Academy of Romanian Scientists
University of Oradea, Faculty of Electrical Engineering and Information Technology

Vol. 2, Nr. 1, 2009

Journal of Computer Science and Control Systems

University of Oradea Publisher

Academy of Romanian Scientists
University of Oradea, Faculty of Electrical Engineering and Information Technology

Vol. 2, Nr. 1, 2009

Journal of Computer Science and Control Systems

University of Oradea Publisher

ISSN 1844 - 6043

This volume publishes papers presented at *The 10th International Conference on Engineering of Modern Electric Systems*, 27-29 May 2009, Oradea, Romania and is sponsored by *The National Authority for Scientific Research*, Romania, within the frame of Grant no. 33M/21.04.2009.

CONTENTS

PART 1. COMPUTER SCIENCE

PART 2. CONTROL SYSTEMS

PLC Programs Design Using Signal Interpreted Petri Networks

Eugen Ioan Gergely*, Geza Husi** and Sahin Yildirim***
*Department of Electrical Drives and Automation,
University of Oradea, Faculty of Electrical Engineering and Information Technology,
1 Universitatii Street, 410087 Oradea, Romania, E-Mail: egergely@uoradea.ro

**Department of Electrical Engineering and Mechatronics,
University of Debrecen, Faculty of Engineering,
2-4 Otemeto Street, H-4028 Debrecen, Hungary, E-Mail: husigeza@mk.unideb.hu

***Mechatronics Engineering Department,
Erciyes University, Faculty of Engineering,
Kayseri, 38039, Turkey, E-Mail: sahiny@erciyes.edu.tr

Abstract *– The paper presents an approach for designing dependable programmable logic controller (PLC) programs. It starts from informal specifications and ends with the final implementation on a real PLC. The approach uses Signal InterpretedPetri Networks (SIPNs) for modeling the control algorithm, model checking for model verification and validation, and automatic Instruction List (IL) program generation. Finally, the IL program is tested on a real PLC. A simple example is used throughout the paper in order to illustrate the framework. The advantage of the approach consists in the correctness of the resulting PLC programs, which makes them much more dependable than direct implemented PLC code.*

Keywords: *PLC, SIPN, model checking, dependability.*

I. INTRODUCTION

More and more manufacturing systems and plants are controlled by programmable logic controllers (PLCs). Due to the growing complexity of these systems and growing demands on safety and fault-tolerance of PLCs, classical methods for PLC programs design, like direct implementation, became prone to design errors.

A key way to handle the program complexity lies in the application of formal methods in the PLC program design process, which allows the application of formal verification and validation (V & V) methods and can guarantee that a program fulfills certain specified properties [1, 7].

The main purpose of using formal methods to design PLC programs is to derive a correct control algorithm prior to implementation. The earlier a problem is found, the easier and cheaper is to solve it.

Furthermore, control algorithms have to be adapted to new control scenarios very often. Therefore, the design has to be well documented and easy to understand, as well.

The paper presents an integrated approach to design PLC programs, which combines the concepts of correctness and transparency, thus leading to more dependable (safe and reliable) control programs [6, 9]. The presen-ted method starts from an informal description of the PLC program, proceeds over the design steps with for-mal V & V and assessment of program quality, and ends with automatic PLC program generation and testing.

For the formal modeling of the PLC programs the presented approach will use Signal Interpreted Petri Networks (SIPNs), which formalism is described in the following section. The verification is done using the model checker SMV, which means that the SIPN has to be translated into SMV input code and the standard functional properties to be verified have to be formalized using temporal logic. After the evaluation of non-functional properties, application specific functional properties are added to the SIPN model, which is validated with SMV (Sections III an IV). Having a validated model, the SIPN software automatically generates the PLC program in the Instruction List (IL) language. The program is then tested and the results are analyzed (Section V). The approach is illustrated using a simple example throughout the paper.

II. SIGNAL INTERPRETED PETRI NETWORKS

Signal Intrepreted Petri Networks SIPNs [3, 10] are an extension of Condition Event Petri Networks, dedicated to provide a graphical language for designing PLC control programs. They provide the possibility to handle signals: *transitions*, associated with firing conditions, and *places*, associated with outputs.

Firing conditions are Boolean functions over the SIPN input signals. A transition is enabled when its associated firing condition is fulfilled, thus passing the control token from one place (*pre-place*) to another one (*post-place*).

An output function of a place is activated while its corresponding place is marked. In general, output functions are sets of assignments to SIPN output signals.

The informal specifications from which the design shoud start consists of a set of properties that can be

splitted in three categories: standard functional properties, application specific functional properties and non-functional properties.

Standard functional properties are:

Safety: a SIPN is safe if the post-places of a transition need not to be checked to determine if the transition fires.

No Dynamic Synchronization (DS): two transitions t_1 and t_2 form a dynamic synchronization if the firing of t_1 implies the simultaneous firing of t_2.

Reachability: a marking m' is said to be reachable from a state m if there exists a sequence of input signal combinations such that a firing sequence starting from m has m' as a stable final marking.

Unambiguity: every control algorithm has to be defined unambiguously. This criterion can be subdivided into four subcriteria:

- **Determinism:** There must be no conflicts during control operation. If the control algorithm is not deterministic, implementational aspects will determine the behaviour of the control.
- **Termination:** In a cyclic control algorithm at least one marking must be stable. A cycle without stable marking would lead to an algorithm that does not terminate (i.e. endless loops).
- **Defined outputs:** There has to be a specification for the value of every output, i.e. 0 or 1, at every reachable marking.
- **Unambiguous outputs:** If two places marked at the same time assign different values to an output signal, an contradictory output setting results.

Liveness means that a transition can always fire again. When a transition or a set of transitions can no longer fire, then part of the control algorithm doesn't work anymore (i.e. dead code).

Reversibility means that the initial marking can always be reached again. This property guarantees that the controller reaches its initial state again.

Application specific functional properties (e.g. safety interlocks, disjoint activation of two output signals) are formalized by the development of the SIPN. During this manual synthesis, the specification of these properties form additional constraints for the designer.

Non-functional properties of the SIPN model are reflected by its **transparency**, which is a means to achieve maintainability, usability and portability. A SIPN model is transparent if:

a) At any time it is easy to see what the control algorithm does and which it will do in the next step.
b) At any time there is the possibility to reinterpret the algorithm (i.e. the algorithm is recognizable).

Transparency has values between 0 and 1. The closer to 1 the value is, the more transparent the algorithm is. Criteria and metrics for transparency are given in [4]:

T1: Comments: There should be a comment at every place/step/transition.

$$T1 = \frac{\#\,\text{Comments}}{\#\,\text{Places} + \#\,\text{Comments}} \quad (1)$$

T2: No trivial input: A defined input signal is trivial if it does not influence the model.

$$T2 = 1 - \frac{\#\,\text{TrivialInputSignals}}{\max(1, \#\,\text{InputSignals} - 1)} \quad (2)$$

T3: No trivial outputs: An output signal is trivial when is set to the same value all the time.

$$T3 = 1 - \frac{\#\,\text{TrivialOutputSignals}}{\#\,\text{OutputSignals}} \quad (3)$$

T4: No redundant output: There is redundant information if an output signal is set to the same value in several activated states.

$$T4 = \frac{\#\,\text{CorrectOutputSettings}}{\max(1, \#\,\text{OutputSettings})} \quad (4)$$

T5: Directionality: The control flow should follow one preferred direction.

$$T5 = \frac{\#\,\text{ArcsIn PreferredDirection}}{\#\,\text{Arcs}} \quad (5)$$

T6: Intersections: There should be not too much intersecting arcs.

$$T6 = 1 - \frac{\min(5, \#\,\text{Intersections})}{5} \quad (6)$$

III. WORKING EXAMPLE

The presented approach is ilustrated by the means of a simple example, which concerns a press used to mould metal parts (see Figure 1), controlled by a PLC.

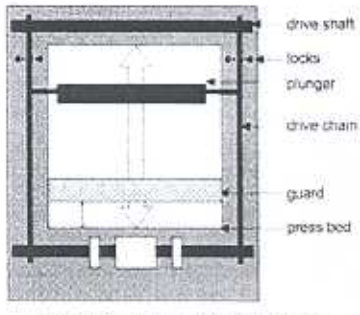

Figure 1. The moulding press

It consists of a heavy plunger which is raised meters above the press bed by the means of a plunger drive. An operator loads sheet metal onto the press bed, then moves away and pushes a button which causes the press to close. The plunger is released and falls under gravity, thereby pressing the workpiece. As safety measures, there are locks to hold the plunger at the top and a physical guard blocks operator access to the press while the plunger is falling. If the operator decides to abort prematurely the operation the plunger drive activates to slow and returns to the top. Table 1 shows PLC inputs and outputs.

A. Informal specification

Regarding the *application specific functional properties*, the control algorithm has to be designed using the following informal specifications:

TABLE 1. Inputs and outputs of the PLC

Type	Name	Meaning
Input	Start	Start one moulding cycle
Input	Plunger on top	The plunger is on the press top
Input	Plunger locked	The plunger is locked
Input	Guard closed	The protection guard is closed
Input	Plunger down	The plunger is on the press bed
Input	Abort	Operator commands cycle abortion
Input	PONR	The press has passed the point of no return
Output	Release plunger	The plunger is released and falls
Output	Move plunger up	The plunger moves up

- if the plunger is up, the guard is closed and the Start buton is pressed, the plunger moves down
- when the plunger is down, the operator opens the guard, takes the work piece and closes the guard
- the plunger moves up where is locked, ready for another cycle
- if the Abort button is pressed with the plunger above the PONR, the drive slows and reverses, moving the plunger back to the top
- if the Abort button is pressed when the plunger is below the PONR, the movement cannot be stoped.

From this informal specification a set of application specific functional properties can be derived. Here are some of them (within others):

P1: Releasing the plunger is always followed by moving it up.

P2: After the Start button has been pressed with the guard closed, the plunger moves down

P3: The plunger never moves down and up simultaneously.

With respect to *standard functional properties*, since the algorithm will be designed using SIPN, we demand that it should be formally correct according to the criteria defined in Section II.

As a measure of the design quality of a SIPN, transparency metrics for *non-functional properties* have been specified in Section II. Criteria T1÷T6 are meant to show that the SIPN:

- is well structured
- is easy to read.

B. Formalization of the algorithm

SIPNs are Petri Nets with binary marking and the following extension for the information flow (I19):

- every transition is associated with a boolean function of the input signals (the firing condition)
- every place is associated with an output function that assigns a set of output signals while it is marked.

The dynamic behaviour of a SIPN consists in a flow of tokens through the net, realized by the firing of transitions. Such a firing removes a token from its pre-places and puts a token on each of its post-places, according to the following rules:

- a transition is enabled if all its pre-places are marked and all its post-places are unmarked
- a transition fires immediately if it is enabled and its firing condition is fulfilled
- all transitions that can fire and are not in conflict with other transitions fire simultaneously.
- the firing process is iterated until a stable marking is reached
- after reaching a stable marking the output signals are computed by the assessment of the output functions of the marked places.

The SIPN for the working example has been realized with the tool `SIPN Editor`[1] and is given in Figure 2.

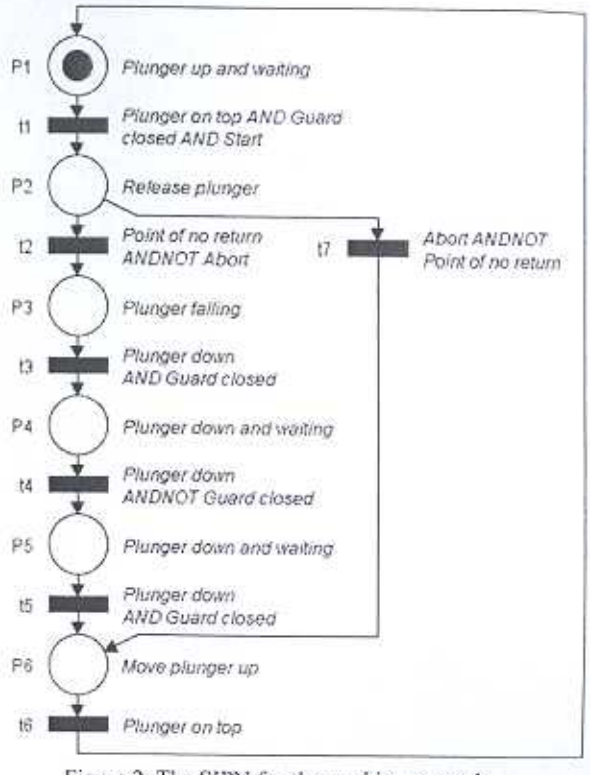

Figure 2. The SIPN for the working example

B1. Formalization of application specific functional properties. In order to perform V& V we well use model checking [2, 5, 8]. Therefore the properties have to be translated into Temporal Logic (TL) formulae. For the properties P1÷P3 we derive the following formulae:

```
P1: SPEC AG ((Release_plunger=1)→
            EF (Move_plunger_up=1))
```

which means that: It is always true (AG) that Plunger moving down (Release_plunger=1) implies (→) that in the future (EF) the plunger will move up (Move_plunger_up=1).

Properties P2 and P3 are defined in a similar manner:

```
P2: SPEC AG ((Start&Guard_closed)→
            EF (Release_plunger=1))
P3: SPEC AG ~((Release_plunger=1)&
            (Move_plunger_up=1))
```

B2. Formalization of standard functional properties. As before, standard properties are also needed as TL formulae. These properties are now automatically

[1] The `SIPN Editor` is available for download at http://www.eit.uni-kl.de/litz/

written in TL formulae by the SIPN Editor during the formalization of the algorithm. As an example, formulae P4 and P5 generated by the SIPN Editor are presented below:

P4 (determinism): for each potential conflicting transitions in the SIPN model, the SIPN Editor generates the necessary properties that have to be verified. Since that in the SIPN model of the working example there are two such situations (t2 & t7, t5 & t7), the SIPN Editor generates the following properties:

```
P4a: SPEC AG ! (t2 & t7)
P4b: SPEC AG ! (t7 & t5)
```

P5 (termination): The algorithm should always reach a stable marking i.e. never run in an infinite loop. In order to check this property the SIPN Editor automatically creates the variable `eoc` (End of Cycle) and the statement for its verification. For the working example its definition is:

```
eoc := !(t1|t2|t3|t4|t5|t6|t7);
```

and the property which `eoc` has to comply with is:

```
P5: SPEC AG EF eoc
```

The other standard functional properties are inferred in a similar manner.

B3. Formalization of non-functional properties. Assessing equations 1÷6 for the SIPN model given in Figure 2 yields a value close to 1. In fact, the criterion T4 is the only one which affects the model's transparency, due to the requirement that each output must be defined in each state.

IV. VERIFICATION & VALIDATION

These two concepts ore oftenly confused. *Verification* means to check if the algorithm complies with the standard functional properties. This step must be the first one, because if all the formalization is done and the verification result is `False`, the algorithm has to be reviewed until a `True` result occurs.

Validation signifies to check wether the verified control algorithm fulfills the application specific functional properties or not. After correcting the algorithm due to a `False` result, verification must be done again.

Although transparency metrics do not affect the correctness of the model, changes in the algorithm make necessary a new round of V & V.

The tool we use for V & V is `Cadence SMV`[2]. It requires a description of the control algorithm written in a text file (.smv) and a set of properties written in Temporal Logic (TL). If a checked property is fulfilled, SMV returns a `True` result. If a property is not fulfilled, SMV gives a `False` response and a counter-example as a `trace`.

Firstly, the verification of standard functional properties (among which are P4 and P5) is realized. The .smv file (automatically generated by the SIPN Editor) is executed by Cadence SMV. The `True` result obtained shows that the SIPN model accomplishes the standard functional properties.

This enables to add to the .smv file the necessary specifications for the validation of the application specific functional properties (P1÷P3). Launching Cadence SMV again on the .smv file yields `True` results, which means that the model is correct according to both standard and application specific functional properties. Figure 3 shows the results for both verification and validation in a compact manner.

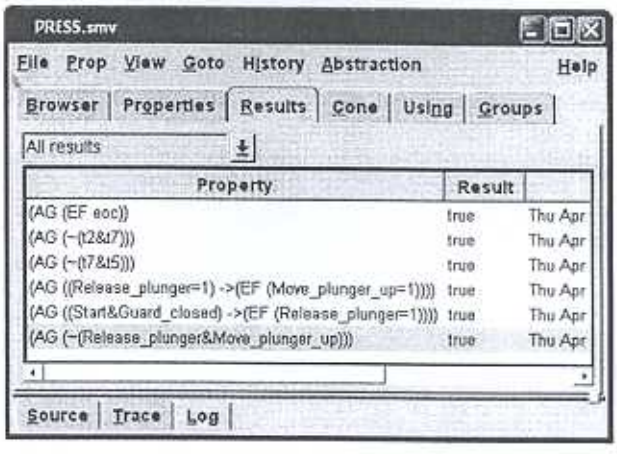

Figure 3. The results of V & V of the SIPN model for the working example

V. GENERATING & TESTING THE PLC PROGRAM

As mentioned in Section I, the SIPN Editor is able to automatically generate the Instruction List (IL) PLC program, according with the IEC 61131-3 Standard.

Although the SIPN Editor is provided with a Soft PLC, which can simulate the generated IL program, the PLC program for the working example has been downloaded and tested on a real PLC.

Minor changes had to be done to the PLC program before download, mainly due to the fact that the used PLC has different sintax for several instructions.

Input signals were generated using switches. The inputs sequences followed both normal operation scenarios and random sequences. The PLC program worked correctly for both kind of input sequences.

Although the obtained PLC program is correct and clear, it is not the fastest. As mentioned in Section II, one of the standard functional properties that has to be accomplished is that in the SIPN model it has to be a specification for the value of every output, i.e. 0 or 1, at every reachable marking (*defined outputs*). For the working example this yields 7 "redundant" instructions:

- the output *Release plunger* has to be set in P2 and reset in P3 and P6, but it is reset also in P1, P4, P5.
- the output *Move plunger up* is set in P6 and reset in P1, but it is reset also in P2, P3, P4, P5.

Despite these considerations, these "redundant" instructions were kept in the PLC program in order to assure its deterministic behaviour.

[2] Cadence SMV has been developed at Cadence Berkeley Labs and is available for download at http://www.kenmcmil.com/smv.html

VI. CONCLUSIONS

In this paper we presented an approach to deliver correct and dependable PLC programs. Detailed steps of the design process were illustrated through a simple example.

For modeling the control algorithm we used the formalism of the Signal Interpreted Petri Networks (SIPNs) and the SIPN Editor[1] tool. The advantage is that the tool allows to easily make the verification and validation of the SIPN model by the automatic generation of the standard functional properties and of the SMV input code.

Standard functional properties were verified using the Cadence SMV[2] tool. Then SMV has been used again in order to validate the application specific functional properties. In both cases SMV gave a `True` response, which means that our control algorithm is correct. Together with a favourable assessment of non-functional properties, this allowed the automatic generation of the PLC program by the SIPN Editor in the Instruction List (IL) language, followed by its testing on a real PLC.

The IL program needed minor (syntactical) adjustments, due to the fact that the used PLC is not strictly conforming to the syntax issues of the IEC 61131-3 standard.

Analyzing the PLC program, it results that in order to preserve its deterministic behaviour we have to admit longer program scan cycles.

The presented approach provides correct, thus dependable, PLC programs. Although there are no metrics to estimate the dependability gain (in terms of safety and reliability), the improvement is undeniable, especially when compared with direct implemented PLC programs.

REFERENCES

[1] Nina Amla, X. Du, A. Kuehlmann, R. P. Kurshan, K. L. McMillan, *"An Analysis of SAT-Based Model Checking Techniques in an Industrial Environment"* Proceedings of 13th Advanced Research Working Conference on Correct Hardware Design and Verification Methods CHARME'05, Saarbrucken, Germany, pp. 254-268, October 2005.

[2] B. Berard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci and P. Schnoebelen, "Systems and Software Verification - Model-Checking Techniques and Tools", Springer, Berlin, New York, 2001.

[3] D. Dimitrova, G. Frey, I. Batchkova, *"Sequential Control at the Supervisory level of Batch Plant using Signal Interpreted Petri Nets"*, Proceedings of the International Conference on Automatics and Informatics, Sofia, Bulgaria, Vol. 2, Part V, pp.17-20, October 2007.

[4] G. Frey and L. Litz, *"Transparency analysis of petri net based logic controllers – a measure for software quality in automation"*, Proceedings of the American Control Conference ACC 2000, Chicago USA, pp. 3182-3186, June 2000.

[5] E. I. Gergely, H. M. Silaghi, *"Using model checking for verification of programmable logic controllers programs written in ladder diagram language"*, Scientific Proceedings of Riga Technical University, Vol. 11, part 4: "Power and Electrical Engineering", Riga, Latvia, pp. 72-78, 2004.

[6] E. I. Gergely, *"Automatic safety analysis of computer controlled plants using model checking"*, Analele Universității din Oradea, Fascicola Electrotehnică, Secțiunea Știința Calculatoarelor și Sisteme de Control, Oradea, Romania, pp. 194-198, 2005.

[7] Y. Hietter, J. M. Roussel, J. J. Lesage, *"Algebraic synthesis of dependable logic controllers"*, 17th IFAC World Congress, Seoul ,Korea, pp. 4132-4137, July 2008.

[8] O. Pavlovic, R. Pinger and M. Kollmann, *"Automation of Formal Verification of PLC Programs Written in IL"*, Proceedings of 4th International Verification Workshop VERIFY'07 Bremen, Germany, pp. 152-163, July 2007.

[9] J.M. Roussel, J.M. Faure, "Designing dependable controllers using algebraic specifications", Control Engineering Practice, Volume 14, Issue 10, pp. 1143-1155, October 2006.

[10] M. B. Younis, G. Frey, *"Formalization of PLC Programs to Sustain Reliability"*, Proceeding of the 2004 IEEE Conference on Robotics, Automation and Mechatronics, RAM-2004, Singapore, pp. 507-511, December 2004.