Boel, R.; Stremerch, G. (eds): Discrete Event Systems, Analysis and Control;
Kluwer Academic Publishers 2000, ISBN 0-7923-7897-0, pp. 275-282

Time-related Modelling of PLC Systems with Time-less Petri Nets *)

Monika Heiner, Thomas Menzel

Brandenburg University of Technology at Cottbus
Department of Computer Science
Postbox 101344, D-03013 Cottbus, Germany
phone: +49-355 69 3885, fax.: +49-355 69 3830
{mh,thm}@informatik.tu-cottbus.de, http://www.informatik.tu-cottbus.de

Key words: Petri nets, programmable logic controller, system program, static analysis.

Abstract: At WODES '98, we introduced an approach to modelling and analysis of PLC systems using ordinary Petri nets. This paper supplements the former one by presenting detailed contributions to a systematic model design of the application program's surroundings. For that purpose, design aspects of the environment model as well as three different kinds of models for the system program are discussed. As a side effect, an obvious principle of timer modelling turns out.

1. INTRODUCTION

Programmable Logic Controllers (PLC) are widely used in many industrial areas, including especially safety-critical applications. Therefore, it is an challenging objective of the responsible professional to validate newly developed PLC against a separately given specification of functional, safety and performance requirements. Any mathematically stringent validation requires some kind of formal model of the system under validation. Among other formalisms, Petri nets represent a well-established and promising approach due to the inherent concurrency of many PLC applications.

One objective of our on-going project is the explicit support of development engineers, who are hardly interested in formal methods. Consequently, the aim is to transform automatically an existing PLC application into a corresponding Petri net - at least as far as possible. The resulting net model allows impartial validation of the given requirements by the numerous available sophisticated Petri net analysis techniques and tools [Heiner98a].

*) This work is supported by the German Research Council under grant ME 1557/1-1

The systematic model generation of a PLC system may be divided into four steps:

1. translation of the **application program** (written e.g. in instruction list notation [IEC1131-3]), as described in [Heiner98b], full version in [Heiner97].
2. modelling the **uncontrolled plant** (plant without controller), using a suitable library of Petri net components of the environment model (see chapter 2),
3. modelling resp. selection of the adequate **system program** (see chapter 3),
4. composition of the three model parts of the steps 1-3 to the total system model.

Step 1 has to be repeated until the application program satisfies the given requirements. Step 2 needs to be done once for each configuration of the uncontrolled plant, while step 3 generally takes place just once during the project development cycle.

This paper is organized as follows: The next chapter discusses some essential aspects of the environment model, and chapter 3 presents three different types of system program modelling and related consequences. We conclude with final remarks in chapter 4.

2. THE ENVIRONMENT MODEL

The validation of PLC systems, as of any kind of reactive systems, does not only require a model of the software - the controlling part of the system, but also an adequate model of the plant - the controlled part of the system. The latter one is shortly called environment model.

Basically, the environment model is a state-oriented one. Each local state, reachable in the uncontrolled plant (e.g. a switch is *on* or *off*), is represented by a corresponding (logical) place. Each actuator, effecting a local state change in the plant, is modelled as a (logical) transition between the related places (e.g. *switch_on* and *switch_off*). For a small example see top of Figure 1.

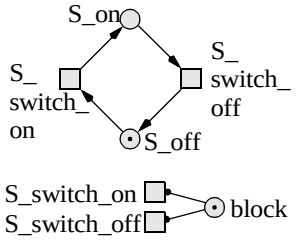

Figure 1: Environment model with blocking place.

Generally, a PLC system can only process discrete values, and analogue values have to be

mh@informatik.tu-cottbus.de

discretized appropriately. Consequently, each significant state or significant set of equivalent states of any analogue sensor is modelled as a (logical) place in the environment model.

Moreover, PLC-based automation engineering generally assumes that there are no state changes in the environment during the application program's execution. With other words: the control program is considered to be fast enough to observe all state changes in the environment. In order to ensure this assumption in the model, a mechanism is needed to block temporarily actuator transitions. For that purpose, a single blocking place is introduced additionally.

The initially marked blocking place is connected with each transition of the environment model by a so-called read arc (highlighted graphically by a black dot instead of the arc arrow, compare bottom of Figure 1). Semantically, read arcs establish side conditions for transition firing. Therefore, the whole environment model can now be blocked by just removing the token from the blocking place.

The synchronisation between PLC and plant, necessary to reflect the correct PLC semantics, is realized by the system program (compare next chapter). It removes the token from the blocking place before starting (the next cycle of) the application program, and it puts back the token after finishing (a cycle of) the application program's execution.

This enforced synchronization behaviour eliminates unrealistic system states in the model, which has two very important consequences for the analysis quality:

- **safety properties** (something bad never happens) become more likely to be analyzable because the model's state space becomes smaller,
- **liveness properties** (something good may happen) become analyzable at all, because unrealistic system states in the model behaviour distort these kind of properties.

However, there are still unrealistic state changes possible in our formal model while the environment model is unblocked. Examples are: nothing happens in the environment, resulting into the repeated execution with identical parameters, or a sequence of changes in the environment takes place before the application program is executed again. The next chapter is devoted to that problem and will present three different approaches to solve it.

3. THE SYSTEM PROGRAM MODEL

Each PLC application program is embedded in a PLC system program, which is basically responsible for two different tasks. At first, it transforms the physical data of the environment into application program values, and vice versa. For that purpose, the system program maps external values and values of internal variables to each other. Figure 2 illustrates how that may look like as Petri net for the small example of Figure 1, whereby S_off/S_on stand for external values and S_0/S_1 stand for internal values.

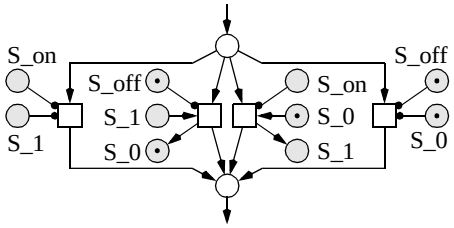

Figure 2: Mapping of external to internal values.

Secondly, the system program drives the (infinite) cyclic execution of the application program. Such a cycle is shortly called PLC cycle. The basic scheme of the interplay of system program and application program within a PLC system is given in Figure 3.

The PLC cycle establishes exactly two time points, at which an interaction between PLC program and environment can take place. The environment state is only propagated to the PLC program by the mapping of external into internal values during the 'system in' phase. There is no communication with the environment during the execution of the application program, therefore any state changes are not observable for that time.

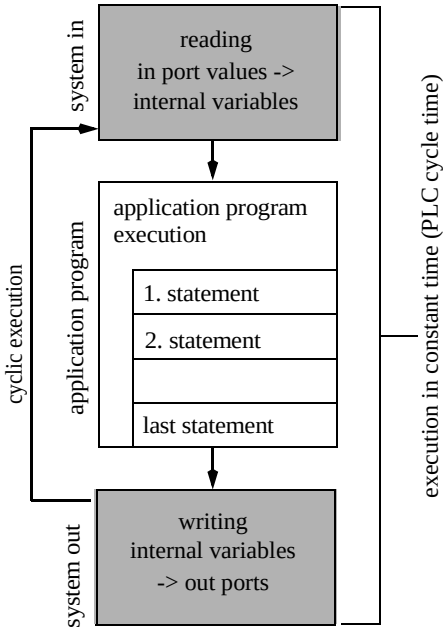

Figure 3: PLC cycle scheme.

Consequently, it is obviously very important for a controlled system behaviour that the execution of the application program happens *fast enough*. If the execution speed of the application program is much higher in relation to the speed of environment changes, then all changes in the environment are

observed and processed by the application program. This closes the loop back to the basic assumption mentioned in the chapter before and explains at the same time the reasoning behind it.

Following this line, absolute execution speeds can been reduced to a relation between speeds: on the one side of the PLC system, determining the execution time, and on the other side of the state changes within the environment. Using this term, the crucial point may now be rephrased as: how much faster is the controller compared with the environment. This relation determines significantly the PLC system behaviour, therefore it should be considered in the model of the system program. Basically, there are three different alternatives:

- State changes in the environment are possible after each PLC cycle (3.1).
- State changes in the environment are possible after a certain number of executions of the application program (3.2).
- State changes in the environment are only possible if the application program has computed completely the output values for the set of input values read at the beginning of the cycle (3.3).

The following sections discuss these three alternatives resulting into different system programs (which generally exhibit different system behaviour).

3.1 One Cycle Blocking

This semantics may be realized by the model sketch shown in Figure 4. The start of the PLC cycle is combined with the token removal from the global blocking place (compare chapter 2). Therefore, no more state changes in the environment model are possible from that point on. Having excluded any enabled transitions within the environment model, all external values are read and mapped to internal variable values.

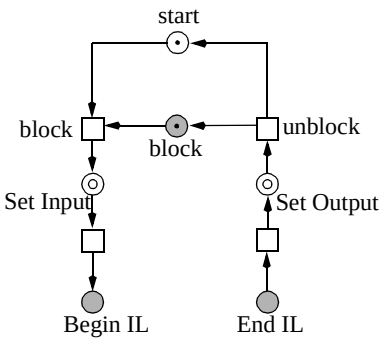

Figure 4: System program with one cycle blocking.

After one complete execution of the application program (between the places ’Begin IL’ and ’End IL’, not given in the figure here), the output values are written and the environment is released by laying back a token on the blocking place.

Obviously, the plant is blocked for just one PLC cycle. In the understanding of the automation engineering it reflects the expectation that the PLC reacts on each input mapping within exactly one cycle. With other words, the corresponding output mapping is produced completely within one cycle.

3.2 Blocking for *n* Cycles

This semantics is considered to be likely to reflect the realistic case. Generally, the execution speed of a PLC is *n* times faster than the environment is able to change. A cycle counter is introduced to model that factor *n*, using the same binary representation of values as described in [Heiner98b] for program variables. The environment is blocked for *n* cycles following the procedure discussed in the preceding section. The resulting model is quite large, therefore Figure 5 just shows a part of it.

Worth mentioning here is the fact that this type of modelling allows at the same time the introduction of timers. In case of PLC programming, timers are no real-time ones, but the timer value represents a relation to a given number of PLC cycles which are all executed in constant cycle time. Therefore, the timer progress may be determined by help of the cycle counter.

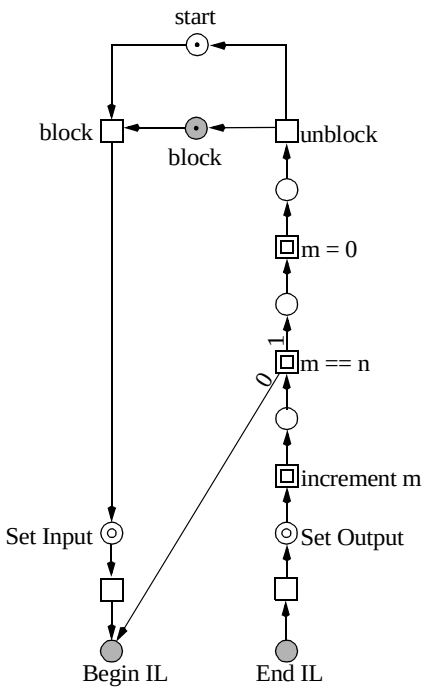

Figure 5: System program with factor n.

3.3 Blocking until Complete Output Mapping

This semantics assumes that a PLC is always as fast as necessary to compute a complete output mapping for a given set of input values before any state changes in the environment may happen.

Therefore, the environment is blocked as long as the application program

changes some variable values. As soon as there are no more changes of variable values, the output mapping has been computed completely, and any more cycles will not bring any news.

There are four IL commands, which can change a variable's value: `R` - Reset, `S` - Set, `ST` - Store, `STN` - Store if not. Each command possesses its own Petri net component, modelling its IL semantics (for details see [Heiner98b]). These basic components have to be extended by a flag indicating any variable changes occurring during the PLC cycle (see Figure 6 for an example of the command `S Step`).

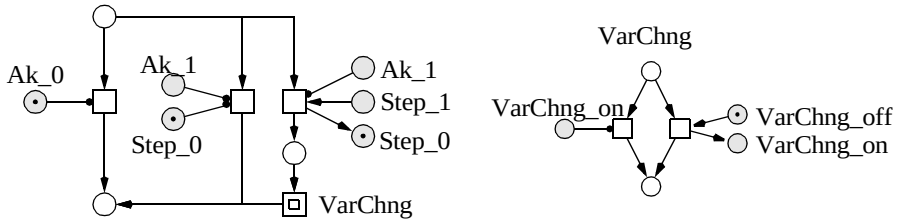

Figure 6: Extension of some Petri net modules.

Having done these extensions, the system program can be modelled as given in Figure 7. This model of the system program checks, whether there has been a change of any variable value. If this is the case, the environment is kept blocking. If the execution has reached eventually a cycle where nothing news happens (no variable obtains a new value), then the environment is released.

Finally, this system model reflects adequately the fact that a PLC is always much faster than the controlled environment. The output mapping is always computed completely before any input values may change.

Again, this model may be improved by a counter, in order to support timer

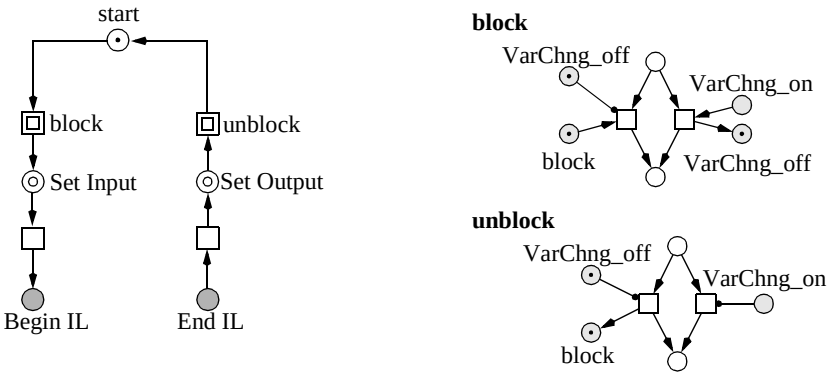

Figure 7: Blocking until complete output mapping.

components. But in this case, a value range check is necessary, because a PLC with a system program of type 3 may run unpredictably long.

4. SUMMARY

In this paper, a method has been described to transform a PLC system into a Petri net in order to allow stringent analyses. Special emphasis has been laid on the system program's modelling style. Different approaches have been discussed to describe timely relations between the speeds of the application program's execution and of any changes in the controlled environment.

In opposite to [Hanisch97], only time-free ordinary place/transition nets are used. All graphically highlighted net extensions are just syntactical sugar. Therefore, the resulting Petri nets are analyzable by the available powerful set of conventional Petri net analysis techniques and tools [Heiner99].

But the most crucial point is that an adequate modelling of the synchronization between controlling and controlled system part avoids unrealistic model behaviour, and therefore allows the analysis of not only safety properties, but also of liveness properties.

References

[IEC1131-3] IEC Standard 1131-3, Programmable controllers - Part 3: Programming languages; Int. Electrotechnical Commission, 1993.

[Hanisch97] Hanisch, H.-M. et al.: Modelling of PLC Behaviour by Means of Timed Net Condition / Event Systems; Proc. 6th IEEE Int. Symposium on Emerging Technologies and Factory Automation (ETFA '97), Los Angeles, Sept. 1997, pp. 361-396.

[Heiner97] Heiner, M.; Menzel, T.: Petri Net Semantics for the PLC User programming language Instruction List (in German); Techn. Report BTU Cottbus, I-20/1997, Cottbus December 1997.

[Heiner98a] Heiner, M.: Petri Net Based System Analysis without State Explosion; Proc. High Performance Computing '98, Boston, April 1998, SCS Int. San Diego 1998, pp. 394 - 403.

[Heiner98b] Heiner, M.; Menzel, T.: A Petri Net Semantics for the PLC Language Instruction List; IEE Workshop on Discrete Event Systems (WODES '98), Cagliari/Italy, August 1998.

[Heiner99] Heiner, M.; Deussen, P.; Spranger, J.: A Case Study in Design and Verification of Manufacturing System Control Software with Hierarchical Petri Nets; Int. J. of Advanced Manufacturing Technology, Springer-Verlag London, (1999) 15, pp. 139-152.

 mh@informatik.tu-cottbus.de