UP (O05 = 1)	x		**X**		x
Vertical Mill Moving DOWN (O06 = 1)	x	**X**		**N**	x
Vertical Mill Motor ON (O07 = 1)					x
Tool changer rotates (O10 = 1)		x	x	x	

Table 2. Output properties.

As an example, the following properties can be extracted:
Prop2: the mill can never been moved up (O*05*=1) and down (O*06*=1) simultaneously.
Prop3: the milling motor must always be on (O*07*=1) while the mill is moved down (O*06*=1).
This example shows that the table is symmetric for exclusive outputs (X) but not for needed ones. Taking into account the example, this can be explained by the fact that the milling motor must be on while the mill moves down, whereas moving the mill down is not necessary to turn on the milling motor.

3 Design using Signal Interpreted Petri Nets

3.1 Formal Definition

In this work, timed Signal Interpreted Petri Nets (tSIPN) are used to design the specification model. In [4], a tSIPN is define as a 10-tuple SIPN = ($\boldsymbol{P}$, $\boldsymbol{T}$, $\boldsymbol{F}$, $\mathbf{m}_0$, $\boldsymbol{I}$, $\boldsymbol{O}$, ö, ù, Ù, τ) with:

($\boldsymbol{P}$, $\boldsymbol{T}$, $\boldsymbol{F}$, $\mathbf{m}_0$) an ordinary PN with places $\boldsymbol{P}$, transitions $\boldsymbol{T}$, arcs $\boldsymbol{F}$, and binary initial marking $\mathbf{m}_0$, with $|\boldsymbol{P}|,|\boldsymbol{T}|,|\boldsymbol{F}|>0$,

$\boldsymbol{I}$ a set of logical input signals with $|\boldsymbol{I}| > 0$

$\boldsymbol{O}$ a set of logical output signals with $\boldsymbol{I} \cap \boldsymbol{O} = \varnothing$, $|\boldsymbol{O}| > 0$

ö a mapping associating every transition $t_i \in \boldsymbol{T}$ with a firing condition ö(t_i) = Boolean function in $\boldsymbol{I}$

ù a mapping associating every place $p_i \in \boldsymbol{P}$ with an output ù(p_i) $\in (0, 1, -)^{|O|}$, where (-) means 'don't care'.

Ù the output function combines the output ù of all marked places Ù: $\mathbf{m} \rightarrow (0, 1, -, c)^{|O|}$. The combined output can be zero (0), one (1), undefined (-) or contradictory (c).

τ a mapping associating every arc f_i that is an input arc to a transition $f_i \in (\boldsymbol{P} \times \boldsymbol{T}) \cap \boldsymbol{F}$ with a time delay $t_i \in \mathbf{R}_0^+$. With $f_i = (p_j, t_k)$, τ_i is the time that a token has at least to stay in p_j before it can be removed by the firing of t_k.

The dynamic Behavior of a tSIPN is given by the firing process defined by four rules:

1. A transition is enabled, if all its pre-places are marked, in the case of timed arcs marked for at least the specified delay time τ, and all its post-places are unmarked (safe enabling rule).
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. This also means the hypothesis that a change of input signal values can not occur during the firing process. That is verified with reactive systems.

After a new stable marking is reached, the output signals are recalculated by applying Ù to the marking.
The firing process can be represented by a flow chart like the one presented in Figure 2.

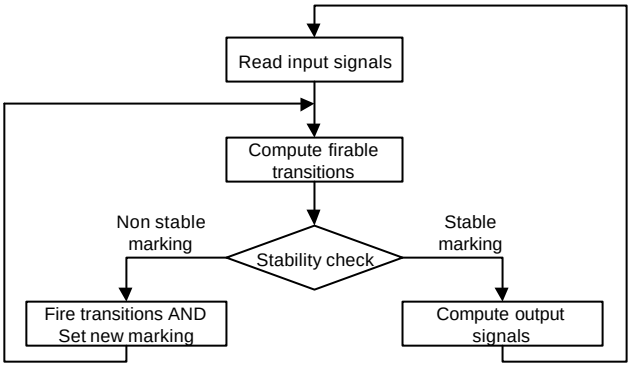

Figure 2. Evolution algorithm of an SIPN.

3.2 Hierarchical SIPN

Nevertheless, for real-world programs, SIPN controllers tend to be large and difficult to handle like for most visual languages. In most cases, certain subnets can be identified which realize certain subtasks and are only active in specific situations. The complete SIPN can, therefore, be replaced by an abstract SIPN where single places are used instead of these subnets. The subnets are subordinated to these hierarchical places and can in turn contain hierarchical places and so on. Hence a hierarchical SIPN so called SIPN^H results. Hierarchy does not only enhance the readability of SIPNs. It is also a valuable means for top-down design of SIPN controllers: Instead of trying to create a single, large net, a programmer can start with an abstract SIPN which is then refined step by step.
The subnets in an SIPN^H contain an input and an output place. The input place is given a token at the same time as the corresponding hierarchical place is marked. The token from the hierarchical place can only be removed together with the token on the output place. With this construction and an additional passivity constraint, it is assured, that the subnet is only influencing the process while it is activated by the hierarchical place. For a formal definition see [4].
In the following section, a place having a subnet is drawn with a dashed line.

3.3 Design of the specification model

To design the specification model of the flexible manufacturing line, timed hierarchical SIPN are used. More precisely, the model stands out of three hierarchical levels with six subnets.

The highest level (Figure 3) is used to synchronize the three machines and their associated conveyors.

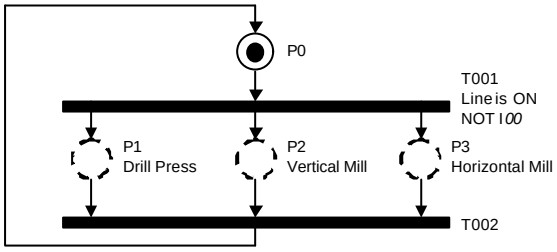

Figure 3. Level 1 of the SIPN specification model.

In the second level, the three machines are described in a quite similar same way. In the upper part, a new part is waited for and in the lower part, the part is machined. For example, the SIPN (Figure 4) is the model of the second level for the vertical milling machine.

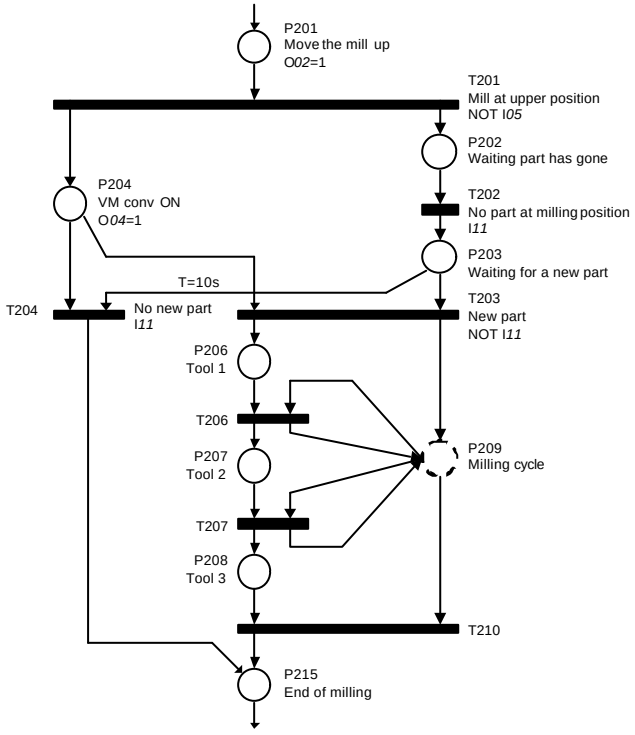

Figure 4. SIPN-level 2 of the vertical milling machine.

In the third and last level, the actions on each machine are described in a linear sequence. Figure 5 illustrates that for the vertical milling machine.

The complete behavioral model stands out of three hierarchical levels, 55 places and 40 transitions.

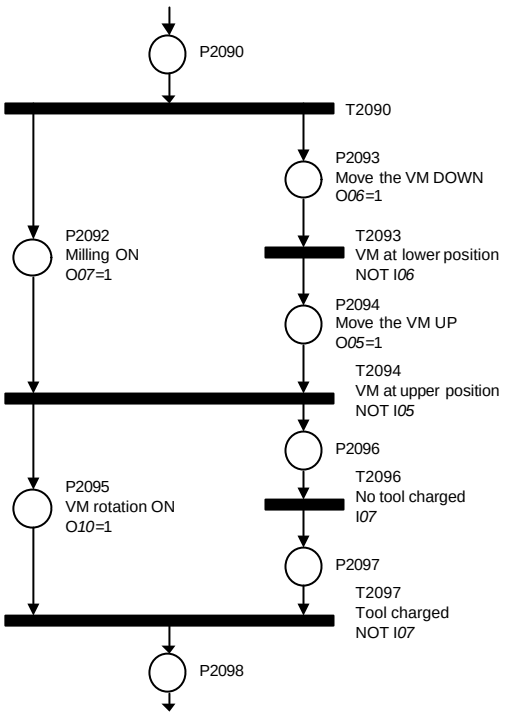

Figure 5. SIPN-level 3 of the vertical milling machine.

4 VALIDATION

4.1 Definition

Validation is often confused with verification. Therefore, let us remind the definitions given by Boehm in [5]:

"Verification: am I building the product right ?"

"Validation: am I building the right product ?"

Hence, according to [6], the verification is the proof that that the internal semantics of a model is correct independently of the modeled system whereas the validation determines if the model agrees with the designer's purpose.

In this contribution, we will focus on validation. Nevertheless, verification will be performed on each model *before* its validation.

The method used to perform validation is symbolic model-checking [7].

4.2 Model-checking

Model-Checking is a technique in which a finite state model of the system is built and the expected properties (specifications of the behavior) of the system are checked on this model [8]. The system is modeled as a finite state automaton, the evolution of this automaton 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 are verified on the finite state transition system or not. Figure 6 shows the model-checking procedure.

In symbolic model-checking, the state space of the finite state automaton is not explicitly built and Binary Decision Diagrams (BDD) are used to represent the system's states.

In this work, the symbolic model-checker Cadence SMV [9] is used to perform validation.

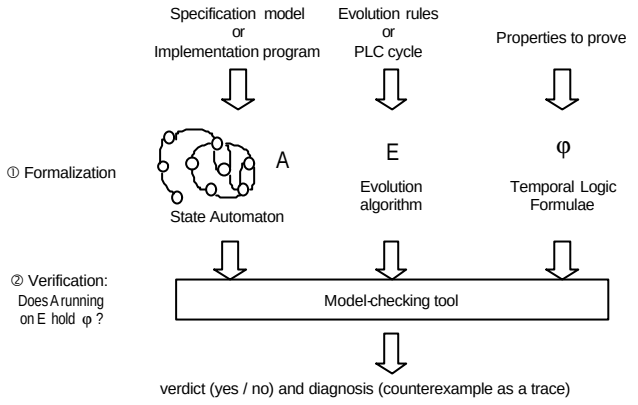

Figure 6. Model-checking process.

4.3 Temporal logic formulas

The assertions written in section 2 must be translated in a temporal logic [7] to be used in Cadence SMV. A variable *eoc* (end of cycle) [10] corresponding to the observable states of the system has been defined. This variable is defined using the evolution algorithm E (Figure 6). This algorithm is:

- The interpretation of the SIPN firing rules (Figure 2) and *eoc* represents: "A stable marking has been reached".
- The cyclic behavior of the Programmable Logic Controller (PLC) shown in Figure 10. In this case, *eoc* means: "The PLC cycle has terminated".

As an example of translation, the first two properties written in section 2 become:

Prop1: SPEC AG (eoc & (O*05*=1) → (I*05* & ~I*07*)) ;

That can be reinterpreted in: It is always true (AG) that the mill moves up (O*05*=1) if (→) it is not at upper position (I*05*) and if there is a tool in position (~I*07*).

Prop2 : SPEC AG ~(eoc & (O*05*=1) & (O*06*=1)) ;

Reinterpretation: It is always true (AG) that we do not (~) have simultaneously (&) the drill moving up (O*05*=1) and down (O*06*=1).

4.4 Validation of the specification model

4.4.1 Coding of the SIPN specification model

Before we can use the symbolic model checker Cadence SMV, we have to translate the structure and the dynamic behavior of the control algorithm (A and E in Figure 6). This coding is performed using the method presented in [11].

4.4.2 Validation

After have made sure that the designed SIPN fulfils the standard SIPN properties (verification), the validation of the problem specific properties has been performed.

The properties extracted from the analysis of the output variables have been validated at the first validation pass.

Properties involving input and output variables are mostly not supported by the model because of the non-model based approach [12]. In this approach, no assumptions about the real behavior of the plant are made. Hence every combination of input signals is possible. As an example, Figure 7 shows the result of the validation of Prop1.

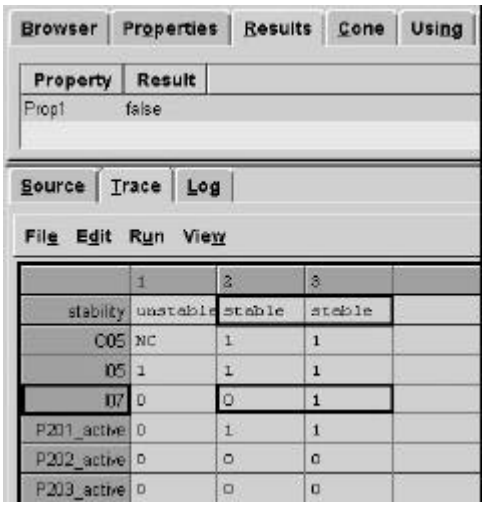

Figure 7. SMV diagnosis for the validation of Prop1 (SIPN).

We can notice that the sensor signal I*07* becomes 1 whereas the tool changer has not been moved. That is not a design failure but it is due to the non-model based approach. To make the model support the property, we formulate an assumption which is: When the sensor signal I*07* has value 0 (tool in position), it remains 0 until the tool changer rotates. This can be written in TL:

Tool: assert G ((I*07*=0) → (I*07*=0) U (O*07*=1)) ;

Using this assumption on the input variables, the property is supported by the model.

Formulating such assumptions, the approach is called constrained based. This approach must be handled carefully because every assumption made about the behavior of the plant has to be realistic and must not hide real behavioral possibilities. However, using this approach, the validation did not show other failures.

The designed formal model of the controller has been validated but an SIPN can not be implemented directly into a PLC. That means that a translation of the former correctly designed SIPN into an implementation language is necessary. This also means that a new formalism with other evolution rules will be used and that part of the information could get lost. In order to make sure that the properties remain unchanged during the translation, a new step of validation will be performed.

5 Implementation of the Control Plan

To implement the control plan, 3 of the five languages defined in the IEC 61131-3 standard [13] are used. After a brief presentation of the standard, the implementation program design is presented in more details before it is validated.

5.1 The IEC 61131-3 standard

The IEC 61131-3 standard defines the syntax and, for a lesser part, the semantics of four programming languages for PLC, as well as a structuring one (Sequential Function Chart).

The four programming languages are Ladder Diagram, Function Block Diagram (graphic programming languages), Structured Text and Instruction List (textual ones). The Ladder Diagram language (LD), based on relay ladder logic diagrams, permits the description of Boolean functions. In the Function Block Diagram language (FBD), the programming features are represented as graphic blocks. The Structured Text language (ST), close to Pascal, allows sequential, conditional and loop statements. The Instruction List (IL) is an assembly code-like language.

The fifth language is defined in order to structure the internal organization of PLC programs or function blocks. This graphic language called Sequential Function Chart (SFC) is inspired by Grafcet [14] and is an extended state machine that contains primitives to describe sequential, parallel and alternative behaviors. It enables the partitioning of a PLC program (or function block) into a set of steps and transitions interconnected by directed links. Each step is associated to a set of actions and each transition is associated to a transition condition.

Because of this step and transition definition, a quite easy translation from the SIPN specification model can be done into this programming language. However there are some major differences that have to be considered [15]:

- In SFC there is, by definition, no transient state. The activity of a step is always held up for at least one PLC cycle.
- In SFC post-steps of a transition are not checked in the firing rule. A step that is already activated can be activated again.
- There is a structural restriction in SFC allowing only one initial step.

Furthermore, the standard defines action qualifiers such as stored, restored, delayed, allowing the replacement of interpreted parallelism by qualified actions. To create the implementation program, following qualifiers have been used: N (Non-stored action), S (Set or Stored action) and R (Reset a former stored action).

5.2 Design of the implementation program

The implementation program is a translation of the former designed Signal Interpreted Petri Nets specification model using SFC to structure the program, LD and FBD to express the transition conditions.

The second hierarchical level of the SIPN model is developed in the main SFC. The machining cycles are described in three other SFC and are called by the main SFC. The different SFC exchange signals to evolve. Furthermore some places of the Interpreted Petri Nets have been suppressed by the use of Functions Blocks to express edges and time delays as shown in Figure 8 and Figure 9. It can also be noticed that parallelism has been replaced by qualified actions.

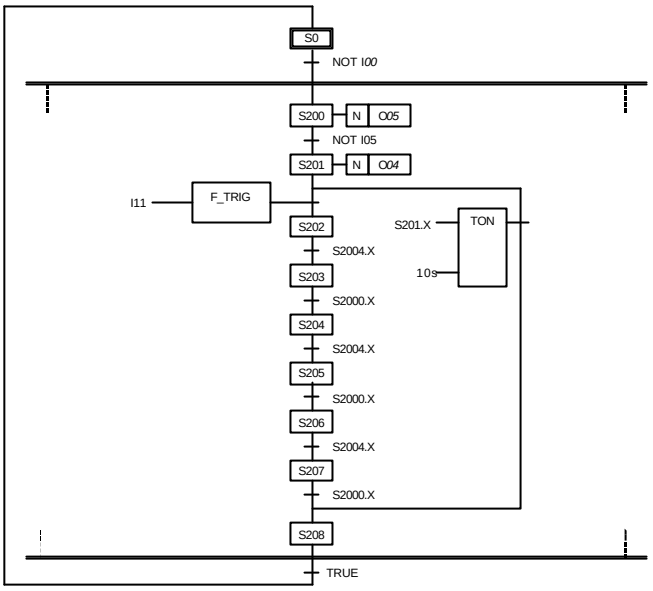

Figure 8. Implementation program (main SFC).

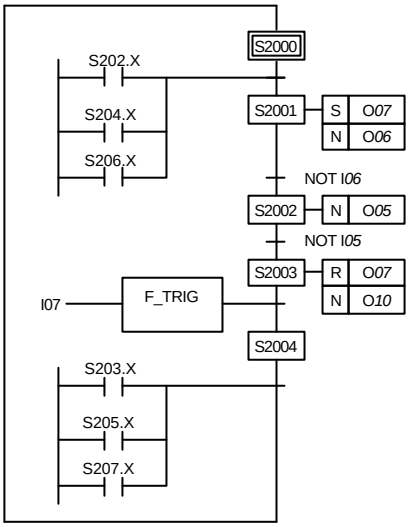

Figure 9. Implementation program (vertical milling machine SFC).

The complete implementation program stands out of 4 SFC with 41 steps, 42 transitions, 4 LD and 7 FBD used to express transition conditions.

To perform validation on this program, we have to describe its structure and behavior in order to use Cadence SMV.

5.3 Validation

Before the SFC program can be validated, we have to describe the cyclic evolution (E) of the PLC in which it will be implemented. This cyclic behavior [16] can be modeled by Figure 10.

After having made sure that the designed SFC was semantically correct (verification), we performed validation. We noticed that property 1 was no longer supported by the pro-

gram. The trace delivered by Cadence SMV is shown in Figure 11.

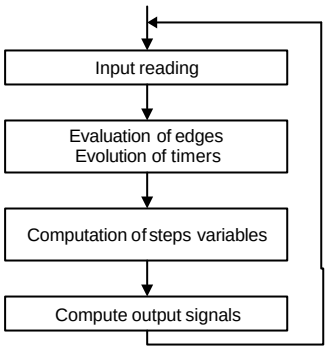

Figure 10. PLC cyclic model

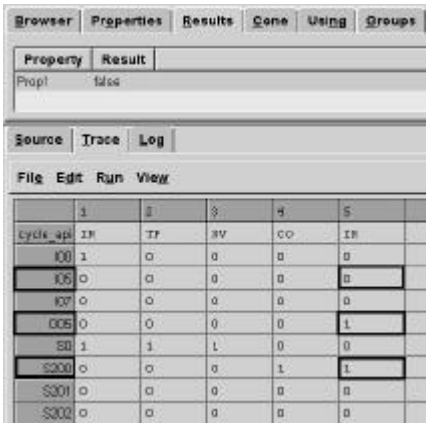

Figure 11. SMV diagnosis for the validation of Prop1 (SFC)

This property is not validated by the implementation program because even if the mill is already at the upper position (*I05*=0), the motor will be set to 1 when the step S200 is reached. This is due to the fact that there are no transient steps in SFC. Actually every step is active at least one cycle time and the associated outputs are emitted. The solution consists of introducing a new step and a new transition in order to pass over the step S200 when the mill is already at the upper position.

This example shows that, even if the SFC is a simple translation of the former validated SIPN, errors can occur because of the different dynamics of the models.

6 Conclusion

In this paper, a formal model of the controller of the Flexible Manufacturing System has been designed using Signal Interpreted Petri Nets. This formal description has been validated via model-checking and then translated into a programming language of the IEC61131. Due to its graphical representation, Sequential Function Chart has been chosen. This program has been validated anew to make sure that the former validated behavior has not been modified during the translation phase.

This example shows that formal methods like validation are well suited for the design of control algorithms. But due to the different languages, and therefore the different semantics, used along the design process, a new step of validation has to be performed after each translation of the controller. None of these validation steps can be omitted if we want to be sure that *What You Prove is What You Get.*

References

[1] University of Michigan http://www-personal.engin.umich.edu/~tilbury/testbed/

[2] H. Maas and G. Frey; "Documentation and Control Scenarios for a Flexible Manufacturing Line", Technical Report I17/2001, Institute of Automatic Control, University of Kaiserslautern, Dec 2001.

[3] T. Filkron, M. Hölzein, P. Warkentin, M. Weiß; "Formal verification of PLC-programs", Proc. of the14th World Congress of IFAC, paper No. C-2a-15-1, 1999.

[4] G. Frey; "SIPN, hierarchical SIPN, and Extensions", Technical Report I19/2001, Institute of Automatic Control, University of Kaiserslautern (Germany), Dec 2001.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] K. Mc Millan: Cadence SMV-Symbolic model checker. http://www-cad.eecs.berkeley.edu/~kenmcmil/

[10] 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.

[11] X. Weng and L. Litz: "Model checking of Signal Interpreted Petri Nets", Proc. of the IEEE international Conference on Systems Man and Cybernetics, SMC 2001, Tuscon (AZ), USA, pp. 2748-2752, Oct. 2001.

[12] G. Frey, L. Litz; "Formal methods in PLC programming", Proc. of the IEEE Conference on Systems Man and Cybernetics SMC 2000, pp. 2431-2436, Nashville, Tennessee (USA), Oct. 8-11, 2000.

[13] IEC, International Standard 61131-3, 2nd Ed., Programmable Controllers - Programming Languages, 1999.

[14] IEC, International Standard 60848, Preparation of Function Charts for Control Systems, 1988.

[15] G. Frey; "PLC programming for Hybrid systems via SIPN", Proc. of the 4th International Conference Automation of Mixed Processes, ADPM 2000, pp. 189-194, Dortmund (Germany), Sept. 2000.

[16] S. Lampérière-Couffin and J.-J. Lesage, "Formal verification of the sequential part of PLC programs", Proc. of the 7th IEE Workshop on Discrete Event Systems, WODES 2000, Gent (Belgium), pp. 247-254, Aug. 2000.