*ICIT 2015 The 7th International Conference on Information Technology*
doi:10.15849/icit.2015.0093 

# *UML Activity Diagrams and Maude Integrated Modeling and Analysis Approach Using Graph Transformation*

*Elhillali Kerkouche, Khaled Khalfaoui*
Dept. Computer Science, University of Jijel,
MISC Laboratory, University Constantine 2,
Algeria
{elhillalik, Kh-khalfaoui}@yahoo.fr

*Allaoua Chaoui*
Dept. Computer Science and its Applications,
MISC Laboratory,
University Constantine 2,
Algeria
chaoui@misc-umc.org

*Ali Aldahoud*
Al-Zaytoonah University of Jordan,
P.O. Box 130, Amman 11733,
Jordan
aldahoud@zuj.edu.jo

***Abstract*—The use of UML Activity Diagrams for modeling global dynamic behaviors of systems is very widespread. UML diagrams support developers by means of visual conceptual illustrations. However, the lack of firm semantics for the UML modeling notations makes the detection of behavioral inconsistencies difficult in the initial phases of development. The use of formal methods makes such error detection possible but the learning cost is high. Integrating UML with formal notation is a promising approach that makes UML more precise and allows rigorous analysis. In this paper, we present an approach that integrates UML Activity Diagrams with Rewriting Logic language Maude in order to benefit from the strengths of both approaches. The result is an automated approach and a tool environment that transforms global dynamic behaviors of systems expressed using UML models into their equivalent Maude specifications for analysis purposes. The approach is based on Graph Transformation and the Meta-Modeling tool AToM$^3$ is used. The approach is illustrated through an example.**

***Keywords— UML Activity Diagrams; Rewriting Logic;Maude language; Meta-Modeling; Graph Grammars; Graph Transformation; AToM3.***

## I. INTRODUCTION

The Unified Modeling Language (UML) [1] has become a widely accepted standard in the software development industry. Some diagrams are used to model the structure of a system while others are used to model the behavior of a system. UML Statecharts, UML collaboration diagrams, UML Sequence Dsiagrams and UML Activity diagrams are used to model the dynamic behavior in UML. UML State chart diagrams model the lifetime (states life cycle) of an object in response to events. A UML Collaboration diagram models the interaction between a set of objects through the messages (or events) that may be dispatched among them. UML Sequence Diagrams describe an interaction by focusing on the sequence of messages (or events) that are exchanged, along with their corresponding occurrence specifications on the lifelines. UML Activity diagram model the global dynamic behavior of systems in term of control flow or object flow with emphasis on the sequence and conditions of the flow. UML Activity diagrams are widely used to model workflow systems, service oriented systems and business processes. Control flow includes support for sequential, choice, parallel and events. Activities may be grouped in sub-activities and can be nested at different levels. However, the UML is a semiformal language which lacks rigorously defined constructs.

Rewriting logic has sound and complete semantics [2] and it is considered as one of very powerful logics in description and verification of concurrent systems. Also, the rewriting logic language Maude [3] is considered as one of very

doi:10.15849/icit.2015.0093 © ICIT 2015 (http://icit.zuj.edu.jo/ICIT15)

powerful languages based on Rewriting logic. However, Maude system offers textual way to the user to create and deal with systems. Execution under Maude system is done by using command prompt style. In this case, the user looses the graphical notations which are important for the clarity, simplicity and readability of a system description.

In this context, UML and Maude language have complementary features: UML can be used for modeling while Maude can be used for verification and analysis. Thus, developing a tool support for modeling and analysis of complex concurrent systems is significant to modelers who use UML to model their systems. UML behavioral models are projected automatically into Maude specifications for analysis and verification to detect behavioral inconsistencies like deadlock, imperfect termination, etc. Then, the results of the formal analysis can be back-annotated to the UML models to hide the mathematics from modelers.

In this paper, we propose a modeling tool and Graph Transformation approach for modeling and verification of global dynamic behavior in UML models using Maude language. Building a modeling tool from the scratch is a prohibitive task. Meta-Modeling approach is useful to deal with this problem, as it allows the modeling of the formalisms themselves [4]. A model of formalism should contain enough information to permit the automatic generation of tool to check and build models subject to the described formalism's syntax. In order to get a more general transformation approach between UML and Maude, we research the transformation at the Meta-Model level. And for reaching an automatic and correct process, we use Graph Transformation Grammars and Systems to define and implement the transformation. Using our approach, the modelers specify the global dynamics of a system by means of UML Activity diagrams. Then, the modelers transform automatically their behavioral specification into its equivalent Maude specification. From the obtained formal specification, they can use Maude Model Checker to verify their models.

With this end, we have defined a simplified Meta-Model for UML Activity diagrams using AToM$^3$ tool [5]. Then, we have used this Meta-Modeling tool to automatically generate a visual modeling tool for UML Activity diagrams according to its proposed meta-model. For the transformation process, we have defined a graph grammar to translate the UML Activity diagrams created in the generated tool to a Maude specification. Then the rewriting logic language Maude is used to perform the verification of the resulted Maude specification.

The rest of this paper is organized as follows. Section 2 outlines the major related work. In section 3, we review the main concepts of UML Activity diagrams, Rewriting logic, Maude language and graph transformation. In section 4, we describe our approach that transforms a UML Activity diagrams to Maude specification. In section 5, we illustrate our approach using an example. The final section concludes the paper and gives some perspectives.

## II. RELATED WORKS

In the literature, several research works has been done about the integration of different UML diagrams and formal methods such as Petri nets [6] [7] [8], Colored Petri nets (CPN) [9], Object-Z [10], B method [11], LOTOS, Communicating Sequential Processes (CSP) [12] and Maude [13].

For the formalization of UML Activity Diagrams, the most important approaches use CSP or CPN formalisms. In [14], the authors present a case study of UML Activity Diagram to CSP transformation using graph transformation. In [15], the authors describe how an UML activity diagram can be transformed into a corresponding CSP expression by using the graph rewriting language PROGRES. In [16], the author explains how activity semantics are translated into colored Petri net semantics.

On the other hand, the rewriting logic language Maude offers the advantage of its sound and complete semantics [2] and it is considered as one of very powerful languages in specification, programming and verification of non-deterministic concurrent systems. In this paper, UML Activity Diagram semantics are defined in terms of rewriting logic. Rewriting logic gives to UML Activity Diagram a simple, more intuitive and practical textual version to analyze, without losing formal semantic (mathematical rigor, formal reasoning).

## III. BACKGROUND

### A. UML Activity Diagrams

UML Activity Diagram is one of the important UML models. It is utilized to describe an operation step by step in a system. Moreover, it models the overall control flow between activities and its relationships among several objects with a lot of parallel process. It supports the following concepts: choice, iteration and concurrency. Its structure is a connected graph in which the nodes are represented by icons and the edges by connections. An Activity Diagram includes the following constructs: Initial Node, Flow Final node, Activity Final node, Decision Node, Merge Node, Fork Node Join Node and transition. Only the last construct is represented by a connection; the others are represented by icons. These constructs are shown in Figure 1.

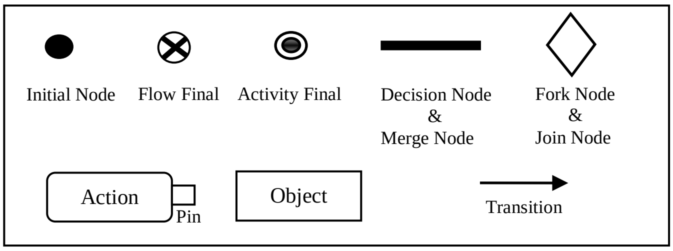



Fig. 1. UML Activity Diagram constructs.

### B. Rewriting Logic & Maude Language

In Rewriting Logic, each concurrent system can be specified by a rewriting theory. A rewrite theory is defined as a 4-tuple (Σ, E, L, R), where the signature (Σ, E) is an equational theory, L is a set of labels and R is a set of possibly conditional labeled rewrite rules that are applied modulo the equations E.

doi:10.15849/icit.2015.0093 

An important consequence of the RL definition is that the rewrite theory can be viewed as an executable specification of the concurrent system that it formalizes. The state is represented by an algebraic term, the transition becomes a rewriting rule and the distributed structure is expressed as an algebraic structure. For more information on the subject see [17].

Maude is a specification and programming language based on Rewriting Logic [18]. It integrates an equational style of functional programming with Rewriting Logic computation. Maude's implementation has been designed with the explicit goals of supporting executable specification and formal methods applications. Three types of modules are defined in Maude specification: The functional modules, the system modules and the object oriented modules. In this work, we will use only functional and system modules

**Functional Modules:** Functional modules define data types and operations on them by means of equational theories. In other words, Functional modules can be seen as an equational-style functional program with user definable syntax in which a number of sorts, their elements, and functions on those sorts are defined.

**System Module:** The system module defines the dynamic behavior of a system. It augments the functional modules by the introduction of rewriting rules. A rewriting rule specifies a local concurrent transition which can proceed in a system. The execution of such transition, specified by the rule, can take place when the left part of a rule matches to a portion of the global state of the system and the condition of the rule is valid. This type of module augments the functional modules by the introduction of rewriting rules.

In addition, Maude integrates a model checker. Model-checking is an automatic method for deciding if a specification satisfies a set of properties (for more details, see [19]).

### C. ATo$M^3$ & Graph Grammar

ATo$M^3$ [5] is a visual tool for Multi-formalism Modeling and Meta-Modeling. By means of Meta-Modeling, we can describe or model the different kinds of formalisms needed in the specification and design of systems. Based on these descriptions, AToM3 can automatically generate tools to manipulate (create and edit) models in the formalisms of interest [20].

ATo$M^3$'s capabilities are not restricted to these manipulations. AToM3 also supports graph rewriting system, which uses Graph Grammar to visually guide the procedure of model transformation. Graph Grammar [21] is a generalization of Chomsky grammar for graphs. It is a formalism in which the transformation of graph structures can be modeled and studied. The main idea of graph transformation is the rule-based modification of graphs as shown in Fig.1.

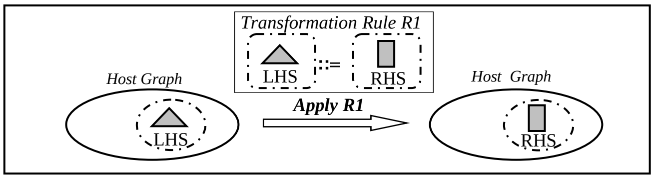



Fig. 2. Rule-based Modification of Graphs.

Graph Grammars are composed of production rules, each having graphs in their left and right hand sides (LHS and RHS). Rules are compared with an input graph called host graph. If a matching is found between the LHS of a rule and a subgraph in the host graph, then the rule can be applied and the matching subgraph of the host graph is replaced by the RHS of the rule. Furthermore, rules may also have a condition that must be satisfied in order for the rule to be applied, as well as actions to be performed when the rule is executed. A graph rewriting system iteratively applies matching rules in the grammar to the host graph, until no more rules are applicable.

## IV. Our Approach

The proposed approach consists of transforming a UML Activity diagram to Maude specification. To reach this objective, we have proposed a meta-model for UML activity diagram and a graph grammar that performs automatically the transformation of a UML Activity diagram. In this work, we focus on control flow which addresses the control part of UML Activity diagram, and we leave the object flow for future works. In the following, we describe in details our approach.

### A. Meta-modeling

To Meta-model Activity diagrams, we proposed the simplified meta-model containing thirteen classes linked by seven associations and twelve inheritances as shown in Figure 3. Each association of this meta-model links an instance of the source class with a single instance of the destination Class. Some classes are described as follows:

***ActionNode*** Class: represents the Action constructs of the diagram. Graphically it is represented by a rectangle with rounded corners. An Action node has *Name* attribute, and it can be connected with all control nodes, others Action nodes, Object nodes or Pin nodes.

***InitialNode*** Class: represents the beginning of an activity diagram. Graphically it is represented by a small solid circle. It has a constraint which prohibits the existence of incoming Arcs.

To fully define our meta-models, we have also specified the graphical appearance of each entity of the formalisms according to its appropriate graphical notation (shown in Figure 1). Given our meta-model, we have used AToM3 to generate a palate of buttons allowing the user to create the constructs defined in meta-model (see Figure 5).

doi:10.15849/icit.2015.0093 

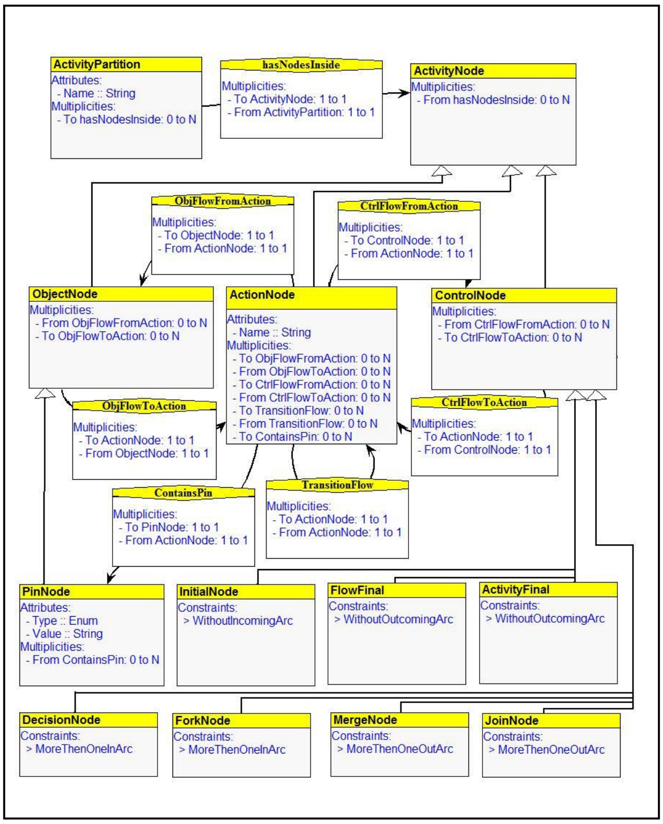



Fig. 3. Simplificed UML Activity Diagram Meta-Model

### B. *Representation of UML Activity Diagram in Maude*

In this section, we will explain how to express a UML Activity Diagram in Maude language by using two Modules. We define first a *Basic_ActivityDiagram* functional module that describes basic operations of *Activity Diagram*. This module is described as follows:

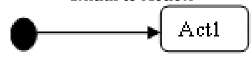
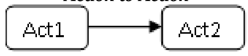
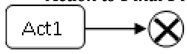
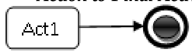
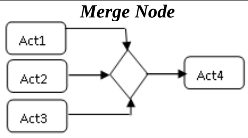
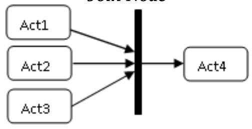
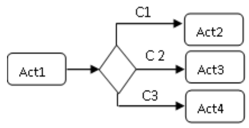
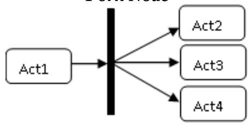
```
fmod Basic_ActivityDiagram is

sort CONFIGURATION .
sorts InitialNode ActivityFinal FlowFinal Action .
subsorts InitialNode ActivityFinal FlowFinal Action < CONFIGURATION .
op null : -> CONFIGURATION .
op _ _ : CONFIGURATION CONFIGURATION -> CONFIGURATION [assoc comm id:null] .
op Isin : ActivityFinal CONFIGURATION -> Bool .
vars E E' : ActivityFinal .
vars S conf : CONFIGURATION .
eq Isin (E, Null) = false .
eq Isin (E, E' S) =  E==E' or Isin (E, S)  .

endfm
```

It contains the declaration of new type called *CONFIGURATION* which represents the current configuration of an Activity diagram instance. The configuration of an Activity diagram consists of Initial Node, Activity Final, Flow Final and/or Actions which are declared as subsorts of *CONFIGURATION*. In addition, this module defines operations used for manipulating configuration elements, as well as equations implementing these operations. For example, The *Isin* operation returns a Boolean value which indicates if Activity Final sub-configuration is in a configuration.

TABLE I. REPRENTATION OF CONTROL STRUCTURES IN MAUDE

| Activity Diagram Control Structures | Corresponding Maude Rewriting Rules |
|---|---|
| ***Initial to Action***   | rl [Initial]: Initial => Act1 |
| ***Action to Action***   | rl [Transition]: Act1 => Act2 |
| ***Action to Final Flow***   | rl [FinalFlow]: Act1 => FinalFlow |
| ***Action to Final Activity***   | rl [FinalAction]: Act1 => FinalActivity |
| ***Merge Node***   | rl [Merge]: Act1 => Act4<br>rl [Merge]: Act2 => Act4<br>rl [Merge]: Act3 => Act4 |
| ***Join Node***   | rl [Joint]: Act1 Act2 Act3 => Act4 |
| ***Decision Node***   | rl [DecisionC1]: Act1 => Act2<br>rl [DecisionC2]: Act1 => Act3<br>rl [DecisionC3]: Act1 => Act4 |
| ***Fork Node***   | rl [Fork]: Act1 => Act2 Act3 Act4 |

The second module is *ActivityDiagram* system module that describes transitions firing and control nodes with their conditions (if any) by rewriting rules as shown in Table I.

We note that all rewriting rules (except Initial rewriting rule) are enabled when the Activity Final is not in the current configuration of Activity diagram.

### C. *Automatic Translation (Graph Grammar)*

To generate automatically Maude specification from a UML Activity diagram, we have proposed a Graph Grammar (GG) to traverse the Activity diagram and generate the corresponding code in Maude. The advantage of using a graph grammar to generate the textual code is the graphical and high-level fashion.

The graph grammar has an *initial Action* which opens the file where the code will be generated and decorates all the elements in the model with temporary attributes to be used in the conditions specified in the GG rules. For each element, we use two attributes: *Current* and *Visited*. The *Current* attribute is used to identify the element in the model whose code has to be generated, whereas the *Visited* attribute is used to indicate

*ICIT 2015 The 7th International Conference on Information Technology*
doi:10.15849/icit.2015.0093 © ICIT 2015 (http://icit.zuj.edu.jo/ICIT15)

whether code for the element has been generated yet. In our GG, we have proposed sixteen rules which will be applied in ascending order by the rewriting system until no more rules are applicable. We are concerned here by code generation, so none of these GG rules will change the Activity diagram models. For lack of space, we only describe the following rules (see Figure 4):

**Rule1:** ***Gen_Rule_InitialNode2Action (priority 1):*** is applied to locate the initial node which is related to an Action node, and generate the corresponding Maude specification.

***Rule5: Gen_LeftPartOfForkNodeRule (priority 3):*** is applied to locate a Fork node which is related to current Action node with an incoming transition, and generate the left part of the corresponding rewriting rule in Maude.

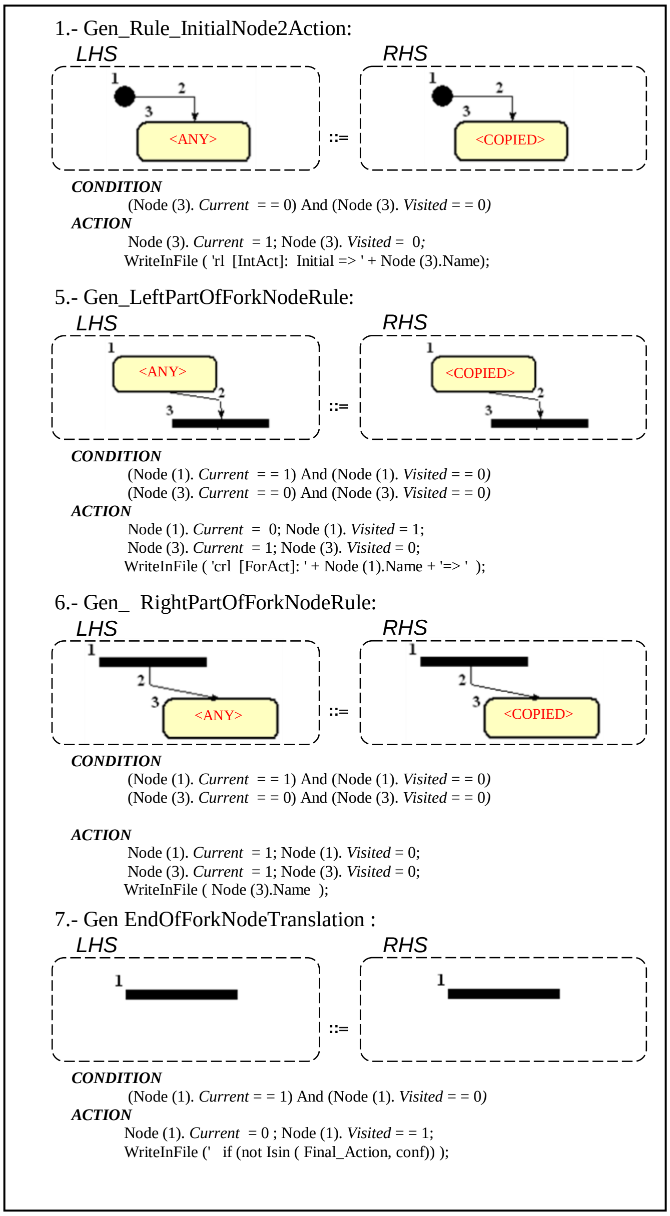



Fig. 4. Graph Grammar to generate Maude specification from an Activity Diagram

***Rule6: Gen_RightPartOfForkNodeRule (priority 3):*** is applied to locate an Action node related to the current Fork node with an incoming transition, and generate its name in the right part of the corresponding rewriting rule in Maude.

***Rule7: EndOfForkNodeTranslation (priority 6):*** is applied to locate the current Fork node whose processing has been terminated, and mark it as *Visited*. In addition, it generates the condition of the rewriting rule.

The graph grammar has also a final action which erases the temporary attributes from the entities and closes the output file.

## V. CASE STUDY

To evaluate the practical usefulness of the proposed approach, we consider a simple example of order processing application. In this diagram, the first action is to receive requested order. After order is accepted and all required information is filled in, payment is accepted and order is shipped. We Note that this example allowing order shipment before invoice is sent or payment is confirmed. The Figure 5 presents the UML Activity diagram of the Process Order created in our tool.

To analyze this behavioral specification of the order processing application, we have to transform this specification into its equivalent Maude specification. To realize this transformation in our tool, we have to execute the proposed Graph Grammar. The resulted Maude specification of the automatic transformation is shown in Figure 6.

In order to perform the analysis by simulation of the resulted Maude specification, we have invoked the rewriting logic Maude system. Simulation consists of transforming the initial state to another by doing one or many rewriting actions. Therefore, in addition to generated file, the user may give to the Simulator the number of rewriting steps if (he/she) wants to check intermediary states. If this number is not given, the Simulator continues the simulation operation until reaching a final state. The Result configuration (final state) of the simulation is given in the same manner as configuration. In our example (see Figure 7), we have asked the application to perform the simulation from the initial node.

doi:10.15849/icit.2015.0093 © ICIT 2015 (http://icit.zuj.edu.jo/ICIT15)

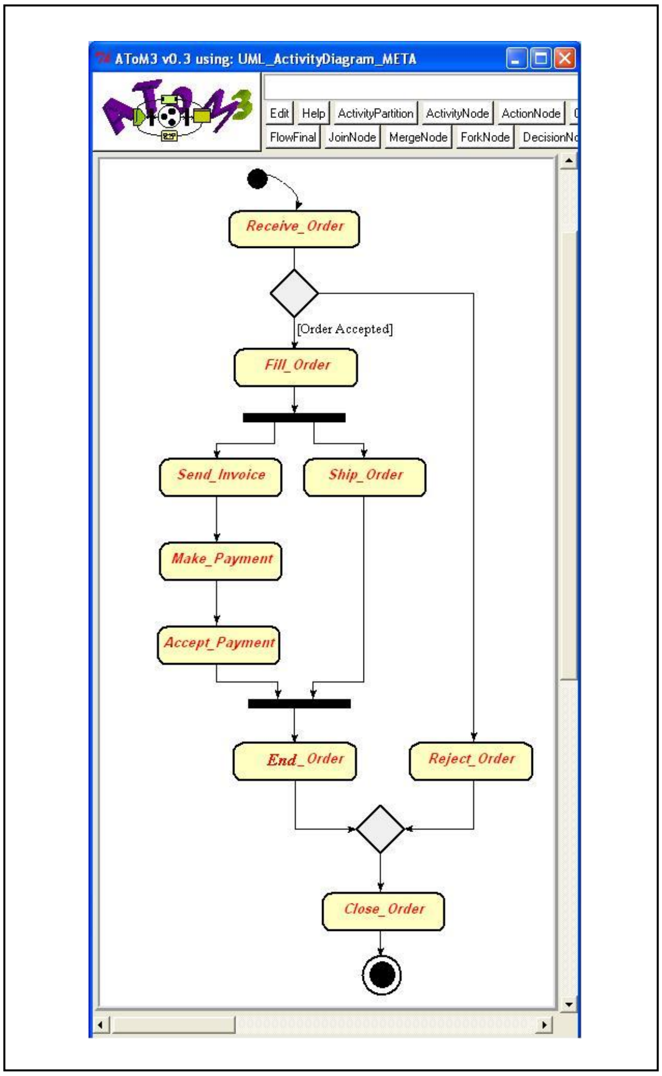



Fig. 5. UML Activity diagram created in our tool

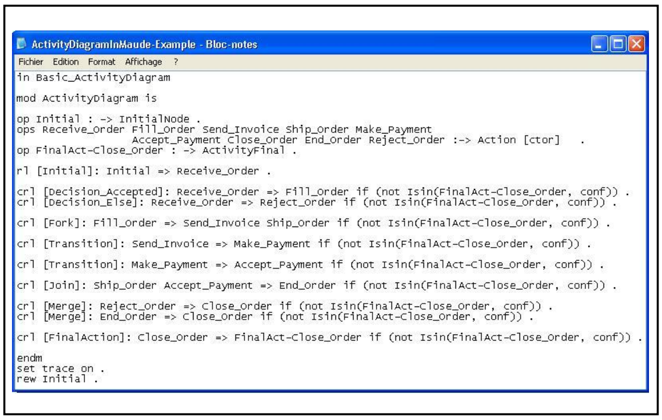



Fig. 6. Generated Maude specification

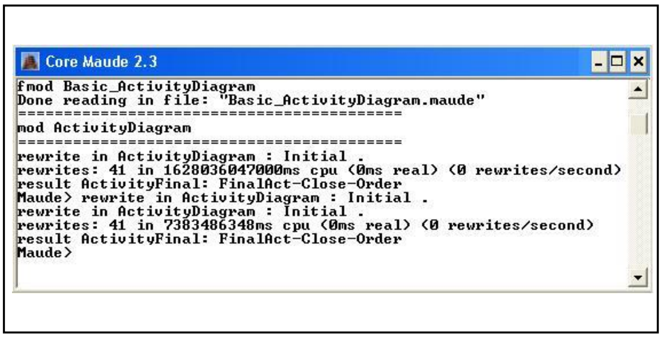



Fig. 7. Execution of order processing example under Maude system.

## VI. CONCLUSION

In this paper, we have presented a formal framework and an environment tools based on the combined use of Meta-Modeling and Graph Grammars for the Modeling and analysis of global dynamic behavior in UML models using Maude language. With Meta-modeling, we have defined the syntactic aspect of UML Activity Diagrams, and then we have used the meta-modeling tool AToM$^3$ to generate its visual modeling environment. By means of Graph Grammar, we have extended the capabilities of our framework to transform UML Activity Diagrams into an equivalent Maude specification. The resulted specification can be used to verify system properties using Maude model checking.

In a future work, we plan to transform composite action nodes and complexes links in Maude specification. We plan also to perform some verification of properties using Maude model checking.

## REFERENCES

[1] G. Booch, I. Rumbaugh and J.Jacobson, "The Unified Modeling Language User Guide", in Addison-Wesley, 1999.

[2] J. Meseguer, "Rewriting Logic as a Semantic Framework of Concurrency: a Progress Report", in Springer-Verlag, Lecture Notes in Computer Science, 119, 1996, pp. 331-372.

[3] J. Meseguer, "Rewriting logic and Maude: a Wide-Spectrum Semantic Framework for Object-based Distributed Systems", In S. Smith and C.L. Talcott, editors, Formal Methods for Open Object-based Distributed Systems, (FMOODS'2000), 2000, pp. 89-117,.

[4] J. De Lara and H. Vangheluwe, "Meta-Modelling and Graph Grammars for Multi-Paradigm Modelling in AToM3", in Software and Systems Modelling, Special Section on Graph Transformations and Visual Modeling Techniques, Vol. 3, 2004, pp. 194–209.

[5] AToM3 Home page, http://atom3.cs.mcgill.ca/

[6] J.A. Saldhana, M. Shatz and Z. Hu, "Formalisation of Object Behavior and Interaction From UML Models", in International Journal of Software Engineering and Knowledge Engineering. Vol. 11, #6, 2001, pp. 643-673.

[7] H. Xinhong, C. Lining, M. Weigang, G. Jinli and X. Guo, "Automatic transformation from UML statechart to Petri nets for safety analysis and verification", Quality, Reliability, Risk, Maintenance, and Safety Engineering (ICQR2MSE), in International Conference on, Conference Publications, Print ISBN: 978-1-4577-1229-6, 2011, pp. 948 - 951.

[8] M. Wang; L. Lu, "A transformation method from UML statechartto Petri nets", in Computer Science and Automation Engineering (CSAE), 2012 IEEE International Conference on, On page(s): 89 - 92 Vol.2, May 2012, pp.25-27.

[9] E. Kerkouche, A. Chaoui, E. Bourennane, O. Labbani, "A UML and Colored Petri Nets Integrated Modeling and Analysis Approach using Graph Transformation", In Journal of Object Technology, vol. 9, no. 4, 2010, pp 25–43.

[10] J. Araujo and A. Moreira. "Specifying the Behavior of UML Collaborations Using Object-Z". in Departamento de Infomatica, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, Portugal, 2000.

[11] H. Ledang and J. Souquières, "Formalizing UML Behavioral Diagrams with B. Tenth OOPSLA Workshop on Behavioral Semantics: Back to Basics", in Tampa Bay, Florida, USA, 2001.

[12] C.A.R. Hoare, "Communicating Sequential Processes". In Prentice Hall International Series in Computer Science.Prentice Hall, April 1985.

[13] P. Gagnon, F. Mokhati, M. Badri: "Applying Model Checking to Concurrent UML Models", in Journal of Object Technology, Vol. 7, no. 1, January- February 2008, pp. 59-84, http://www.jot.fm/issues/issue_2008_01/article1/

[14] D. Bisztray, K. Ehrig, and Reiko Heckel, "Case Study: UML to CSP Transformation". Available at http://www.informatik.uni-marburg.de/~swt/agtivecontest/UML-to-CSP.pdf

[15] E. Weinell and U. Ranger, "Using PROGRES for Transforming UML Activity Diagrams into CSP Expressions". Available at www.se.rwthaachende/files/agtivetc/UML_to_CSP.pdf.

[16] H. Störrle, "Structured Nodes in UML 2.0 Activities", in Nordic Journal of Computing, Vol. 11, No. 3, Sep 2004, pp. 279-302.

[17] J. Meseguer, "Conditional rewriting logic as a unified model of concurrency", in Theoretical Computer Science, Vol. 96(1), 1992, pp. 73-155.

[18] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer and C.Talcott, "Maude manual (version 2.2)", Internal Report, SRI International, December 2007.

[19] S. Eker, J. Meseguer and A. Sridharanarayanan, "The Maude LTL model checker", in Proceedings of the 4th International Workshop on Rewriting Logic and Its Applications (WRLA), Electronic Notes in Theoretical Computer Science, Vol. 71, 2002.

[20] J. De Lara and H. Vangheluwe, "Meta-Modelling and Graph Grammars for Multi-Paradigm Modelling in $AToM^3$", in Software and Systems Modelling, Special Section on Graph Transformations and Visual Modeling Techniques, Vol. 3, 2004, pp. 194–209.

[21] R. Bardohl, H. Ehrig, J. De Lara and G. Taentzer, "Integrating Meta Modelling with Graph Transformation for Efficient Visual Language Definition and Model Manipulation", in Wermelinger, M., Margaria-Steffen, T. (eds.) FASE 2004. LNCS Springer, Heidelberg, Vol. 2984, 2004, pp. 214–228.