Research Journal of Applied Sciences, Engineering and Technology 11(2): 169-175, 2015
DOI: 10.19026/rjaset.11.1704
ISSN: 2040-7459; e-ISSN: 2040-7467

Submitted: March 31, 2015 Accepted: June 14, 2015 Published: September 15, 2015

Research Article

Formalizing Semantics for UML Activity Diagram through Regular Expression Translation

[1]Bramah Hazela, [1]Deepak Arora and [2]Vipin Saxena
[1]Department of Computer Science and Engineering, Amity University,
[2]Department of Computer Science, Babasaheb Bhimrao Ambedkar University, Lucknow, India

Abstract: Formalization of UML models now becomes a requisite action by most of the software designers. UML is semiformal in nature. So it becomes necessary to formalize the UML which would reduce the overall complexity of software design. Today as software becoming more interactive and distributed in nature, the formal syntax and automated verification analysis of behavioral aspect of any model becomes very important in order to reduce overall software development cost and time. UML Activity diagram has become widely acceptable tool for documenting the artifacts related to Control flow and complexity of the software system. Here Authors proposed the semantics for activity diagram of UML by means of regular expression and its equivalent transition system. UML has now become one of the most widely acceptable standards for visual modeling related to object based software development. Since inception, continuous adoption of various design patterns and profiles of software have been included to make it more flexible and capable to represent different views of software design at early phases of its development. It is also found that the mapping of these visual modeling structures to some pre-established formal graphical notations of data structures like graph certainly provides more realistic and robust automated verification and validation ground for these models. The available literature shows the tremendous research work is being carried out to make it more adoptable and reliable visual modeling platform across the globe. Although UML has a richer and wider visual modeling skill set, but still it is not very easy to find better ground for establishing, set of rules and semantics for UML model verification and validation. The research work also proposes a formal verification and traceability method for any activity model with the help of Arden's lemma. The correctness of proposed verification method has been shown with supporting case studies after generating its corresponding formal regular expression.

Keywords: Arden's lemma, finite state machine, regular expression, transition system, UML activity diagram

INTRODUCTION

Unified Modeling Language (UML) is a framework or platform that is used for writing blueprints of software by industry experts. UML is process independent framework. Unified Modeling Language (UML) (Rumbaugh *et al*., 2001) specification provides the details of various software systems having varying complexity and hardware designs (Martin and Müller, 2005; Berlin Müller *et al*., 2006). UML provides a set of graphical notation to visualize different views of a system. UML models can also be customized for a specific domain. UML activity diagram is a widely accepted tool for software designing at the system level. The semi-formal specification of the activity diagram is specifically useful to describe the concurrent behavior of complex logic operations of software systems.

In past, various verification techniques have been proposed to validate software design through automated generation of test case (Ammann *et al*., 1998).

Traditional verification and validation methods of software system are less efficient as they are basis of handwork. These software methods took more time to developed and hence increased of overall development cost. Now-a-days, software testing highly demands its automaton due to development of testing advancement. The effective and maintainable testing of software can only be guaranteed through automated generation of test cases. Software testing industries are now choosing a methodology known as Model based testing. The main potential advantage of this testing approach includes the faults detection in software at early stage and reducing the overall time for the development of software. In Literature review, different diagram of UML has been considered for generation of various test

Corresponding Author: Bramah Hazela, Department of Computer Science and Engineering, Amity University, Lucknow, India, Tel.: +91-09839123463

cases (Wang *et al*., 2005; Li and Lam, 2005; Chandler *et al*., 2005; Chen *et al*., 2006; Xu *et al*., 2005).

Activity diagram of UML are most suitable for the study of behavioral aspects of a given system. Behavior modeling for a service-oriented architecture system and large software system can be allowed through workflows specifications (Eshuis *et al*., 2002; Alonso *et al*., 2004).

Over the past decade, in Software development process it has been found that software models can be best expressed by Unified Modeling Language (UML). One of the limitations of UML is that it lacks formal semantics as it focuses only on syntax for system modeling. Today, implementation of formal methods is a very hard task to automate software engineering by software designers because of complex mathematical theory behind them. However, formal semantics based on UML specification have already been discussed (OMG, 2007; Baresi and Heckel, 2002; Ehrig *et al*., 1999).

A research work on formal semantics for activity diagram had already been given in many research work using different formal languages. Phenomenon of Dynamic Meta Modeling was explained by Hausmann (2005). This modeling was done by graph transformation system. In his study, Hausmann (2005) defined a concept named "Rule Invocation" to extend the traditional graph rules. DMM is compromise of two rules named as big and small step rules. The rules of Big-step are traditional in nature that invoked small-step rules. Author also used the concept of Dynamic Meta Modeling to define the semantics for activity diagram. DMM and semantics were used by Engels *et al*. (2007) for modeling and workflows verification. In their study, author change the rules of GROOVE (Rensink, 2004) which was also verified by GROOVE, to check the action reachability and deadlock freeness properties of work flows. In reference to the previous work, the given approach provides more flexibility for system verification properties. Rule Invocation and different Small or big step rules defined by Hausmann (2005) can't be modeled directly by the use of existing tool by the designers. Petri-net was used by Störrle and Hausmann (2005) that provides semantic background for the UML activities.

In their work, authors examine the activities described in the UML through the de-notational semantics.

This approach describes basic data and control flow, exception handling and expansion nodes. Authors also found that some standard constructs are not easy to be formalized by Petri nets. So, they concluded that Petri nets cannot be possibly used to complete semantics in UML 2.0 due to transverse. Strong fairness property was used to verify the functional requirement of state chart-like semantics for activity diagrams of UML 1.5. This property states that the infinite loops should be avoided by model as discussed by Eshuis (2006). Cimatti *et al*. (2000) has suggested methods for verification and correctness of the strong fairness property in an expression of LTL where NuSMV model checker is used. The approaches discussed in Börger *et al*. (2000) and Bolton and Davies (2000) are the predecessor of UML 1.5. Baldan *et al*. (2005) define static model of a system by using instance graph. They used the hyper graph with synchronized feature to control the applications of the rules that are defined for each action. Instead of defining the semantics for activity, Synchronized Hyper graphs were used to show the dynamic model behavior by using UML activities. They implemented monadic second-order logic for the verification of hyper graphs without introducing any tools. Furthermore, they found that activity diagrams are largely useful to describe the flow of events. This assignment is necessary for transformation of test case information into activity diagram. It generates an activity graph in which every edge represent the flow in activity diagram and each node representing a test case construct from activity diagram. UML is semiformal in nature and formalization of the UML diagram has now become the dominant area of research.

In this research work, authors have proposed a model transformation approach in order to generate regular expression, corresponding to any UML activity diagram. The target regular expression will be suitable for verification and determining the consistency of the system. The proposed method represents the relationship of UML activity diagram with well establish mathematical structure like activity graphs. This relationship can further be useful to generate equivalent transition diagram for a given activity model. The authors have covered the concept of strings, languages and regular expression of computation theory to establish the relation between finite state automata and UML activity graph. Authors have also explained a transition system by referring the transformation approach of UML activity diagram into state diagram. The main idea of Activity diagram formalization has given by discussing the Arden's theorem for generating regular expression from the transition system. The result of this formalization will reduce the overall software development complexity.

MATERIALS AND METHODS

Authors have given the concept of UML activity diagram along with its equivalent transition system. Further, based on the existing concepts of formal language theory, different patterns of strings generated by regular expressions of a transition system have also explore by means of different case studies.

Activity diagram: UML Activity Diagrams refer to a method of software engineering that is modeled to describe business processes, procedures and work flows. Activity diagram of UML is a kind of behavioral modeling. Any UML Activity Diagram may have its equivalent UML activity graph and both are somewhat similar to state machine diagrams.

Modeling of different computational as well as organizational processes can be done through Activity diagram of UML. Activity diagrams show the flow of activities in a stepwise manner, constructed with small number of shapes and are connected through arrows. The most important shape type includes:

- Actions denoted by rectangles having rounded corner. Diamond shape represents decisions during activity.
- Start (split) or end (join) of concurrent activities represents by horizontal Bars.
- Initial state or start of the workflow is represented by black circle.
- Solid black fill encircled represents the end or final state.
- Arrows represent the order in which activities happen from start to end state.

Transformation of Activity diagram of UML into Finite State Machine for description of activity diagram semantics have already been specified by Object Management Group (2005) and Friedl (2006).

Strings, languages and regular expressions: Theory of formal languages is the base for any programming languages. Formal theory refers to the fact that all the rules for the languages are stated explicitly in terms of what strings of symbols can occur. A Symbol is an abstract or user defined identity. Letters and digits are example of a set of symbols, which is finite in nature and is also known as alphabet, for example, x = {*a*, *b*} is an alphabetic set with symbols *a* and *b*. Another alphabet is *Y* = {*tiger*, *elephant*, *snake and python*} is a set of alphabet with symbols tiger, elephant, snake and python. A String can be defined as a finite sequence of symbols from that alphabet. Strings play an important role in testing of patterns. Finite state machine takes strings as inputs and test for its acceptability. Thus a string serves as a test input. A language is a collection of strings. For example, the language of binary numbers is the combination of all strings defined over zeros and ones. Total number of symbols count in a string gives the length of string which is written as |*s*|. For example, |1010| = 4 and |*hot, pot*| = 3. An empty String must be of length 0, which can be written as $\in$. Concatenation of two strings s_1 and s_2 is denoted by $s_1 \bullet s_2$. For example, given the alphabet *X* = {*a*, *b*} and two strings *abb* and *bab* over *X*, then concatenation of two strings would be *abb***.** *bab* = *abbbab*.

A Set of strings can be described through Regular expression based on certain syntax rules (Raschke, 2009). Regular expressions are used by most of the application software for searching and manipulating text patterns as in word processors and programming languages. Short description of a set can be given as Regular expressions, without elements listing. For example, a set of two strings *"green"* and *"grean"* would be a regex *"gre* [*ea*] *n"*. Regular expression

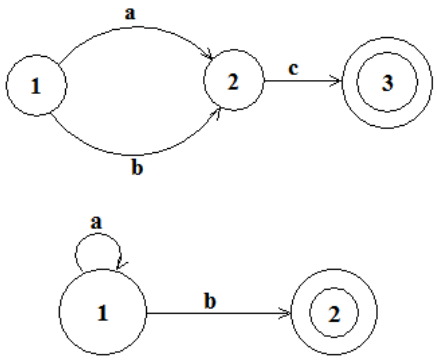

Fig. 1: Language and its equivalent regular expression

provides a compact representation of sets of strings. For example, the regular expression (*ab*)* represent the set of strings that consists of the empty string, the string *ab* and all strings obtained by concatenating the string ab with itself one or more times. Expression (*ab*)* denotes an infinite set. Formally, a regular expressions, generates various patterns of strings that can be recognized by finite state automata. Suppose *X* is a set of finite alphabet, then regular expressions over *X* are given as here:

If $p \in X$, then *'p'* is a regular expression that represent the set {*p*}. If we consider regular expressions Re_1 and Re_2 over the alphabet *X* that represents the two set L_1 and L_2, respectively, then concatenation of two regular expressions written as Re_1. Re_2 is also regular expression represented by the set $\{L_1 \bullet L_2\}$. Similarly, Union of two regular expressions Re_1, Re_2 written as $Re_1 + Re_2$ is also a regular expression represented by the set $\{L_1 \cup L_2\}$. If Re_1 is a regular expression, then Re_1* is also a regular expression known as Kleen closure of Re_1.

Regular expressions are useful in expressing both finite and infinite test sequences. For example, if a program takes a sequence of zeroes and ones and flips from 0 to 1 and vice versa, then a few possible sets of test inputs are 0*, (10)+010|100. Regular expressions are also useful in defining the all possible inputs to a finite state machine that will give transition of the machine from one arbitrary state to another state. Every regular expression *'Re'* can be recognized by a given transition system and finite automaton. For further explanation, consider two different languages and their finite state automaton that accepts the language defined by regular expression as given in Fig. 1:

Example: 1
L = (*a*|*b*) *c*
ex. {*ac*, *bc*}

Example: 2
L = *a** *b*
ex. {*b*, *ab*, *aab*, *aaab*, …….}

For every regular expression *'Re'*, there exists a corresponding finite automaton that accepts the regular

set given by *'Re'* due to equivalent expressive power of Regular expressions and finite automata.

So we can say that regular languages, regular expressions and finite automata are all different representation of the same string. In this reference, Arden's Theorem defines the transformation rules and semantics to convert a transition system to its corresponding regular expression. According to Arden's method, let the P and Q are regular expressions over alphabet Σ and if P does not have null string, then given equation in *'R'*, written as $R = (Q+RP)$ have a solution $R = QP^*$. This unique solution can be found by substituting the value of $R = QP^*$ in the R.H.S. of $R = Q + RP$, further $R = Q+(Q+RP)\ P = Q+QP+RP^2$, Now by substituting the value of R again and again, we will get the following set of equations:

$$R = Q+QP+QP^2+QP^3.....$$
$$R = Q\ (1+P+P^2 + P^3+ .....)$$
$$R = Q\ (\epsilon+P+P^2+P^3+ .....)$$

The second part of the product on the L.H.S can be replaced with the help of kleen closure property. So the final equation becomes $R = QP^*$, which is unique solution as stated above. For getting regular expression for the given automata we first create equations of the given form for all the states:

$$q_1 = q_1w_{11}+q_2w_{21}+...+q_nw_{n1}+\epsilon$$
(q_1 *is the initial state*)
$$q_2 = q_1w_{12}+q_2w_{22} + ...+q_nw_{n2}+.....$$
$$q_n = q_1w_{1n}+q_2w_{2n}+...+q_nw_{nn}$$

Here, w_{ij} is the regular expression representing the set of labels of edges from q_i to q_j. For parallel edges, we find similar expressions for all states in the expression. Then, these equations are solved to get the equation for q_i in terms of w_{ij} and that expression is the required solution, where q_j is a final state.

RESULTS AND DISCUSSION

Formalization and Generation of Regular Expression is shown by a simple login process as in Fig. 2.

The user will enter the login name and password. And then, these details must be validated by the system to check the correctness of unique pairing, user name and password. In case of valid entry, system allows to login the user. In case of invalid information, system will ask from its user to re-enter their credentials.

Based on notion of semantics for translation of UML Diagrams into finite state machines for Model Checking (Deepak *et al*., 2012; Raschke, 2009) and formalization of UML activity diagram using finite state machine (Rodrigues, 2000), it is concluded that a transition system can be drawn from the given UML activity diagram as shown in Fig. 3.

Through massive literature survey it can be concluded that regular expressions are one of the important algebraic expression in theory of computation that can also be implemented by Finite State Automaton. A Finite State Automata is a significant tool of computational linguistics. A regular expression is a formula in a special language that is used for characterizing a set of strings known as pattern. Any regular expression can be modeled through finite state automaton. Based on the Fig. 2 one can arrive at the following set of equations:

$$q_1 = q_{21}y_3+ \Lambda \quad (1)$$

(As q_1 is the initial state a *'* Λ *'* should be added by Arden's theorem for generating regular expression from transition system):

$$q_2 = q_1y_1 \quad (2)$$

$$q_{21} = q_2n_1 \quad (3)$$

$$q_{22} = q_2y_2 \quad (4)$$

$$q_3 = q_{22}y_4 \quad (5)$$

$$q_4=q_{21}n_2+q_3y_5 \quad (6)$$

From Eq. (2):

$$q_2 = (q_{21}y_3+ \Lambda)\ y_1 = (q_{21}y_3y_1+y_1)$$

Or,

$$q_2 = y_1+q_{21}y_3y_1$$

Or,

$$q_2 = y_1+q_2n_1y_3y_1$$

By Eq. (3):

$$q_2 = y_1+q_2\ (n_1y_3y_1) \quad (7)$$

By Arden's theorem for generating regular expression from the transition system, if regular expression is of the form of $R = Q+RP$ then the unique solution of this regular expression is $R = QP^*$:

$$q_2 = y_1\ (n_1y_3y_1)^* \quad (8)$$

From Eq. (6), we get:

$$q_4 = q_{21}n_2+q_3y_5 = q_2n_1n_2+q_3y_5$$

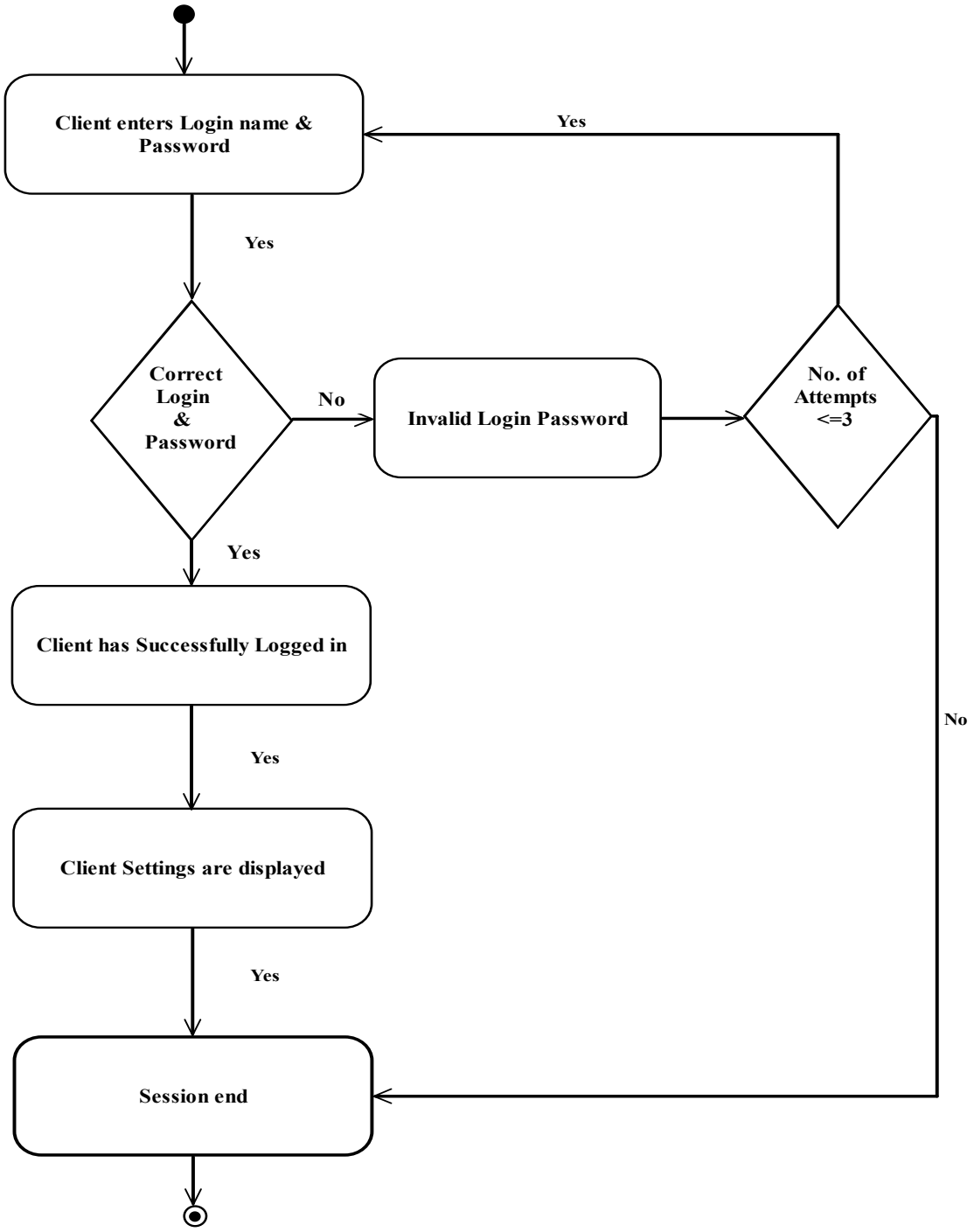

Fig. 2: Activity diagram for login process

By Eq.

$$= q_2 n_1 n_2 + q_{22} y_4 y_5 \quad (9)$$

By Eq.

$$= q_2 n_1 n_2 + q_2 y_2 y_4 y_5 \quad (10)$$

By Eq.

By substituting of q_2 from Eq. (7), we get the Regular expression as:

$$q_4 = y_1 (n_1 y_3 y_1)^* n_1 n_2 + y_1 (n_1 y_3 y_1)^* y_2 y_4 y_5$$

This expression gives the desire sequence of activities performed throughout the login process. The complete regular expression can be verified through the following two cases:

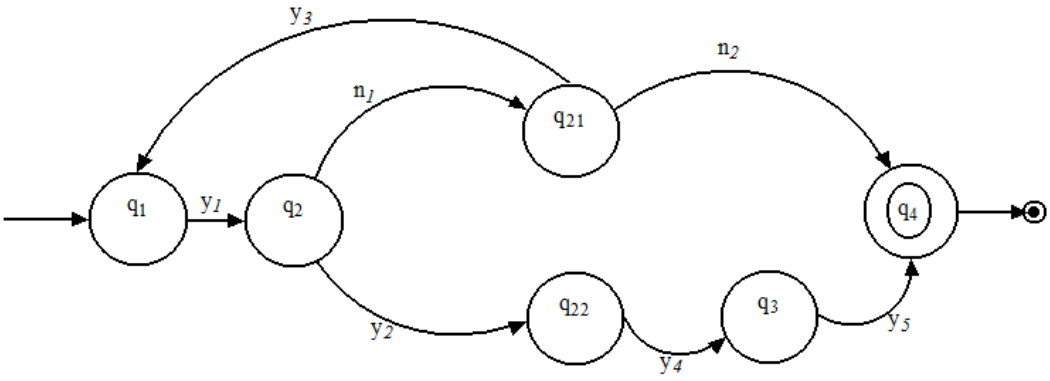

Fig. 3: Transition system for login process

Case 1: First activity includes login name and password entered by user then transit to next activity which required further validation. If the login and password entered by the user is incorrect, then system allows two more attempts before the termination of activity. As soon as the valid attempts exhaust, the system will terminate the session. This sequence of activities can be best described by the regular expression, $y_1 (n_1y_3y_1)^*n_1n_2$. This expression is concatenated form of y_1 and $(n_1y_3y_1)^*n_1n_2$, which confirms that without capturing the desire inputs for login, the next sequence of activities can't be performed.

Case 2: The inputs to the system may leads to transition of new activity. If the client successfully logged into the system, the client settings of the system are displayed for further customization. If the client does not require customization there then at this juncture there remains an option of logout from the system by terminating the session. This sequence of activity can be described by the regular expression, $y_1(n_1y_3y_1)^*y_2y_4y_5$. This expression confirms that with correct login and password entered by the user the end user can customized their settings.

CONCLUSION

In this study, authors have established a set of semantics to generate corresponding regular expressions for a UML activity diagram.

These semantics proves that the verification can be done by the means of generated regular expressions. The use of Arden's theorem make it more concise in order to assure the generation of corresponding regular expression without transition system. This will reduce overall effort towards verification of any UML activity diagram.

For illustration purpose different case studies have been shown to prove the correctness of proposed methodology.

These regular expressions can be further helpful in verifying the UML activity diagrams at early stages of software development. The language generated by these regular expressions is also useful in generating and validating different test cases.

The proposed research work can be extended towards other behavioral diagrams for the development of important phase of compiler design known as lexical analysis.

ACKNOWLEDGMENT

The authors are very grateful to respected Mr. Aseem Chauhan, Chairman, Amity University Lucknow and Maj. Gen. K.K. Ohri, AVSM (Retd.), Pro-VC, Amity University, Lucknow Uttar Pradesh, India, for providing excellent research infrastructure facilities in the University campus. Authors also pay their best regards to Wg. Cdr. (Dr.) Anil Kumar, Director Amity School of Engineering and Technology, AUUP, Lucknow, Prof. S.T.H. Abidi and Brig. U.K. Chopra, Director Amity Institute of Information Technology, Amity University, Lucknow for giving their motivational support and help to carry out the present research work.

REFERENCES

Alonso, G., F. Casati, H. Kuno and V. Machiraju, 2004. Web Services: Concepts, Architectures and Applications. Springer-Verlag, Berlin, Heidelberg, pp: 354.

Ammann, P., P. Black and W. Majurski, 1998. Using model checking to generate tests from specifications. Proceeding of the International Conference on Formal Engineering Methods (ICFEM), pp: 46-54.

Baldan, P., A. Corradini and F. Gadducci, 2005. Specifying and verifying UML activity diagrams via graph transformation. In: Priami, C. and P. Quaglia (Eds.), GC, 2004. LNCS, 3267, Springer-Verlag, Berlin, Heidelberg, pp: 18-33.

Baresi, L. and R. Heckel, 2002. Tutorial introduction to graph transformation: A software engineering perspective. In: Corradini, A. *et al*. (Eds.), ICGT,

2002. LNCS 2505, Springer-Verlag, Berlin, Heidelberg, pp: 402-429.

Bolton, C. and J. Davies, 2000. On giving a behavioural semantics to activity graphs. In: Evans, A., S. Kent and B. Selic (Eds.), UML 2000. LNCS 1939, Springer, Heidelberg.

Börger, E., A. Cavarra and E. Riccobene, 2000. An ASM semantics for UML activity diagrams. In: Rus, T. (Ed.), AMAST 2000. LNCS 1816, Springer-Verlag, Berlin, Heidelberg, pp: 293-308.

Chandler, R., C.P. Lam and H. Li, 2005. An automated approach to generating usage scenarios from UML activity diagrams. Proceeding of the 12th Asia-Pacific Software Engineering Conference.

Chen, M., X. Qiu and X. Li, 2006. Automatic test case generation for UML activity diagrams. Proceeding of the International Workshop on Automation of Software Test (AST '06).

Cimatti, A., E. Clarke, F. Giun Chiglia and M. Roveri, 2000. NuSMV: A new symbolic model checker. Int. J. Softw. Tools Technol. Trans., 2(4): 410-425.

Deepak, A., H. Bramah and S. Vipin, 2012. Semantics for UML model transformation and generation of regular grammar. ACM SIGSOFT, 37: 1-5.

Ehrig, H., G. Engels, H.J. Kreowski and G. Rozenberg, 1999. Handbook on Graph Grammars and Computing by Graph Transformation: Vol. 2: Applications, Languages and Tools. World Scientific Publishing Co. Inc., River Edge, NJ, USA.

Engels, G., C. Soltenborn and H. Wehrheim, 2007. Analysis UML activities using dynamic meta modeling. Proceeding of the 9th IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS), 4468: 76-90.

Eshuis, R., D. Jansen and R. Andwieringa, 2002. Requirements-level semantics and model checking of object-oriented statecharts. Requir. Eng. J., 7: 243-263.

Eshuis, R., 2006. Symbolic model checking of UML activity diagrams. ACM T. Softw. Eng. Meth., 15(1): 1-38.

Friedl, J.E.F., 2006. Mastering Regular Expressions. O'Reilly, New York, ISBN: 0596528124.

Hausmann, J.H., 2005. Dynamic META modeling: A semantics description technique for visual modeling languages. Ph.D. Thesis, University of Paderborn, Germany.

Li, H. and C.P. Lam, 2005. Using anti-ant-like agents to generate test threads from the UML diagrams. In: Khendek, F. and R. Dssouli (Eds.), TestCom 2005. LNCS 3502, Springer-Verlag, Berlin, Heidelberg, pp: 69-80.

Martin, G. and W. Müller, 2005. UML for SOC Design. Springer, US.

Müller, W., A. Rosti, S. Bocchio, E. Riccobene, P. Scandurra, W. Dehaene and Y. Vanderperren, 2006. UML for ESL design-basic principles, tools, and applications. Proceeding of the IEEE/ACM International Conference on Computer-Aided Design, pp: 73-80.

Object Management Group, 2005. UML Specification 2.0. Retrieved from: http://www.omg.org/technology/documents/modelingspeccatalog.htm.

OMG, 2007. UML Superstructure V2.1.2. Retrieved from: http://www.omg.org/spec/UML/2.1.2/.

Raschke, A., 2009. Translation of UML 2 activity diagrams into finite state machines for model checking. Proceeding of the 35th Euro Micro Conference on Software Engineering and Advanced Applications (SEAA, 2009), pp: 149-154.

Rensink, A., 2004. The GROOVE simulator: A tool for state space generation. In: Pfaltz, J.L., M. Nagl and B. Böhlen (Eds.), Applications of Graph Transformations with Industrial Relevance (AGTIVE). LNCS 3062, Springer-Verlag, Berlin, Heidelberg, pp: 479-485.

Rodrigues, R.W.S., 2000. Formalising UML activity diagrams using finite state processes. Online Proceeding of the UML 2000 Workshop on Dynamic Behaviour in UML Models: Semantic Questions.

Rumbaugh, J., I. Jacobson and G. Booch, 2001. The Unified Modeling Language User Guide. Addison Wesley, Boston.

Störrle, H. and J.H. Hausmann, 2005. Towards a Formal Semantics of UML 2.0 Activities. In: Liggesmeyer, P., K. Pohl and M. Goedicke (Eds.) Software Engineering. LNI, GI, 64: 117-128.

Wang, L., J. Yuan, X. Yu, J. Hu, X. Li and G. Zheng, 2005. Generating test cases from UML activity diagram based on gray-box method. Proceeding of the 11th Asia-Pacific Software Engineering Conference, pp: 284-291.

Xu, D., H. Li and C.P. Lam, 2005. Using adaptive agents to automatically generate test scenarios from the UML activity diagrams. Proceeding of the 12th Asia-Pacific Software Engineering Conference.