Design of Safe PLC Programs by Using Petri Nets and Formal Methods

EUGEN IOAN GERGELY¹, LAURA COROIU¹, ALEXANDRU GACSADI²
¹Department of Electrical Drives and Automation
²Department of Electronics
University of Oradea
1 Universitatii Str., Oradea 410087
ROMANIA
egergely@uoradea.ro, lcoroiu@uoradea.ro, agacsadi@uoradea.ro

Abstract: - The paper presents an approach which combines the formalism of Petri Nets and of model checking in order to deliver correct and dependable PLC programs. Based on SIPNs, a variant of PNs, the complete controller development process from an informal specification to the final implementation on a PLC is discussed. The SIPN formalized according to corresponding plant model enables the derivation of standard functional properties and the specific functional properties of the PLC control algorithm. These properties are verified and validated with the model checker tool Cadence SMV. In order to do this, the SIPN is translated into SMV input code, in which there are inserted the functional properties to be verified and validated, expressed by using Temporal Logic formulae. This correct SIPN is used as a basis for implementation. For the realization there are used standard PLC programming languages according to IEC 61131. It resulted that the approach provides correctness of the resulting PLC programs, which makes them much more dependable than direct implemented PLC code. In order to illustrate the approach steps a working example was used.

Key-Words: - programmable logic controllers, signal interpreted Petri Networks, model checking

1 Introduction

Today programmable logic controllers (PLCs) are the main equipment of automation in all areas from single machines to manufacturing plants and continuous processes. The increasing complexity of the control software and the rising of user-defined safety and functionality requirements necessitates new methods to provide proper control software in view of given requirements [1, 2]. Because of this growing complexity classical methods of designing PLC programs, like direct implementation, are no longer feasible.

A method to handle these requirements lies in the application of formal methods in the PLC program design. It allows the application of formal verification and validation methods, which can assure that a program fulfills certain specified properties [3, 4, 5, 6]. The main purpose of using formal methods to design PLC programs is to derive a correct control algorithm prior to implementation and realization.

In model-based approaches a model of the process under control is included in the analysis. Petri Networks (PNs) have shown good properties in modeling control algorithms [7, 8]. They proved to be able to express the causality and the concurrency of control algorithms in a transparent way.

Signal Interpreted Petri Networks (SIPNs) are an extension of the basic PN framework. In addition to the potential of graphical representation and mathematical treatment of PNs, SIPNs allow explicit treatment of input/output facilities [9, 10].

The approach presented in the paper combines SIPNs and model checking in a framework which provides correct, thus dependable, PLC programs. The paper is structured as follows. In the next section are presented the basic concepts of SIPNs. Section 3 describes how to use formal methods in the control design process. In order to illustrate the approach steps, a complete controller development process, which includes the steps of design, verification, validation, implementation and realization on a PLC is presented in Section 4 through a working example.

2 The Formalism of SIPNs

SIPNs are an extension of ordinary PNs [11] with input and outputs elements. A SIPN is a 10-tuple SIPN=(P, T, F, m₀, I, O, φ, ω, Ω, ν) with [9]:

(P, T, F, m₀) - an ordinary PN with places P, transitions T, arcs F and binary initial marking m₀
I - a set of input signals
O - a set of output signals
φ - a mapping associating every transition with a firing condition
ω - a mapping associating every place with an output
The \( \text{formalization} \) represents the conversion of an informal specification into a formal specification (e.g., a SIPN). This conversion can be done by using computers, but it is not fully automatic and requires human expertise.

The formalization of the informal specification consists in the following tasks:
1. **Formalization of specific properties**, which produces a set of properties to be fulfilled by the PLC or the controlled process.
2. **Formal modeling of the uncontrolled process**, which results in a process model that is needed in model-based approaches.
3. **Direct formal modeling of the control algorithm**, which can be done if the control problem given by the informal specification is very clear.

Depending on the formal methods used, not all of these tasks have to be done. This paper focuses on the formalization of specific properties using temporal logic and model checking [13].

The **implementation** is the process of deriving the target-system dependent realization from the formal specification. Using one of the standardized PLC languages, the formal description of the control algorithm is implemented in a direct manner (by using a compiler) or indirectly (by using an interpreter implemented in the PLC).

Generally, the realization includes hardware and software. Assuming a standard hardware with a well-defined functionality, the realization is in fact the program of the control algorithm (i.e., the software). For this, PLC languages according to [12] are more and more accepted.

The **verification and validation** (V & V) are the main areas for applying formal methods in PLC programming [14].

Verification means the application of formal methods in order to prove that the control algorithm fulfills a given specification (i.e., standard functional properties), which yields important information about the correctness of the control algorithm. The properties investigated by verification are standard and hence can be assumed as already formalized. Therefore, the verification can be fully automated.

In validation application specific functional properties of the control algorithm have to be formalized. Validation shows if the controlled process behaves as it should. The validation process uses as inputs the information from the informal specification and from realization. Hence, validation cannot be fully formal and cannot be fully automated, requiring the designers’ expertise.

In this paper, in order to perform V & V it is used the same method: symbolic model checking. This is
a technique in which finite model of the system is built and the expected properties of the system are checked on this model. The system is modeled as a finite state transition system and the properties are expressed in a Temporal Logic [15]. Then, a search procedure is used to check whether the expected properties hold on the finite state transition system or not. In symbolic model checking the state space of the finite state transition system is not explicitly built, so Binary Decision Diagrams (BDDs) are used to represent the system states.

The tool we use is Cadence SMV (http://w2.cadence.com/webforms/cbl_software/index.aspx). It requires on the one hand side a description of the control algorithm given in a text file and on the other hand a set of properties written in Temporal Logic. As a result, the model checker gives us a verdict (True/False) and a diagnosis which is a counterexample given as a trace. So, in order to use SMV, we have to translate the SIPN describing the control algorithm into SMV input code.

### Table 1. The PLC I/O signals.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Start_button</td>
<td>Starts one mixing cycle</td>
</tr>
<tr>
<td>Input</td>
<td>Tank_empty</td>
<td>The tank is empty</td>
</tr>
<tr>
<td>Input</td>
<td>Tank_half</td>
<td>The tank is filled with the necessary quantity of liquid A</td>
</tr>
<tr>
<td>Input</td>
<td>Tank_full</td>
<td>The tank is filled with the necessary quantity of liquid B</td>
</tr>
<tr>
<td>Output</td>
<td>Open_Valve_1</td>
<td>Liquid A is filled into the tank</td>
</tr>
<tr>
<td>Output</td>
<td>Open_Valve_2</td>
<td>Liquid B is filled into the tank</td>
</tr>
<tr>
<td>Output</td>
<td>Open_Valve_3</td>
<td>The A+B mix is emptied from the tank</td>
</tr>
<tr>
<td>Output</td>
<td>Motor_on</td>
<td>The A+B mix is stirred</td>
</tr>
</tbody>
</table>

The informal specification (i.e., the expected behaviour) given above is formalized by developing the SIPN of the PLC control algorithm (see Fig. 3).

The algorithm in Fig. 3 works as follows: in the initial state only $P1$ is marked and hence the output of the net is $(Open_{Valve_1}, Open_{Valve_2}, Open_{Valve_3}, Motor_{on}) = (0, 0, 0, 0)$. Transition $t1$ fires when the start button is pressed ($Start_{button}=1$) and the tank is empty ($Tank_{empty}=1$). The token from $P1$ is removed and is generated in $P2$. Valve 1 is open, so the new output of the net is $(1, 0, 0, 0)$. If in $P1$ the tank is not empty ($Tank_{empty}=0$), then $t6$ fires, also removing the token from $P1$ but putting it in $P5$.
After the filling level for liquid A is reached (\(Tank\_half=1\)) when in \(P2\), \(t2\) can fire, thus moving the token from \(P2\) to \(P3\), which closes \(Valve\_1\) and opens \(Valve\_2\). When the filling level for liquid B is reached (\(Tank\_full=1\)), then \(t3\) fires and the token is moved from \(P3\) to \(P4\), thus starting the stirring motor \(M\) (\(Motor\_on=1\)) for 60 seconds. The expiration of this time will fire \(t4\), which moves the token from \(P4\) to \(P5\). A \(A+B\) mix is evaporated from the tank (\(Open\_Valve\_3=1\)) till the tank is empty. This will fire \(t5\) which removes the token from \(P5\) and puts it in \(P1\), thus resulting the initial state again.

The SIPN in Fig. 3 is used to formalize the standard functional properties and the specific functional properties of the PLC control algorithm. As mentioned in Section 3, the standard functional properties will be verified and the specific functional properties will be validated by model checking. For both of these the same tool -Cadence SMV- will be used. Therefore, all functional properties will be expressed as Temporal Logic formulae. The standard functional properties that the SIPN in Fig. 3 should fulfill are given in Table 2.

It can be easily seen that the SIPN is implicitly safe due to its nature, so this property does not need any verification. Other properties, such as input dependence, can be verified by the analysis of the SIPN using Table 1.

In order to verify that the algorithm is deterministic we have to examine each potential conflicting transitions in the SIPN. Since that in the SIPN model of the working example there are two such situations (\(t1 \& t6\), \(t4 \& t6\)), the following properties are written:

\[
P1a: \text{SPEC AG } \neg (t1 \& t6) \quad (1)
\]

\[
P1b: \text{SPEC AG } \neg (t4 \& t6) \quad (2)
\]

which means that it is always true (AG) that in the future (EF) transitions \(t1 \& t6\) and respectively transitions \(t4 \& t6\) do NOT (\(\neg\)) fire simultaneously.

For verifying that the algorithm always terminates and never run in an infinite loop we create the variable \(eoc\) (End of Cycle) and the statement for its verification. For the working example its definition is:

\[
eoc := \neg (t1|t2|t3|t4|t5|t6) \quad (3)
\]

and the property which \(eoc\) has to comply with is:

\[
P2: \text{SPEC AG EF eoc} \quad (4)
\]

In order to verify the output correctness property, it should be checked if the output signals are defined in every stable marking reached. For example, for the Motor on output this would be:

\[
P3: \text{SPEC AG EF}
\]

\[
((eoc \& Motor\_on=0) | (eoc \& Motor\_on=1)) \quad (5)
\]

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

<table>
<thead>
<tr>
<th>Property</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Safety</td>
<td>A SIPN is safe if the post-places of a transition need not to be checked to determine if the transition fires.</td>
</tr>
<tr>
<td>Liveness</td>
<td>When a transition or a set of transitions is no longer fireable, then part of the control algorithm doesn't work anymore (i.e. dead code).</td>
</tr>
<tr>
<td>Reversibility</td>
<td>The initial marking can always be reached again.</td>
</tr>
<tr>
<td>Reachability</td>
<td>A marking (m') is reachable from a state (m) if there exists a sequence of inputs combinations such that a firing sequence starting from (m) has (m') as a stable final marking.</td>
</tr>
<tr>
<td>No Dynamic Synchronization</td>
<td>Two transitions (t1) and (t2) form a dynamic synchronization if the firing of (t1) implies the simultaneous firing of (t2).</td>
</tr>
<tr>
<td>Determinism</td>
<td>The algorithm is deterministic if the transitions firing conditions at every branching are disjoint.</td>
</tr>
<tr>
<td>Termination</td>
<td>In a cyclic control algorithm at least one marking must be stable. The algorithm terminates if there is no self-loop at any place (i.e. never run in an infinite loop).</td>
</tr>
<tr>
<td>Output correctness</td>
<td>The output signals have to be formally correct, i.e. 0 or 1 at every stage of the algorithm.</td>
</tr>
<tr>
<td>Input dependence</td>
<td>Every input signal should have an influence on the control algorithm.</td>
</tr>
</tbody>
</table>

For verifying the standard functional properties we use the tool Cadence SMV. It requires a description of the control algorithm written in a text file (.smv), in which is inserted the set of properties \(P1a, P1b, P2\) and \(P3\) written in Temporal Logic. 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.

The verification shows that all above specified properties are fulfilled. This enables the inference of application specific functional properties (e.g. safety interlocks, disjoint activation of two output signals, etc.). These are formalized from the SIPN by manual synthesis, thus requiring the designer expertise.

Some of the application specific functional properties which we want to be fulfilled by the working example are the followings:

\(P4\). It is forbidden that all three valves are open at the same time. Written in Temporal Logic this property is as follows:
P4a: SPEC AG
\(~((\text{Open\_Valve\_1}=1) \land (\text{Open\_Valve\_2}=1))\) (6)
P4b: SPEC AG
\(~((\text{Open\_Valve\_1}=1) \land (\text{Open\_Valve\_3}=1))\) (7)
P4c: SPEC AG
\(~((\text{Open\_Valve\_2}=1) \land (\text{Open\_Valve\_3}=1))\) (8)

P5. Stirring in an empty tank cannot occur. In Temporal Logic this is:
P5: SPEC AG
\((\neg (\text{Tank\_empty}) \rightarrow EF ~\neg (\text{Motor\_on}=1))\) (9)

P6. Stirring starts only in a fully filled tank, which in Temporal Logic is written as:
P6: SPEC AG
\((\neg (\text{Tank\_full}) \rightarrow EF (\text{Motor\_on}=1))\) (10)

P7. Stirring does not start while a valve is open.
The validation of the application specific functional properties will be done with symbolic model verifier Cadence SMV as well. In order to do this, the Temporal Logic formulae of the specific functional properties P4 - P7 are inserted in the same .smv file as the Temporal Logic formulae of the standard functional properties. 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. Fig. 4 shows the results for both verification and validation in a compact manner.

The V & V demonstrates that the SIPN in Fig. 3 is correct. Due to the fact that it gives visual feedback of the control flow, it is easy to apply and to implement, this correct SIPN serves as a basis for implementation.

For the realization there are used standard PLC programming languages according to IEC 61131 [12]. In order to guarantee a correct realization the generation of the PLC code has to preserve the dynamic behaviour of the SIPN.

5 Conclusion
The paper presents an approach to deliver correct and dependable PLC programs. Based on SIPNs, a variant of PNs, the complete controller development process from an informal specification to the final implementation on a PLC is discussed. This process includes the steps of design, verification, validation, implementation and realization. In order to illustrate the approach steps we used a working example.

In the PLC programming SIPNs represents a tool that is capable of graphically describing sequential and concurrent algorithms, gives visual feedback of the control flow, it is easy to apply and easy to implement, resulting in fast codes.

The SIPN formalized for the working example enabled the derivation of standard functional properties and the specific functional properties of the PLC control algorithm.

Standard functional properties were verified using the Cadence SMV tool. For this, the SIPN model has been translated into a SMV input file, in which were inserted the standard functional properties also expressed in Temporal Logic. 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.

The verified and validated SIPN according to corresponding working example model has been implemented in an IEC 61131 programming language. For the industrial realization of a controller standard PLC programming languages according to [2] are used. The properties of the SIPN can only be guaranteed for the implemented controller if the generation of PLC code from the SIPN preserves the dynamic behaviour of the latter.

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

This work was partially supported by a grant from the Romanian National University Research Council, PNCDI Program, ID-668/2008.

References: