Petri Net Based Specification in the Design of Logic Controllers with Exception Handling Mechanism

Michał Doligalski and Marian Adamski

Abstract—Hierarchical Petri nets beside UML state machine diagrams, sequential function charts (SFC) and hierarchical concurrent state machines are common solution for specification of logic controllers. These specification formats provide both concurrency and modeling on multi levels of abstraction (hierarchical approach). But only state machine diagrams support exceptions handling in direct way. Program model presented in form of state machine diagram may be later transformed into a program in the SFC language or transformed in the Petri Net and implemented in the FPGA structure. Similarity between SFC language and Petri Nets give us lot of tools for analysis such control system. Article presents new approach for exceptions handling in hierarchical Petri nets as formal specification for logic controllers. Proposed method of specification can be used independently or as a part of dual specification (correlated state machine diagram and hierarchical configurable Petri Net).

Keywords—Logic controller, dual specification, hierarchical Petri net, UML, state machine diagram.

I. INTRODUCTION

The application of reconfigurable logic controllers in the form of FPGA systems or PLCs is the most commonly used solution in the field of controllers development [1]. Commercially manufactured reprogrammable systems are much cheaper than the dedicated solutions – designed specially for an individual production line or a particular controlling process. Logic controllers can exist as autonomous industrial process control equipment or as a part of embedded system (SoPC). Example of the manufacturing process was shown in Fig. 1. The process task is to mix two liquid substances in the reactor tank. Two additional tanks (A and B) are used to measure proper quantities of substrates. Mixed product is poured into containers. There is six discrete sensors (x1, x2, ..., x6) responsible for level measuring, four momentary switches on the operator panel (start, resumption, defect, failure), six normally closed solenoid valves responsible for liquid flow control (y1, .., y4, y6), agitator relay (y5) and emergency reactor emptying valve (EV1).

The presented textual specification describes both behavior of the control system and structure of controlled industrial object. However, even the best textual specification is an informal specification that can be interpreted in different ways. While maintaining the diligence and compliance with the standard, formal methods provide unambiguous interpretation of the specification. The paper is focused on two formal behavioral method: Petri nets and UML state machine diagram.

Fig. 1. The manufacturing process.
II. INTERPRETED CONTROL PETRI NETS

Interpreted control Petri nets are widely used form of logic controller specification [2], [3]. Besides a clear description Petri nets provide wide range of formal method and algorithms for formal verification.

Interpreted control Petri Net can describe a binary control algorithm but only on one hierarchy level. This net can be extended in a simple way with the possibility of modeling at different abstraction levels through the introduction of macroplaces ($P_m$):

Hierarchical interpreted Petri Net is defined as:

$$PN_{CTRL} = \langle P, T, A, M_0, \mu, C, X, Y, \delta, \lambda, \theta \rangle,$$

where:

- $P$ is a finite non-empty set of places, $P \subseteq P_o$;
- $P_o$ is a finite non-empty set of net operational places, $P_o \subseteq P \cup P_m$;
- $P_s$ is a finite non-empty set of simple places, $P_s = \{ p_1, \ldots, p_m \}$;
- $P_m$ is a finite set of macroplaces, $P_m = \{ m_1, \ldots, m_l \}$;
- $T$ is a finite non-empty set of transitions, $T \subseteq T_o$;
- $T_o$ is a finite non-empty set of operational transitions, $T_o = \{ t_1, \ldots, t_n \}$;
- $A$ is a finite non-empty set of arcs, $A \subseteq A_o$;
- $A_o$ is a finite non-empty set of operational arcs, $A \subseteq A_{oPT} \cup A_{oTP}$;
- $A_{oPT}$ is a finite non-empty set of operational arcs, $P_o \rightarrow T_o$;
- $A_{oTP}$ is a finite non-empty set of operational arcs, $T_o \rightarrow P_o$, $A_{oTP} = (T_o \times P_o)$;
- $M_0$ is an initial net marking, $M_0 : P \rightarrow Z^+$;
- $\mu$ is a number of tokens in each place, $\mu : P \rightarrow K$, $K \in \mathbb{N}$.
- $X$ is a finite non-empty set of inputs, $X = \{ x_1, \ldots, x_p \}$;
- $Y$ is a finite non-empty set of outputs, $Y = \{ y_1, \ldots, y_k \}$;
- $C$ is a finite non-empty set of logic conditions in the Boolean algebra, $C = \{ c_1, \ldots, c_t \}$;
- $c_i$ is a function in the Boolean algebra:

$$C = \{ c_1, \ldots, c_t \}$$

$$\delta$$ is a function of transitions conditions defined as:

$$\delta : T \rightarrow C$$

$$\lambda$$ is a function of outputs in such a way that:

$$\lambda : P \rightarrow 2^Y$$

$$\theta$$ is a function of hierarchy in such a way that:

$$\theta : P_m \rightarrow PN_{CTRL}$$

and

$$a, b \in P_m, \forall a \neq b \Rightarrow \theta(a) \neq \theta(b).$$

The above definition was oriented towards the possibility of its configuration through distinguishing operational (simple) places and transitions – executing the control algorithm.

The behavioral description of logic controller was shown in Fig. 2. It represents only main control process and omits both kinds of exceptions. For interpreted Petri nets exceptions handling and resumption mechanism should be described indirectly by means of places and transitions (Fig. 3). Transitions (t9, ..., t16) are responsible for token expropriation after an exception, one transition for each place in net. These transitions have a higher priority than transitions of the control

![Fig. 2. Interpreted Petri net.](image-url)

![Fig. 3. Interpreted Petri net with exceptions handling.](image-url)
process \((t2, ..., t8)\). In order to achieve better transparency, it has been omitted from the diagram but each transition condition for \((t2, ..., t8)\) should be extended by the \(\text{defect}\) condition. Otherwise the non-deterministic transition firing will occur. Expropriated tokens are moved to synchronizing places \((p10, p11, p13, p14)\), one place for each parallel path. Tokens are merged by means of transitions \((t17, t19)\) then will exception handling process. Places \(p12, p16\) generate alarm signal and place \(p15\) perform emergency reactor emptying by means of \(e1f\) valve. Indirect exceptions handling implementation increase number of additional transitions and places. The presented net does not have a mechanism of resumption – to execute the mechanism it is necessary to include additional places and transitions which would allow to keep a token for the time of expropriation. This method is analogous to the one presented in dissertation [4]. Considerable increase in the number of places and transitions entails not only worse transparency of the diagram but also the increase in the use of the FPGA device hardware resources.

III. BEHAVIORAL SPECIFICATION BY MEANS OF UML STATE MACHINE DIAGRAM

The Unified Modeling language (\(UML\)) supports exceptions handling mechanism, that is strictly connected with object-oriented programming. In particular, the state machine diagrams allow state oriented behavioral specification with native exceptions handling mechanism. These diagrams are also suitable for logic controllers specification [5]. For given industrial process the UML state machine diagram was elaborated (Fig. 4). The state machine diagram consists of two levels of abstraction. Two composite states that distinguish submachines for which exception handling is performed. Composite states: \(CS2\) and \(CS3\) perform substrates measuring and mixing final product, respectively. According to textual specification two exceptions were also included. Placed shallow history states indicate that exceptions \(\text{defect}\), handled from \(CS2\) state is noncritical and after the resumption, the process will continue.

In proposed dual specification [6] of logic controllers, the UML state machine diagram is the user interface and the hierarchical configurable Petri net is used for formal verification and synthesis.

IV. HIERARCHICAL CONFIGURABLE PETRI NETS

Hierarchical configurable Petri Net is defined as follows:

\[
HCfgPN = \{PN_{CTR}, P_e, T_e, A_e\},
\]

where:

- \(P_e\) is a finite non-empty set of net configurable places, \(P_e = \{p^{\text{init}}, p^0, p^1\}, P \subseteq P_e \cup P_e; \)
- \(T_e\) is a finite non-empty set of net configurable transitions, \(T_e = \{t^{\text{init}}, t^0, t^1, t^w, t^{\text{fin}}\}, T \subseteq T_e \cup T_e; \)
- \(A_e\) is a finite non-empty set of net configurable arcs, \(A_e = \{(p^{\text{init}}, p^0), (p^0, t^0), (t^0, p^1), (p^1, t^w), (t^w, p^{\text{init}}), (t^{\text{fin}}, p^{\text{init}})\}, A \subseteq A_e \cup A_e; \)
- \(P_{\text{init}}\) is an initial net place; and
- \(M_0(p^{\text{init}}) = 1, \forall p \in P \setminus \{p^{\text{init}}\}, M_0(p) = 0; \)

- \(A_{\text{OPT}}\) is a finite non-empty set of arcs \(P_o \rightarrow \{T_o, T_{\text{fin}}\}, \)
- \(A_{\text{OPT}} = (P_o \times \{T_o, T_{\text{fin}}\}); \)
- \(A_{\text{OTP}}\) is a finite non-empty set of arcs \(\{T_o, T_{\text{init}}\} \rightarrow P_o, \)
- \(A_{\text{OTP}} = (\{T_o, T_{\text{init}}\} \times P_o); \)
- \(p^0\) is an active configuration place;
- \(t^{\text{init}}\) is the net initial transition with net activation value of \(\delta(t^{\text{init}}) = c^{\text{init}}, c^{\text{init}} \in C\);
- \(t^{\text{fin}}\) is the net final transition with net activation value of \(\delta(t^{\text{fin}}) = c^{\text{fin}}, c^{\text{fin}} \in C; \)
- \(t^0\) is an active configurable transition with reactivation value of \(\delta(t^0) = c^0, c^0 \in C; \)
- \(t^1\) is an idle configurable transition with the net deactivation value of \(\delta(t^1) = c^1, c^1 \in C; \)
- \(t^w\) is an expropriation final transition with the net expropriation value of \(\delta(t^w) = c^w, c^w \in C; \)

The name characterizing presented new class of Petri nets is quite complex and, this is why, this net class was named in short as hierarchical configurable Petri Net and was given the following acronym: \(HCfgPN\).

Configurable transitions \(t^{\text{init}}, t^{\text{init}}, t^0, t^w\) are fired when the following conditions for a specific transition are fulfilled at the same time:

- transition is enabled,
- condition of the transition execution assumes the following value 1.

The \(t^{\text{fin}}\) configurable transition is fired when the following conditions are fulfilled at the same time:

- transition is enabled,
- condition of the transition execution assumes the value of 1,
- execution condition of the \(t^w\) transition assumes the logic value of 0,
- execution condition of the \(t^1\) transition assumes the logic value of 0.

In case of the net which executes a resumption (reactivation) mechanism, an expropriation condition should be assigned to an idle transition. Firing of the \(t^1\) transition will cause running into an idle state: operational transition execution and output signal generation will be withheld. Resumption
of a net takes place after firing of the \( t^u \) active configurable transition to which the resumption condition of an idle net was assigned. An idle \( t^i \) configurable transition is fired when the following conditions are simultaneously fulfilled in respect of the transition:
- the \( t^i \) transition is enabled,
- execution condition of the \( t^i \) transition assumes the logic value of 1,
- execution condition of the \( t^w \) transition assumes the logic value of 0.

The \( t_n \in T_n \) operational transition is fired only when the following conditions are fulfilled simultaneously in respect of the transition:
- the \( t_n \) transition is enabled,
- execution condition of the \( t_n \) transition assumes the logic value of 1,
- execution condition of the \( t^i \) transition assumes the logic value of 0,
- execution condition of the \( t^w \) transition assumes the logic value of 0,
- the active configuration place \( p^a \) is marked.

In case of the interpreted control Petri Net an output function assumes output subsets to places. For example, in respect of both transitions:
- \( \gamma \) transition is given a higher execution priority. This is why, simultaneous assignment of conditions to the \( t^i \) and \( t^w \) transitions is possible. Simultaneous execution of these transitions is forbidden but at the same time the \( t^w \) expropriation transition is given a higher execution priority. This is why, it is permissible to specify the same execution conditions in respect of both transitions: \( \delta(t^w) = \delta(t^i) \).

The task of configurable places is to supervise the execution of the net operational block of the HCfgPN through blocking or permitting the transition execution and output signals generation. If the \( p^a \) configurable place is marked then the net is in active mode: transitions execution and output signals generation is permissible.

Execution of the \( t^i \) transition is the reason for the fact that an active configurable place loses a token in favor of an idle \( p^i \) configurable place. When the \( p^i \) place is marked then the net is in an idle mode (frozen mode). Despite the fact that operational places are marked transition execution in the operational subnet and output signals generation are withheld. Resumption of the net is executed through the \( t^w \) transition: token is transferred to the \( p^a \) place, execution of operational transitions and generation of a signal is possible again as the net is active.

The \( t^i, t^w, t^i\text{ in}, t^w\text{ in} \) transitions, as opposed to operational transitions and the \( t^w\text{ init} \) transitions, have a default logic condition which is ,,0” (false). Otherwise, the net would operate incorrectly and just after its activation it would be preempted.

V. HCfgPN Based Logic Controller Specification

In the dual specification hierarchy is maintained at every stage of the design process. The top level state machine and two composite states CS2 and CS2 were transformed into hierarchical configurable Petri nets: Net1, Net2 and Net3, respectively.

The Net1 (Fig. 5) contains two macroplaces:
- \( Mp2 \) corresponding to subnet Net2 (Fig. 6) with noncritical exception defect caught by T3 transition and handled by place \( p3 \).
- $M_{p3}$ corresponding to subnet $Net3$ (Fig. 7) with critical exception failure caught by $T7$ transition and handled by places $p4$, $p5$.

The logic condition "$0$ (false) have been assigned to $Net1$ transitions $T^a$, $T^i$, $T^w$ and $T^i ini$. At the highest level of abstraction there is no exception handling, listed transitions have been intentionally left in order to ensure coherence of HCfgPN diagrams.

Transitions $T1$ and $T2$ are treated as global variable [7] and were used as firing conditions for transitions $T^i ini$ for subnet 2 and 3. Transitions $T5$ and $T7$ also are treated as global variable and were used as firing conditions for transitions $T^a$ for subnet 2 and $T^w$ for subnet 3.

Firing of $Net2$ $T^i$ transition perform subnet freezing, as it was described in previous section. Transitions $T10$ and $T11$ become frozen and output signal generation from places $P6$, ..., $P9$ was disabled. Reactivation of $M_{p2}$ macroplace executed by the $T6$ transition firing, perform $Net2$ reactivation: transition $T6$ is a firing condition of $Net2$ $T^w$ transition.

Transition $T7$ is a firing condition of $Net3$ $T^w$ transition and is responsible for the $Net3$ expropriation. After $T^w$ firing all tokens from places $P10$, ..., $P17$ are removed (killed) and the subnet goes to the initial marking.

- $MP_{i}$ was treated as a global variable.

VI. SYNTHESIS, IMPLEMENTATION AND IN-CIRCUIT VERIFICATION

Subnets were described by means of Verilog HDL [8]: separated module (block) for each subnet. The logic controller top level diagram was presented in Fig. 8.

Global variables were specified as output signals. Synthesis and implementation steps were carried out using third party tools (Xilinx ISE). Implementation results were presented in Tbl. 1.

One-hot encoding results one flip-flop to one place assignment. Places $P_i$ from $Net1$ and $Net3$ were automatically omitted during synthesis because transitions $T^i$ from these subnets are impossible to be fired.

For simulation and in-circuit verification purpose stimulus generator was elaborated. The stimulus block was implemented using synthesizeable Verilog constructs. This makes
it useful not only during the simulation, as was the case in the
testbench blocks, but also during the in-circuit verification.
Stimulus generator connected with logic controller bloc was
presented in Fig. 9. The application of the same stimulus
generator during simulation and verification simplify failures
detection and speeds up the comparison of the results of both
processes.

Simulation results (Active-HDL) were presented in Fig. 10,
in particular subnet configuration is presented (as merged bus).
The scenario included both a critical and noncritical
exceptions handling. Simulation results (Fig. 10) are consistent
with prototype verification by means of Tektronix TLA 5204
logic analyzer (Fig. 11).

VII. CONCLUSION

In the paper approach to exceptions handling in hierarchical
configurable Petri nets based specification for logic controllers
was presented. Redefined semantic for new Petri net class was formally defined. Example show how handle critical and
noncritical exceptions.

Fig. 10. Simulation results.

REFERENCES

reprogrammable controllers,” in Design of embedded control systems,
1994.
digital complex system,” in Proceedings of the 2008 IEEE Computer
Society Annual Symposium on VLSI. Washington, DC, USA: IEEE
controllers using uml,” Ph.D. dissertation, University of Zielona Góra,
2010.
[5] F. Basile, P. Chiacchio, and D. Del Grosso, “A two-stage modelling archi-
tecture for distributed control of real-time industrial systems: Application
of uml and petri net,” Comput. Stand. Interfaces, vol. 31, pp. 528–538,
123–125, 2010.
in statecharts for fpga implementation,” in Design Test
DOI: 10.1109/EWDDS.2008.5580158.
of logic controllers,” Proceedings of SPIE : Photonics Applications in
Astronomy, Communications, Industry, and High-Energy Physics Exper-