Journal of Object Technology
Published by ETH Zurich, Chair of Software Engineering, © JOT 2010
Online at http://www.jot.fm.

A UML and Colored Petri Nets Integrated Modeling and Analysis Approach using Graph Transformation

Elhillali Kerkouche[a] Allaoua Chaoui[b] El Bay Bourennane[c]
Ouassila Labbani[c]

a. Department of Computer Science, University of Oum El Bouaghi, Algeria
b. Department of Computer Science, University of Constantine, Algeria
c. Laboratoire LE2I, University of Bourgogne, Dijon, France

Abstract Nowadays, UML is considered to be the standardized language for object-oriented modeling and analysis. However, UML cannot be used for automatic analyses and simulation. In this paper, we propose an approach for transforming UML statechart and collaboration diagrams to Colored Petri net models. This transformation aims to bridge the gap between informal notation (UML diagrams) and more formal notation (Colored Petri net models) for analysis purposes. It produces highly-structured, graphical, and rigorously-analyzable models that facilitate early detection of errors such as deadlock and livelock. The approach is based on graph transformations where the input and output of the transformation process are graphs. The meta-modeling tool AToM3 is used. A case study is presented to illustrate our approach.

Keywords UML, Colored Petri Nets (CPN), Models Transformation, Meta-modeling, Graph Grammars, Graph Transformation, AToM3.

1 Introduction

It is now recognized that Unified Modeling Language (UML) [3] is considered to be a standard object-oriented software modeling language for specifying and visualizing software systems. UML consists of a large number of diagrams. Some diagrams are used to model the structure of a system while others are used to model the behaviour of a system. UML statecharts and UML collaboration diagrams are widely used to model dynamic behaviour 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.

However, despite its success as being a unified and visual notation in software engineering field, UML still lacks a precise formal semantics, which hinders the formal analysis and verification of system design. So, UML needs a well-defined semantic base for its notation. Petri nets [20] and in particular coloured Petri nets (CPNs) [13] are a formal and graphical language appropriate for systems modeling and analysis. Thus developing a tool environment (framework) for modeling and analysis of dynamic behaviour in UML models would be a significant aid to modellers who use UML to model their systems. UML models are projected into CPN models for analysis, and the results of the formal analysis can be back-annotated to the UML models to hide the mathematics from modellers.

Building a modeling tool from the scratch is a prohibitive task. A meta-modeling approach is useful to deal with this problem, as it allows the (possibly graphical) modeling of the formalisms themselves [19]. A model of these formalism should contain enough information to permit the automatic generation of a tool to check and build models subject to the described formalism's syntax. If this specification is made graphically, the time to develop a modeling tool can be drastically reduced to a few hours. Since both meta-model and model are stored as graphs, further manipulations of the models can be described (modelled) graphically and formally as graph grammars [23], thus they become high-level modes, reducing the need for coding to a minimum. Some of these manipulations are model simulations or animations; model optimisations, for example, to reduce its complexity; model transformations into another model (equivalent in behaviour), expressed in a different formalism; and the generation of textual model representations for use by existing simulators or tools. These ideas presented above are implemented in AToM3 (A Tool for Multi-formalism and meta-modeling). It is developed at the Modeling, Simulation and Design Lab in the School of Computer Science of McGill University [2].

In this paper, we propose a graph transformation approach and tools based on the combined use of meta-modeling and Graph Transformation grammars and systems for modeling and analysis of dynamic behavior in UML models using the Colored Petri Nets formalism. Using our approach, the modellers specify the dynamics of a system by means of a set of statechart diagrams and collaboration diagram. Then the modellers transform automatically their behavioral specification into its equivalent intermediate system-level CPN-like model of the system under study. The obtained CPN-like model can be used as a foundation for formal analysis and simulation techniques.

We have defined meta-models for statechart diagrams, UML collaboration diagrams and a CPN-like formalism. The meta-modeling tool AToM3 is used to automatically generate a visual modeling tool for each formalism according to its proposed meta-model. To reach an automatic and correct process of transformation, we propose to use graph transformation grammars and systems to define and implement the transformation. So, we have defined two graph grammars. The first one performs the transformation of statechart diagrams to Flat State Machine (FSM) [25] models. The second graph grammar converts the obtained FSM models into CPN-like models and relates each other according to the messages passing described in collaboration diagram. The resulting model of this step is single CPN-like model which is a form of Object Petri Nets called Object Net Models [24](discussed in Section3).

This paper is organized as follows: section 2 outlines the major related work. Section 3 summarizes briefly the pertinent concepts which we require in our approach.

In section 4, we give an overview of AToM3. In section 5, we define meta-models for statechart and collaboration diagrams and formalisms for generating visual tools for all formalisms. In section 6, we propose two graph grammars. The first one converts automatically statechart diagrams into a set of flattened FSM models whereas the second graph grammar transforms the resulting FSM models and the collaboration diagram into system-level CPN-like model which represent intermediate model. The intermediate model can be used to generate input descriptions for the tool analyzer. In section 7, we illustrate our framework with an example. Finally, section 8 concludes the paper.

2 Related Work

In addition to AToM3, there are several visual tools to describe formalisms using metamodeling like Generic Modeling Environment (GME) [9], MetaEdit+ [14] and other tools from the Eclipse Generative Modeling Tools (GMT) project such as Eclipse Modeling Framework (EMF) [6], Graphical Editing Framework (GEF) [8] and Graphical Modeling Framework (GMF) [10]. In most of these tools, model transformations have to be described textually and user-friendly support for visual analysis and testing is generally missing. In AToM3, the user expresses such transformations by means of graph grammar models. Graph grammars are a natural, declarative, and general way to express transformations.

There are also similar tools which manipulate models by means of graph grammars, such as PROGRES [21], GReAT [11], FUJABA [7], TIGER [26] and AGG [1]. However, none of these has its own meta-modeling layer. Some of them are complemented with support for meta-modeling (for example, The GReAT model transformation engine is combined with GME).

The combined use of meta-modeling and graph grammars in AToM3 allows users not only to benefit from the advantages of both (meta-modeling and graph grammars) but also to model with multi-paradigm modeling [17]. The AToM3 tool has been proven to be powerful, enabling the meta-modeling and the transformations of known formalisms. In [18] the authors presented a transformation from statecharts (without hierarchy) to Petri Nets. In [5], we have provided the INA Petri net tool [12] with a graphical environment. First, we have proposed a meta-model for Petri net models and used it in the meta-modeling tool AToM3 to generate automatically a visual modeling tool to process models in INA formalism. Then we defined a graph grammar to translate the models created in the generated tool to a textual description in INA language (INA specification). Then the INA is used to perform the analysis of the resulting INA specification. In [15], we have presented a formal framework (a tool) based on the combined use of meta-modeling and Graph Grammars for the specification and the analysis of complex software systems using G-Nets formalism. Our framework allows a developer to draw a G-Nets model and transform it into its equivalent PrT-nets model automatically. To perform the analysis using PROD analyzer, our framework allows a developer to translate automatically each resulting PrT-Nets model into PROD's net description language. To this end, we have defined a meta-model for G-Nets formalism and another for PrT-Nets formalism. Then the meta-modeling tool AToM3 is used to automatically generate a visual modeling tool for each formalism according to its proposed meta-model. We have also proposed two graph grammars. The first one performs the transformation of the graphically specified G-Nets models to semantically equivalent PrT-Nets models. The second one translates the resulting PrT-Nets models into PROD's net description language.

In this paper we propose an extended version of our paper published in [16]. It consists of an approach for transforming UML statechart and collaboration diagrams to Colored Petri nets models. More precisely, we have proposed an automated approach and a tool environment that formally transforms dynamic behaviours of systems expressed using UML models into their equivalent Colored Petri Nets (CPN) models for analysis purpose.

3 Background and Context

The intuitive and graphical notations of behaviour diagrams in UML greatly improve the expressive power for system modeling dynamic aspects of complex systems. However, the lack of a formal semantics makes it difficult to apply direct, mathematical analyses. To reach this goal, one has to translate these diagrams into a mathematical model. Coloured Petri Nets (CPNs) [13] — a well proven formal model for resource sharing and synchronisation — are widely used to formalise UML diagrams.

Our objective is to establish an integrated tool environment based on meta-modeling and graph transformation in which software engineers can specify and analyse dynamic behavior in UML models using CPN formalism. In this work, we assume that the behavior of the system is specified as a set of statechart diagrams and that the collaboration diagram is used to represent object interactions.

Our approach consists of a process with two steps. First, the behavioral specification is transformed into an equivalent intermediate single model. In the second step, the obtained intermediate model is used to generate the equivalent description in the input language of an analyzer tool (such as INA, Design/CPN, etc.) for further analysis. The approach is illustrated in Figure 1.

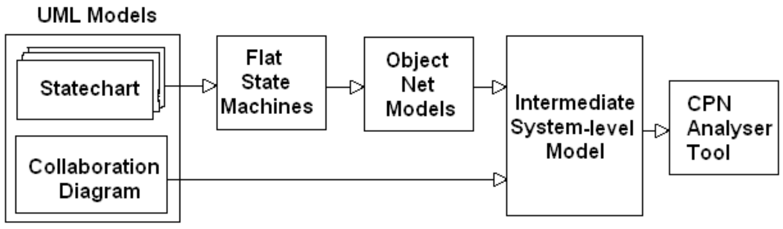

Figure 1 – Architecture of the approach [24]

In the first step, we have adopted the approach proposed by Saldhana and Shatz [24]. This approach suggests that statechart diagrams are first converted to Flat State Machines (FSM) which contain just simple states and arcs. Since statechart diagrams may contain hierarchical or nested states, effective conversion to Petri nets requires that the nested states be "flattened". These state machines are then converted to a form of Object Petri Nets (OPN) called Object Net Models (ONM). Finally, the UML collaboration diagrams are used to connect these object Net models (ONMs) to derive an intermediate single system-level model for the system under study.

The structure of Object Net Models (ONM) consists of a lifetime behavior model (LM) and a token routing structure as shown in Figure 2. LM represents a CPN-like model that is derived from the statechart of an object. Three places (input place (IP), output place (OP) and event Dispatcher place (ED)) and four transitions (T1, T2, ITA and OTA) defined the token routing structure for an object. The input place of the object holds the event-tokens that will be used by the object. The output place of the object holds the event-tokens that will be routed to other objects. The event router place holds the event-tokens that are generated by the object. When the object generates an event-token, the token can have a type which is either external or internal. For precise details see [24].

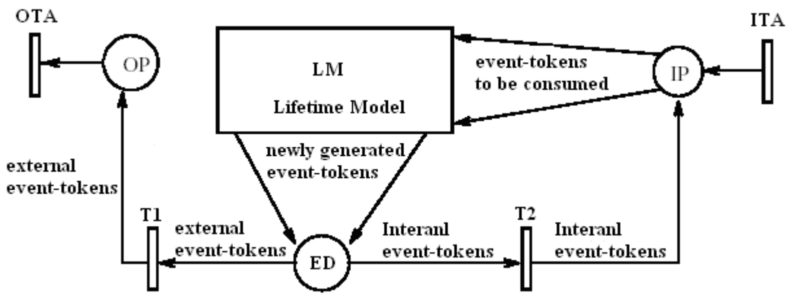

Figure 2 – The structure of an Object Net Model [24]

4 Grapg Grammar and AToM3

AToM3 [2] is a visual tool for multi-formalism modeling and meta-modeling. Being implemented in Python [22], it is able to run without any change on all platforms for which an interpreter for Python is available.

By means of meta-modeling, we can describe or model the different kinds of formalisms needed in the specification and design of systems. The AToM3 meta-layer allows a high-level description of models using Entity Relationship (ER) formalism or UML Class Diagram formalism extended with the ability to express constraints. Based on these descriptions, AToM3 can automatically generate tools to manipulate (create and edit) models in the formalisms of interest [19].

AToM3's capabilities are not restricted to these manipulations. AToM3 also supports graph rewriting, which uses Graph Grammars to visually guide the procedure of model transformation. Model transformation refers to the (automatic) process of converting, translating or modifying a model of a given formalism into another model that might or might not be in the same formalism. Graph Grammars [4] 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 Figure 3.

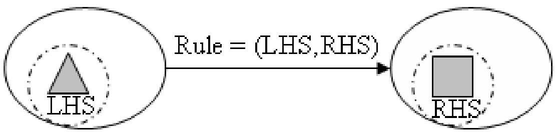

Figure 3 – Rule-based Modification of Graphs

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 rewriting system iteratively applies matching rules in the grammar to the host graph, until no more rules are applicable.

In the next sections, we will discuss how we use AToM3 to create our formal framework for modeling and verifying UML models using Colored Petri nets.

5 Meta-Modeling of Used UML Diagrams and Formalisms

In this section, we will use the meta-modeling tool AToM3 to define four metamodels for used UML diagrams and formalisms. To define a diagram or a formalism, one has to provide abstract syntax (denoting entities allowed in the formalism with their attributes, how they may be connected, what cardinalities bebetween them are, and constraints) as well as concrete graphical syntax information (the appearance of entities, relationships, and possible graphical constraints in the visual tool). The meta-formalism used in our work is the UML Class Diagram and the constraints are expressed in Python code.

5.1 StateChart Diagram Meta-Model

A statechart diagram [3] contains states (simple and composite) and transitions (events and actions). A state has several parts, namely Name (textual string for identification, can be anonymous), Entry/exit actions (actions executed on entering and exiting the state respectively). Composite states have nested structure of a states which can be sequentially or concurrent. Statechart transitions are denoted by standard finite state machine arcs that define a change from one state to a successor state. Our meta-model for UML statechart diagrams (see Figure 4) is composed of the following classes:

Statechart. This has a Name and represents a statechart in the diagram.

SC_State. This class describes simple states and has tree attributes, namely Name, EntryAction and ExitAction.

SC_CompositeState. This class represents composite states. It inherits from SC_State all its attributes, multiplicities and associations.

SC_Initial. These kinds of entities mark the initial state of a statechart or the initial state/states when reaching a composite state.

SC_Final. Represents the final state of a statechart (if any).

The following associations are also included in the meta-model:

StatechartStart. This association allows the connection of a statechart and its initial state.

SC_InitialConnection. This association allows the connection of an initial state and a state.

SC_FinalConnection. This association allows the connection of a state and a final state.

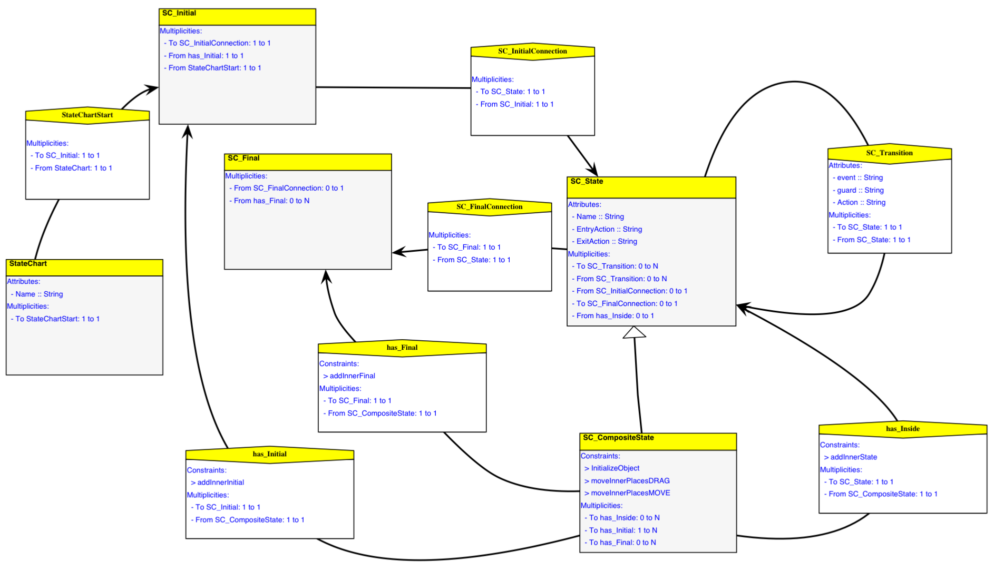

Figure 4 – Meta-model of statecharts

SC_Transition. This association represents the transition from source state to destination state (which may be the same state). It contains three attributes: event, guard and Action.

has_Inside. This is an association between a composite state and a state (which may be in it turn composite). It expresses the notion of hierarchy: states are inside composite state.

has_Initial. This association expresses the notion of hierarchy between a composite state and its initial state.

has_Final. This association expresses the notion of hierarchy between a composite state and its final state.

Since the associations: has_Inside, has_Initial and has_Final are a means to express hierarchy they are drawn as invisible links.

5.2 UML Collaboration Diagram Meta-Model

Since a UML collaboration diagram consists of objects that interact by sending each other messages, we propose to meta-model UML collaboration diagrams with one class named CollaborationObject for representing objects and one association named CollaborationLink for representing messages as shown in Figure 5.

5.3 Flat State Machines (FSM) Meta-Model

A Flat State Machines (FSM) is State Machine without composite states. States in FSM are denoted by rounded boxes, while transitions between states are represented with arcs. The Figure 6 presents our meta-model for FSM formalism

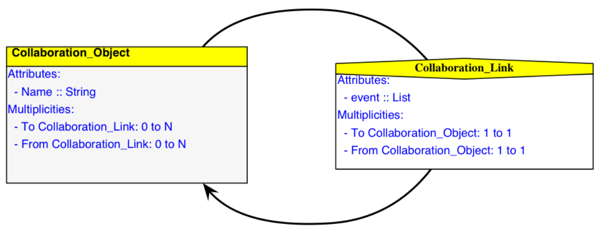
Collaboration_Object
Attributes:
- Name :: String
Multiplicities:
- To Collaboration_Link: 0 to N
- From Collaboration_Link: 0 to N
Collaboration_Link
Attributes:
- event :: List
Multiplicities:
- To Collaboration_Object: 1 to 1
- From Collaboration_Object: 1 to 1

Figure 5 – Meta-model of UML Collaboration Diagram

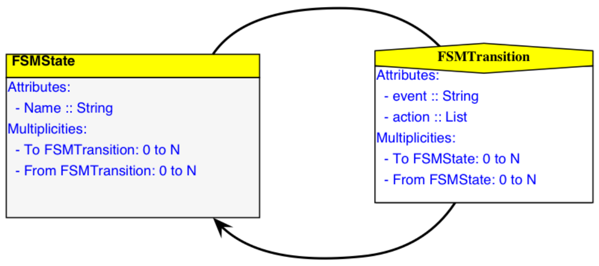

Figure 6 – Meta-model of Flat State Machines (FSM)

5.4 Object net model (ONM) Meta-Model

The last formalism used in our work is Object net model (ONM). ONM is a form of Object Petri Net which uses places and transitions to build models. To define meta-model for ONM we propose two classes: "ONMPlace" class to describe places and "ONMTransition" to describe transitions. These classes are related with two associations named "InputArc" and "OutputArc" which represent input arcs and output arcs as shown in the right part of Figure 7.

To fully define our meta-models, we have also specified the graphical appearance of each entity of the formalisms according to its appropriate notation.

Given our meta-models, we have used AToM3 tool to generate a visual modeling environment for all these formalisms. More precisely, AToM3 generates for each formalism a palate of buttons allowing the user to create the entities defined in its meta-model. Since AToM3 is a visual tool for multi-formalism modeling, we can show in a user interface of AToM3 all generated tools at the same time (see Figure 13).

6 Graph Transformation Approach of UML Models

In this section, we propose a Graph transformation approach to transform statechart diagrams and collaboration diagrams into CPN-like models for analysis purpose. For this regard, we have defined two graph grammars. The first one converts each statechart diagram into its equivalent flattened FSM model, whereas the second graph grammar constructs the single system-level model from the obtained FSM models and UML collaboration diagrams. Then any CPN analyzer tool can be used. In the following subsections we describe these graph grammars.

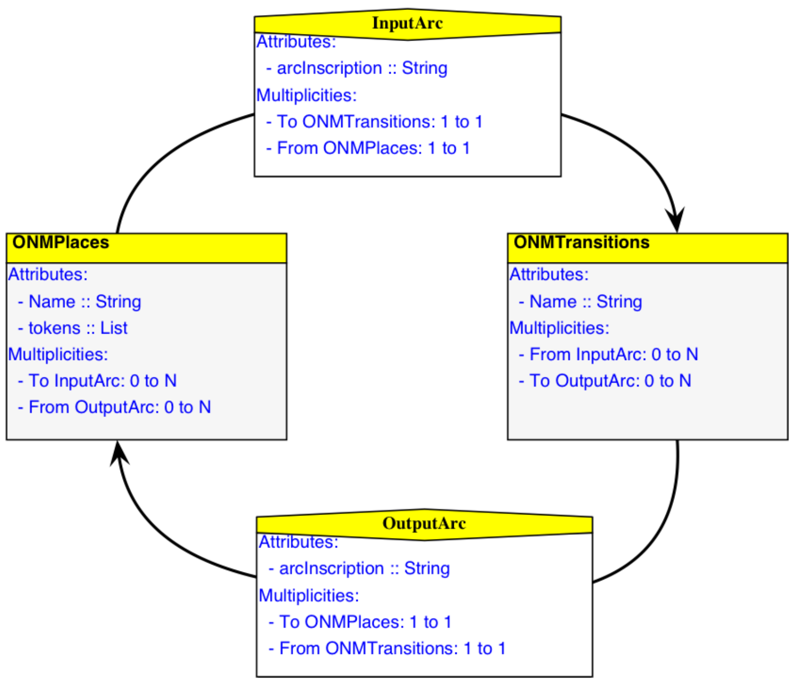

Figure 7 – Meta-model of Object net model (ONM)

6.1 1st GG: Converting Statechart Diagrams into FSM Models

We have named this graph grammar Statechart2FSM. To transform a statechart into its equivalent FSM model using our Statechart2FSM grammar, we have proposed twenty three rules (see Figures 8 and 9) which will be applied in ascending order. Note that each rule has a priority.

The idea of the transformation in Statechart2FSM grammar can be summarized on the following main steps:

The first step is to select a statechart form the set of statechart diagrams and to convert its initial state to FSM state (rule #22 for simple state or rule #23 for composite state).

The second step is to traverse the selected statechart from its initial state (which is processed in the first step). Each exit transition of a processed statechart state is converted into FSM transition and the destination statechart state is also converted if it has not previously processed (rule #5 and rule #6).

For the composite state in statechart, the graph grammar use flag attributes indicating that this composite state is traversed (T = = 1) or processed (P= = 1). For example, see rules #7, #8 and # 9.

In third step, all composite states in the statechart (if any) will be converted. The process of the conversion is the same for statechart diagrams (rule #1 and rule #2).

The last step is to relate the equivalent FSM segment of composite state to FSM model of the statechart. For example, rule #14 converts exit transiotion from composite state to simple state into its equivalent transition in FSM model.

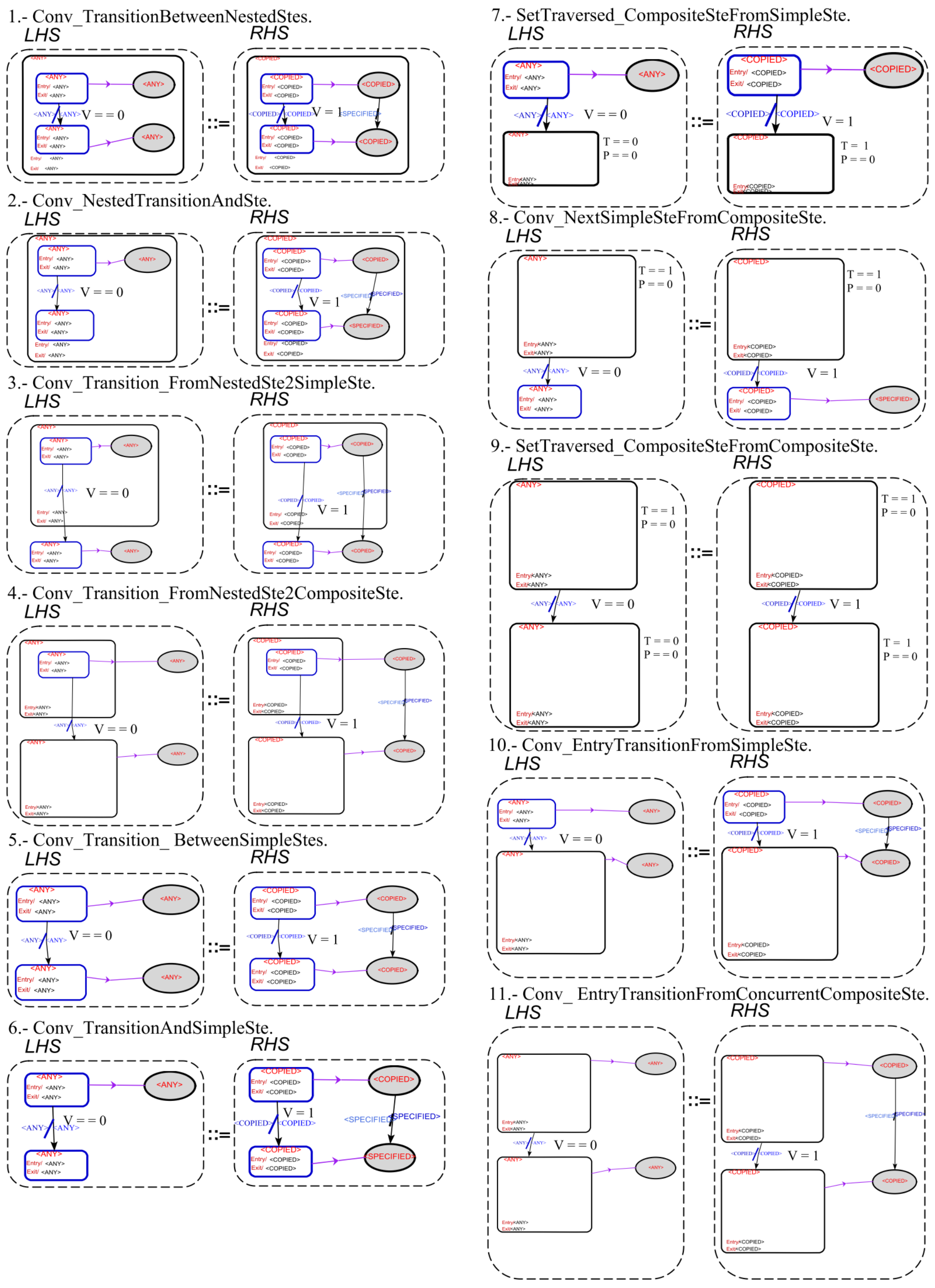
1.- Conv_TransitionBetweenNestedStes.
2.- Conv_NestedTransitionAndSte.
3.- Conv_Transition_FromNestedSte2SimpleSte.
4.- Conv_Transition_FromNestedSte2CompositeSte.
5.- Conv_Transition_ BetweenSimpleStes.
6.- Conv_TransitionAndSimpleSte.
7.- SetTraversed_CompositeSteFromSimpleSte.
8.- Conv_NextSimpleSteFromCompositeSte.
9.- SetTraversed_CompositeSteFromCompositeSte.
10.- Conv_ EntryTransitionFromSimpleSte.
11.- Conv_ EntryTransitionFromConcurrentCompositeSte.
LHS
RHS
::=
V = = 0
V = 1
T = = 1
P = = 0
T = = 0
T = 1
<ANY>
<COPIED>
<SPECIFIED>
Entry/
Exit/

Figure 8 – 1st Graph Grammar Rules 1-11

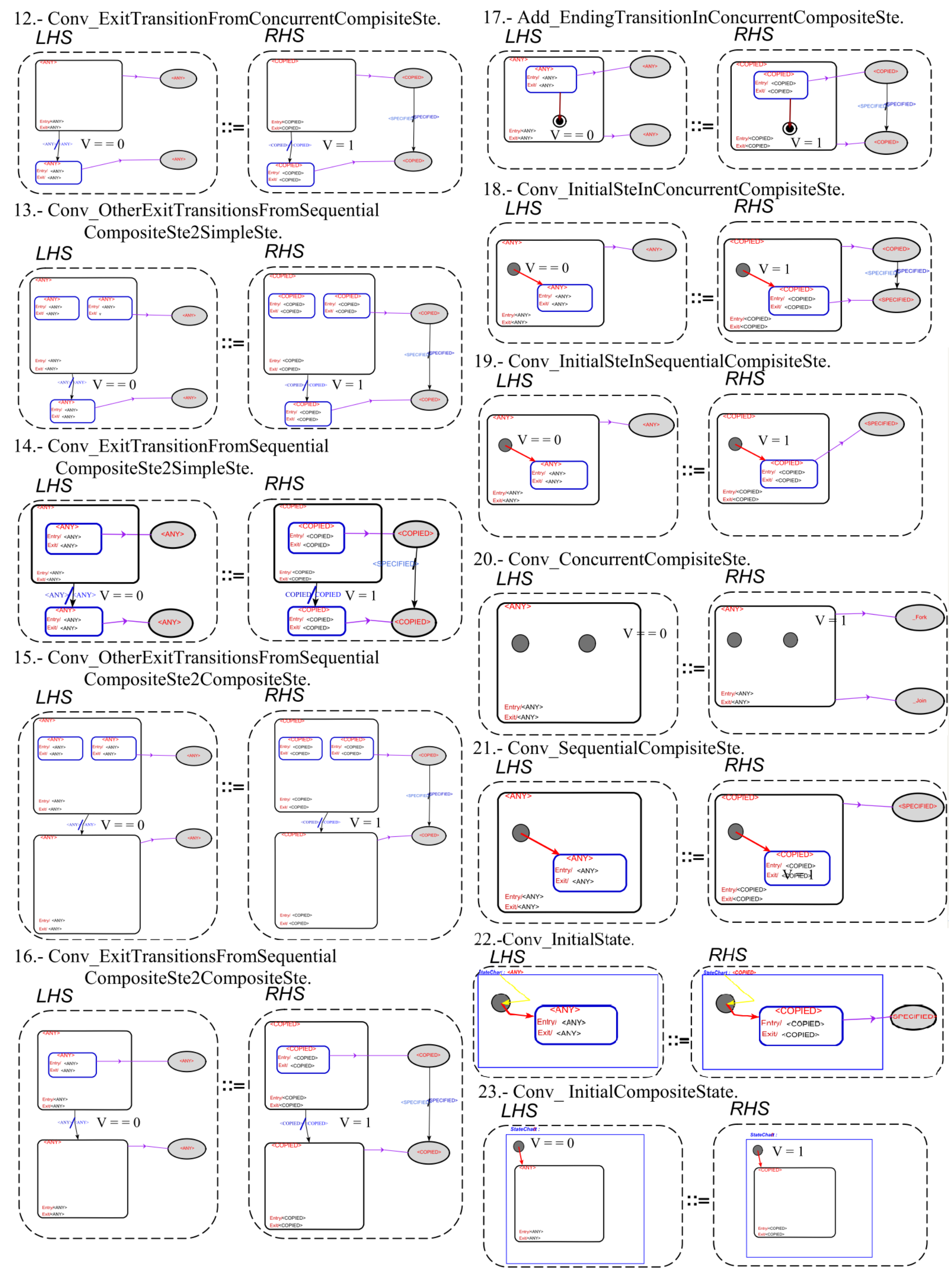
12.- Conv_ExitTransitionFromConcurrentCompisiteSte.
17.- Add_EndingTransitionInConcurrentCompositeSte.
13.- Conv_OtherExitTransitionsFromSequential CompositeSte2SimpleSte.
18.- Conv_InitialSteInConcurrentCompisiteSte.
19.- Conv_InitialSteInSequentialCompisiteSte.
14.- Conv_ExitTransitionFromSequential CompositeSte2SimpleSte.
20.- Conv_ConcurrentCompisiteSte.
15.- Conv_OtherExitTransitionsFromSequential CompositeSte2CompositeSte.
21.- Conv_SequentialCompisiteSte.
22.-Conv_InitialState.
16.- Conv_ExitTransitionsFromSequential CompositeSte2CompositeSte.
23.- Conv_ InitialCompositeState.
LHS
RHS
::=
V = = 0
V = 1
<ANY>
<COPIED>
<SPECIFIED>
Entry/ <ANY>
Exit/ <ANY>
Entry/ <COPIED>
Exit/ <COPIED>
_Fork
_Join
StateChart

Figure 9 – 1st Graph Grammar Rules 12-23

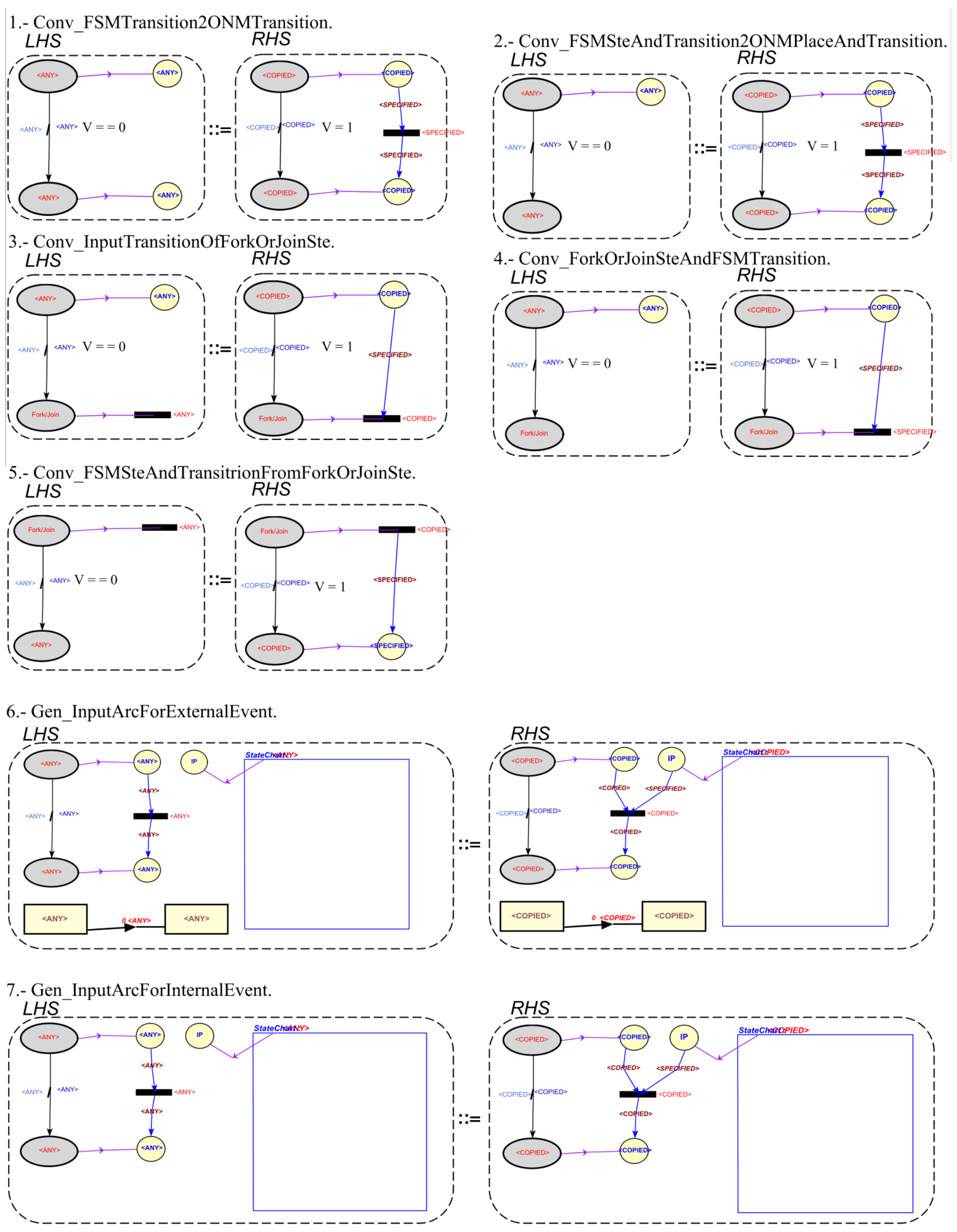
1.- Conv_FSMTransition2ONMTransition.
LHS
RHS
V = = 0
::=
V = 1
2.- Conv_FSMSteAndTransition2ONMPlaceAndTransition.
LHS
RHS
V = = 0
::=
V = 1
3.- Conv_InputTransitionOfForkOrJoinSte.
LHS
RHS
V = = 0
::=
V = 1
4.- Conv_ForkOrJoinSteAndFSMTransition.
LHS
RHS
V = = 0
::=
V = 1
5.- Conv_FSMSteAndTransitrionFromForkOrJoinSte.
LHS
RHS
V = = 0
::=
V = 1
6.- Gen_InputArcForExternalEvent.
LHS
RHS
::=
7.- Gen_InputArcForInternalEvent.
LHS
RHS
::=

Figure 10 – 2nd Graph Grammar Rules 1-7

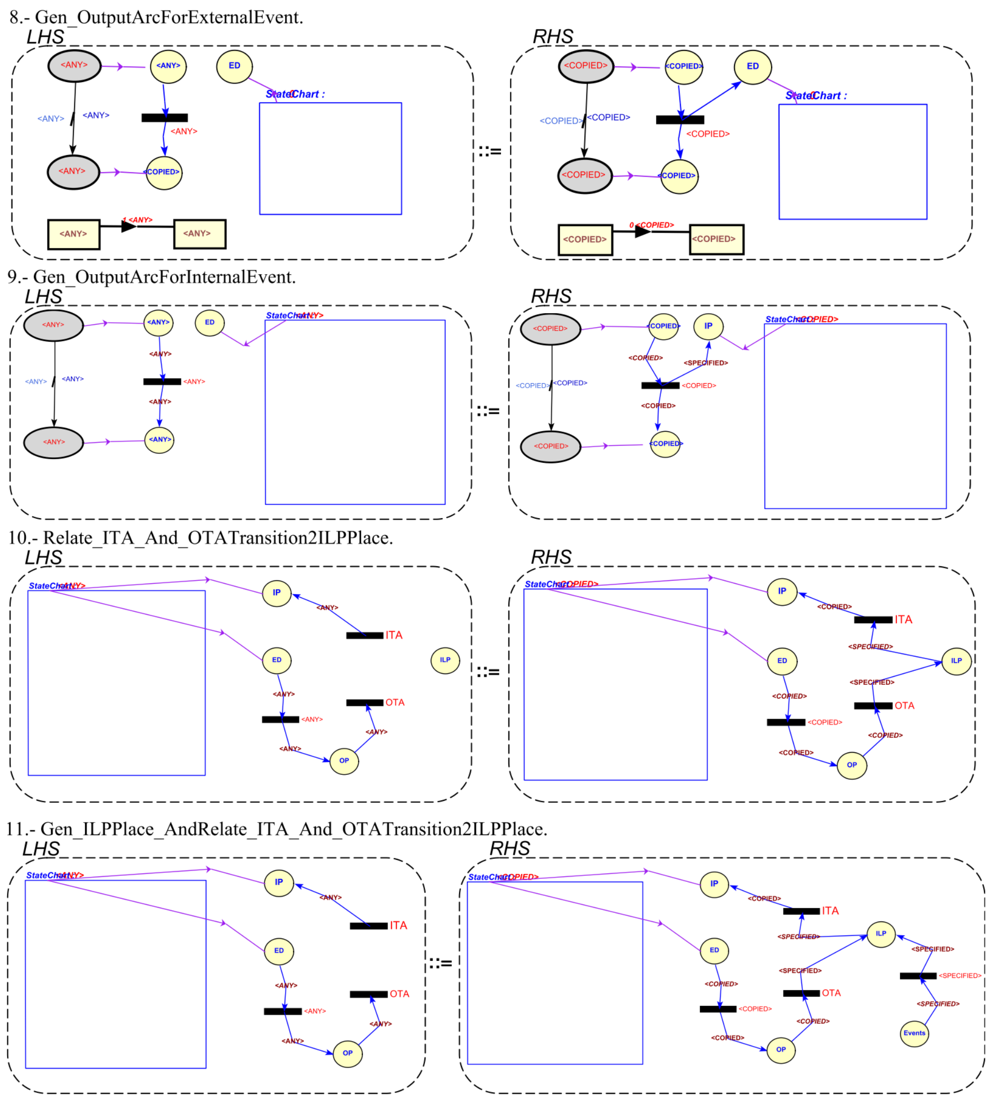
8.- Gen_OutputArcForExternalEvent.
LHS
RHS
::=
StateChart :
9.- Gen_OutputArcForInternalEvent.
LHS
RHS
::=
10.- Relate_ITA_And_OTATransition2ILPPlace.
LHS
RHS
::=
ITA
OTA
11.- Gen_ILPPlace_AndRelate_ITA_And_OTATransition2ILPPlace.
LHS
RHS
::=
ITA
OTA
Events

Figure 11 – 2nd Graph Grammar Rules 8-11

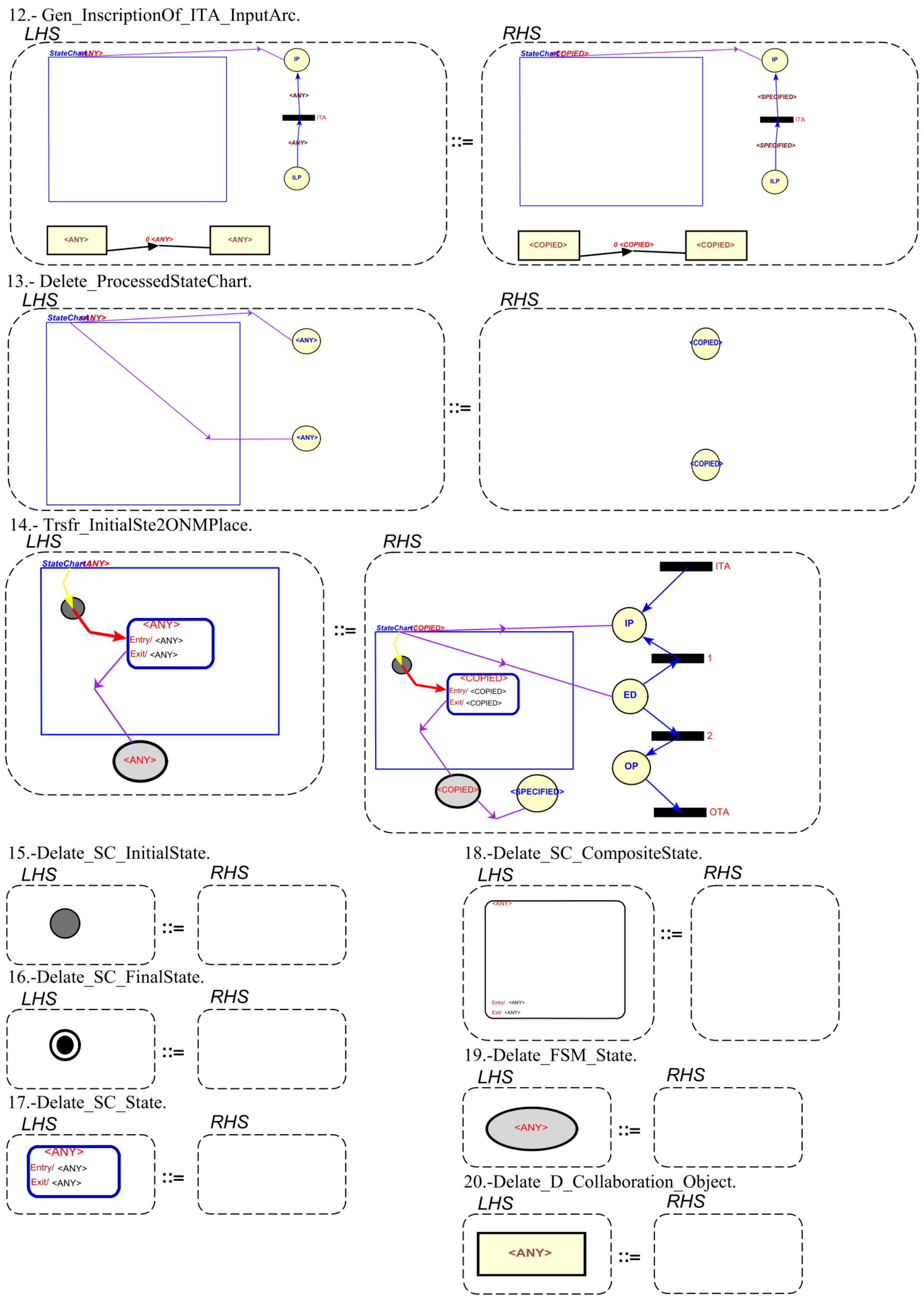
12.- Gen_InscriptionOf_ITA_InputArc.
LHS
RHS
StateChart<ANY>
IP
<ANY>
ITA
ILP
0 <ANY>
StateChart<COPIED>
<SPECIFIED>
<COPIED>
0 <COPIED>
::=
13.- Delete_ProcessedStateChart.
LHS
RHS
<COPIED>
14.- Trsfr_InitialSte2ONMPlace.
LHS
RHS
Entry/ <ANY>
Exit/ <ANY>
Entry/ <COPIED>
Exit/ <COPIED>
ED
OP
1
2
OTA
15.-Delate_SC_InitialState.
16.-Delate_SC_FinalState.
17.-Delate_SC_State.
18.-Delate_SC_CompositeState.
19.-Delate_FSM_State.
20.-Delate_D_Collaboration_Object.

Figure 12 – 2nd Graph Grammar Rules 12-20

6.2 2nd GG: Constructing Intermediate System level Model

We have named the second graph grammar FSM2ONMs. When the FSM2ONMs graph grammar execution finishes, the resulting model is the system-level model. Figure 10, 11 and 12 show the proposed rules.

The idea behind the transformation can be described in the following steps. The first one selects a statechart and creates its token routing structure (rule #14). The second step is to create for each FSM state an equivalent ONM place, and for each FSM Transition an equivalent ONM transition (rules #1 and #2). Note that the rules #3, #4 and #5 are used for transforming Fork or Join states into transitions. The transformation process is based on traversing FSM model as the same manner used in the first graph grammar.

At this point, LM should be linked to the token routing structure. For each FSM transition that generates events, an arc will be created from the equivalent ONM transition to ED place in the token routing structure. If the event is an exit event of the object in the collaboration diagram, the created output arc will have an inscription as external event (rule #8). Otherwise, it will be considered as an internal event (rule #9). Linking IP place to LM will be processed in same way as for the ED place but with input arc (rules #6 and #7). Then, ITA and OTA transitions are related to ILP place (rule #10 or rule #11 if ILP place is not previously created).

Finally, Statechart2FSM graph grammar deletes all elements of statechart diagrams, collaboration diagrams and FSM formalisms in order to have only the system-level model in ONM formalism (rules #13, #15, #16, #17, #18, #19 and #20).

7 Case Study: ATM Machine

We consider an example of an ATM machine, dispensing cash to a user [3]. The statechart diagrams and the collaboration diagram created in our tools are shown in Figure 13.

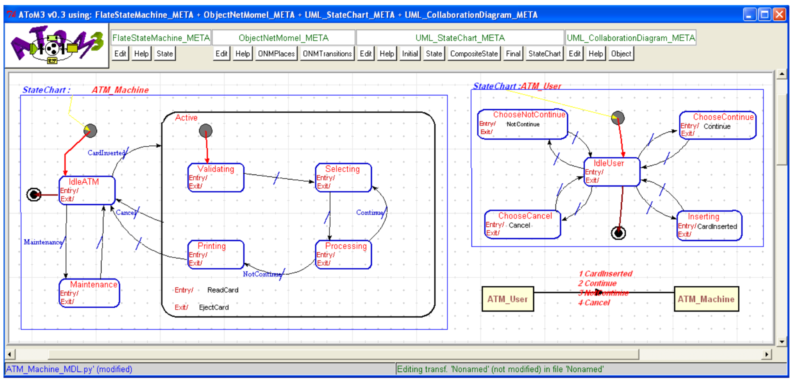

Figure 13 – Behavior specification of ATM Machine

To analyze this behavioral specification of the ATM Machine, we have to transform this specification into its equivalent system-level CPN-like model. To realize this transformation in our tools, we have to execute the Statechart2FSM graph grammar to obtain flattened FSM models from composite statechart diagrams. Then we have to execute also the FSM2ONMs graph grammar to synthesize the intermediate system-level model from the obtained FSM models and collaboration diagram. The resulting intermediate system-level model of the automatic transformation is shown in Figure 14.

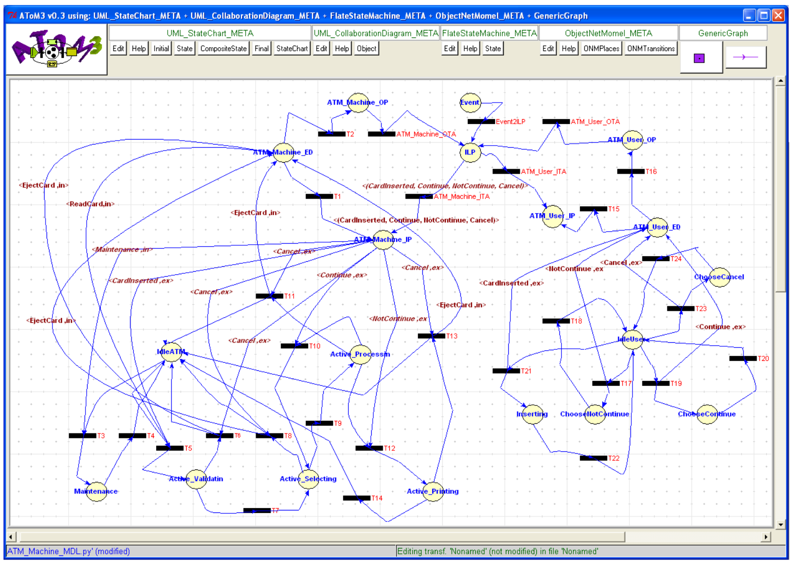

Figure 14 – Intermediate system-level model of ATM Machine

8 Conclusion

In this paper we have proposed an approach for transforming UML statechart and collaboration diagrams to Colored Petri nets. This transformation is intended to bridge the gap between informal notation (UML diagrams) and more formal notation (Colored Petri nets) for analysis purposes. It produces highly-structured, graphical, and rigorously-analyzable models that facilitate early detection of errors like deadlock and livelock. The approach is based on graph transformation where the input and output of the transformation process are graphs. The meta-modeling tool AToM3 is used. An example illustrates our approach. This paper is a step in project that is exploring means to exploit the INA analyzer tool. In future work we plan to transform other UML diagrams to Colored Petri nets and use the well-known reduction technique on the obtained models before performing the analysis to optimize the models.

References

[1] AGG Home page, `http://tfs.cs.tu-berlin.de/agg/`

[2] AToM3 Home page, version 3.00, `http://atom3.cs.mcgill.ca/`

[3] Grady Booch, Ivar Rumbaugh and Jim Jacobson: The Unified Modeling Language User Guide, Addison-Wesley, 1999.

[4] Spencer Borland and Hans Vangheluwe: "Transforming Statecharts to DEVS", Summer Computer Simulation Conference, Student Workshop, pp. 154-159, Society for Computer Simulation International (SCS), Montréal, Canada, 2003.

[5] Raida El Mansouri, Elhillali Kerkouche, and Allaoua Chaoui: " A Graphical Environment for Petri Nets INA Tool Based on Meta-Modeling and Graph

Grammars", Proceedings of World Academy of Science, Engineering and Technology, ISSN 2070-3740, vol. 34, pp.471-475, October 2008.

[6] EMF Home page, `http://www.eclipse.org/emf/`

[7] FUJABA Home page, `http://www.fujaba.de/`

[8] GEF Home page, `http://www.eclipse.org/gef/`

[9] GME Home page, `http://www.isis.vanderbilt.edu/gme/`

[10] GMF Home page, `http://www.eclipse.org/gmf/`

[11] GReAT `http://www.escherinstitute.org/Plone/tools/`

[12] INA Home page, `http://www2.informatik.hu-berlin.de/~starke/ina.html`

[13] Kurt Jensen: Coloured Petri Nets, Vol 1: Basic Concepts, Springer-Verlag 1992.

[14] Steven Kelly, Kalle Lyytinen and Matti Rossi: "MetaEdit+: A fully configurable Multi-User and Multi-Tool CASE and CAME Environment", Advanced Information System Engineering; LNCS 1080. Berlin, 1996. MetaEdit+ Home page, `http://www.MetaCase.com`

[15] Elhillali Kerkouche and Allaoua Chaoui: "A Formal Framework and a Tool for the Specification and Analysis of G-Nets Models Based on Graph Transformation", International Conference on Distributed Computing and Networking -CDCN'09-, LNCS 5408, pp. 206–211, Springer-Verlag Berlin Heidelberg, India, 3-6 January, 2009.

[16] Elhillali Kerkouche, Allaoua Chaoui, El Bay .Bourennane, and Ouassila Labbani: "Modeling and verification of Dynamic behaviour in UML models, a graph transformation based approach", proceedings of SEDE'2009, Las Vegas, Nevada, USA, 22-24 June 2009.

[17] Juan De Lara and Hans Vangheluwe: "AToM3: A Tool for Multi-Formalism Modeling and Meta-Modeling". LNCS 2306, pp.174-188. Presented also at Fundamental Approaches to Software Engineering - FASE'02 , in European Joint Conferences on Theory And Practice of Software - ETAPS'02, Grenoble, France, 2002.

[18] Juan De Lara and Hans Vangheluwe: "Computer aided multi-paradigm modeling to process petri-nets and statecharts", International Conference on Graph Transformations (ICGT), Lecture Notes in Computer Science, vol. 2505, pp. 239-253, Springer-Verlag, Barcelona, Spain, 2002.

[19] Juan De Lara and Hans Vangheluwe: "Meta-Modeling and Graph Grammars for Multi-Paradigm Modeling in AToM3". Software and Systems Modeling, Vol. 3, pp. 194-209, Springer-Verlag, Special Section on Graph Transformations and Visual Modeling Techniques, 2004.

[20] Tadao Murata: "Petri Nets: Properties, Analysis and Applications", Proceedings of the IEEE, Vol. 77, no, pp. 541-580, 4 April 1989.

[21] PROGRES Home page, `http://www-i3.informatik.rwth-aachen.de/research/projects/progres/main.html`

[22] Python Home page, htpp://www.python.org

[23] Grzegorz Rozenberg: Handbook of Graph Grammars and Computing by Graph Transformation, World Scientific, 1999.

[24] John Anil Saldhana and Sol M. Shatz and Zhaoxia Hu: “Formalisation of Object Behavior and Interaction From UML Models”, International Journal of Software Engineering and Knowledge Engineering. Vol. 11, #6, pp. 643-673, 2001.

[25] Sally Shlaer and Stephen Mellor: Object Life Cycles – Modeling the World in States, Yourdon Press, Prentice Hall. 1992.

[26] TIGER Home page `http://tfs.cs.tu-berlin.de/tigerprj/`

About the authors

Elhillali Kerkouche is Assistant Professor in the department of Computer science, University of Oum Ebouaghi, Algeria. His research field is UML, formal methods and Distributed Systems. He can be reached at `elhillalik@yahoo.fr`.

Allaoua Chaoui is with the department of computer science, Faculty of Engineering, University Mentouri Constantine, Algeria. He received his Master degree in Computer science in 1992 (in cooperation with the University of Glasgow, Scotland) and his PhD degree in 1998 from the University of Constantine (in cooperation with the CEDRIC Laboratory of CNAM in Paris, France). He has served as associate professor in Philadelphia University in Jordan for five years and University Mentoury Constantine for many years. During his career he has designed and taught courses in Software Engineering and Formal Methods. Dr Allaoua Chaoui has published many articles in International Journals and Conferences. He supervises many Master and PhD students. His research interests include Mobile Computing, formal specification and verification of distributed systems, and graph transformation systems. He can be reached at `chaoui2001@yahoo.com`.

El Bay Bourennane is Professor of Electronics at the laboratory LE2I, (Laboratory of Electronics, Computer Science and Image). University of Bourgogne, Dijon, France. His research field is UML and embedded systems. He can be reached at `ebourenn@u-bourgogne.fr`.

Ouassila Labbani is an Associate Professor at the laboratory LE2I, (Laboratory of Electronics, Computer Science and Image). University of Bourgogne, Dijon, France. Her research field is UML, embedded systems and formal methods. She can be reached at `Ouassila.Labbani@u-bourgogne.fr`.