ijecs open access www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 2 Issue 3 March 2013 Page No. 798-805

TRANSFORMATION OF UML ACTIVITY DIAGRAMS INTO PETRI NETS FOR VERIFICATION PURPOSES

Bhawana Agarwal

M.Tech Scholar (CSE), Mewar University, Chittorgarh, India
Email:agarwalbhawana06@gmail.com

Abstract:

In the software engineering world, modeling has a rich tradition, dating back to the earliest days of programming. The most recent innovations have focused on notations and tools that allow users to express system perspectives of value to software architects and developers in ways that are readily mapped into the programming language code that can be compiled for a particular operating system platform. The current state of this practice employs the Unified Modeling Language (UML) as the primary modeling notation. The UML allows development teams to capture a variety of important characteristics of a system in corresponding models. Transformations among these models are primarily manual. However, potential faults that violate desired properties of the software system might still be introduced during the process. Verification technique is well-known for its ability to assure the correctness of models and uncover design problems before implementation. This paper presents a set of rules that allows Software engineers to transform the behavior described by a UML 2.0 Activity Diagram (AD) into a Petri Net (PN). The main purpose of the transformation to Petri nets is to use the theoretical results in the Petri nets domain to analyze the equivalent Petri nets and infer properties of the original workflow. Furthermore, we implement a tool to support the transformation process.

Key words: Activity Diagram, Petri Nets, Verification and Validation, AD2Petri.

1. Introduction

Although complex systems are, by their nature, hard to build, the problem can be ameliorated if the user requirements are rigorously and completely captured. This task is usually very difficult to complete, since clients and developers do not use the same vocabulary to discuss. For behavior-intensive applications, this implies that the dynamic behavior is the most critical aspect to take into account. This contrasts with database systems, for example, where the relation among data types is the most important concern to consider. A scenario is a Specific sequence of actions that illustrates behaviors, starting from a well defined system configuration and in response to external stimulus. Petri nets are used to formalize the behavior of some component, system or application, namely those that have a complex behavior. Since Petri nets are a formal model, they do not carry any ambiguity and are thus able to be validated.

2. Background

In this section we briefly introduce UML Activity Diagrams and Petri Nets.

2.1 Activity Diagram

Activity diagram [1] is basically a flow chart to represent the flow form one activity to another activity. The activity can be described as an operation of the system. So the control flow is drawn from one operation to another. This flow can be sequential, branched or concurrent. Activity diagrams deals with all type of flow control by using different elements like fork, join etc.
The focus of activity modeling is 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 behaviors coordinated by these models can be initiated because other behaviors finish executing, because objects and data become available, or because events occur external to the flow.

2.2 Petri Nets

A Petri net (also known as a place/transition net or P/T net) is one of several mathematical modeling languages for the description of distributed systems. A Petri net[2,3] is a

directed bipartite graph, in which the nodes represent transitions (i.e. events that may occur, signified by bars) and places (i.e. conditions, signified by circles).
A Petri net consists of places, transitions, and arcs. Arcs run from a place to a transition or vice versa, never between places or between transitions. The places from which an arc runs to a transition are called the input places of the transition; the places to which arcs run from a transition are called the output places of the transition. In the diagram of a Petri net, places are conventionally depicted with circles, transitions with long narrow rectangles and arcs as one-way arrows that show connections of places to transitions or transitions to places.

3. Activity Diagram to Petri net Model Transformation Rules

In this section we show how to translate some of the High-level operators available in the UML 2.0 AD, into a behaviorally equivalent PN. To accomplish this, we explain the semantics of the operator, we describe in an informal way how the transformation is achieved, and additionally we show the result of applying these ideas to some illustrative examples.

3.1. Transitions from one Activity to another activity

We consider a semantic for AD with an order relation between control flow such that the emission requires the reception of the preceding action. The AD presented in Fig. 3.1.1 represents an interaction without high-level operators. There are two Actions and one control flow between them. The obtained PN (see Fig. 3.1.2) associates a transition for message in the AD.

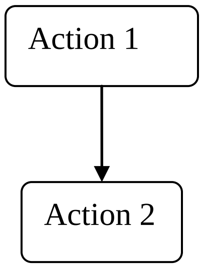

Fig. 3.1.1: A UML Activity diagram

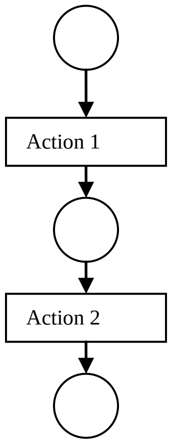

Fig 3.1.2 Obtained Petri Net

3.2. Transition from one Activity to Parallel Activities (using fork):

Fig. 3.2.1 represents a parallel interaction. In figure after Action 1 two transitions namely Action 2 and Action 3 occurs parallel. The obtained PN is shown in Fig 3.2.2.

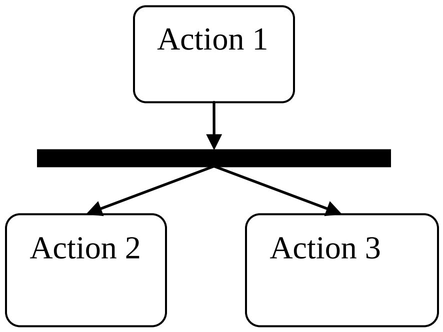

Fig 3.2.1: (a) A UML Activity diagram

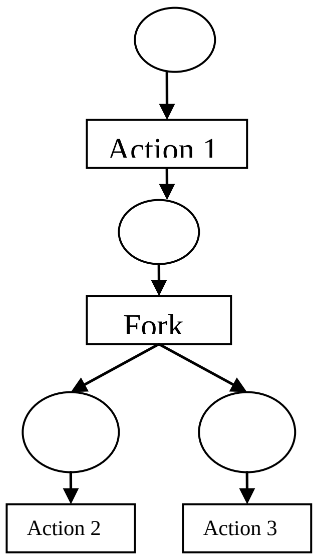

Fig 3.2.2 Obtained Petri Net

3.3. Transition from two Parallel Activities to one Activities (using join):

Fig 3.3.1. Indicate two parallel Activities namely Action1 and Action 2 combines in Action 3. This denotes the end of parallel processing. Fig 3.3.2 shows its corresponding PN.

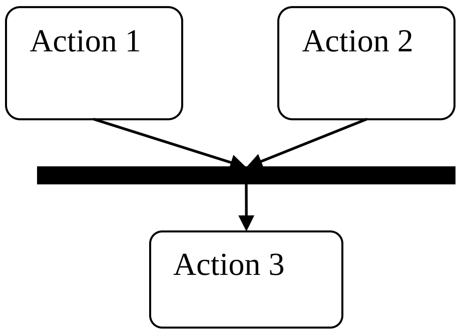

Fig 3.3.1 A UML Activity diagram

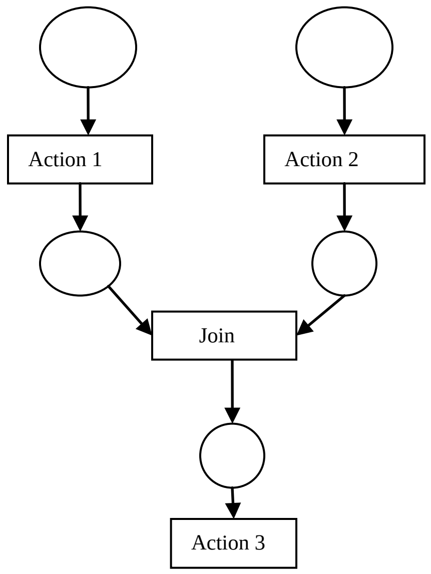

Fig 3.3.2 Obtained Petri Net

3.4. Transition from one Activity to Parallel Activities (using decision):

The AD represented in Fig 3.4.1 shows that If condition Action 1, then perform action 2, else do action 3. This signifies If-Else statement. Its corresponding PN is shown in Fig 3.4.2.

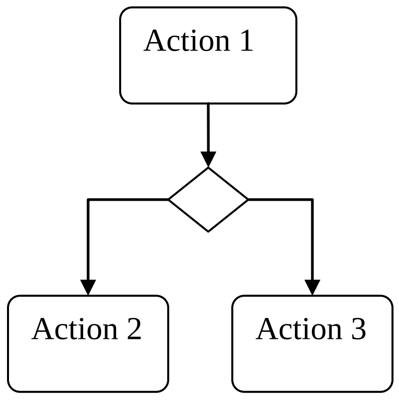

Fig 3.4.1 A UML Activity diagram

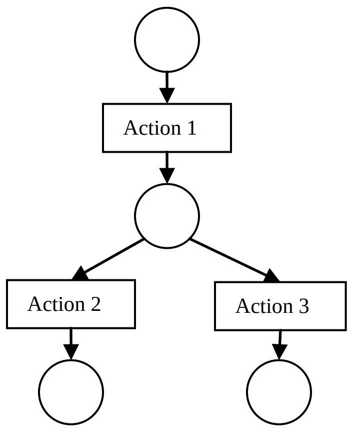

Fig 3.4.2 Obtained Petri Net

3.5. Transition from two Parallel Activities to one Activities (using merge):

Fig 3.5.1 shows that If condition Action 1 and condition Action 2 holds, then do Action 3 i.e. merging two actions. The transformation of this AD into PN is shown in Fig 3.5.2.

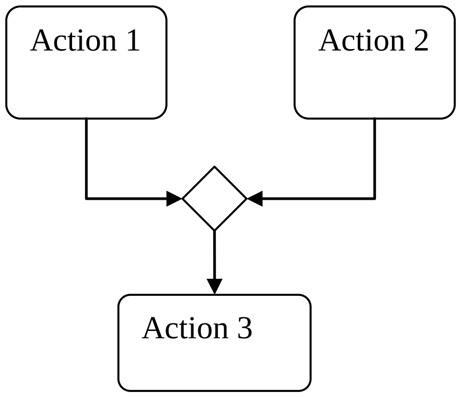

Fig 3.5.1

A UML Activity diagram

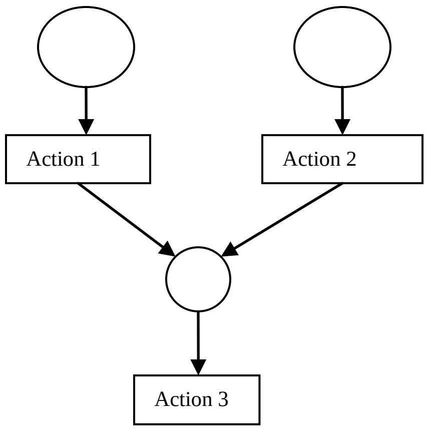

Fig 3.5.2 Obtained Petri Net

3.6. Looping Transition:

A loop node is a structured activity node. In fig 3.6.1 looping occurs in second condition of decision i.e. in Action3 indicates that While condition true do Action 3. The Fig 3.6.2 indicates looping in Petri nets.

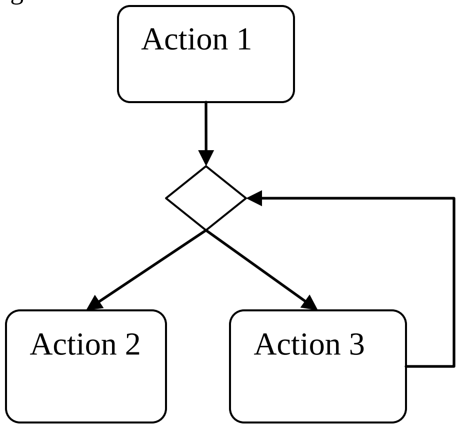

Fig 3.6.1 A UML Activity diagram

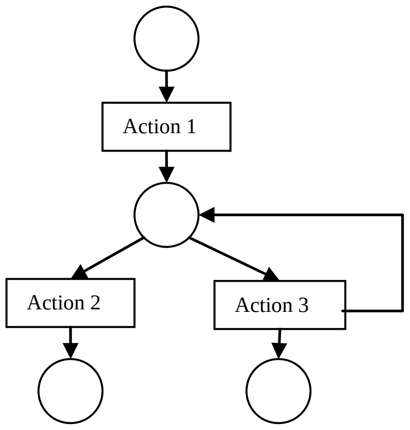

Fig 3.6.2 Obtained Petri Net

3.7. Precedence Transition:

Precedence means Action 1 should precede Action 3. This is shown in Fig 3.7.1.Its corresponding PN shown in Fig 3.7.2.

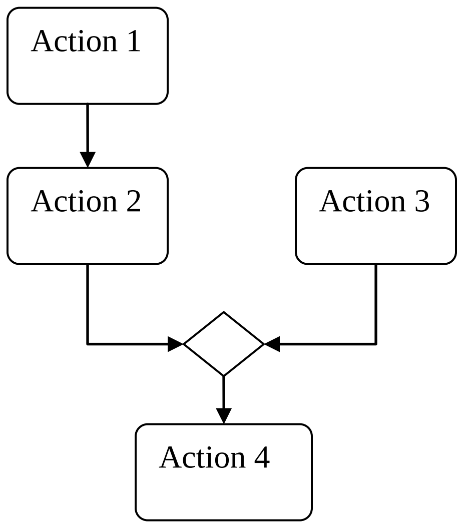

Fig 3.7.1 A UML Activity diagram

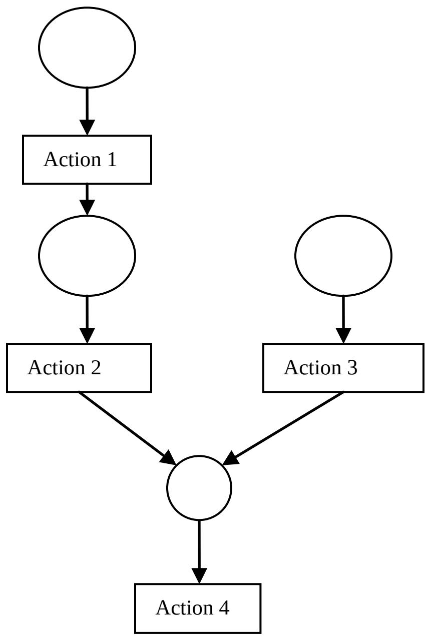

Fig 3.7.2 Obtained Petri Net

3.8. Timing Transitions:

The result value contains the time at which the occurrence transpired. Such an action is informally called a wait time action. This is shown in Fig 3.8.1 indicates after k seconds do Action 1. Its corresponding PN is shown in fig 3.8.2.

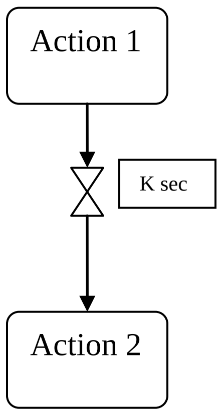

Fig 3.8.1 A UML Activity diagram

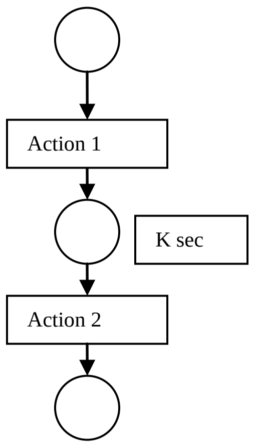

Fig 3.8.2 Obtained Petri Net

4. Running Example

To validate the proposed transformation rules we apply them to one Example namely Order Management System. The following is an example of an activity diagram for order management system. In the diagram Seven activities are identified which are associated with conditions. One important point should be clearly understood that an activity diagram cannot be exactly matched with the code. The activity diagram is made to understand the flow of activities and mainly used by the business users.

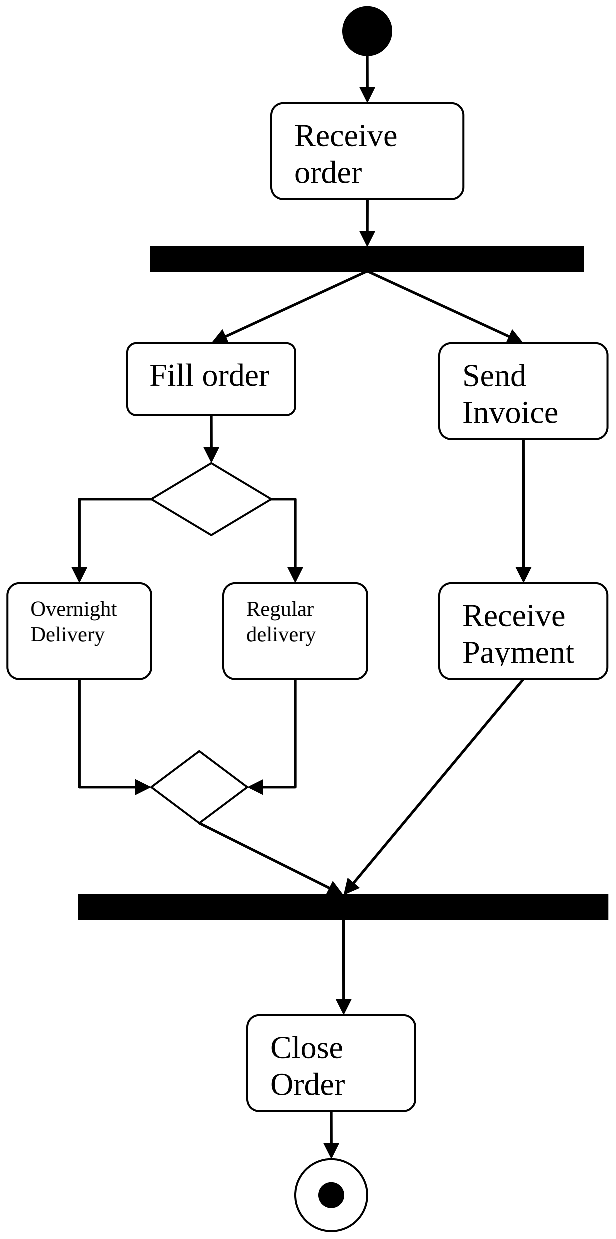

Fig 4.1 A UML Activity Diagram

Fig 4.1 shows the AD of Order Management System. After apply all the rules defined in section 3 we convert this AD into PN shown in Fig 4.2.

Fig. 4.2 shows the PN obtained from the AD in Fig. 4.1, where we can find transitions which are links to a PN [4]. An Activity diagram can be mapped to a Petri net which includes all kinds of control flow [5]. Here activity and fork nodes are mapped to Petri net transitions [6] and start, end, and decision nodes are mapped to places. Connections are mapped in such a way that always there is an arc either from transition to place or place to transition. The converted Petri net model can be represented using Petri Net Markup Language (PNML) [7]. PNML is an XML based interchange format for Petri nets. This is useful for importing and exporting a Petri net model.

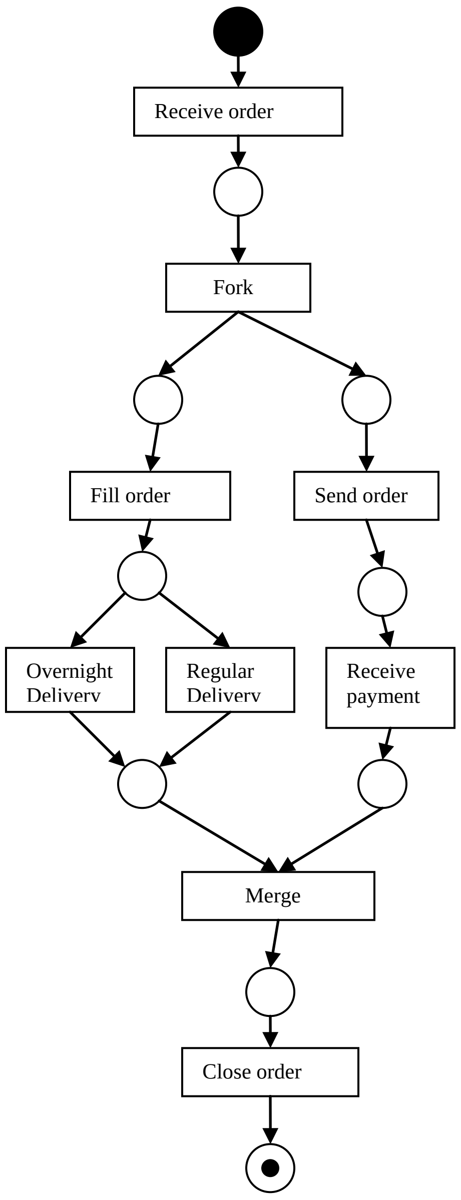

Fig 4.2 Obtained Petri Net

5. Transforming from Activity Diagrams to Petri Nets

Based on the mapping rules in [8], we construct a
Pseudo code to transforming activity diagrams to Petri nets and implement in our tool to provide automatic transformation support. The pseudo code is described in Table 1. The transformed Petri net is a bi-simulation of the activity diagram, which means they are semantically equal. So we can achieve the verification of the activity diagram by verifying the equivalent Petri net against same system properties.

```
Read Activity Diagram
{
set source node of edge=$N_i$;
set similar place or transition of $N_i$=$S_i$;
set target node of edge=$N_j$;
set similar place or transition of $N_j$= $S_j$;
set place=P;
set transition =T;
set subsidiary transition= $T_s$;
set subsidiary place= $P_s$;
set Arc=A;
set  token=$M_0$;
for(each node= N)
{
if(N= initial node, final node, decision node, or
merge node);
{
Create similar P;
}
else
{
Create similar T;
}

for(each edge= E)
{
if ($N_i$ and $N_j$= initial node, final node, decision node,
or merge node)
{
Create $T_s$;
Create A= $S_i$ to $T_s$;
Create A= $T_s$ to $S_j$;
}
else if($N_i$ and $N_j$=action node, fork node, join node)
{
Create $P_s$;
Create A=$S_i$ to $P_s$;
create A=$P_s$ to $S_j$;
}
else
{
Create A= $S_i$ to $S_j$;
}

for(each P without incoming A)
{
Create $M_0$;
}

display PN
}
}
}
```

Table 1 Pseudo code for transforming Activity diagram into Petri net

6. Tool Implementation

We implemented a tool named AD2petri based on Eclipse java Platform. As Figure 5 shows, the full function of tool which is consisting of 4 main parts: Activity diagram generated in UML2 plug-in of Eclipse, AD2petri, Petri-Net, Petri- net tool for Verification. The AD2petri converts an activity diagram to a Petri net automatically. The inputs of AD2petri are UML diagrams designed by UML2 in the form of XML file and the outputs of the tool are Petri net files which are readable for various Petri net tools to perform verification tasks.

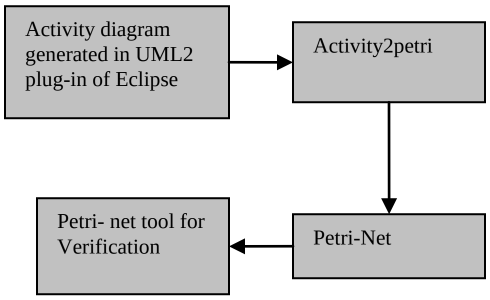
Activity diagram generated in UML2 plug-in of Eclipse
Activity2petri
Petri-Net
Petri- net tool for Verification

Fig 6.1 Framework of AD2Petri

7. Analysis of Petri net

The Petri net is subjected to three analysis methods namely, Liveliness, Boundness and Reachability analysis[10][9].The liveliness is determined through the absence of Deadlocks in the Petri Net[11] while Boundness is computed through a P-invariant calculation. The result or analysis[12] confirms that the Petri net is live and bounded. Through the P-invariant calculation it is revealed that the Petri Net is safe also.

8. Conclusion and Results

In this paper we show a set of rules and tool implementation to transform AD into equivalent PN. In UML 2.0, AD is quite expressive and this work explores the new constructors that allow several plain activities to be combined in a unique AD. Thus the rules allow the generation of a PN that covers several sequences of behaviors. The Verification result shows whether the Petri net satisfy the requirements or not this work is in progress so we plan to develop it further. In this work we only have a validation of transformation through analysis of UML AD.
In future we plan to implement any of the Petri net verification tool to this project so we automatically get verification results of the inputted AD.

9. Acknowledgement

I would like to thank Mr. B.L Pal and Mr. G. Balakrishna for his support, guidance, and patience they gave throughout writing this paper. Their encouragement and dedication is numerous to mention.

10. References

1. OMG, OMG Unified Modelling Language (UML) Superstructure 2.1, available at www.omg.org. 2007.

2 . Christensen, S. and L. Petrucci, Modular Analysis of Petri nets. The Computer Journal, 2000. **43**(3): p. 224-242.

3. Petri Nets: Properties, Analysis and Applications, by Tadao Murata, in: Proceedings of the IEEE, vol. 77, no. 4, April 1989

4. Zhou CH,The modeling of UML diagrams based on the Petri Net[M], Shandong University of Science and Technology. 2004: 19-31.

5. Harald Storrle, Semantics of UML 2.0 Activities Workflow management coalition [Online]. http://www.wfmc.org/standards/docs/TC-1011_term_glossary_v3.pdf

6. Carl Adam Petri and Wolfgang Reisig (2008) Petri net. Scholarpedia, 3(4):6477.

7. Billington et al., The Petri Net Markup Language: Concepts,Technology, and Tools [Online]. Available: http://www.informatik.huberlin.de/top/pnml/download/about/PNML_CTT.pdf

8. H. Störrle. Semantics of Control-Flow in UML 2.0 Activities. In Proceedings of the 2004 IEEE Symposium on Visual Languages - Human Centric Computing, 2004, pp. 235-242

9. G. Rozenburg, J. Engelfriet, Elementary Net Systems, in: W.Reisig, G. Rozenberg (Eds.), Lectures on Petri Nets I: Basic Models - Advances in Petri Nets, volume 1491 of Lecture Notes in Computer Science, Springer,1998, pp. 12-121

10. J.L. Peterson. Petri net theory and the modeling of systems. Prentice Hall, Englewood Cliffs, 1981.

11. Adamski, M.: Direct Implementation of Petri Net Specification. In:7th International Conference on Control Systems and Computer Science. (1987) 74–85.

12. R.E. Barlow and F. Proschan. Statistical Theory of Reliability and Life Testing. Holt, Rinehart and Winston, New York, 1975.