Michał GROBELNY[1], Andrzej PIECZYŃSKI[2]

University of Zielona Góra, Media and Information Technology Department (1), Institute of Control and Computation Engineering (2)

Exception Handling in Logic Controller Design by means of UML Activity Diagrams and Control Interpreted Petri Nets

Abstract. *Modelling of all aspects of logic controller in design phase is very important. The paper presents handling of possible exceptions with usage of two specification techniques – UML activity diagrams and control interpreted Petri nets. Sample control process of transportation of friable goods is a base for further discussion about complexity of exception handling in each technique. Furthermore, the target is to model exceptions with usage of standard syntax of each technique.*

Streszczenie*. Szczególnie ważnym etapem procesu projektowania sterownika logicznego jest modelowanie poszczególnych jego perspektyw. Artykuł przedstawia możliwość reprezentacji obsługi wyjątków z wykorzystaniem dwóch technik modelowania – diagramów aktywności języka UML oraz interpretowanych sieci Petriego sterowania. Aspekty modelowania obsługi wyjątków dla obu wymienionych technik zostały zaprezentowane na przykładzie specyfikacji procesu sterowania transportem materiałów sypkich. Dodatkowo, celem jest zrealizowanie obsługi sytuacji wyjątkowych z wykorzystaniem tylko i wyłącznie standardowych elementów obu technik modelowania. (**Możliwość reprezentacji obsługi wyjątków z wykorzystaniem dwóch technik modelowania – diagramów aktywności języka UML oraz interpretowanych sieci Petriego sterowania**).*

Keywords: exception handling, UML activity diagrams, control interpreted Petri nets.
Słowa kluczowe: obsługa sytuacji wyjątkowych, diagramy aktywności UML, interpretowane sieci Petriego sterowania.

Introduction

The modelling is an important phase of whole logic controller design [4]. Structural and behavioural aspects of target devices are defined. Appropriate specification without errors (or rather with a minimized amount of them) is needed to achieve the quality goal. Moreover, specification which covers all aspects of designed solution is highly recommended. Therefore combining exception handling mechanisms in target behavioural specification of logic controller can play the key role in design reliability. In the paper exception handling mechanisms in control processes, both in UML activity diagrams [6] and Control Interpreted Petri Nets [2], are proposed and presented. Furthermore, usage of transformation [5] with proposed interpretation of exception handling mechanism in logic controller design with both specification techniques enables formal verification of complete system specification. Therefore, it increases the probability of full consistency of client requirements with the design.

The paper is organised as follows. Section 2 describes logic controller design process with impact on behavioural specification phase. Section 3 presents UML activity diagrams and the possibility of their implementation in discreet control design. Section 4 introduces exception handling mechanisms in UML activity diagrams basing on sample process of transportation of friable good. Control interpreted Petri nets and mechanisms of exception handling in this modelling technique is presented in section 5. Finally, section 6 concludes the paper.

Logic controller design

Logic controller design process [4] is usually realised in couple phases. First steps of this process are basic concepts usually saved using some sentences in natural language describing the idea. Next step is to formalize the notation and to give shape to the desired device. Formalization is realised with usage of various techniques. Behavioural specification is one of formalisms used during logic controller design, which can be realised with such techniques as Petri nets or UML diagrams. Both techniques have possibilities of exception handling mechanisms, but in both cases different results are reached.

UML activity diagrams

Originally UML [6] during its definition was dedicated to model software systems and their implementation in frames of software engineering. However, with the gain of importance, UML started to appear in different domains than previously desired. The fact influenced the UML popularity and in the same time the strength of its evolution with even faster spread of the technology in various new regions. In the same time, UML seems to be very efficient in information flow simplification. Moreover, the technology is easily readable and understandable by non-technicians, especially considering system design and its complete functionality. It is a motivation to use it during consultations with the client part.

Currently, UML is present even is such domains as business modelling, workflow description, production systems design or discrete systems specification. In the last area multiple diagram types are exploited for structural and behavioural description. Behavioural system specification is usually created using state machines, sequence diagrams and finally activity diagrams. The scope of the paper concentrates on hardware behavioural modelling acquired with activity diagrams of UML which describe system dynamic.

Object Management Group in UML Superstructure version 2.4.1 [6], dated august 2011, introduces activity modelling (in other words activity diagrams) as described below:

> "*Activity modeling emphasizes the sequence and conditions for coordinating lower-level behaviors, rather than which classifiers own those behaviors. These are commonly called control flow and object flow models. The actions coordinated by activity models can be initiated because other actions finish executing, because objects and data become available, or because events occur external to the flow.*"

In other words activity diagrams present consecutive sequence of actions and activities with decisions made considering external signals or events. This definition matches to the logic controller behavioural specification characteristics and thus, it can be successfully implemented in the domain.

However, in logic controller design not all elements and aspects of UML activity modelling are taken into account. Usually, object flow is not implemented due to the fact of signal oriented nature of the domain without object-oriented characteristics. In logic controller design input and output signals play the key role. Thus, specification of control

process by means of UML activity diagrams also concentrates on signals and these are conditions of actions or activities executions.

Formally, UML activity diagram dedicated to describe logic controller specification or control process can be defined as a 6-tuple:

$AD = (A, F, S, E, G, Z),$

where:

- A is a finite non-empty set of actions/activities;
- F is a finite non-empty set of flow relations between the actions and activities;
- S is a finite non-empty set of initial nodes;
- E is a finite non-empty set of final nodes;
- G is a finite set of guard conditions;
- Z is a finite set of output signals.

UML activity diagrams in discreet logic controller design usually involve one initial node and one final node. Final node is sometimes omitted, due to cyclic nature of control processes. Then, instead of a final node a final loop is specified, which restarts the process after finishing of one process execution.

Moreover, systems not always can be specified on a simple one-page diagram. Sometimes a system consists even of thousands of actions. Therefore, system design decomposition and hierarchical representation are important issues in such a large control system. This aspect enforces decomposition of complex designs into hierarchical structures concerning autonomous segments of the design. Decomposition increases readability and understanding of logic controller specification. On the other hand, when a reconfigurable logic controller (RLC) is considered, decomposition of specification can increase modularity of target implementation with the decrease of resource usage.

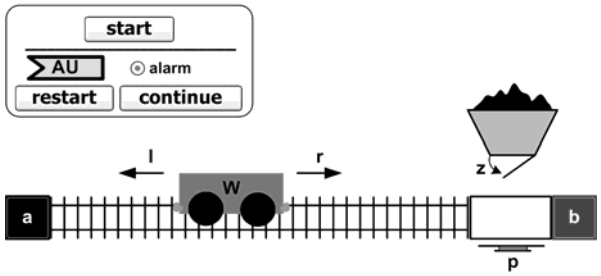

Fig.1. Real model of discussed transportation process

A process for transportation of friable goods [1] is a base to present exception handling mechanisms in logic controller design by means of UML activity diagrams and Petri nets. Real model of described process is presented in Fig. 1. Process is realized as follows. Initially, the carriage is in starting points (input signal *a*). The process is started after pressing the *start* button. Firstly, carriage is moving to the right (active output signal *r*). The movement is realised until the carriage reaches its destination point (input signal *b*). When the carriage reaches its own destination point, the chute is opened (active output signal *z*) and as the result friable goods are dropped into the carriage, until it is full (indicated by *p* input signal). Then the chute is closed (output signal *z* is deactivated) and the carriage begins to move left (active output signal *l*) until it reaches its staring point (input signal *a*). Input signals *AU*, *restart* and *continue* or output signal *alarm* are used to model handling exception mechanisms and will be discussed further in the paper.

Specification of simple process without exception handling mechanism is presented with usage of UML activity diagrams in Fig. 2. The specification is basing on target signals and it is a flat diagram, which means that it is realized without hierarchy.

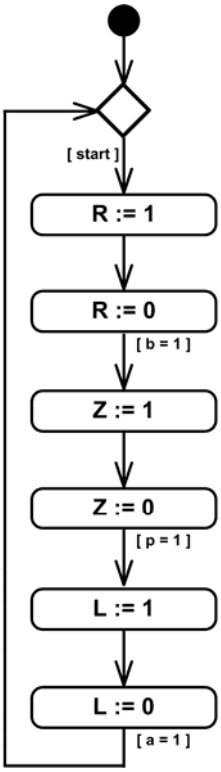

Fig.2. Real model of discussed transportation process

Exceptions in UML activity diagrams

Logic controllers are commonly used in many domains. Mainly controllers supervise industrial automation processes. Due to the fact, that logic controller is only supervising machinery systems, there is a possibility that unexpected scenario may happen. Discussed example with specification of transportation of goods depends on operational aspects of carriages and tracks. Possibilities of defects should be taken into account during specification preparation. Thus, exception handling mechanisms in logic controller specification by means of UML activity diagrams and Petri nets are presented. Exceptions and defects in discreet process are different to defects common in software engineering. Therefore, exception handling has to realise resumption process to ensure a proper functionality of controlled system.

The sample process (real model in Fig. 1 and UML activity diagram in Fig. 2) can be divided into three main functionality parts. The first part is realising movement to the right, the second is responsible for carriage filling and finally, the third controls movement to the left. Presented exception handling mechanism (Fig. 3) also covers these three parts separately. *InterruptibleActionRegion* [6] of UML is covering the guarded part of the diagram.

Indication of signal *AU* starts exception handling mechanism and stops normal control process execution. Presented handling mechanism guarantees resumption of control process and complete process restart. First possibility is used to continue the process whenever exception was not critical and the process can by realised further. The schema is achieved in Fig. 3 by pressing *continue* button. On the other hand, whenever process exception was critical and there is no possibility to execute it further, deep restart can be realized. The functionality is executed by pressing *restart* button. Pressing *continue* and *restart* buttons does not influence normal control process execution.

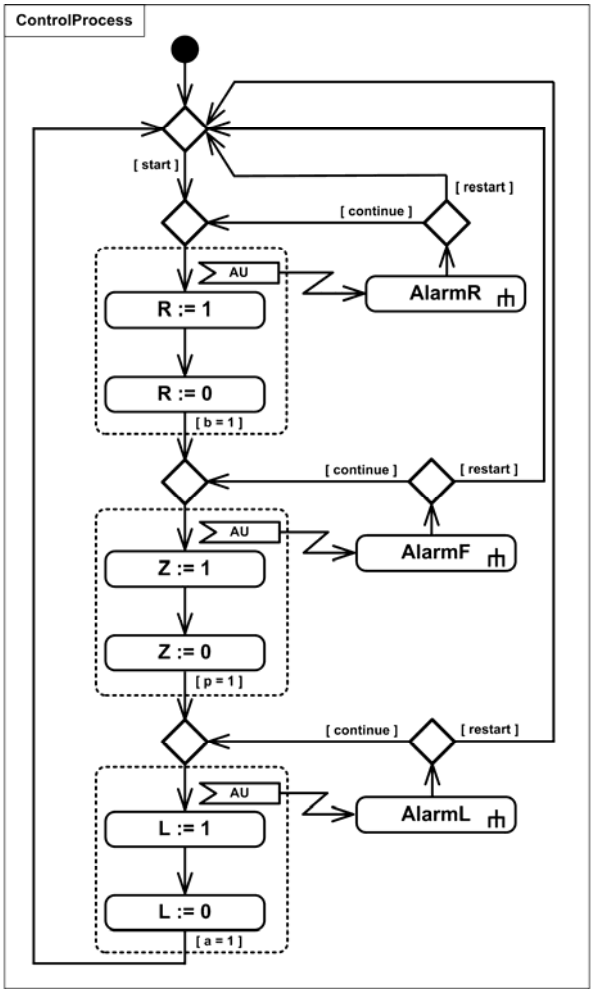

Fig.3. UML activity diagram with exception handling

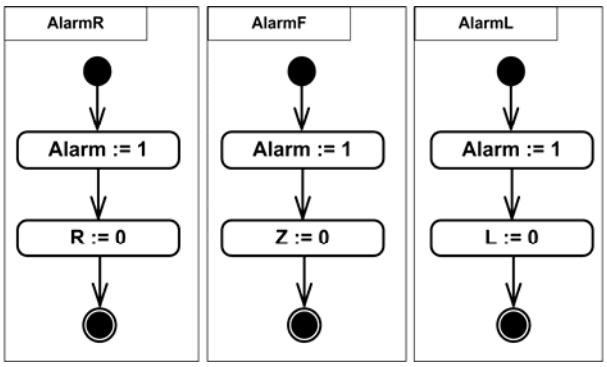

Fig.4. Exception handlers

Furthermore exception handling of every part of the system is specified separately. Every part has its own exception handler (*AlarmR*, *AlarmF* and *AlarmL*). All handlers are complex and their implementation is shown in Fig. 4. Moreover, exception handler is responsible for *alarm* signal activation to inform about abnormal process functionality and also for current process signals deactivation (e.g. handler *AlarmR* first to the right in Fig. 4 deactivates control signal *R* realizing carriage movement to the right).

Realisation with usage of control interpreted Petri nets

UML activity diagrams are one of possible specification techniques available to define logic controller behavioural properties. Petri nets [2], available for over 50 years, are mathematical model to describe systems built of states and transitions. Control interpreted Petri nets [1, 2] are one from various modifications of Petri nets dedicated to specify control processes.

Formally, a control interpreted Petri net can be defined as a 8 tuple:

$$PNIO = (P, T, F, X, Y, \rho, \lambda, \gamma),$$

where:

P is a finite and non-zero set of places;
T is a finite and non-zero set of transitions;
F is a finite and non-zero set of flows between elements of P and T or elements of T and P (it is forbidden to create connections between two elements of the same type e.g. T and T or P and P);
X is a finite and non-zero set of input states;
Y is a finite and non-zero set of output states;
ρ is a function $T \rightarrow 2^X$, that each transition assigns the subset of input states $X(T)$; 2^X states for the set of all possible subsets of X;
λ is a function $M \rightarrow Y$ of Moore outputs, that each marking M assigns the subset of output states $Y(M)$;
γ $(M \times X) \rightarrow Y$ is a function of Mealy outputs, that each marking M and input states X assigns the subset of output states Y.

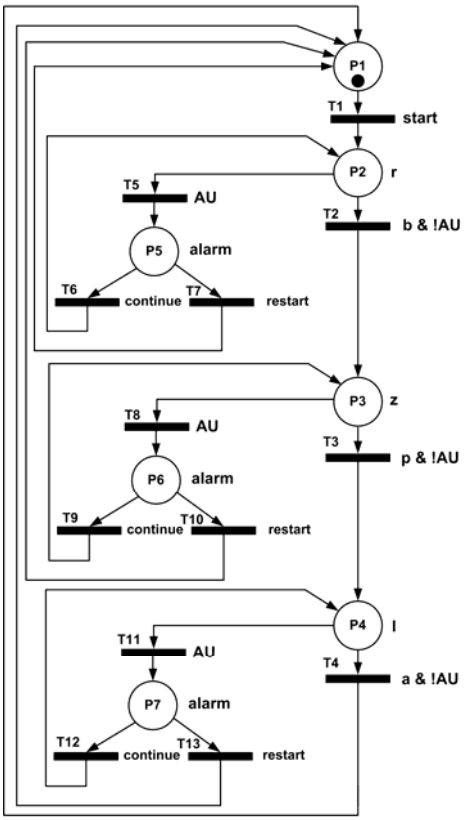

Fig.5. Control interpreted Petri net with exception handling

Control interpreted Petri nets, in contrast to UML activity diagrams, do not have special syntax elements dedicated to realise exception handling mechanisms. Designer must manually extend the already existing specification with usage of extra places and transitions to present exception handling. Another disadvantage is no possibility of interruptible region marking. There has to be an extra flow added to every place of the net with possible exception. This fact makes Petri net specification less readable and may sometimes lead to specification faults. Therefore exception handling mechanism in Petri nets is hardly realisable.

In Fig. 5 sample control interpreted Petri net with exception handling corresponding to UML activity diagram from Fig.3 is presented. Whenever *AU* signal is activated, place dedicated handling transition (*T5, T8 and T11)* becomes active and token is removed from control part place (*P2, P3 and P4)* and added in handler place (*P5, P6, P7*). Due to control interpreted Petri net specification, output signals are assigned to places, therefore whenever token is removed from a place – output signal becomes inactive. Moreover, whenever token is placed in handler place of the net, *alarm* signal is turned on. Resumption of the process in every handler is realised with usage of two transitions. First, with guard *continue* (transitions *T6, T9, T12* for adequate handler) which is responsible for simple continuation of interrupted process. Other transitions (*restart* guard, transitions *T7, T10, T13*) are responsible for whole control system reset and its return into initial state.

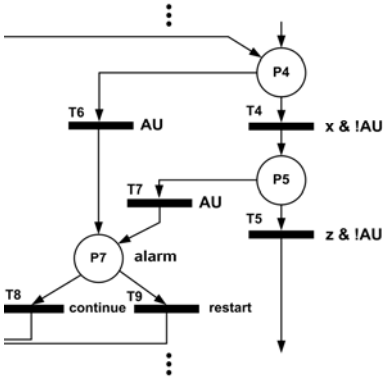

Fig.6. Control interpreted Petri net with multiple places exception handling

Furthermore, whenever exception handling is considered, there is no possibility to cover graphically a part of process with one exception handler. In UML activity diagrams syntax, a part of control process is covered with *InterruptibleActionRegion* and one flow directs to exception handler. In control interpreted Petri nets, there has to be a connection (transitions with flows) between every place in the covered part with one dedicated handler. Fig.6 presents covering of two places (*P4, P5*) with one dedicated exception handling place. There has to be a dedicated exception handling transition connected with every covered place to realise exception handling mechanism. Transition *T6* is realising exception interruption of place *P4* and transition *T7* is realising the same mechanism of place *P5*. In very complex control systems such multiplication of handling transitions can cause significant growth of the net. Therefore, exception handling mechanism in control interpreted Petri net is much less efficient than it is in UML activity diagrams based specifications. Thus, some combination of both techniques with usage of transformation [5] may be reasonable.

There is also a new interpretation of Petri net proposed in [3], which has exception handling possibilities referring to the UML techniques. However, the main disadvantage of proposed solution is the fact, that the new interpretation creates a new syntax of Petri nets.

Conclusion

Exception handling mechanisms in control process behavioural specification play an important role fulfilling design process. Implementation of such mechanism is presented in the paper with usage of UML activity diagrams and control interpreted Petri nets. Presented mechanisms cover situations when after defect removal normal further continuation of process is possible. Moreover, critical situations are also covered and complete system restart is then realized.

In the paper two hardware behavioural specification techniques are considered. UML activity diagrams have dedicated exception handling elements and therefore handling mechanisms are realisable without meaningful increase of the diagram size. On the other hand, Petri nets are supported with enormous set of analysis, optimisation and verification tools and techniques. Authors propose the interpretation of exception handling mechanisms in logic controller design by means of UML activity diagrams and control interpreted Petri nets dedicated to their common use with help of transformation technique [5]. Combined usage of both Petri nets and UML activity diagrams enriches design possibilities and may enhance target product quality. Transformation method between UML activity diagrams and Petri nets [5] may be useful to connect both techniques in one design engine.

Acknowledgments

Michał Grobelny is a scholar within Sub-measure 8.2.2 Regional Innovation Strategies, Measure 8.2 Transfer of knowledge, Priority VIII Regional human resources for the economy Human Capital Operational Programme co-financed by European Social Fund and state budget.

REFERENCES

[1] Adamski, M. and Chodań, M. Modelling of discreet control circuits using SFC nets. Politechnika Zielonogórska Press, 2000.
[2] David, R. and Alla, H. Discrete, Continuous, and Hybrid Petri Nets, Springer Verlag, 2010.
[3] Doligalski, M. and Adamski, M. (2010). Exceptions and deep history state handling using dual specification. Electrotechnical Review, 86 (2010), nr 9, pp. 123 – 125 (in Polish).
[4] Gomes, L. and Barros, J.P. and Costa, A. Modeling formalisms for embedded system design, Embedded Systems Handbook, Taylor & Francis Group, LLC, 2006.
[5] Grobelna, I. and Grobelny, M. and Adamski, M. Petri Nets and activity diagrams in logic controller specification – transformation and verification, In Proceedings of the 17th International Conference Mixed Design of Integrated Circuits and Systems, 2010, pp. 607 – 612.
[6] Object Management Group. OMG Unified Modeling Language, Superstructure, Version 2.4.1, 2011.

***Authors**: mgr inż. Michał Grobelny, University of Zielona Góra, Media and Information Technology Department, Al. Wojska Polskiego 69, 65-762 Zielona Góra, E-mail: M.Grobelny@kmti.uz.zgora.pl; dr hab. inż. Andrzej Pieczyński, Prof. UZ, University of Zielona Góra, Institute of Control & Computation Engineering, ul. Ogrodowa 3b, 65-462 Zielona Góra, E-mail: A.Pieczynski@issi.uz.zgora.pl.*