From UML Statecharts to LOTOS Expressions Using Graph Transformation

Salim Djaaboub[1,3(✉)], Elhillali Kerkouche[2,3], and Allaoua Chaoui[3]

[1] Department of Mathematics and Computer Sciences, Centre Universitaire de Mila, Mila, Algeria
[2] Département of Computer Sciences, Université de Jijel, Jijel, Algeria
[3] MISC Laboratory, Université de Constantine 2, Constantine, Algeria
{dja_salim,elhillalik}@yahoo.fr, a_chaoui2001@yahoo.com

Abstract. The use of UML Statecharts for modeling dynamic behaviors of systems is very widespread. UML Statechart diagrams support developers by means of graphical notation, but the lack of formal semantics for this diagram makes the detection of errors and behavioral inconsistencies difficult. On the other hand, the specification language LOTOS has been proved to be an essential technique for specifying and verifying distributed and communicating systems. However, the strong and obscure textual algebraic notation of LOTOS makes its utilization in software development an onerous and complex task. In the order to combine UML Statecharts with LOTOS for the formal specification and verification of system behaviors, we propose in this paper an approach and a tool environment to automatically generate behavioral LOTOS expressions from UML Statechart diagrams. The approach uses the meta-modeling tool AToM^3.

Keywords: UML · Statechart diagram · LOTOS · Formal methods · Graph transformation · Atom^3

1 Introduction

It is now recognized that UML (Unified Modeling Language) [14] is considered nowadays as a standard software modelling language. UML consists of many diagrams. Some diagrams are used to model the structure of a system while others, such Statechart diagram, are used to model the behavior of a system. The use of UML Statecharts for modeling the dynamic behaviors of systems is very widespread. Such diagrams provide an effective graphical notation for the specification and design of system behaviors. However, the lacks of a precise formal semantics of Statechart diagrams hinders the formal analysis and verification of system design.

In the other hand, the specification language LOTOS (Language Of Temporal Ordering Specification) [3] has been proved to be an essential technique for specifying distributed and communicating systems. It is based on a rigorous mathematical model and provides many verification tools that allow the early detection of errors in a specification before its implementation. However, the obscure and textual algebraic notation

G. Dregvaite and R. Damasevicius (Eds.): ICIST 2015, CCIS 538, pp. 548–559, 2015.
DOI: 10.1007/978-3-319-24770-0_47

of LOTOS makes its utilization in software development an onerous task, which limits its widespread adoption in industry and confines most of the research activity to academia [2, 15].

So, combining UML Statecharts with LOTOS is a promising approach for modelling and verification of dynamic system behaviors. We can benefit from the conceptual visuality by UML Statechart to model systems behavior and from the formal notation of LOTOS for early detection of errors in specifications before their implementation.

In this paper, we propose an approach to automatically generate LOTOS processes from UML Statechart diagrams. This approach is based on the combined use of Meta-Modelling and Graph Transformation grammars which are supported by the Metamodeling tool AToM3. Our approach will allow developers to model system behaviors using a set of UML Statechart diagrams and automatically convert them into their equivalent LOTOS processes. Then, these processes can be composed to form the global specification of the system using a set of predefined LOTOS operators.

The rest of this paper is organized as follows. Section 2 outlines the major related work. Section 3 gives a brief description of the UML Statechart, LOTOS, graph grammars and AToM3. Section 4 discusses the transformation rules used in our approach. Section 5 is the main section which presents the proposed approach. An ATM machine example and its transformation into LOTOS are presented in Sect. 6. Section 7 concludes the paper and gives some perspectives.

2 Related Work

Several works have focused on the UML models transformation towards formal methods such as Petri nets [16–18], Collared Petri nets [10], Maude [7], Object-Z [1], B method [11] and LOTOS. In this paper we have chosen the specification language LOTOS as one of the predominant and standard formal methods. It can be used in different application domains and provides a great modularity to structure and to analyze complex specifications. LOTOS specifications are precise, unambiguous and can be rigorously analyzed, verified and validated using several tools and methods based on theorem-proving, equivalence test or model-checking. The verification tools existing around LOTOS, such as CADP [20], provide much functionality such as model-checking, equivalence test (bisimulation, trace equivalence, observational equivalence … etc.), counter example visualization and code generation.

No much research has been done about the transformation of UML Statechart LOTOS. In [5], the authors present a model transformation of UML Statechart to LOTOS applied in the automotive industries. LOTOS is used in [13] to give a semantic model for compositional UML Statecharts. In [8], a transformation approach from a subset of UML Statechart to LOTOS behavior expressions is presented. In [15], the authors present how one can use LOTOS operators to gradually transform a particle form of UML State Machines into verifiable LOTOS specifications, in spite of not proposing precise transformation rules. However, the complexity of these approaches is high due to their manual, rigid and complex transformation rules. In the other hand, these approaches require the familiarity with LOTOS and the transformation process.

From the practical point of view, we believe that developers cannot use these approaches to convert Statechart diagrams to LOTOS.

In this paper, we propose a graph transformation based approach to automatically generate LOTOS expressions from UML Statechart diagram. The approach which is based on graph transformation is implemented as a tool environment using the meta-modeling tool AToM3. This approach will allow developers to model the system behaviors using UML Statechart diagrams and then automatically convert them into LOTOS for the verification purposes.

3 Background

3.1 UML Statechart

The Unified Modelling Language (UML) is a language for specifying, constructing, visualizing, and documenting the artifacts of a software development process [4]. It consists of a large number of diagrams. Some diagrams are used to model the structure of a system while others, such as Statechart diagram, are used to model the behavior of system.

A Statechart diagram [14] is a graph that models the lifetime of an object in response to events. Statechart diagrams describe state machines by emphasizing the potential states and transitions between them. A state can be considered as a situation during which some invariant condition holds. States can be either simple, composite or concurrent that can have several orthogonal regions. A transition is a relationship between two states indicating that an object in the transition source state may leave it and after that enters to the state target by the transition. Graphically, states and transition are represented as state boxes and transition arrows.

3.2 Lotos

LOTOS (Language Of Temporal Ordering Specification) is a standard formal description technique developed within ISO for specifying, among others, distributed concurrent information processing systems [2, 3]. This language adopts most of its concepts from Hoare's CSP [9] and Milner's CCS [12]. LOTOS is based on process algebraic methods for the description of process behaviors and interactions. For the description of data structures, it is based on the abstract data type language ACT ONE [6]. In this paper, we focus on the control aspect of the system, which is also called Basic LOTOS, and we leave the abstract data type description for future research.

In Basic LOTOS, a distributed concurrent system is seen as a process, possibly consisting of several subprocesses. A sub-process is a process in itself, so that in general a LOTOS specification describes a system via a hierarchy of process definitions [3]. A process is an entity able to perform internal, unobservable actions, and to interact with other processes, which form its environment. The typical structure of a basic LOTOS process definition is as presented in Fig. 1.

The essential component of a process definition is its behavior expression, which describes its behavior and its interaction with the environment through a number of small

```
process <proc_name> [gate list]:functionality:=
behaviour
     Behavior expression
where
     Process definitions
endproc
```

Fig. 1. Structure of basic LOTOS process

atomic actions denoted as gates. Behavior expressions are built from actions and other behavior expressions by using a predefined set of operators.

3.3 Graph Grammar and AToM^3

AToM^3 [21] 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. The AToM^3 meta-layer allows a high-level description of models using UML Class Diagram formalism or Entity Relation-ship (ER) formalism extended with the ability to express constraints. Based on these descriptions, AToM^3 can automatically generate tools to manipulate models in the formalisms of interest [22].

AToM^3 also supports graph rewriting, which is based on Graph Grammars to visually guide the procedure of model transformation. Model transformation refers to the process of translating, converting or modifying a model of a given formalism into another model that might or might not be in the same formalism. Graph Grammars [22] are a generalization of Chomsky grammars 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. 2.

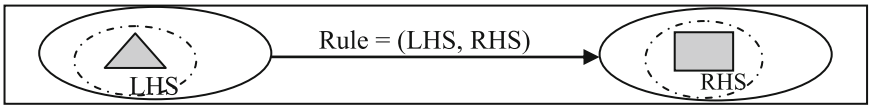

Fig. 2. Rule-based modification of graphs

4 Overview of the Transformation Rules

In LOTOS, behaviors of systems are usually specified as a set of processes, each of them specifies the behavior of an entity of the system. The global specification of the system is formed by composition of these processes using a set of predefined operators [2, 3]. For this reason, we assume in this work that the behavior of the system is specified as a set of Statechart diagrams, each of them models the behavior of an entity or an object of the system. These diagrams are translated into a set of processes in LOTOS. Then, these processes are composed to form the global specification using the fallowing

process composition operators: parallel operators (Interleaving, Partial synchronization, full synchronization), enable and disable operators.

Converting Statechart diagrams directly into LOTOS is a very complex process. This is due to the complexity of the transformation rules, which makes its automation a very complex task. For this reason, we have chosen to flatten the hierarchical and concurrent structure of composite states in the Statechart diagram before its conversion into LOTOS process. Thus, Statechart diagrams are first converted to Flat State Machine (FSM) models which contain just simple states and arcs. Then, these FSM are converted into LOTOS processes.

Several algorithms and tools have been proposed to flatten Statechart diagrams [19]. In this work, we will use the graph transformation rules proposed in [10]. These rules are presented in Sect. 5. In the following, we will present only the mapping rules to convert an FSM diagram into a LOTOS process.

1. *Events to actions*. Events and actions in the FSM diagram are specified as actions in the equivalent LOTOS process.
2. *FSM states to LOTOS processes*. For each state in the FSM diagram, a LOTOS process is created as that presented in Fig. 3. The name of this process is the name of the state and its gate list ([gatelist]) is the list of events and actions in the transitions outgoing from the state.

```
process <proc_name> [gate list]:functionality:=
behavior
     Behavior expression
endproc
```

Fig. 3. The LOTOS process created for a state

3. *State Transitions to the behavior expression of the equivalent process*. For each transition, a sequential behaviors expression is created using the action prefix ";" that expresses the sequential combination of actions before a behavior expression. This operator is used to compose the transition event, the transition actions and the instance of the process is created from the state target by the transition as follows. In the case where there are several transitions from a single state, the choice "[]" operator is used to compose the behavior expressions corresponding to these transitions.
4. *Final state to stop process*. Final state is translated to "stop" process. "stop"is a predefined basic behavior used to express the process that finishes its execution or to represent deadlock situation.
5. *The FSM diagram to a LOTOS process*. The format of the LOTOS process generated for an FSM is presented in Fig. 4. The name of this process (proc_name) is the name of the class of objects modelled by the Statechart diagram. Its gate list ([gate list]) is the list of events and actions in the FSM diagrams. The behavior expression of this process is the instantiation of the process generated from the state target by the initial transition. Processes created from states are inserted as sub-processes of this process.

```
process <proc_name>[gate list]:functionality:=
behavior
     state1_name [gatelist1]
where
     process <state1_name> [gatelist1]:functionality:=...
     process <state2_name> [gatelist2]:functionality:=...
     . . .
endproc
```

Fig. 4. The structure of LOTOS specification created for an FSM.

5 The Proposed Approach

In this section, we describe our automated approach that transforms UML Statechart diagrams into their equivalent LOTOS processes. The approach performs the transformation as follows: first, flattening Statechart diagrams into Flat State Machines (FSM). Then, converting FSM models results into LOTOS processes.

For automating our approach using the meta-modeling tool AToM3, we have performed two following steps as follows (see Fig. 5): The first step consists of Meta-Modeling of UML Statechart and FSM formalism. The second step is the definition of graph transformation grammars used to perform the transformation. So, we have defined two Graph Grammars: the first Graph Grammar (1st GG): converts the Statechart diagrams to FSM models. The second Graph Grammar (2nd GG): generates LOTOS processes from the obtained FSM models.

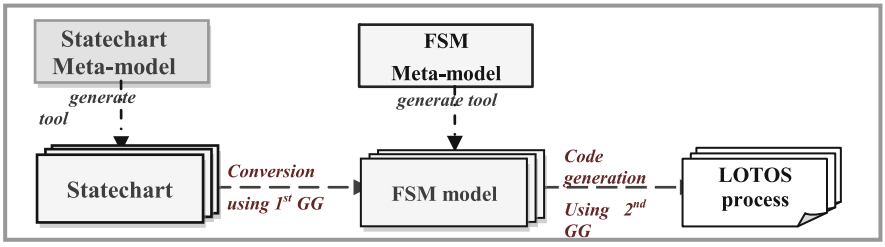

Fig. 5. The proposed approach

5.1 Meta-Modeling of UML Statechart and FSM Model

Since the visual tool AToM^3 has a meta-modeling layer that allows us to graphically model the different formalisms (Fig. 6), we have created two meta-models, Statechart diagrams meta-model and FSM meta-model. Then, we have used AToM^3 to generate a visual modeling tool for each of them according to their proposed Meta-Models.

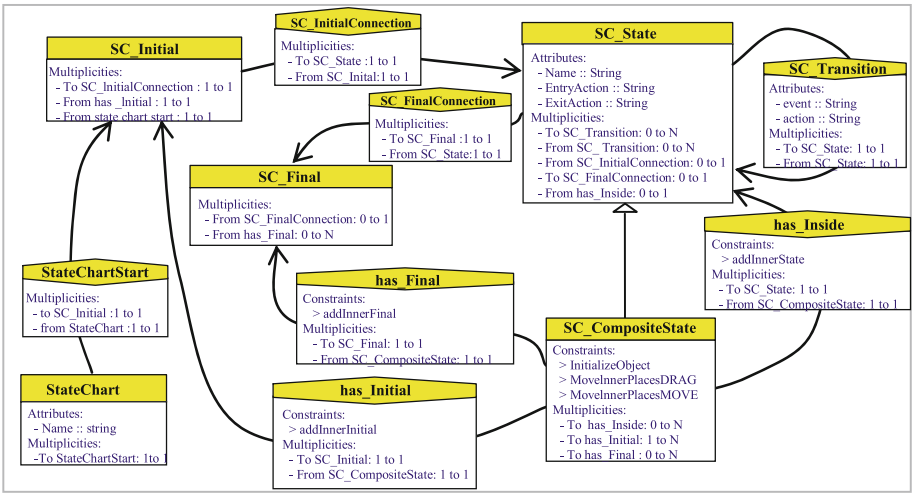

Fig. 6. UML Statechart meta-model

In this paper, we deal with a subset of UML Statechart which consists of states (simple and composite), transitions, events, actions that generate events, and initial and final states. The created meta-model for UML Statechart diagram is composed of five classes linked by seven associations as shown in Fig. 6.

A FSM is a State Machine without composite states. States in FSM are represented by rounded boxes, while transitions between them are represented with arcs. The meta- model of the FSM model consists of one class to represent FSM states and one association for representing FSM transitions.

5.2 Graph Transformation Grammars

In this sub-section, we describe the two graph grammars used to reach an automatic and correct transformation of UML Statecharts to LOTOS process.

1St GG: Converting Statechart Diagrams into FSM Models. To transform a Statechart into its equivalent FSM model, we have used the graph transformation grammar proposed in [10]. This graph grammar is named Statechart2FSM and containing twenty three rules which will be applied in ascending order. Only some representative rules are of this graph grammar shown here (see Fig. 7).

The idea of the transformation in Statechart2FSM grammar can be summarized by the following main steps: The first step is to select a Statechart and convert its initial state to FSM state using rule 22. The second step is to traverse the Statechart from its initial state (which is processed in the first step). Each exit transition of a processed Statechart state is converted into the corresponding FSM transition and the destination Statechart state is also converted if it has not previously processed (see rule 6). For each composite state in the statechart, the graph grammar uses a flag attribute indicating that this composite state is traversed but not processed (see rule 7). In third step, all composite states in the Statechart

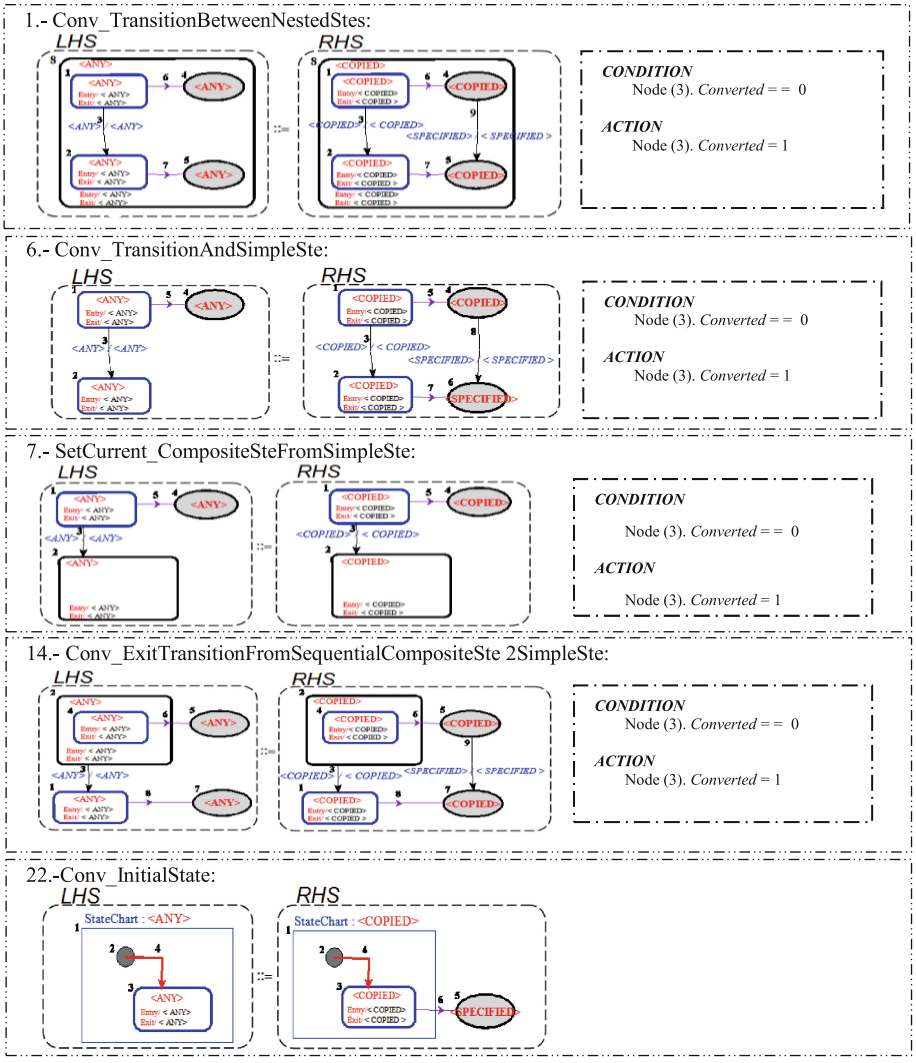

Fig. 7. Some rules of the first graph grammar (1St GG)

(if any) will be converted. The process of the conversion is the same for Statechart diagrams (see rule1). The last step is to relate the equivalent FSM segment of composite state to FSM model of the Statechart (see rule 14).

2nd GG: Generating LOTOS Processes from FSM Models. In this grammar, we have concerned by the automatic generation of the LOTOS code. We have named this second graph grammar FSM2LOTOS which is composed of 14 rules. Applying this grammar, after the first grammar, leads to the generation of a file ".LOT" containing the corresponding LOTOS processes.

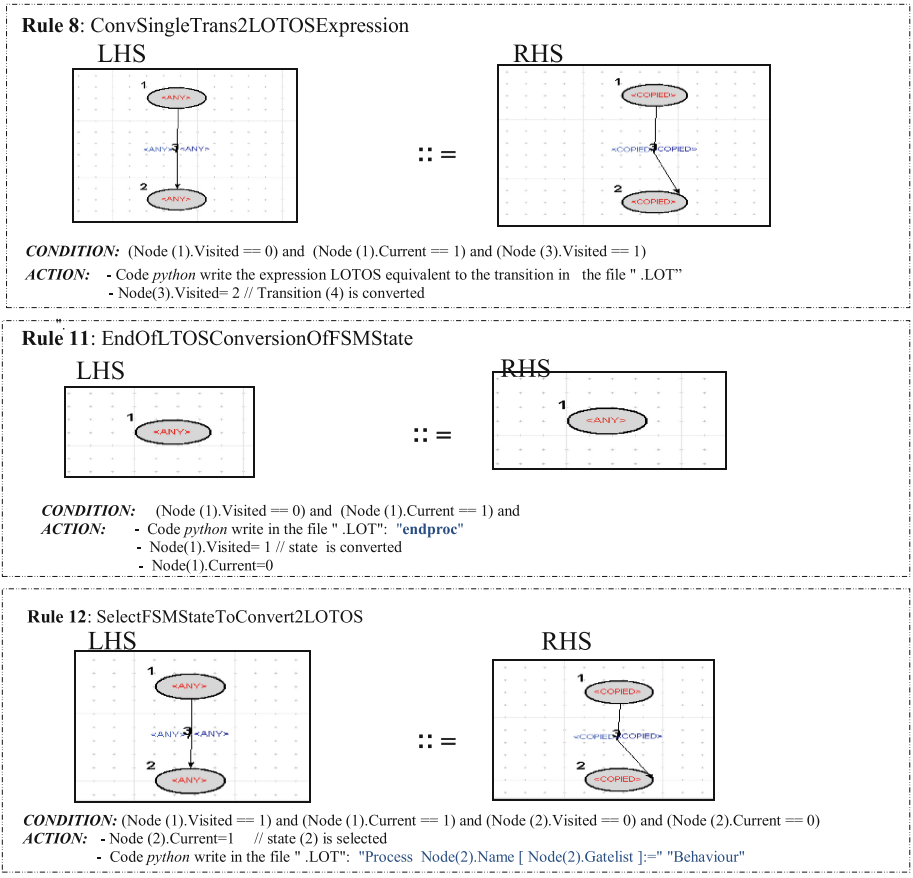

Fig. 8. Some rules of the second graph grammar (2nd GG)

Figure 8 shows some rules of the second graph grammar. These rules are used to convert a state with a single transition into a LOTOS process as that presented in Fig. 3. These rules are executed in the following order: first, Rule 12 selects a state no yet converted into a LOTOS process. Then, Rules 8 convert the transition outgoing from the state into the behavior expression of the process equivalent. At last, the Rule 11 finishes the conversion and actualises the flag attributes.

6 Example

We consider an example of an ATM machine, dispensing cash to a user. The system is composed of two objects ATMmachine and User. The Statechart diagrams model the behaviors of these objects are shown in Fig. 9.

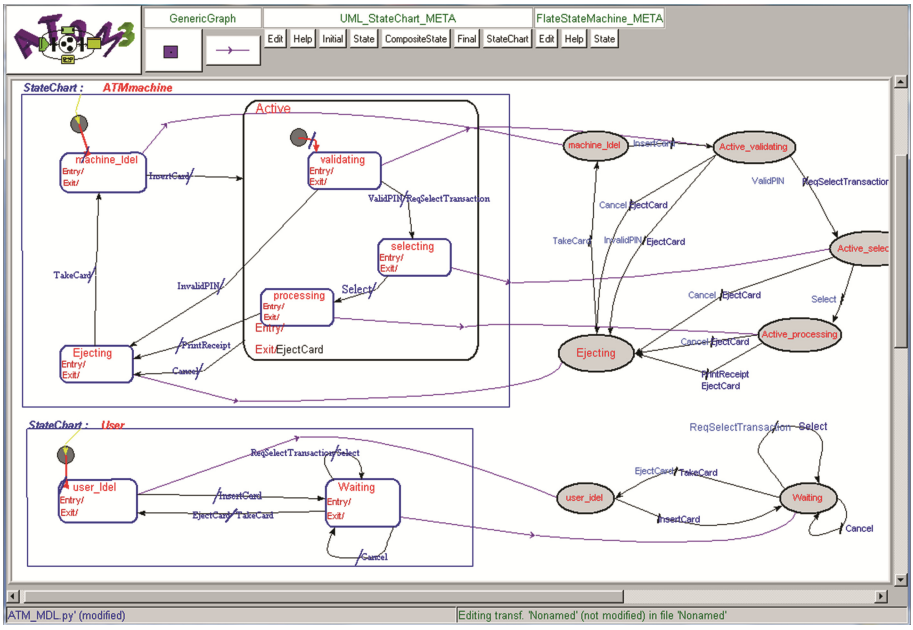

Fig. 9. Statecharts ATMmachine and user with their equivalent FSM models.

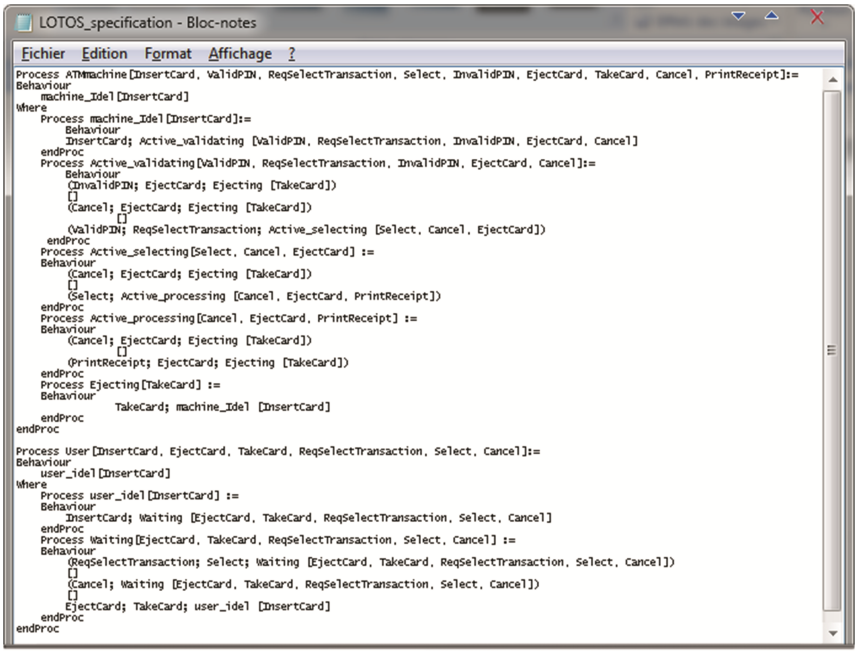

Fig. 10. ATMmachine and user processes generated using our tool

To transform these Statecharts into LOTOS processes, we have executed the Statechart2FSM graph grammar to flatten Statechart diagrams to FSM models as shown in Fig. 9. Then, we have executed the FSM2LOTOS graph grammar which generates the LOTOS processes shown in Fig. 10. These processes are named ATMmachine and User, respectively for FSM ATMmachine and FSM User.

For space limitation reasons, only the process ATMmachine is described here. The process name is the name of the Statechart ATMmachine. States in the FSM ATMmachine are formalized as sub-processes of the equivalent process. The gate list of ATMmachine process is the list of events and actions in the FSM ATMmachine. The behavior expression is the instantiation of the machine_Idle process, which is the process generated for the state targeted by the initial transition.

7 Conclusion

We have proposed an approach to transform UML Statechart diagrams to LOTOS performed by AToM^3. This approach is based on the combining of metamodeling and graph grammars. This transformation aimed to combine visual UML Statechart diagrams and the formal language LOTOS for modeling and verification of system behaviors. It produces a formal and verifiable specification that facilitates the early detection of errors like deadlock, livelock, etc..... The transformation is based on the graph grammars that are natural, visual and high level formalism to describe the transformations. Furthermore unlike to the existing approaches, the transformation in our approach is automatically performed. The use of FSM model in our approach makes the translation rules less complex which allows their automation as a tool environment using AToM^3. This tool allows the modelling of system behaviors using visual UML Statecharts and then automatically generating their equivalent LOTOS processes.

In the future works, we plan to enhance our approach by using the complete version of LOTOS to represent data and other Statechart elements such transition guards. Secondly, we plan also to exploit the CADP [20] verification tool to provide some verification of system properties.

References

1. Araujo, J., Moreira, A.: Specifying the behavior of UML collaborations using object-Z. Departamento de Infomatica, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, Portugal (2000)
2. Babich, F., Deotto, L.: Formal methods for specification and analysis of communication protocols. IEEE Commun. Surv. **4**, 2–20 (2002)
3. Bolognesi, T., Brinksma, E.: Introduction to the ISO specification language LOTOS. Comput. Netw. ISDN Syst. **14**, 25–59 (1987)
4. Booch, G., Rumbaugh, J., Jacobson, I.: The unified modeling language user guide. Addition-Wesley, Object Technology Series (1998)

5. Chimisliu, V., Schwarzl, C., Peischl, B.: From UML statecharts to LOTOS: a semantics preserving model transformation. In: Ninth International Conference on Quality Software (2009)
6. Ehrig, H., Fey, W., Hansen, H.: ACT ONE : An Algebraic Specification Language with two levels of Semantics, Research Report Nr. 83-03, Department of Computer Science, TU Berlin (1983)
7. Gagnon, P., Mokhati, F., Badri M.: Applying model checking to concurrent UML models. J. Object Technol. **7**(1), 59–84 (2008). http://www.jot.fm/issues/issue_2008_01/article1/
8. Hnatkowska, B., Huzar, Z.: Transformation of dynamic aspects of uml models into lotos behaviour expressions. Int. J. Appl. Math. Comput. Sci. **11**(2), 537–556 (2001)
9. Hoare, C.A.R.: Communicating Sequential Processes. Prentice Hall International Series in Computer Science. Prentice Hall, New Jersey (1985)
10. Kerkouche, E., Chaoui, A., Bourennane, E., Labbani, O.: A UML and colored petri nets integrated modeling and analysis approach using graph transformation. J. Object Technol. **9**(4), 25–43 (2010)
11. Ledang, H., Souquières, J.: Formalizing UML behavioral diagrams with B. In: Tenth OOPSLA Workshop on Behavioral Semantics: Back to Basics, Tampa Bay, Florida, USA (2001)
12. Milner, R.: Formal a calculus of communication systems. LNCS, vol. 92. Springer, Heidelberg (1980)
13. Mrowka, R., Szmuc, T.: UML statecharts compositional semantics in LOTOS. In: ISPDC 2008, pp. 459–463 (2008)
14. OMG, Object Modeling Group: Unified Modeling Language Specification, version 2.0 (July 2005)
15. Babaee, R., Babamir, S.M.: From UML state machines to verifiable lotos specifications. In: Pichappan, P., Ahmadi, H., Ariwa, Ezendu (eds.) INCT 2011. CCIS, vol. 241, pp. 121–129. Springer, Heidelberg (2011)
16. Saldhana, J.A., Shatz, M., Hu, Z.: Formalisation of object behavior and interaction from UML models. Int. J. Softw. Eng. Knowl. Eng. **11**(6), 643–673 (2001)
17. Xinhong, H., Lining, C., Weigang, M., Jinli G., Guo, X.: Automatic transformation from UML statechart to petri nets for safety analysis and verification. In: ICQR2MSE 2011, Conference Publications, pp. 948–951 (2011). ISBN 978-1-4577-1229-6
18. Wang, M., Lu, L.: A transformation method from UML statechart to petri nets. In: IEEE International Conference on Computer Science and Automation Engineering (CSAE) 2012, vol. 2, pp. 89–92, 25–27 May 2012
19. Devroey, X., Perrouin, G., Cordy, M., Legay, A., Schobbens, P.Y., Heymans, P.: State machine flattening: mapping study and assessment. CoRR abs/1403.5398 (2014)
20. CADP: Construction and analysis of distributed processes. http://cadp.inria.fr/
21. AToM3 Home page. http://atom3.cs.mcgill.ca/
22. Lara, J.D., Vangheluwe, H., Alfonseca, M.: Meta-modelling and graph grammars for multi-paradigm modelling in AToM^3. In: Software and Systems Modelling, vol. 3, pp. 194–209. Springer, Heidelberg (2004) (Special Section on Graph Transformations and Visual Modeling Techniques)