International Journal of Recent Technology and Engineering (IJRTE)
ISSN: 2277-3878, Volume-1, Issue-5, November 2012

Some Rules to Transform Activity Diagrams into Colored Petri Nets

Bhawana Agarwal

Abstract*: 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 Colored Petri Net (CPN). ADs in UML 2.0 are much richer than in UML 1.x, namely by allowing several traces to be combined in a unique diagram, using high-level operators over interactions. 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. Thus, non-technical stakeholders are able to discuss and validate the captured requirements. The usage of this model is an important topic , since it permits the user to discuss the system behavior using the problem domain language. A small control application from industry is used to show the applicability of the suggested rules.

***Index Terms*: Activity Diagram, Petri Nets, Colored Petri Nets, Verification and Validation.**

I. 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.

II. UML DIAGRAMS FOR INTERACTION

The dynamic part of the system can be specified in UML 2.0 through various behavioral diagrams, such as: activity diagrams, sequence diagrams and state machines diagrams. These diagrams use behavioral constructs, namely activities, interactions, and state machines.

An interaction is formed by lifelines and messages between them, that sequence is important to understand the situation. Although data may be also important, its manipulation is not the focus of interactions. Data is carried by the messages, and stored in the lifelines, and can be used to decorate the diagrams.

Activity diagram is another important diagram in UML to describe dynamic aspects of the system. Activity diagram is basically a flow chart to represent the flow from 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 basic purposes of activity diagrams are similar to other four diagrams. It captures the dynamic behavior of the system. Other four diagrams are used to show the message flow from one object to another but activity diagram is used to show message flow from one activity to another.

Activity is a particular operation of the system. Activity diagrams are not only used for visualizing dynamic nature of a system but they are also used to construct the executable system by using forward and reverse engineering techniques. The only missing thing in activity diagram is the message part.

There are several possible operators, whose meaning is described informally in the UML 2.0 Superstructure specifications [2]:

Fig 1(a): Activity states, represent the performance of a step within the workflow.

Fig (b): Transitions show what activity state follows after another. This type of transition can be referred to as a completion transition. It differs from a transition in that it does not require an explicit trigger event; it is triggered by the completion of the activity that the activity state represents.

Fig(c): Merge node is a control node that brings together multiple alternate flows. It is not used to synchronize concurrent flows but to accept one among several alternate flows. A merge node has multiple incoming edges and a single outgoing edge.

Fig (d):Join node is a control node that synchronizes multiple flows. A join node has multiple incoming edges and one outgoing edge.

Fig (e): Fork node is a control node that splits a flow into multiple concurrent flows. A fork node has one incoming edge and multiple outgoing edges.

Fig (f): Decision node accepts tokens on an incoming edge and presents them to multiple outgoing edges. Which of the edges is actually traversed depends on the evaluation of the guards on the outgoing edges.

Fig (g): Initial node is a control node at which flow starts when the activity is invoked. An activity may have more than one initial node.

Revised Manuscript Received on 30 November 2012.

* Correspondence Author

Bhawana Agarwal*, M.tech Scholar, Computer Science, Mewar University,Chittorgarh,India.

Fig (h): Activity may have more than one activity final node. The first one reached stops all flows in the activity.

Fig (i): If the occurrence is *a time event occurrence*, the result value contains the time at which the occurrence transpired. Such an action is informally called a wait time action.

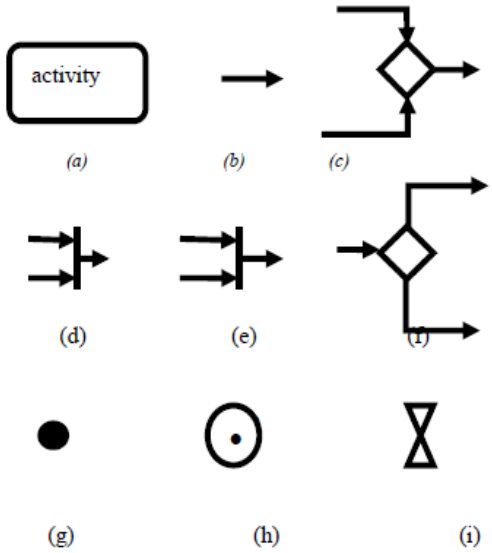

Fig 1: Symbols of Activity Diagram

III. TRANSFORMING ACTIVITY DIAGRAMS INTO PETRI NETS

In this section we show how to translate some of the high-level operators available in the UML 2.0 ADs, into a behaviorally equivalent CPN. 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.

A. Transitions from one Activity to another activity

First of all we look to Interaction Fragments without any of the high-level operators. An Interaction Fragment is a set of Lifelines, each of which has a sequence of Event Occurrences associated with it. We consider a semantic for AD with a order relation between messages such that the emission requires the reception of the preceding message.

The AD presented in Fig. 2a represents an interaction without high-level operators. There are two Lifelines and one messages between them. The obtained CPN (see Fig. 2b) associates a transition for message in the AD. Character F denotes Fusion in all the concerned diagrams..

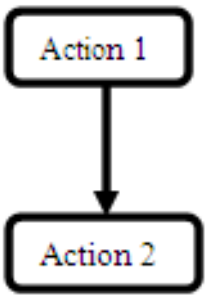

Fig 2: (a) A UML ACTIVITY DIAGRAM

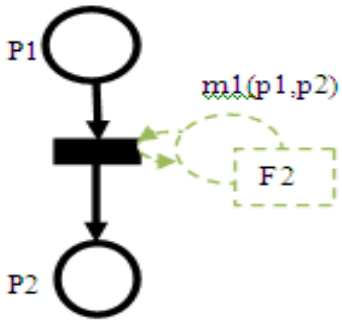

Fig 2: (b) OBTAINED CPN

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

Fig. 3 a. represents an parallel interaction. As in figure after Action 1 two transitions namely Action 2 and Action 3 occurs parallely. T he obtained CPN shown in Fig 3(b).

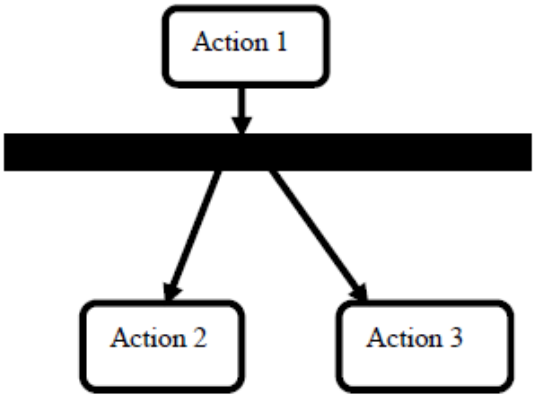

Fig 3: (a) A UML ACTIVITY DIAGRAM

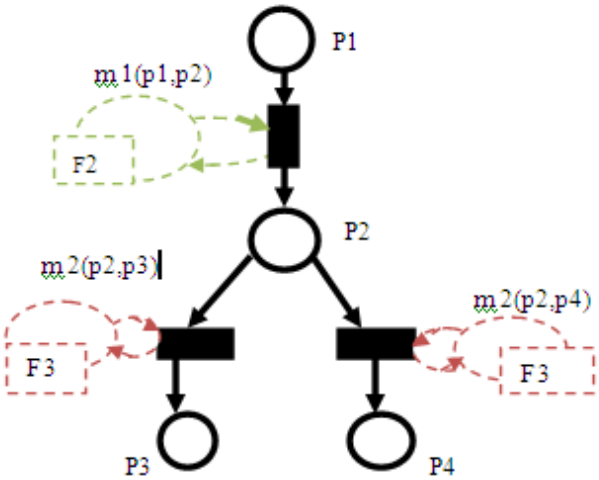

Fig 3 (b) OBTAINED CPN

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

Fig 4(a). Indicates two parallel Activities namely Action1 and Action 2 combines in Action 3. This denotes the end of parallel processing. Fig 4(b) shows its corresponding CPN.

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)

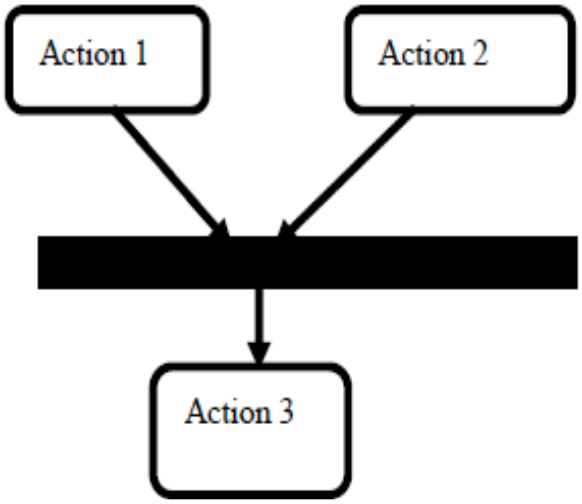

Fig 4: (a) A UML ACTIVITY DIAGRAM

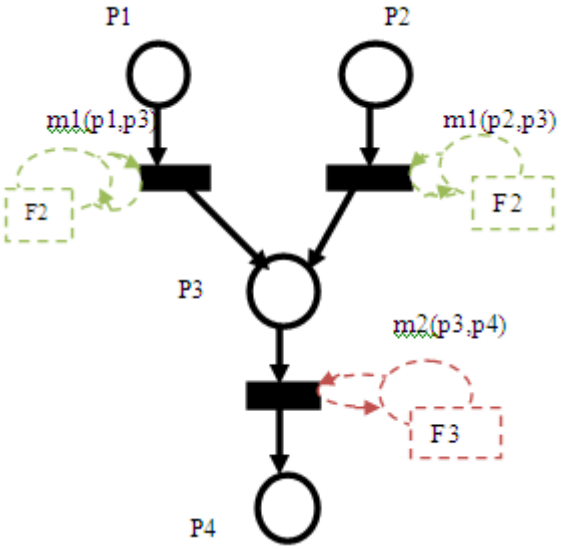

Fig 4: (b) OBTAINED CPN

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

The AD represented in Fig 5(a) shows that If condition Action 1, then do action 2, else do action 3. This signifies If-Else statement. Its corresponding CPN shown in Fig 5(b).

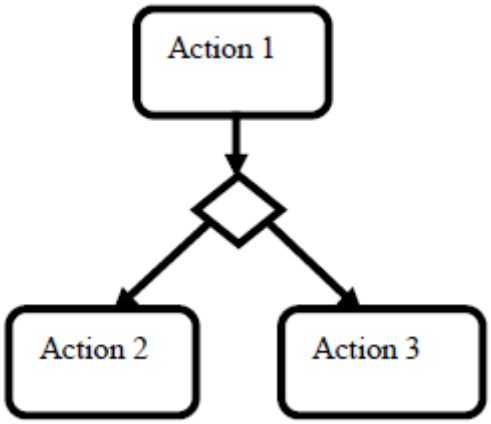

Fig 5: (a) A UML ACTIVITY DIAGRAM

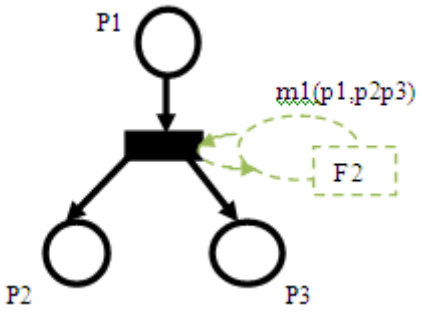

Fig 5: (b) OBTAINED CPN

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

Fig 6(a) shows that If condition Action 1 and condition Action 2 holds, then do action 3. This transforms of this AD into CPN is shown in Fig 6(b).

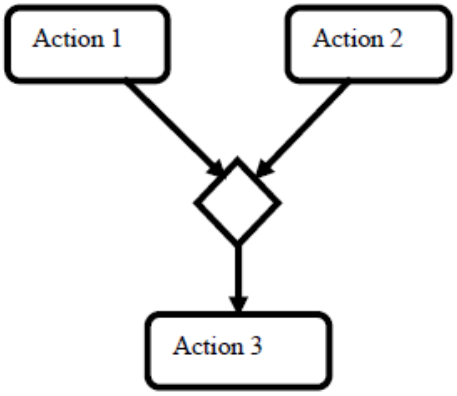

Fig 6: (a) A UML ACTIVITY DIAGRAM

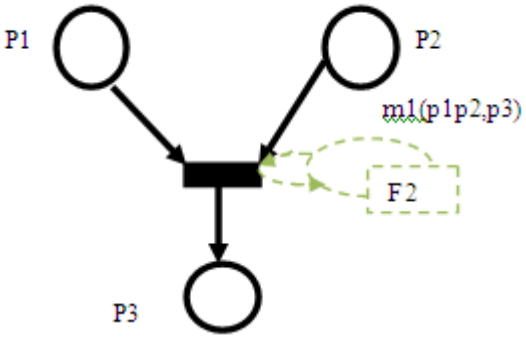

Fig 6: (b) OBTAINED CPN

F. Looping Transition:

A loop node is a structured activity node that represents a loop with setup, test, and body sections. In fig 7(a) looping occurs in second condition of decision i.e in Action3 indicates that While condition true do Action 3.

The Fig 7(b) indicates looping in petri nets.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

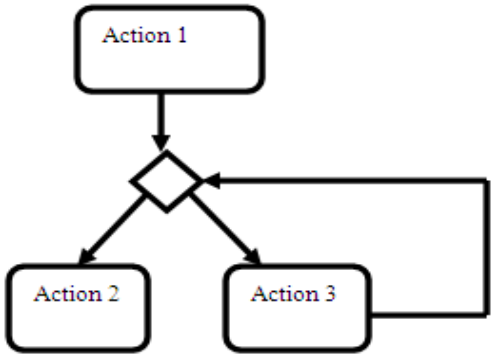

Fig 7: (a) A UML ACTIVITY DIAGRAM

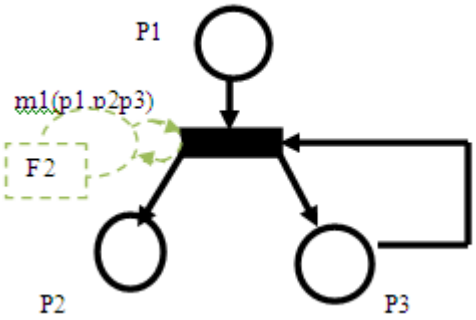

Fig 7: (b) OBTAINED CPN

G. Precedence Transition:

Precedence means Action 1 should precede action 3 .This is shown in Fig 8(a).Its corresponding CPN shown in Fig 8(b).

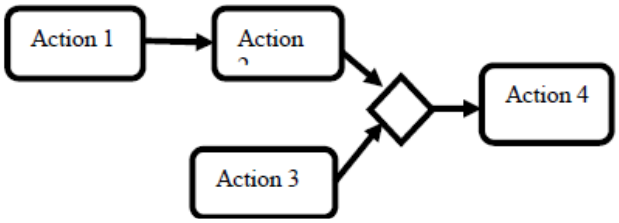

Fig 8: (a) A UML ACTIVITY DIAGRAM

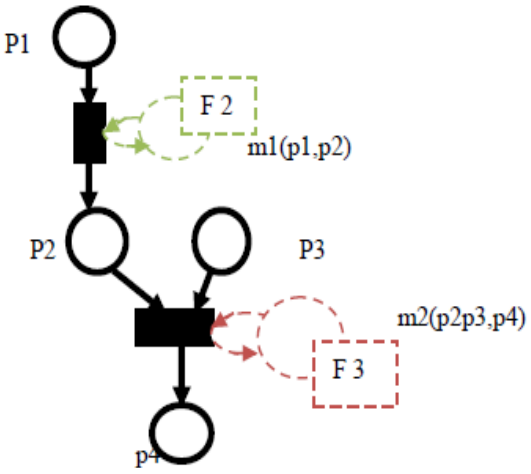

Fig 8: (b) OBTAINED CPN

H. 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 9(a) indicates after k seconds do Action 1. Its corresponding CPN is shown in fig 9(b).

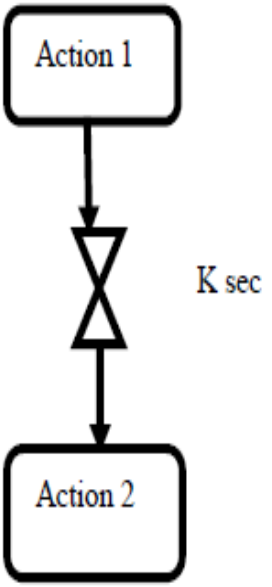

Fig 9: (a) A UML ACTIVITY DIAGRAM

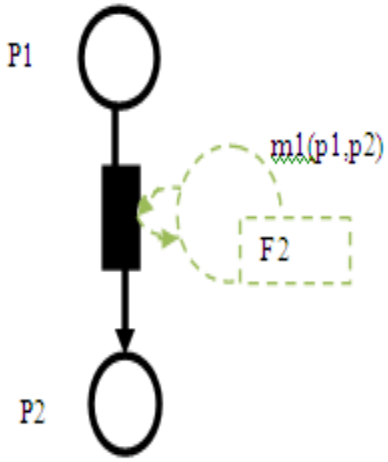

Fig 9: (b) OBTAINED CPN

IV. VALIDATION OF THE RULES

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 Six 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.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

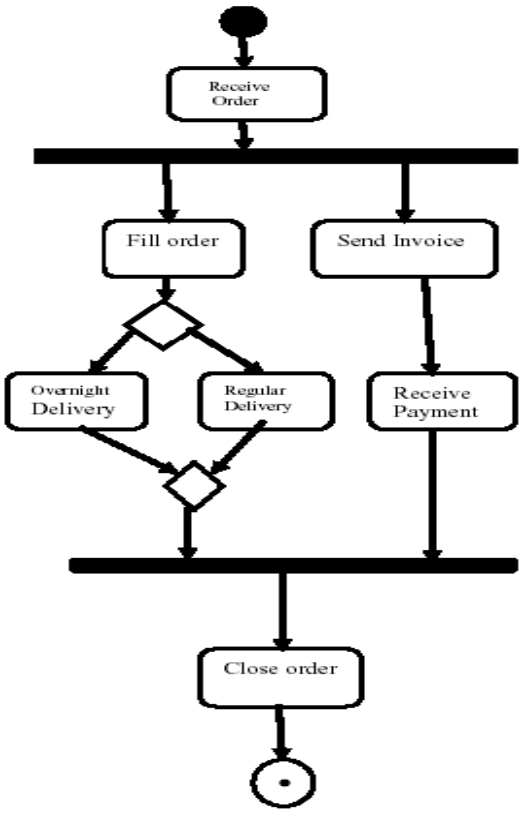

Fig 10: (a) A UML ACTIVITY DIAGRAM

Fig 10(a) shows the AD of Order Management System.

After apply all the rules defined in section III we convert this AD into CPN shown in Fig 10(b).

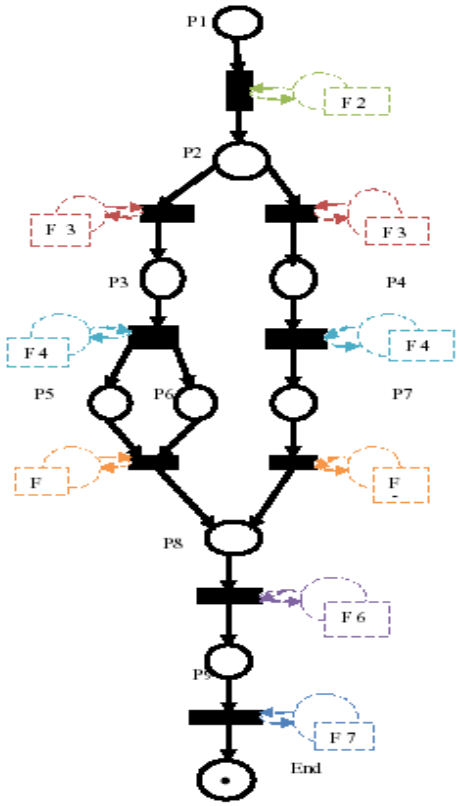

Fig 10 (b): OBTAINED CPN

Fig. 10(b) shows the CPN obtained from the AD in Fig. 10(a), where we can find transitions which are links to a CPN[5][6]. An Activity diagram can be mapped to a Petri net which includes all kinds of control flow [4]. Here activity and fork nodes are mapped to Petri net transitions[8] 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) [3]. PNML is an XML based interchange format for Petri nets. This is useful for importing and exporting a Petri net model.

V. ANALYSIS OF PETRI NETS

The petri net is subjected to three analysis methods namely, liveliness, boundness and reachability analysis[10][11].

The liveliness is determined through the absence of Deadlocks in the Petri Net[7] 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. Subsequently a reachability [9] analysis is performed on the Petri Net,resulted in a reachability Graph presented in Fig 11.

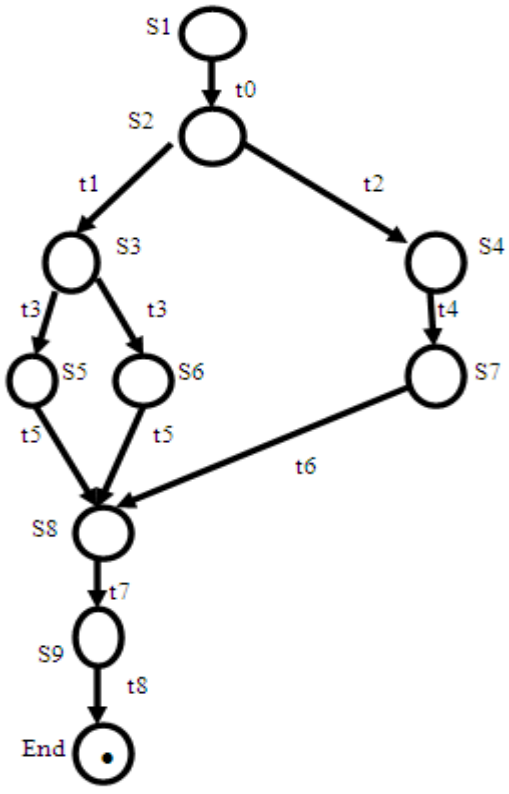

Fig 11: REACHABILITY GRAPH OF Fig 12(b).

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

VI. CONCLUSIONS AND RESULTS

In this paper we show a set of rules to transform AD into equivalent CPNs. In UML 2.0, ADs are quite expressive and this work explores the new constructors (in relation to UML 1.x) that allow several plain sequences to be combined in a unique AD. Thus the rules allow the generation of a CPN that covers several sequences of behaviors. This work is in progress so we plan to develop it further. Firstly, we need to better tune the rules, to realize if they can be automated. In the future we would like to use a UML-based tool to draw the AD diagrams and apply automatically the rules to obtain a CPN. Probably this automation requires a second set of rules that "optimizes" the CPN by eliminating redundant parts. In this work we only have a validation of transformations though analysis and synthesis of UML AD, we plan to study the soundness and completeness of the approach.

Finally the usage of the rules in real-world projects is planned, since we believe that methods and tools for software engineers need to be evaluated by them in complex industrial projects.

ACKNOWLEDGMENT

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.

REFERENCES

1. Jensen, K.: Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Brauer, W. and Gozenberg, G. and Salomaa edn. Volume Volume 1, Basic Concepts of Monographs in Theoretical Computer Science. Springer-Verlag (1997) ISBN: 3-540-60943-1.
2. Fowler, M.: UML Distilled: A Brief Guide to the Standard Object Modelling Language. Addisson-Wesley (2003)
3. Billington et al., The Petri Net Markup Language: Concepts,Technology, and Tools [Online]. Available:
4. http://www.informatik.huberlin.de/top/pnml/download/about/PNML_CTT.pdf
5. Harald Storrle, Semantics of UML 2.0 Activities Workflow management coalition [Online].
6. http://www.wfmc.org/standards/docs/TC-1011_term_glossary_v3.pdf
7. Machado, R.J., Lassen, K.B., Oliveira, S., Couto, M., Pinto, P.: Execution of UML Models with CPN Tools for Workflow Requirements Validation. In: Sixth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools.(2005).
8. Zhou CH,The modeling of UML diagrams based on the Petri Net[M], Shandong University of Science and Technology. 2004: 19-31.
9. Adamski, M.: Direct Implementation of Petri Net Specification. In: 7th International Conference on Control Systems and Computer Science. (1987) 74–85.
10. Carl Adam Petri and Wolfgang Reisig (2008) Petri net. Scholarpedia, 3(4):6477.
11. P. Küngas. Petri Net Reachability Checking Is Polynomial with Optimal Abstraction Hierarchies. In: Proceedings of the 6th International Symposium on Abstraction, Reformulation and Approximation, SARA 2005, Airth Castle, Scotland, UK, July 26–29, 2005.
12. 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
13. J.L. Peterson. Petri net theory and the modeling of systems. Prentice Hall, Englewood Cliffs, 1981.
14. R.E. Barlow and F. Proschan. Statistical Theory of Reliability and Life Testing. Holt, Rinehart and Winston, New York, 1975

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)