International Journal of Enhanced Research in Science Technology & Engineering, ISSN: 2319-7463
Vol. 3 Issue 2, February-2014, pp: (18-24), Impact Factor: 1.252, Available online at: www.erpublications.com

# Generating Automated Test Cases using Model Based Testing

Prof. Kirti Patil[1], Prof. M. M. Ganeshwade[2]
Maharashtra, India

**Abstract: The quality of software depends on verification and validation of software. Software testing is an important phase of software development life cycle. It is the process of finding defects by executing a program on a set of test cases and comparing the actual results with expected results. For generating the test cases, Information regarding the behavior of system must be understood first, which can be taken from the models created in design phase. UML is widely accepted and used by industry for modeling and design of software and of embedded systems also. Nowadays Model based testing of real time embedded system is in boom. This paper presents an approach to generate test cases from UML activity diagrams for real-time embedded systems. We first transform extended activity diagrams into a Petri Net Graph (PNG). PNG then transform to generate Test sequences. We have used workflow analysis for analyzing PNG. Petri Net properties are analyzed and then prioritized test cases generated for verification of dynamic model of Real time Embedded system. It will surely improve the efficiency and at the same time to reduce the high cost of the manual testing in design phase itself.**

**Keywords: Software Testing, UML Models, Extended Activity diagram, Petri Net Graph (PNG), Test Sequence.**

## I. INTRODUCTION

This paper presents a Novel approach to Generate the test cases using extended activity diagram for Real Time Embedded System. The more focus is on verification of dynamic model of real time embedded system. Today, UML is capturing much attention not only of software-intensive systems but also in the Real Time Embedded Software community and considered an important source of information for generating test cases. Therefore if it is correctly designed it will reduce testing cost and effort and at the same time improve the software quality. Here we used UML 2.x layered activity diagram which provides abstracted and complicated workflow for concurrent embedded system. The core of testing work is to verify the workflow of system. In this way, activity diagram is very useful to embedded software testing.

This work demonstrates the possibility by developing an Editor which can be used to model the workflows using Activity Diagrams. Further In this paper we present a formal method to analyze these activity workflow models by transforming it into Petri Net Graph. Petri net properties and generated prioritized test cases will verify the real time embedded system.

## II. LITERATURE SURVEY

Model Based Testing is highly ranked type of testing which provides a solid foundation for generating the test cases from the business requirements that are formally represented by a model. In the literature, Unified Modeling Language (UML) and its models are widely used as modeling mechanisms to specify, analyze and design requirements of the system, for test automation and for model-based testing. The UML 2.0 includes fourteen kinds of diagrams. UML provides both the static (or structural) views and dynamic (or behavioral) views of the system. Static view includes class diagram, object diagram and Dynamic view includes activity diagram, state chart diagram, sequence diagram etc. A lot of research has been done in the literature using these UML models to verify the dynamic model of system. Based on UML diagrams, Anders Hessel and Paul Pettersson [1] have described extended tool of UPPAAL to verify test-cases derived from a network of timed automata specifying the expected behavior of the system. Particularly, they presented the symbolic reachability analysis algorithm of the tool to generate traces. The generated traces satisfies simple coverage criteria, and used as test sequences or suites to test real-time systems. Authors in [2] tried to convert an activity diagram to a directed graph. Then test cases can be generated from the directed graph. But this method lacks concrete rules for the automatic transformation. Wang Linzhang et al. [3] proposed an approach of gray-box testing using activity diagrams according to dynamic scenario of system. Test cases are generated following basic path coverage criterion and by representing a method of a class to an activity and a class to a swim lane, which represent the expected structure and behavior of SUT. Chen Mingsong et al. [4] presented an idea to obtain the random generation of test cases for Java programs using activity diagram. Java programs are

transformed to intermediate level called program execution traces (pet) and finally, reduced test suite is obtained by comparing the simple paths with program execution traces.

In today's era, it becomes necessity to verify the correctness of complex system specification and it can be done with formal verification to guarantee the quality of UML models. UML activity diagram adopts Petri-net semantics which describes the concurrent behavior and very few researchers [10, 11, 16] used Petri Net like semantics approach to generate the test cases from activity diagram. In [10, 11, 16] the authors proposed an approach. The UML activity diagram is translated into a formal model NuSMV [6], and then the properties in the form of CTL or LTL [7] are analyzed. Their formulas can be generated from the coverage criteria. Finally, the negative properties are applied to the formal model using the model to produce the required test cases. However, the translation is used to verify the consistency between UML activity diagrams and class diagrams of two different models. Dianxiang Xu [10] implemented a tool ISTA for automated test generation and execution by using high-level Petri nets. Author [10] considered high-level Petri nets as finite state test models. It also provides a language for mapping the elements in test models as Petri nets for implementation constructs. This test code can be executed against the SUT.

Jörg Desel And Andreas Oberweis, [11] presented a technique which generates test cases from the cause-effect graphing (a method for program code testing) for the validation of high-level Petri nets in a systematic way. The difference between both methods is the representation of the relationships between causes and effects. Additionally, the method also creates test input and test output data to the test cases.

Although UML activity diagram adopts Petri-net Semantics, it only focuses on workflow modeling. Petri nets combine a well defined mathematical theory with a graphical representation of the dynamic behavior of systems. Here we propose a formal method to convert activity diagram into Petri net through which formal analysis of a model can be done. With the combination of workflow analysis of activity diagram and support of mathematical analysis of Petri net, test paths are generated to verify the correct design of real time embedded system. There are a number of tools in the market on model-based software testing. These tools can be categorized as commercial tools and academic tools. Commercial tools include AGEDIS, Spec Explorer ,MaTeLo,Conformiq Qtronic , TorX , T-VEC, JUMBLE, REACTIS and many more and Acadamic tools include SPECTEST, COW SUITE, TORX/CADP, TGV/CADP, TOSTER, MULSAW etc.

Based on tool survey [27], it is clear that most tools used abstract models presenting either usage-based scenarios or state-based system behaviors. Most existing model-based test tools focus on test generation and execution for a particular software or hardware system. Clearly, this is not enough in test automation for many complex real application systems.

To the best of our knowledge, there is no existing tool that can automatically derive test cases only from UML extended activity diagram to show the system behavior and which will give verification of system for implementation. This paper proposes a prototype tool to elaborate the abstract activity diagram and analyze the properties of diagram for further system verification.

The comparison is done with respect to the analogous tool TorX which uses UML Sequence diagram to show the behavior of system. The output is generated in the form of Tree structure and It is also unclear from output whether or not the tool provides expected results for each test case to verify the system. Also it doesn't check any property to verify the system for correct implementation.

## III. BACKGROUND

### Extended Activity Diagram and Petri Net Concept

Activity diagram is used for business modeling, control flow modeling, complex operation modeling etc. There are three main advantages to use an Activity diagram 1) It is very simple like a flowchart and 2) It is very easy to understand the flow of logic of the system. 3) It can be used as the initial diagram to study new or existing systems. UML 2 Activity Diagram presents concepts at a higher abstraction level compared to other UML diagrams. Hence finding test information from this model is a terrified task. Among all UML diagrams UML activities closely represent a wide spectrum of properties and are targeted towards different levels of stakeholders needs to express explicit system workflow. Activity diagram can be used to design the System behavior for different purposes and at all levels. Hence they need to be described in detailed. For better describing the complicated actions of embedded real-time system, we use layered activity diagrams in system modeling. The upper layer activity diagram is the main work flow of at higher abstraction level. The under layer activity diagrams expanded from active nodes of the upper layer activity are used to describe the sub-activity of System.

In this paper, we use two STEREOTYPES <<Task>> and <<Combined Task>> to realize the dynamic modeling of test cases generation for the real-time extension of activity diagram. The UML specification [6] has given the explanation on the different types of activities in form of rules for node execution based on something similar to token flows. Still, activity diagram has some consequences. The activities are informal models that require verification and testing. Hence, we need something more formal to analyze the workflow. Petri net and its properties offer that. Petri net is analytical as well as mathematical tool. As an analysis and mathematical tool it reveals various properties of the model and hence of the actual physical system. Thus, one can draw important conclusions about the system without performing lengthy calculation of conventional system modeling. Therefore, transforming activities into Petri nets seems to be a best choice for system verification. The Supporting evidence exists presenting the advantages of mapping activities into Petri nets [12-15, 19]. In this paper the possible mapping process has been discussed and used [23-26].

**Activity Diagram to Petri Net Translation**

In UML 2.0, the activity diagram semantics can be easily mapped to the Petri net semantics. In general Petri nets consist of places and transitions. Places and transitions are linked by directed arcs. Arcs run from a place to a transition or vice versa, but never between two nodes of the same kind. Each arc is associated with a non negative number called weight. As shown in figure1, simple but important mappings are used. Activity nodes are represented by places and decision nodes, forks, joins, initial nodes, activity final nodes are shown as transitions. The parallel execution of transitions is denoted by an AND-split like fork and An AND-join on the other hand is used to resynchronize parallel flows like join. The alternative transition - decision is modeled with an OR-split. Edges are made transitions with one input and one output arc.

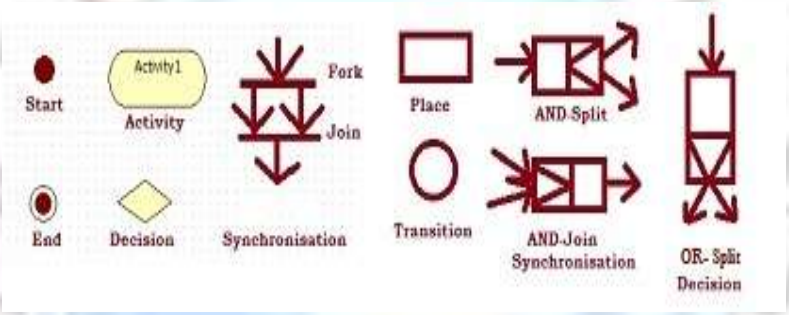


**Figure 1: Semantics of Activity diagram and its corresponding Petri Nets**

**Analysis of Petri Net**

The Formal definition of Petri nets are defined as follows [29, 30]:

**Definition 1:**
A Petri net G is a four-tuple (P, T, IN, OUT) where
$P = \{p_1, p_2, p_3, \ldots, p_n\}$ is a set of places
$T = \{t_1, t_2, t_3, \ldots, t_n\}$ is a set of transitions
And $P \cup T$ and $P \cap T = \varnothing$
where IN: $(P \times T) \rightarrow N$ is an input function that defines directed arcs from places to transitions and where OUT: $(P \times T) \rightarrow N$ is an output function that defines directed arcs from transitions to places.

If $IN (p_i, t_j) = k$, where $0 \leq k \leq 1$ is an positive integer, a directed arc from place p, to transition $t_j$ is drawn with label k.

If $IN (p_i, t_j) = 0$, no arc is drawn from $p_i$ to $t_j$. Similarly, if $OUT (p_i, t_j) = k$, a directed arc is included from transition $t_j$ to place $p_i$ with $k = 1$. If $k = 0$, no arc is included from $t_j$ to $p_i$.

**Definition.2:**
Let $2^P$ be the power set of P. We then define functions $IP: T \rightarrow 2^P$ and $OP: T \rightarrow 2^P$ as follows:

$IP (t_j) = \{p_i \varepsilon P: I N (p_i, t_j) = 0\} V t_j \varepsilon T$
$OP (t_j) = \{p_i \varepsilon P: OUT (p_i, t_j) = 0\} V t_j \varepsilon T$

Where $IP (t_j)$ is the set of input places of $t_j$, and $OP (t_j)$ is the set of output places of $t_j$.

**Definition 3:**
A transition $t_i$ of a PN is said to be enabled in a marking M if

$$M(p_i) \geq I\,N(p_i, t_j)\ V\ p_i \varepsilon P(t_j)$$

An enabled transition $t_j$ can fire at any instant of time. When a transition $t_j$ enabled in a marking M fires, a new marking M' is reached according to the equation

$$M'(p_i) = M(p_i) + OUT(p_i, t_j) - IN(p_i, t_j)\ V\ p_i \varepsilon P.$$

We say marking M' is reachable from M and write

$$M \xrightarrow{t_j} M'.$$

**Definition 4:**
Given a marked net $(G, M_O)$, a reachable marking $M \varepsilon R[M_O]$ is called a deadlocked marking (or a deadlock) if no transition is enabled in M. A marked net $(G, M_O)$ in which no reachable marking is deadlocked is said to be deadlock free.

## IV. PROPOSED APPROACH WITH AN EXAMPLE

As shown in Figure 2, our technique is based on following steps:

**a)** Augmenting the activity diagram with necessary test information.
**b**) Converting activity diagram into PNG.
**c)** Identify all the activity flows from PNG.
d) Prioritize the flows based on weight of all transitions in the sequence activity flow.
**e)** Generate the test cases.

Figure 2 shows Workflow of Proposed tool. Figure 3 shows the activity diagram for "Cash withdrawal" to model the workflow of system. It contains two types of activities: 1) Combined activity and 2) Activity denoted with the help of proposed stereotypes.

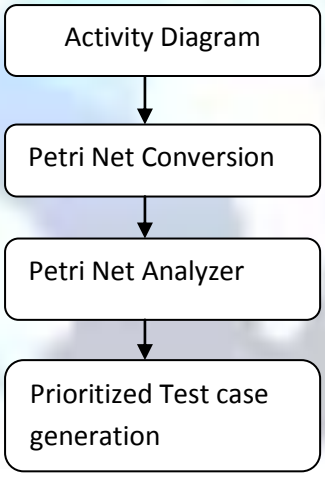


**Figure 2: Workflow of proposed tool**

After translating activity diagram into Petri net Incidence matrix [23] 'I' is used to calculate reachable marking M from a given marking after firing a particular transition and properties of Petri Net are analyzed.
The reachability graph is constructed with initial marking M0 using the following algorithm [29, 30]:

- Label the initial marking M, as the root and tag it "new."
- While "new" markings exist, do the following:
  - Select a new marking M.

If M is identical to a marking on the path from the root to M, then tag M

- "old" and go to another new marking.
- If no transitions are enabled at M, tag M
  "Dead-end."
- While there exist enabled transitions at

M, do the following for each enabled transition t at M:

- Obtain the marking M' that results from firing t at M.
- On the path from the root to M if there exists a marking M" such that M'(p)>M"(p) for each place p and M' ≠ M", i.e., M" is coverable, then replace M'(p) by ω for each p such that M'(p) > M"(p).
- Introduce M' as a node, draw an arc with label t from M to M', and tag M' as "new".

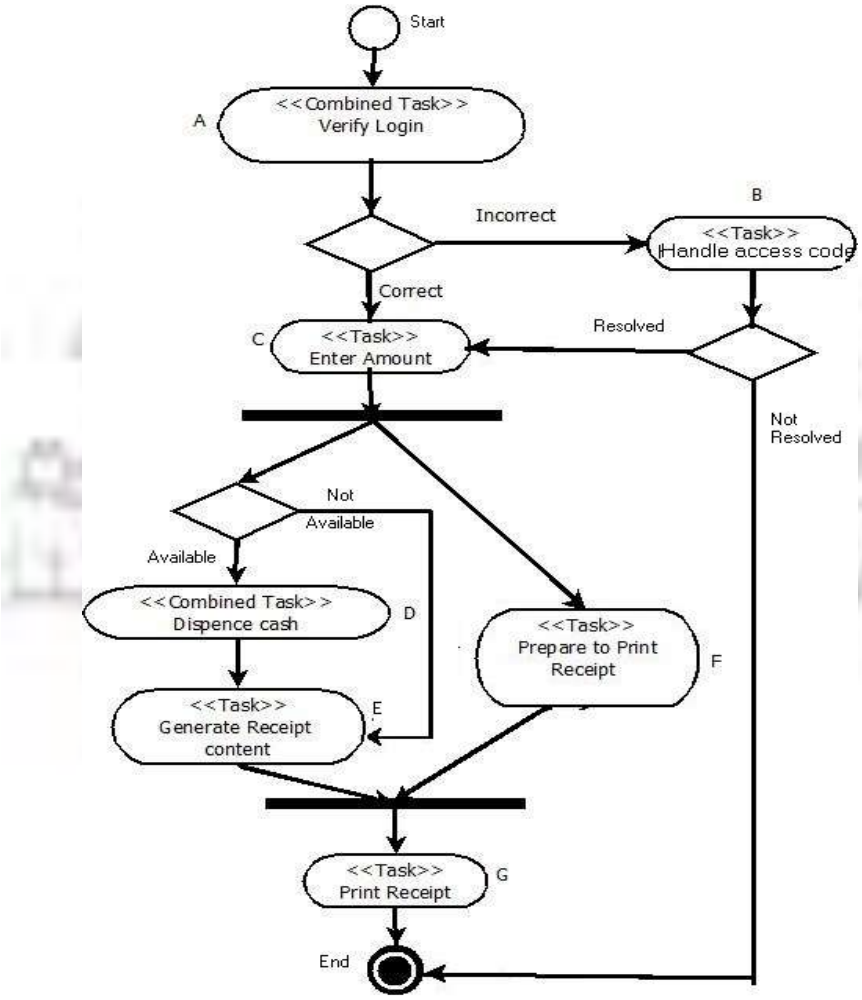


**Figure 3: UML Activity diagram for "Cash Withdrawal"**

The Petri net properties are helpful in analyzing corresponding workflow for constructed activity diagram. Presence of boundedness indicates that a particular place have finite number of tokens if there is only one token in the start place and further same token in other places and absence of boundedness indicates that we can never reach the end place if there is no token in other places. Safeness property in workflow domain will ensure there is no need of processing two same objects when one is needed. Liveness implies the absence of deadlocks. Deadlock property is an important from workflow modeling point of view as it indicates that the corresponding workflow has some activity which cannot be reached hence the design has some flaws.

## V. TEST CASE GENERATION

1) START -> A -> B -> C-> D -> E -> F -> G -> END
2) START -> A -> B -> C -> E -> F -> G -> END
3) START -> A -> B -> C -> F -> E -> G -> END
4) START -> A -> C -> D -> E -> F -> G -> END
5) START -> A -> C -> E -> F -> G -> END
6) START -> A -> C -> F -> E -> G -> END
7) START -> A -> B -> END

**Figure 4:- Generated test sequnces**

According to Petri net analysis workflow of concurrent activities generates two flows with same activities changing its sequence to test whether both activities can work together or not, even though they can not included in simple path set. According to authors in paper [7], they have selected only such test cases which cover all activities in simple path. Our work addresses these activity path coverage criteria.

The final test sequences as shown in figure 4 are generated by prioritizing them so that most important flows are tested. It will also reduce the duration of testing and ultimately it reduces testing cost. For every generated activity flow we use 'weight' of flow. If there are more than one combination of flows covering all the edges we need to select combination with highest flow 'weight'. For every edge in flow we assign a positive integer 'w' whose value may vary from 0 to 1. If activity has only one outgoing edge, weight of the outgoing edge is 1.

If node has n outgoing edges (for decision nodes), sum of the weight of all outgoing edges must be 1.

**Weight of a flow** is defined as the sum of all the weights of the edges in the flow. With this formula we can automatically find the paths covering maximum number of activities and transitions. Hence maximum activity coverage and transition coverage criteria also satisfied.

## VI. COMPARISON WITH RELATED WORK

No evidences found in investigated work that generates test cases with the use of only activity diagrams and petrinet concept. Modeling power and its decision power are the two major factors for the success of any model. First refers to the ability to correctly represent the system to be modeled whereas second refers to the ability to analyze the model and determine properties of the modeled system for correct implementation. In our work we used activity diagram for modeling purpose which shows workflow of system. Mapping same diagram into Petri net is used for formal analysis for testing purpose. Petrinet is the mathematical tool which can be used to analyze the behavior of complex system. Our approach targets black box testing involving interaction among activities. Our work is comparable with the work presented in [3, 4, 5, 6, 7], Linzhang et al. [3] propose a method to automatically generate test cases from UML activity diagrams using a gray-box method. In their method they generate test cases directly from UML activity diagrams. Their proposed method exploits the advantage of black box testing to analyze the expected external behavior and white box testing to cover the internal structure of the activity diagram of the system under test to generate test cases. In paper [4] authors have used petri net like semantics and considered basis path coverage to generate random test cases. Practically it is not possible to test large set of random test cases generated for large complex system. Also they have not addressed minimal loop testing and reducing number of paths. In [5, 6] paper authors addressed the criteria of activity path coverage but some concurrent paths are missing which should be tested. Also not provide any criteria for prioritizing the test cases whereas our approach generates prioritized test flows. In [7] paper author has used ORT relations but again here loop tesing is not done hence testing may have some flaws. None of mentioned papers checked the properties of system such as reachability, safeness, liveness, deadlock, boundedness. Hence they cannot completely specify whether their drawn activity diagram is correct or incorrect for implementation. Our approach has worked on those properties and can find all these properties in system through an activity diagram for complex real time embedded system.

## VII. CONCLUSION

We have presented a prototype tool to generate test cases automatically from UML activity diagrams. Our approach has addressed the different properties of Petri net and we can conclude that drawn activity diagram model of system is safe for implementation or not. Such system can contain deadlocks and we can detect them by specifying proper input to it.

## ACKNOWLEDGEMENT

Our special thanks to Department of Computer science and engineering of MIT College of engineering, Aurangabad.

## REFERENCES

[1]. Anders Hessel and Paul Pettersson, "A Test Case Generation Algorithm for Real-Time Systems" Proc. of 3rd International Workshop on Formal Approaches to testing of Software, number 2931 in Lecture Notes in Computer Science, pages 136–151. Springer–Verlag, 2003.
[2]. Kim H, Kang S, Baik J, Ko I "Test cases generation from UML activity diagrams." In: Software engineering, artificial intelligence, networking, and parallel/distributed computing 2007, pp 556–561.
[3]. W. Linzhang, Y. Jiesong, Y. Xiaofeng, H. Jun, L. Xuandong, and Z. Guoliang "Generating test cases from UML activity diagram based on gray-box method". In 11th Asia-Pacific Software Engineering Conference (APSEC04), pp. 284-291, 2004.
[4]. C. Mingsong, Q. Xiaokang, and L. Xuandong. "Automatic test case generation for UML activity diagrams." In 2006 international workshop on Automation of software test, pp. 2-8, 2006
[5]. M. Chen, P. Mishra, D. Kalita. "Coverage-driven Automatic Test Generation for UML Activity Diagrams", Proceedings of the 18th ACM Great Lakes symposium on VLSI, Orlando, Florida, USA, 2008.
[6]. Mingsong Chen · Prabhat Mishra · Dhrubajyoti Kalita "Efficient test case generation for validation of UML activity diagrams". Published online: 18 June 2010 © Springer Science+Business Media, LLC 2010.
[7]. Abdelkamel Hettab1, Allaoua Chaoui1, and Ahmad Aldahoud2 "AUTOMATIC TEST CASES GENERATION FROM UML ACTIVITY DIAGRAMS USING GRAPH TRANSFORMATION" 6th ICIT May 8, 2013.
[8]. A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Roveri. "NuSMV 2: An OpenSource Tool for Symbolic Model Checking", 2002.
[9]. Christel Baier and Joost-Pieter Katoen "Principles of Model Checking" The MIT Press Cambridge, Massachusetts London, England 2007.
[10]. Dianxiang Xu, L.M. Kristensen and L. Petrucci (Eds.)"A Tool for Automated Test Code Generation from High-Level Petri Nets" PETRI NETS 2011, LNCS 6709, pp. 308–317, 2011.© Springer-Verlag Berlin Heidelberg 2011
[11]. Jörg Desel And Andreas Oberweis, Torsten Zimmer, Gabriele Zimmermann "VALIDATION OF INFORMATION SYSTEM MODELS: PETRI NETS AND TEST CASE GENERATION."
[12]. H. Störrle, Structured Nodes in UML 2.0 Activities, Nordic Journal of Computing, Vol.11, No. 3, Sep 2004, pp. 279-302
[13]. H. Störrle, "Semantics of Control Flow in UML 2.0 Activities", Proc. of 2004 IEEE Symposium on Visual Languages and Human Centric Computing, USA, 2004, pp. 235-242
[14]. H. Störrle, J.H. Hausmann, "Reasoning about UML Activity Diagrams", Publ. Assoc. Nordic Journal of Computing, Vol. 14 No. 1, 2005, pp.43-64
[15]. T. Spiteri Staines, "Intuitive Mapping of UML 2 Activity Diagrams into Fundamental Modeling Concept Petri Net Diagrams and Colored Petri Nets", Proc. of the 15th ECBS conference, IEEE, Apr 2008, pp. 191-200
[16]. A. Spiteri Staines, "A TGG Approach for Mapping UML 2 Activities into Petri Nets", Proc. 9th WSEAS, SEPADS, Univ. of Cambridge, UK, Feb 2010, pp. 90-95
[17]. LaQuSo (2007). LaQuSo Work Group / Project, LaQuSo Repository, Eindhoven, www.Laquso.com
[18]. J.P. Lopez-Grao, J. Campos, "From UML Activity Diagrams to Stochastic Petri Nets" Application to Software Performance Engineering, WOSP'04, Redwood CA, 2004,pp. 25-26
[19]. N. Yang, H. Yu, H. Sun, Z. Qian," Mapping UML Activity Diagrams to Analyzable Petri Net Models" Proc. of the 2010 10th Int. Conf. on Quality Software, IEEE, Zhangjiajie,Jul 2010, pp. 369-372
[20]. N. Feng, W. Ming-Zhe, Y. Cui-Rong, T. Zhi- Gong, "Executable Architecture Modeling and Validation", 2nd Int. Conf. ICCAE, IEEE, Singapore, Feb 2010, pp. 10-14
[21]. J. L. Garrido, M. Gea, "A Colored Petri Net Formalization for a UML-based Notation Applied to Cooperative System Modeling", Interactive Systems: Design, Specification and Verification, LNCS 2545, Springer, 2002, pp.16-28
[22]. L. Baresi, M. Pezze, "Improving UML with Petri Nets, Electronic notes in Theoretical Computer Scie"nce, Elsevier, Vol 44., No. 2, Jul 2007, pp. 107-119.
[23]. Tadao Murata, "Petri Nets: Properties, Analysis and Applications", Proceedings of the IEEE, Vol. 77, No. 4
[24]. Rik Eshuis, Roel Wieringa. "A formal semantics for UML Activity Diagrams – Formalising workflow models", Technical Report CTIT-01-04, U. Twente, Dept. Of Computer Science, 2001.
[25]. Rik Eshuis, Roel Wieringa. "Verification support for workflow design with UML activity graphs", In Proc.24th Intl. Conf. on Software Engineering (ICSE'02), pages 166-176. IEEE, 2002.
[26]. Rik Eshuis, Roel Wieringa." A real time execution semantics for UML activity diagrams", In H. Hussmann, editor, Proc. 4th Intl. Conf. Fundamental approaches to software engineering (FASE'01), number 2029 in LNCS, pages 76-90. Springer Verlag, 2001.
[27]. Alan Hartman," MODEL BASED TEST GENERATION TOOLS" A G E D I S C O N S O R T I U M, www.agedis.de.
[28]. Andrea BOBBIO, A.G. Colombo and A. Saiz de Bustamante (eds.), "SYSTEM MODELLING WITH PETRI NETS" System Reliability Assessment,
Kluwer p.c., pp 102-143, (1990)
[29]. Jiacun Wang , "Petri Nets for Dynamic Event-Driven System Modeling"
[30]. Phillip Laplante, "Real time system analysis and design" 3rd edition, Wiley, IEEE Press.