AN INTEGRATED APPROACH FOR THE DEVELOPMENT OF LOGIC CONTROLLERS BASED ON SIGNAL INTERPRETED PETRI NETS

GEORG FREY
Kaiserslautern University of Technology, Juniorprofessorship Agentbased Automation, Erwin-Schrödinger-Str. 12, 67653 Kaiserslautern, Germany, frey@eit.uni-kl.de

Abstract. In this contribution an integrated design approach for logic control systems is presented. The approach is based on Signal Interpreted Petri Nets (SIPN), a Petri Net model extending condition event nets with means for the input and output of signals. The presented method starts from an informal description of the controller and proceeds over the steps design, formal verification and validation (V&V), evaluation of software quality, simulative validation, and automatic code generation for PLCs to an implemented controller. After implementation, the monitoring of the running controller by visualization of the token flow in the Petri net is possible.

Key Words. Design, Verification and Validation, Logic Control, DES, Petri Nets

1. INTRODUCTION

In today’s flexible, customer-oriented, automated production the creation of control software for the machines and plants is an essential time and cost factor. The increasing complexity of modern software and the rising user-defined safety and functionality requirements cause the urge for formal methods to develop the control algorithms. These allow not merely to test the algorithms but to prove their correctness. Furthermore, control algorithms often have to be adapted to new production scenarios. In most cases, the original developer does not perform this adaptation. Therefore, it is important, that a controller is well documented and easy to understand. Hence, a method is required that allows the evaluation of the quality of a controller in the light of understandability.

One way to achieve an improvement of the controller development process is the use of Petri Nets, e.g. [3]. There are two main Petri Net based approaches:

- Petri Nets as a modeling tool (design) and
- Petri Nets as an analyzing tool (V&V).

In the presented approach both ideas are combined and completed by a third, one:

- Petri Nets as a visual documentation tool (quality evaluation).

Based on these three ideas an integrated design approach for logic control systems is presented. The approach uses Signal Interpreted Petri Nets (SIPN), a Petri Net model extending condition event nets with means for the input and output of signals. The presented method starts from an informal description of the controller and proceeds over the steps design, formal verification and validation (V&V), evaluation of software quality, simulation, and automatic code generation for PLCs to an implemented controller. After implementation, the monitoring of the running controller by visualization of the token flow in the Petri net is possible.

While the simulation of the controller model together with a model of the process under control is intended to give the designer some visual feedback on the functionality of his solution and allows the test of temporal behavior, formal verification and validation (V&V) allows rigid proofing of functional properties. In the presented approach formal V&V is performed non-model-based or constraint-based [6]. This means that the properties are checked under no or minimal assumptions about the process under control. Thus possible faulty behavior of the process and instrumentation is included in the analysis resulting in a robust control algorithm.

A tool for the graphical editing of SIPN was developed. The tool, SIPN Editor, provides interfaces to several other tools via the generation of code in spe-
cial formats. These are the symbolic model verifier SMV for formal verification and validation based on symbolic model checking, the object oriented simulation environment Dymola for the simulation of the controller together with models of the process under control (discrete or continuous), and the IEC 61131 compliant PLC programming tool OpenPCS for the implementation and monitoring of the controller.

In the following section, the basic concepts of Signal Interpreted Petri Nets (SIPN) are presented. Section 3 describes the design process and Section 4 shows how the different steps in this process are integrated into the network of tools grouped around the SIPN Editor. Prior to a summary and an outlook on further work, Section 5 presents an example for the successful laboratory application of the SIPN approach.

2. SIGNAL INTERPRETED PETRI NETS

Signal Interpreted Petri Nets (SIPN) [6] are an extension of Condition Event Petri Nets, especially developed to provide a graphical language for the design of PLC control algorithms. They offer a means for the communication with the process under control. This is achieved via the handling of signals: Transitions are associated with firing conditions and places with output functions.

Firing conditions are Boolean functions over the input signals of the SIPN. An enabled transition (pre places of the transition marked and post places un-marked) fires immediately when its firing condition is fulfilled.

The output function of a place is activated while the corresponding place is marked. In general, the output function consists of a set of assignments to the output signals of the SIPN.

To handle real-world problems, the SIPN has been extended with timing and hierarchy concepts [5]. Arcs from places to transitions can be associated with a time delay. This delay represents the minimal time a token has to spend in the place before the transition is enabled. The hierarchy concept allows replacing an identified quite independent part of an SIPN by a single hierarchical place.

2.1. Example

To illustrate the concept, the controller of a heating tank realized with a hierarchical timed SIPN is shown in Fig. 1. The informal specification of the tank is given by four properties:

1. when the tank is empty and the Start button is pressed, the tank is filled,
2. if the temperature of the full tank is less than required, the liquid is heated until the required temperature is reached,
3. after waiting 20 seconds, the tank is emptied,
4. the contents are stirred during the whole process.

Let us show in details how this SIPN works: In place P1, the control algorithm is waiting for the start of the process and all output signals are set to 0. After it has been checked that the tank is empty ($i_1 = 1$) and the start button has been pressed ($i_4 = 1$), transition T1 fires. Place P1 becomes unmarked whereas P2, P3 and P3A get marked. Hence the stirring motor is set ON ($o_1 = 1$ in place P2) and the tank is being filled ($o_2 = 1$ in place P3A). Once the tank is full ($i_2 = 1$ in transition T3A and T3B), it is simultaneously checked whether the contents have the desired temperature or not. If they have not ($i_3 = 0$ in transition T3B), they are heated ($o_4 = 1$ in place P3B) until the desired temperature is reached ($i_3 = 1$ in transition T3C). Once this temperature has been reached (T3A or T3C fires), the contents are stirred during 20 seconds (timer on place P3C). After this delay, transition T3D fires and the tank is emptied ($o_3 = 1$ in place P3D). The marking of P2, P3 and P3D enables transition T2 which fires as soon as the tank is empty ($i_1 = 1$ in transition T2).

Fig. 1. SIPN for the heating tank.

3. INTEGRATED DESIGN APPROACH

3.1. Design

In general, the designer of a controller starts from an informal specification. This specification consists of a set of properties that can be sorted into three categories:
1. Problem independent functional properties (e.g. termination of the algorithm).
2. Problem specific functional properties (e.g. disjoint activation of two output signals or filling of a tank).
3. Non-functional properties (fulfillment of several quality criteria like use of comments in the code).

Additionally, there may be detailed information about the plant to be controlled.

The problem specific functional part of the informal specification of the control algorithm is formalized by the development of an SIPN. During this manual synthesis, the other parts of the specification form additional constraints for the designer. After the design, it has to be checked whether all properties are fulfilled. In the case of functional properties, this is done by the application of verification and validation methods.

3.2. Formal Verification & Validation
Let us remember the definition of verification and validation as it was given by Boehm [1] in 1979:

Verification: Am I building the product right?
Validation: Am I building the right product?

Following this definition, in verification it must be checked whether the designed SIPN is correct according to formal, problem independent criteria. These are properties, which have to be fulfilled by a controller regardless of its problem specific functions like for example determinism (absence of conflicts) or termination (absence of endless loops). For a list of formal correctness criteria see [6]. During the verification of the SIPN, it is checked whether the designed SIPN is unambiguous, i.e. that its behavior does not depend on the way it is implemented in the PLC. However, an SIPN that is verified to be formally correct does not necessarily fulfill the problem specific part of the specification. These properties are checked during formal validation of the SIPN.

To perform formal verification and validation, symbolic model checking is used. In this technique, a finite model of the system is built and the expected properties (specifications of the behavior) 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 [2]. A search procedure is then used to check whether the properties are fulfilled or not.

However, to use a model checker, the SIPN with its behavior has to be translated into input code of the model checker. Furthermore, the properties to be checked have to be translated into temporal logic, in a form readable by the model checker, finally the output of the model checker has to be interpreted in the SIPN. Therefore care is taken in the translation steps to keep an easy to follow correspondence between places in the SIPN and variables in the model checker code.

Formal verification and validation in the presented approach do not allow the analysis of temporal behavior. Therefore a simulation of the controller together with a model of the controller hardware and the process under control is proposed.

3.3. Simulative Validation
Simulative validation is an option whenever a model of the process under control is available or can be generated with reasonable effort. To apply simulative validation the SIPN Controller has to be translated into a form that could be used in a simulation tool. During this transformation, the temporal behavior of a PLC is added to the model. As simulation environment, the object oriented simulator Dymola is used. In addition to the interesting features of its simulator it is capable of interpreting models described in the modeling language Modelica [12]. This language provides a text-based interchange format for models and therefore could be used for the export of the controller model from the SIPN Editor and the input into Dymola.

3.4. Evaluation
For non-functional properties additional tests on the control algorithm have to be performed. In the case of SIPN, a set of metrics for the evaluation of quality criteria was developed. The evaluation of those metrics on the control algorithm gives a numerical value describing the fulfillment of the criteria. Based on this result, the user has to decide whether the controller is good enough or has to be improved.

3.5. Implementation
After the SIPN has been verified and validated, it can be translated into code in order to be executed on a PLC.

To use SIPN on a very large variety of PLC hardware, it is best to translate the resulting algorithm into one of the standardized PLC language as for example Instruction List (IL) [8, 9]. The direct implementation of an SIPN compiler would only work on a single PLC, whereas an IL according to the specifications of IEC 61131 standard can be executed on nearly every PLC.

Besides the differences in the specific Petri net type considered, the presented code generation differs from other approaches in one main aspect. This is the one-to-one correspondence of net elements to code segments that is used. This correspondence allows to easily re-interpret produced code. This possibility of re-interpretation is of special importance for the acceptance of the formal approach with experienced PLC programmers.

4. TOOL-SUPPORT

4.1. SIPN Editor

The idea of the SIPN Editor is to have a tool to support the new way of graphical programming of PLC controllers offered by SIPN. A Java application has been developed to achieve this goal. The SIPN Editor has been generated using DiaGen (Diagen Editor Generator), an environment for rapidly developing diagram editors from a formal specification of the diagram language based on hypergraph grammars and hypergraph transformation [7, 11]. The main task of the editor is translating a “drawing” which is supposed to be a correct diagram (an SIPN) into a semantic representation (an equivalent IL code). During this translation process, the editor checks the drawing for correctness and has to provide feedback to the editor user if the drawing contains errors.

![Fig. 2: SIPN Editor with Interfaces](image2)

4.2. Design

The editing tool consists of a drawing canvas which contains the SIPN diagram and the usual control widgets. A parser checks the syntactic correctness of the diagram and provides visual feedback by coloring of the erroneous component. A diagram layouter takes care of diagram beautification as of snapping to a grid and adjusts arrows when places and/or transitions are moved so that places, transitions and arrows remain connected. Using this SIPN Editor, it is possible to design and later implement hierarchical and timed SIPN.

![Fig. 3. Snapshot of the SIPN Editor](image3)

Let us now show how in detail the elements of an SIPN are handled in the SIPN Editor.

The definition of a place consists of a name (mandatory and unique), a comment and a piece of code (optional). The name given to a place is a piece of text using the characters allowed by the IEC 61131-3 standard [8]. This name will later be used during the code generations (SMV model checker, Modelica, and IL) as a Boolean variable. The comment is a piece of text that will be displayed near the place on the editor screen. The associated code represents the output function of the place and is written using IL. Since the editor does not check the syntax of this code, a formal analysis of the net should always be performed prior to the translation of the SIPN into IL code. Moreover, a place can be associated with a subnet. It is then drawn with a bold dashed line.

As for places, a transition must be associated with a name and, optionally, with a comment and a piece of IL code. For the name and the comment, the same remarks apply as for the places. The IL code associated with a transition represents its firing condition. If no code is associated to a transition, it is implicitly supposed to be true. As for the output function of a place, this code will not be evaluated during the drawing of the net and it should be checked using formal methods before the translation into IL code.

4.3. Formal Verification

An automatic code generator for the model-checker SMV [10] has been integrated in the SIPN Editor. Thus the translation of the SIPN into input code for SMV is transparent for the designer and the steps of formal verification and validation is made much easier. During the translation of the SIPN, the editor automatically generates the CTL (Computation Tree...
Logic) formulae [2] corresponding to the mandatory properties every SIPN has to verify.

4.4. Formal Validation
For the purpose of formal validation, a window allowing the input of CTL formulae has been integrated in the SMV menu of the SIPN Editor. In this TL Editor, properties and assumptions can be given. These will then be inserted in the SMV code during its generation.

4.5. Simulative Validation
To apply simulative Validation the SIPN-Editor generates a block for the simulation tool Dymola. This block represents the controller including the – simplified – dynamics of the controller hardware (PLC cycle with a fixed sample time as model parameter)

4.6. Implementation
The generation of PLC code is the primary goal of the SIPN Editor. In contrast to the SMV code generation where the SIPN is virtually flattened before the code is generated, the hierarchical structure remains unchanged during the generation of IL code. To achieve that, for each subnet a function block according to IEC 61131 is defined. Once again, this allows an easy reinterpretation of the generated IL.

4.7. Download and visualization
The IL code can be downloaded on a PLC or a Soft-PLC running on an IPC. Furthermore, a visualization process has been implemented to visualize the controller’s state within the editor and its SIPN while the controller is running. While the generated code could be implemented on all PLCs conforming to the IEC 61131 standard, the download and visualization features have to be programmed especially for each PLC programming tool. To demonstrate the feasibility, this was done for one tool: OpenPCS by infoteam software. During the generation of IL code, additional variables are generated to let the SIPN Editor known the current marking. In the visualization, these variables are used to shade the active places.

5. APPLICATION EXAMPLE
As an example, the model of a flexible manufacturing line (Fig. 4) is used. The 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).

To design the control algorithm of the manufacturing line, we study a complex operations scenario given as a timing chart. In this scenario, the three machines of the line are working concurrently.

Fig. 4. Flexible Manufacturing line.

To design the SIPN controller for the flexible manufacturing line, timed hierarchical SIPN are used. More precisely, the model consists of three hierarchical levels with six subnets.

The highest level (Fig. 5) is used to synchronize the three machines and their associated conveyors.

Fig. 5: Highest level of the hierarchical SIPN.

In the second level, the three machines are all described in a quite similar way. In the upper part, the machine waits for a new part and in the lower part, the part is machined. For example, the SIPN in Fig. 6 is the model of the second level for the vertical milling machine. In the third and last level, the actions on each machine are described in linear sequences. Fig. 7 illustrates that for the vertical milling machine.

The complete SIPN consists of three hierarchical levels, 55 places and 40 transitions.

Several rounds of verification and validation have been necessary before the presented SIPN was reached. However, after implementation only one problem in the controller remained: Sometimes one of the machines missed a work-piece. Analysis showed that this was due to timing problems. The signal indicating the arrival of a work-piece is an impulse shorter than the cycle time of the used PLC. In this case the signal can be missed by the controller if it arrives at the wrong time in the cycle. (A PLC reads its input signals, computes its algorithm, sets its output signals and then starts the cycle again. The change of an input signal is only recognized if it is present at the input reading step.) This error could have been avoided by using simulation prior to the implementation because the cyclic behavior of the PLC is automatically integrated into the simulation model.

Fig. 6. Sub-Net for the control of the vertical mill.

Fig. 7: Third level SIPN of the vertical mill.

6. CONCLUSIONS AND OUTLOOK

An approach for the graphical design, verification, validation and implementation of Signal Interpreted Petri Nets has been presented. The approach is supported by the tool SIPN Editor. This tool offers the designer a graphical editor to draw and visualize hierarchical and timed SIPN. After the design of the SIPN, it can be automatically translated into input code for the model checker SMV and into a simulation object for the simulator Dymola. Thus it can be made sure that the designed SIPN is correct prior to its implementation on a PLC. This last step is also supported by the editor with the automatic generation of IL code according to the IEC 61131-3 standard. This code can then be downloaded on a PLC. Furthermore, with the appropriate communication between PLC and PC, the marking of the SIPN can be visualized while the controller is executed.

Currently an extension of the SIPN Editor is under development that will calculate the SIPN reachability graph, and based on this will allow the automatic determination of the quality metrics.

Concerning the analysis via model checking, it is projected to visualize the results produced by SMV in the SIPN Editor. If a property is not satisfied, SMV gives a counter-example as a trace leaving from the initial marking to the faulty one. The goal is to simulate this trace in the SIPN editor.

7. REFERENCES

12. www.modelica.org

The SIPN Editor is free software and available under http://www2.informatik.uni-erlangen.de/DiaGen/SIPN/