From UML Object Behavior Description into Petri Net Models

Towards Systematic Development of Embedded Systems

Wojciech Szmuc, and Tomasz Szmuc

***Abstract*—The paper describes a translation of object behavior specified by UML into semantically equivalent Petri net models. The translation focuses on object behavior with event handling implemented in UML. The resulted Petri net allows to check UML model properties not only by simulation but also in formal way. For possibly closest congruence between UML and Petri net model an event queue is defined. Each state machine assigned to an object has its own event queue which is available as long as the machine exists. It allows modeling not just a simple message passing but also cases, when state machine cannot handle an event. A higher priority of sub-machine's event queue is also considered. The presented solution is a part of wider conversion algorithm from UML model into Petri nets [12]. However, the paper is intended to describe the issue in such a detailed way and focuses on aspects crucial for embedded systems development.**

***Index Terms*—UML, state machine, event, queue, object, Petri, formal, model**

I. Introduction

EMBEDDED systems are built as a set of cooperating concurrent processes (threads, tasks), interacting deeply with their environment by reaction on events, reading data and sending computed control values. Correctness requirements (of such system) focus not only on functional and time compliance of response, but also deal with dependability of running system in changing environment and possible failures of hardware devices installed in the system or its environment. High degree of concurrency makes system more flexible, but increases fuzziness of relation between input demand (e.g. for service of input signals) and response by the system. This feature makes difficult verification and validation (V&V) of embedded systems and significantly increases development costs. Therefore research aiming at improvement of V&V and making the process systematic and more efficient are of great interest. A role of formal methods in the improvement is crucial for development safety embedded systems, see [11] for details.

UML is one of the most widely spread instruments for system modeling. Although, it has many advantages such as different modeling perspectives, there is one important drawback – the standard [2, 6] does not define a formal model, so semantics of UML artifacts may be sometimes interpreted in many ways. Therefore, many conversion algorithms were proposed to build formal model representing some UML diagrams [12]. Usually the main focus is set to visible part of diagrams conversion and does not take into account aspects influencing system behavior but not visible on diagrams [5, 10]. Object behavior and event handling are crucial aspects in development of embedded systems. Analysis of correct system/component response on events is a challenge in embedded system V&V. Formal modeling enables proving required properties making the verification systematic and therefore more efficient and solvable (esp. in the case of complicated multi-processes systems). The paper focuses on conversion of the two main concepts state machine describing class/object behavior and event handling specified communication.

Colored Petri nets [7] have been chosen for modeling translated constructs. The formalism provides several constructs (coloring, hierarchical description) allowing modeling of complicated structures. Moreover, many modeling/analysis tools exist. For visualization purposes CPN Tool 4.0 was used to build and verify proprieties of proposed solutions. Several attempts of translation from UML into Petri nets have been carried out. Some representative examples may be found in [2-5, 8-10]. More detailed analysis is given in [12]. The main advantage of the proposed solution is complete approach for the transformation, taking into account wide spectrum of UML constructs. This scope makes possible translation of UML artifacts defined during analysis and design phases of development process. The paper focuses on behavioral perspective in the context of objects, the main challenge in embedded systems design and verification. The contents is modified and extended version of the paper [13]. State machine constructs are added to the domain and their integration with event handling has been carried out.

II. Class and Object Representation

Events handling influences state machines assigned to objects. Therefore, to understand the concept class representation is needed to be described. Fig 1. depicts Petri net for object modeling. It is composed of constructor, object counter, destructor and a place, where object is stored for processing.

W. Szmuc and T. Szmuc are with the Department of Applied Computer Science, AGH University of Science and Technology, Kraków, Poland (e-mails: wszmuc@agh.edu.pl, tsz@agh.edu.pl)

ISSN 2080-8755 *Copyright © 2016 by Department of Microelectronics & Computer Science, Lodz University of Technology*

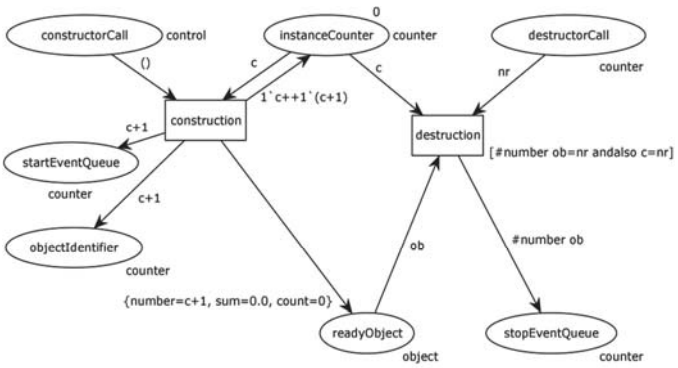

Fig. 1. Class representation in Petri net

Constructor (transition "construction") is started by putting a token in place "constructorCall". This increases value of counter of objects ("1`c++1`(c+1)" to instanceCounter), activates events queue ("startEventQueue") and creates new object which stores its number and attributes. Additionally, a number of created object is returned ("objectIdentifier") to assign the object to proper user.

When in place "readyObject", object could start state machine execution.

Call of destructor needs, as an argument, object number to be destroyed. "destruction" takes object token, stops event queue by sending number of object which is destroyed and decreases number of instances ("c" from "instanceCounter"). The object that is destroyed has to be the one, that destructor was called to ("[#number ob=nr andalso c=nr]").

Definition of the net requires following declarations:

```
colset float=real;
colset counter=int;
var c, nr: counter;
colset object=record number: counter*sum: float*count: counter;
var ob: object;
```

III. Translation of State Machine Diagram

State machine diagram describes behavior providing together with class diagram modeling of whole artifact. Considering the modeling scope, state machine is assigned to class of which behavior it specifies. This paragraph describes translation of most commonly used elements of state machine diagram. Other, more sporadic constructs are described in [12].

The main elements in the state machine are state and transition. State is represented in Petri nets by place. Since states with the same names are considered as one when translating a fusion should be assigned to them. Transition which provides control and data flow between states is converted as transition between places.

However, due to the fact the behavior of transition in UML could be more complex several transitions (with separating places) may need to be used. In the state diagram the transition could be fired by incoming event (described in next paragraph) or guard. The guard allows to transmit data or control if its expression evaluates to true. The same requirements should meet guard in Petri nets thus translation should just pay attention on syntax (e.g. inequality in UML is designated as "!=" whilst in Petri nets as "<>").

Transition provides also data processing functionality, which could be realized with Petri nets in two steps. Firstly, in the arc inscription of incoming (to the transition) arc the value should be assigned to the variable. Secondly, in the inscription of outgoing arc expressions could be evaluated on variables assigning value to returned token.

The guard concept could be used in more expanded construct called decision. A simple example is presented in Fig. 2. It depicts a part of control algorithm for heating and air conditioning system. The first symbol is responsible for calculation of the difference between the set and the measured temperature. Next symbols – guards split the control flow to 3 cases: heating, cooling and measuring. The last mode (with "else") is chosen when the measured temperature is within the limits.

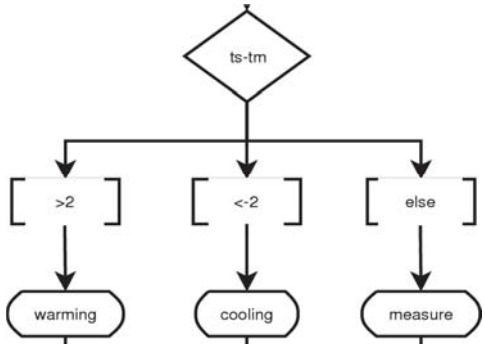

Fig. 2. Decision representation in UML

The corresponding Petri net is presented in Fig. 3. It starts from calculation of the difference between temperature values ("ts-tm") in arc inscription incoming to the place "result". The colset of the place is of integer type. From the place 3 transitions could take a token. First with guard "[td>2]" which is fired, when measured temperature ("tm") is lesser by more than 2 from set temperature ("ts"). The next transition with guard "[td< ~2]" is responsible for cooling. And the last one represents "else" branch in decision construct. This place does not have any guard however, its priority is set to "P_LOW". This guaranties deterministic behavior of the proposed net.

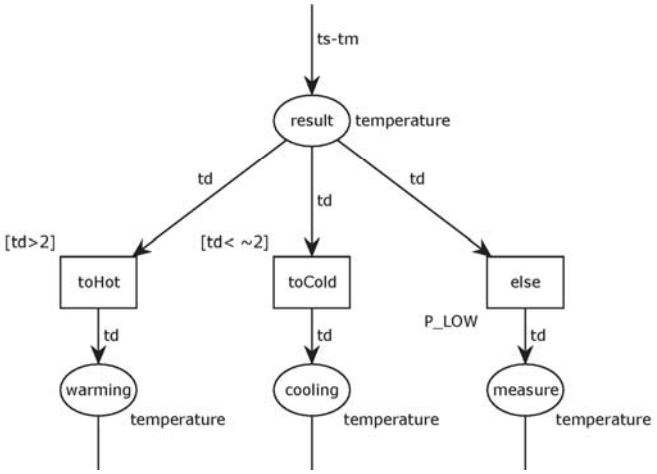

Fig. 3. Decision representation in Petri net

IV. Simple Event Handling Constructs

The following description focuses only on most important aspects of constructs. Object identification is not described (works like in paragraph II) despite it appears in the net.

A. Event queue

Relevant color set has been defined in order to handle different events. String type defined below is sufficient for simple cases:

```
colset events=string;
var evs: events;
```

Event queue is stored in a token defined as list:

```
colset elist: list events;
var l: elist;
colset evlist=record nrlist: elist*number: counter;
```

Petri net for event queue is depicted in Fig. 4. Place "constructor" is connected either as port place in substitution transition or by fusion with net representing class. Token in this place allows queue to be activated. Transition "activate" sends one token to place "isActive" and another to place "queue". After this step place "queue" contains token with empty list ("[]").

Place "destructor" is also connected with net representing class. This place is used to deactivate event queue. If a token is present in this place a "deactivate" transition could be fired. This transition takes token from place "isActive" and "queue".

Proper ratio in putting tokens in places "constructor" and "destructor" is provided by the net of class.

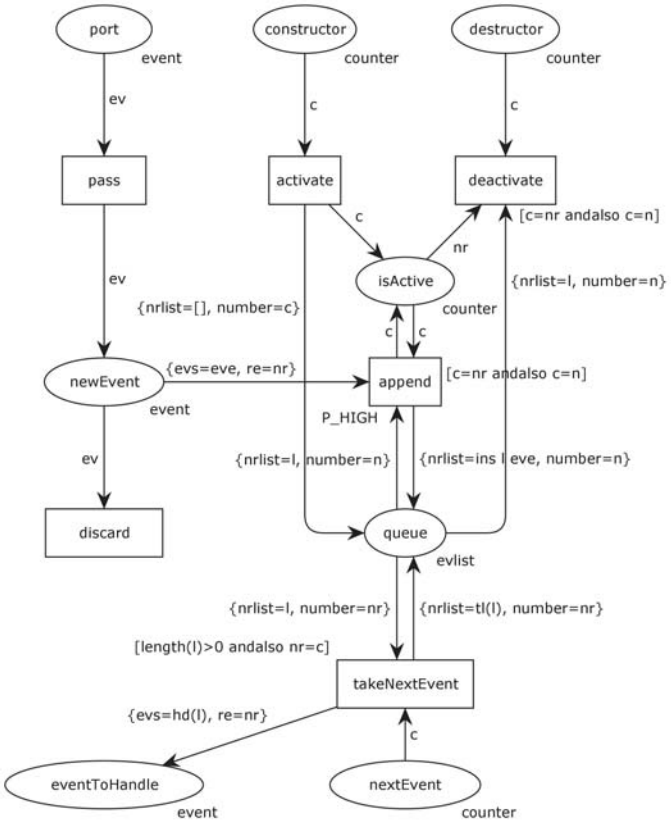

Fig. 4. Event queue model in Petri net

When event queue is active an event could be received. New event is represented as token in place "port". It is then passed by transition "pass" to place "newEvent". This part of net is prepared to assign restrictions (as guard of transition) to events that could be received by certain event queue. After that step transition "append" could be fired. It takes a token from place "isActive" and "queue". New event is appended at the end of the list ("ins l eve"). If there is no token in place "isActive", transition "discard" is fired. To prevent non deterministic behavior when event queue is active, transition "append" has priority set to "P_HIGH".

When receiver is ready to handle event, it should put token in place "nextEvent". Transition "takeNextEvent" takes list of events from place "queue" and passes first event ("hd(l)") to place "eventToHandle" if list is not empty ("length(l)>0"). List without previously first element ("tl(l)") is returned to place "queue".

B. Receive construct

To receive an event receiver has to notify receive construct. Since this part is shared there is a need to distinguish who is waiting for the event. Therefore, the following form of addressing is defined:

```
colset receiver=int;
var reci: receiver;
colset receiverr=record re: counter*number: counter;
colset event=record ev: events*re: receiver;
colset eventr=record evs: events*re: receiver*number: counter;
```

Petri net for event queue is depicted in Fig. 5. Place "receive" is connected with net representing behavior (state machine) which accepts certain event. Token in this place is a record containing information about id of receiver and desired event. Place "eventToHandle" is connected with place, with the same name in event queue. When a new event occurs it could be handled by one of two transitions. "discarding" transition takes token form places "receive" and "eventToHandle" and compares desired event with released by event queue. The transition could be fired if received signal is not the desired one ("eve<>evt"). "receiving" transition is fired, when provided event is the same as desired. This transition sends id of receiver to place "received". The place is connected to net representing behavior. Since this place is shared, the proper part of behavior is identified by receiver id.

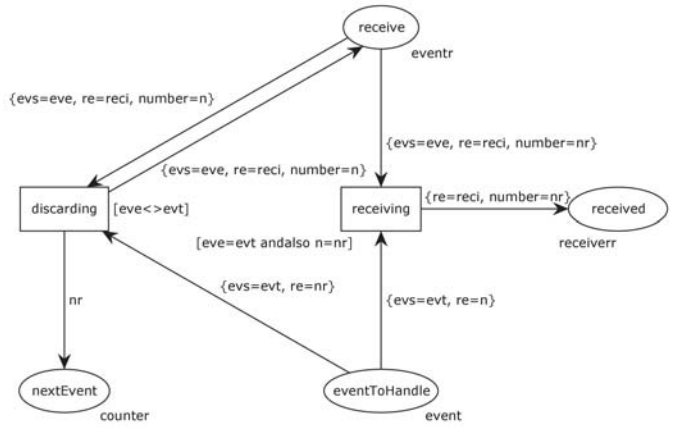

Fig. 5. Petri net for receive construct

This solution has been chosen in order to ensure proper event passing in the case when there is more than one receiver which could handle it.

C. Save construct

Save construct is used to prevent discarding event. It is used, when behavior is not currently in the state that could handle the event but is supposed to do it later. This construct should be put between place, where event is released from event queue and corresponding place in receive construct.

Petri net for save construct is depicted in Fig. 6. Place "save" is connected with net representing behavior. Token in this place defines event, which should be saved. "eventToHandle" is connected with event queue. When this place will contain event to save, transition "saving" will be fired. This transition sends event to "port" what results in appending event to the end of event queue. The transition also allows event queue to release subsequent event ("nextEvent"). If there is no need to save event, "noSaving" transition is fired. To prevent non deterministic behavior when saving an event, transition "saving" has priority set to "P_HIGH". Place "eventToHandleS" should be connected with place "eventToHandle" in receive construct.

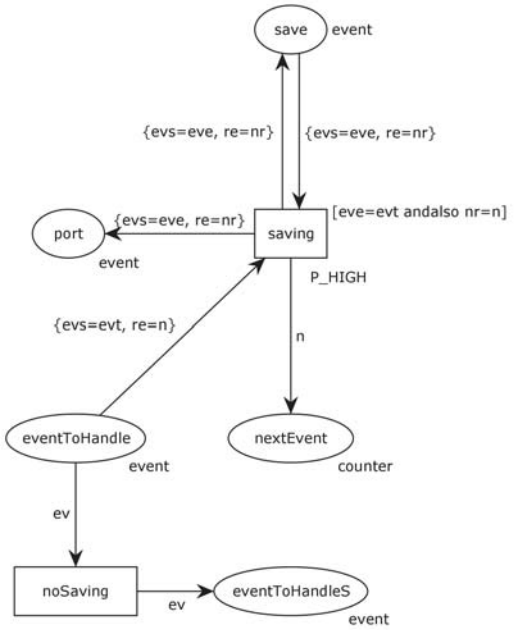

Fig. 6. Save construct in Petri net

V. More Complex Cases

The proposed event handling solutions could be used also for modeling timeouts. To achieve this feature definition of color set "timed" should be added:

colset events=sting timed;

Example arc inscription (to "port") to send timeout event for 10 time units:

"timeout"@+10

Events could have parameters what implies change in color set definition. Example definition for two parameter events:

colset nm=string;
colset tp=int;
colset events=record nm: str*frst: tp*scnd: tp;

Since it is not possible to assign more than one color set to a place, destined type used in signal handling should be defined to allow passing each of available events.

In UML it is possible to address events to certain objects. This could be achieved in Petri nets by adding another component to record representing address. To assure correct event routing additional restrictions may need to be added to transition "pass" in event queue.

When defining composite states place "eventToHandle" of event queue should be connected with place "eventToHandle" in sub-machine receive construct. If event is discarded it should be passed to upper-level machine receive construct. It could be carried out by adding place "discarded" in sub-machines receive construct and connecting with place "eventToHandle" in upper-level machine. This assures higher priority of receiving events in composite state then in upper-level machine.

Event queue should be activated before any other operations performed by constructor. This requirement should be fulfilled by net representing class.

VI. Summary

The described solution allows transformation of UML architecture (class diagram) and behavior (state machine diagram) with event handling model into equivalent Petri net. Presented constructs i.e. class, state machine, event queue, receive and save were tested using available in CPN Tools methods. Starting from simple constructs general concept has been proposed, and then more detailed models has been refined. The elaborated constructs are building blocks of algorithm for automatic translation from UML into equivalent Petri nets. The obtained formal model may be used to prove required properties of system with event handling feature. Proving properties may be carried out directly using CPN Tools, as well as in indirect way, i.e. taking the generated coverability tree as a model for proving properties described by other formalism, e.g. Temporal Logic. The transformation together with others proposed in [12] may be used for to construct parallel path in modeling of artifacts during development using UML language. The additional formal path allows to prove properties of the formal models [11] which are equivalent to the UML artifacts. The approach will make the verification process more systematic and complete supporting development of correct embedded/real-time systems. This is especially important for development of safety related systems, including intelligent/autonomous cars being currently a promising area in research and applications.

References

[1] Bauskar B. E. Mikolajczak B.: *Abstract Node Method for Integration of Object Oriented Design with Colored Petri Nets*, Information Technology: New Generations, 2006. ITNG 2006. Third International Conference on Year: 2006 Pages: 680 – 687

[2] Baresi L., Pezze M.: *On Formalizing UML with High-Level Petri Nets.* Concurrent Object-Oriented Programming and Petri Nets. Lecture Notes in Computer. Springer Berlin Heidelberg, 2001 Pages: 276-304

[3] Bernardi S., Donatelli S., Mersehuer J.: *From UML Sequence Diagrams and Statecharts to Analyzable Petri Nets Models.* Proceedings of the 3rd International Workshop on Software and Performance. WOPS'02. ACM, 2002 Pages: 35-45

[4] Dennis A., Wixom B., Tegarden D.: Systems Analysis and Design with UML. John Wiley & Sons, 2012

[5] Feng X. Liu Q. Wang Z.: *AUV Modeling and Analysis using a Colored Object-Oriented Petri Net,* Computer and Computational Sciences, 2006. IMSCCS '06. First International Multi-Symposiums on Year: 2006, Volume: 2 Pages: 405 – 409

[6] http://www.omg.org/spec/UML/2.5

[7] Jensen K., Kristensen L.: *Coloured Petri nets. Modelling and Validation of Concurrent Systems*. Springer, Heidelberg, 2009

[8] Kerkouche E., Chaoui A., Bourenane E. B., Labbani O.: *A UML and Colored Petri Nets Integrated Modeling and Analysis Approach using Graph Transformation.* Journal of Object Technology. Vol. 9. No 4, 2010, Pages: 25-43

[9] El Miloudi K., El Amrani Y., Ettouhami A.: *An Automated Translation of UML Class Diagram into a Formal Specification to Detect UML Inconsistency.* Proceedings of the 6th International Conference on Software Engineering Advances. ICSEA. 2011 Pages: 432-438

[10] Mukhin D., Mikolajczak B.: *A Method of Concurrent Object-Oriented Design Using High-Level Petri Nets,* Systems, Man, and Cybernetics, 1998. 1998 IEEE International Conference on Year: 1998, Pages: 295 - 300 vol.1

[11] Szmuc T., Szpyrka M.: *Formal Methods – Support or Scientific Decoration in Software Development?*, Proceedings of the 22nd International Conference Mixed Design of Integrated Circuits and Systems, June 25-27 Toruń, Poland, 2015 Pages: 24-31

[12] Szmuc W.: *Modelling of Selected UML 2.0 Diagrams with Coloured Petri Nets*. PhD Report. Supervisor Szpyrka M., AGH 2014

[13] Szmuc W., Szmuc T.: *Modeling of UML Object Event Handling with Petri Nets. Towards improvement of embedded systems analysis and design.* Proceedings of the 23rd International Conference Mixed Design of Integrated Circuits and Systems. June 23-25, 2016 Pages: 454-457

Tomasz Szmuc received MSc in Electrical and Control Engineering from the AGH University of Science and Technology (AGH) in 1972. Employed since the beginning at AGH University of Science and Technology, where received Ph.D. (1979) and Sc.D. (1989) degrees (both in Computer Science), and Professor title (1999). The research focuses on Software Engineering, in particular applications of formal methods (Petri Nets, Temporal Logics, Process Algebras) for modelling and support of software development. Development of real-time systems and embedded systems constitute the main stream in application related research. He is author or co-author of 10 books and more than 180 articles mainly related the specified above Software Engineering and application categories. He was supervisor of 15 Ph.D. thesis. He is a member of IEEE Computer Society, ACM, Computer Science Committee of PAN, and 2 Scientific Committees of PAU and many other scientific committees.

Wojciech Szmuc received MSc in Automatics and Robotics (Computer Science in Control and Management) from the AGH University of Science and Technology (AGH) in 2001. Employed since 2006 at AGH University of Science and Technology, where received Ph.D. (2015) in Computer Science. The research focuses on formal methods (Petri Nets, Temporal Logics, Process Algebras) in application of Software Engineering modelling and support of software development. Although, the main concern are real-time systems and embedded systems some research was made also in Digital Watermarking and Steganography. This work resulted in achieving US patent as one of co-authors.