

YASMINA RAHMOUNE  
ALLAOUA CHAOUI

## AUTOMATIC BRIDGE BETWEEN BPMN MODELS AND UML ACTIVITY DIAGRAMS BASED ON GRAPH TRANSFORMATION

**Abstract** *Model-driven engineering (MDE) provides the available tools, concepts, and languages for creating and transforming models. One of the most important successes of MDE is model transformation; it permits the transformation of models that are used by one community to equivalent models that can be used by another one. Moreover, each community of developers has its own tools for verification, testing, and test-case generation. Hence, a developer of one community who moves to another community needs a transformation process from the second community to his/her own community and vice versa. Therefore, the target community can benefit from the expertise of the source one, and the developers do not begin from zero.*

*In this context, we propose an automatic transformation in this paper for creating a bridge between the BPMN and UML communities. We propose an approach and a visual tool for the automatic transformation of BPMN models to UML activity diagrams (UML-AD). The proposed approach is based on meta-modeling and graph transformation and uses the AToM<sup>3</sup> tool. Indeed, we were inspired by the OMG meta-models of BPMN and UML-AD and implemented versions of both meta-models using AToM<sup>3</sup>. This latter one allows for the automatic generation of a visual-modeling tool for each proposed meta-model. Based on these two meta-models, we propose a graph grammar that is composed of 58 rules that perform the transformation process. The proposed approach is illustrated through three case studies.*

**Keywords** MDE, BPMN, Business Process Models, UML-AD, Meta-Modeling, Graph Grammars, Models Transformation, AToM<sup>3</sup>

**Citation** Computer Science 23(3) 2022: 411–447

**Copyright** © 2022 Author(s). This is an open access publication, which can be used, distributed and reproduced in any medium according to the Creative Commons CC-BY 4.0 License.

## 1. Introduction

A business process model (BPM) describes the business processes of organizations and modern enterprises. BPM is used to understand information and activities in order to reach certain goals. For describing BPM, researchers can use different notations such as BPMN [34], UML-AD [35], Eriksson-Penker's notation [21], etc.

Recently, the business process model and notation (BPMN) has become one of the standard graphical modeling languages of business processes. It is used by business process managers for creating and representing their organizations' models (diagrams). BPMN contains a variety of symbols to describe the process in a detailed and efficient way.

The unified modeling language (UML) is more suitable for software developers and its supported tools. UML is the de-facto standard for software design; it has been widely adopted in the industry. The UML activity diagram specifies the dynamic (behavioral) structure of a system by describing the activities, choices, interactions, and concurrency of a workflow or a process [6]. The major advantage of the activity diagram is its simplicity and its ease for understanding the logical flow of a modeled system [43]. Hence, BPMN and UML are two alternative modeling specifications for business process models. The communication between these communities allows for taking benefits from both the tools of business process management and software development.

In this paper, we propose an automatic approach for transforming BPMN to UML-AD based on a transformation model of model-driven engineering. Furthermore, our global objective is to create a bridge between the BPMN and UML communities and create a framework that permits developers to reuse the existing BPM of the first notation (BPMN) in the second notation (UML). We propose an automatic transformation approach and a visual tool that are based on meta-modeling and graph grammars. To do so, we propose two meta-models that are associated with BPMN and UML-AD, respectively. Based on these meta-models, we propose a graph grammar that contains 58 rules that automatically perform the transformation process. The proposed graph grammar is executed using AToM<sup>3</sup> (a tool for multi-formalism and meta-modeling) [10, 12]. This tool proves its capacity and power for enabling meta-modeling and transformations of known formalisms [23].

In this work, we focus on the transformation of flow object elements (events, activities, and gateways) and data elements (data objects, data inputs, data outputs, data stores, etc.) that were not studied and discussed in the previous works that addressed the mapping of BPMN and UML-AD.

The principal objectives of this approach are as follows:

- Exploiting the principles and techniques of MDE approaches (such as meta-modeling and graph grammars) to create a bridge between two different communities: BPMN and UML-AD. This transformation takes the benefits of both tools for business process management and software development. This also allows the

exchange between these communities. Moreover, the target community (UML) can use BPMN as a source model even if it is unknown to the community.

- Proposing an automatic visual tool for transforming business process models to UML activity diagrams. Our tool allows us to re-express and rewrite a business process model into its equivalent model (UML-AD in this case). BPMN contains several rich, complex, and non-atomic elements. The proposed tool extracts these elements and represents them in a UML activity diagram. For example, the semantics of a message start event in a BPMN is the combination of accepting a message and starting a process; our tool decomposes this event into three elements in UML-AD (Initial node, Control Flow, and AcceptCallAction – see Case Study 3).
- Enabling the existing business process models to benefit from all of UML's advantages (methods, development tools, and verification) and validation techniques. Actually, we can check some of the properties (deadlock, incoherence, inconsistencies, etc.) of the target model (UML-AD) by using existing methods such as the one that was proposed in [36]. This verification provides feedback on the properties of the source model (BPM modeled in the input). It is noteworthy that we use the techniques of MDE for model transformation as well as their verifications by different specifications such as those that were presented in [15, 16, 24, 26, 33]. Furthermore, we can use generation test cases like those that were presented in [22].

This paper is organized as follows; Section 2 presents related works, and Section 3 recalls the basic notions of model transformation (meta-modeling, graph grammar and the AToM<sup>3</sup> tool). We present the proposed automatic approach in Section 4, while Section 5 presents some examples for illustrating our automatic approach. In order to ensure the correctness of our transformation, we present the verification of the most important properties in Section 6. In the last section, we give our conclusions and draw some perspectives from the work.

## 2. Related Works

Over the past decade, much research has been done in the domain of business process modeling and raised many challenges in modernizing enterprises; BPMN and UML-AD are the most used notations in this field. In [44], the author reviewed the ability of representing 21 workflow patterns for describing the behavior of business processes by BPMN and UML-AD. The author compared the results of the two notations with respect to the technical ability to represent the patterns as well as their readability. In [41, 45], the authors studied the suitability of UML 2.0 activity diagrams (ADs) for business process modeling, and they used workflow patterns (WPs) as an evaluation framework. They presented all of the proposed groups of WPs as well as their abilities. They provided a complete evaluation of the capabilities of UML 2.0 ADs, BPMN, and the business process execution language (BPEL) and showed their strengths and weaknesses when utilized for business process modeling.

In [37], UML and BPMN were compared based on their complexity levels; it was discovered that BPMN had a very high level of complexity as opposed to UML. In [7], the authors presented the results of a comparison of both notations during the process of creating an application model by business users; this indicated that an activity diagram is just as useful as the BPMN model in any case.

Likewise, another study [42] presented a method for translating business models such as BPMN and DMN (decision model and notation) into a set of consistent UML models (which can be later used by business analysts and developers for understanding and implementing the system). As a single-notation design, this allows users to take advantage of software that supports UML modeling and consistency checking. The limitation of this method is that, for business users, UML is too technical and complex in the preparation of models.

In addition, a UML profile is used to extend UML's general-purpose language. There has been some research that studied the transformation of BPMN to a UML profile (e.g., [2, 27, 29]).

In [27], the author developed a UML profile for an event-driven process chain (EPC) to facilitate software developers, as a UML activity diagram is close to the design/model of a software project. EPC is widely used in the industry, but it is not a standard language. In contrast, BPMN has been an OMG standard since 2005.

In [29], a UML profile for business process modeling was developed to help software developers view business process modeling in familiar notations. For this purpose, two perspectives were considered; business perspective (which focused on goals, deliverables, and customers) and sequence perspectives (which was used to refine the business perspective).

In [2], the authors proposed a UML profile for business process modeling notation (UMLPBPMN). Their goal was that a UML model would be accessible to a software engineer without him/her being forced to master the particulars for understanding BPMN or relate it with any software requirements. Moreover, the communication time was significantly reduced, and the understandability and synchronization were highly increased; this could ultimately boost the productivity of a software product.

Other works have been proposed to translate BPMN to a UML use case diagram. We can cite the work in [8] where the authors developed an MDA approach (BPMN2UC) to generate UML use cases that represent the user requirements of an information system (IS) to bridge the gap between BPMs and information system models.

In [9], the author discussed the importance of this mapping to satisfy business process requirements and facilitate the alignment among the BP and IT solutions of building a BPM in business-driven development (BDD). The author claimed that the translation of BPMN diagrams to UML activity diagrams was required for the development of systems, because UML had become the de-facto standard modeling language for object-oriented systems. Besides, UML has rich tools for modeling software systems and is easier to be read by end users. The author presented the challenges of defining and implementing this translation. The author proposed the use



of the ATLAS Transformation Language (ATL) as a model transformation language but did not detail the proposal.

In [30], Macek and Richta presented a transformation from BPM to UML activity diagrams using XSLT (XSL Transformations). They used an extensible markup language (XML) document as an intermediate notation between BPM and UML-AD. The authors discussed the need for transforming business process models to UML-ADs but not to other UML diagrams. They explained the drawbacks of the transformation of BPM to use case diagrams (presented in [38]) and to class diagrams (presented in [39]). The disadvantages of this approach consist of the use of intermediate models and the manual creation of the graphical layout of the AD.

This paper [18] presented a synthetic analysis of BPMN and UML-AD according to three criteria: the capacity of being readily understandable, the adequacy of the graphical elements of BPMN and UML AD to represent the real business processes of an organization, and the mapping to business process execution language (BPEL).

In the previous works, the researchers discussed the necessity for transforming BPMN to UML diagrams and proposed general ideas (UML-AD, a UML profile, a use case model) that lacked a complete transformation.

The strong point of our approach is the use of a graph transformation approach to realize the automatic transformation of BPMN modes to a UML-AD model and take the primary benefit of the important advantage of graph transformation (which is graph grammar – it has a mathematical foundation and is a generalization of Chomsky grammars for graphs [40]). This represents the first advantage of this research. The second advantage consists of the creation of a visual tool for manipulating a business process model, an UML-AD model, and their automatic transformation. The third advantage of our proposal is that the transformation is performed at a high level of abstraction, as our rules are defined in the meta-model level. Finally, we believe that the transformation between these two notations can serve as a bridge between the tools that support the business process management and the UML tools for software development.

### 3. Background of Model Transformation

#### 3.1. Graph transformations

Model transformation is a very important operation in any model-driven approach. Actually, such transformations assure the translation of operations regarding one or more models of a given level of abstraction as well as to one or more other models of the same level (horizontal transformation) or a different level (vertical transformation). Graph transformation is a particular and well-known type of model transformation. Generally, graph transformations consist of two steps: meta-modeling and graph grammar.

- Meta-modeling techniques are widely used when describing the different kinds of formalisms that are needed for the specifications and designs of systems. The definition of meta-models requires the definition of two syntaxes. In the first syntax, we define the abstract formal syntax to designate the formalism's entities, their relationships, their attributes, and the constraints. In the second one, we define the concrete graphical syntax to represent the graphical appearance of these entities and their relationships. The advantage of this technique is that the generated tool accepts only syntactically correct models according to the formalism definition. For more details, the reader is referred to [11].
- A graph grammar [40] has a mathematical foundation and is a generalization of Chomsky grammar for graphs. This is a formalism in which the transformation of graph structures can be modeled and studied.

### 3.2. Graph grammar

According to Rozenberg [40], a graph grammar is defined as follows:

A graph grammar ( $G$ ) consists of a set of production rules ( $P$ ) and a start graph ( $G_0$ ). A sequence of direct derivations ( $p = (G_0 \xrightarrow{p_1} G_1 \xrightarrow{p_2} \dots \xrightarrow{p_n} G_n)$ ) constitutes a derivation of the grammar (also denoted by  $G_0 \Longrightarrow^* G_n$ ). The  $L(G)$  language that is generated by grammar  $G$  is the set of all graphs  $G_n$  such that  $G_0 \Longrightarrow^* G_n$  is a derivation of the grammar.

Graph grammars are composed of production rules, with each rule having graphs on the left-hand and right-hand sides (LHS and RHS, respectively). The LHS of the rules are compared with an input graph that is called the host graph. If a match is found between the LHS of a rule and a sub-graph in the host graph, the rule can be applied, and the matching sub-graph of the host graph is replaced by the RHS of the rule. A rewriting system iteratively applies matching rules in the grammar to the host graph until no more rules can be applied [14]. In our case, AToM<sup>3</sup> allows for an order of rules that is based on a user-assigned priority [13].

### 3.3. Principle of transformation rules

A graph transformation rule is defined by  $r = (L, R, K, \text{glue}, \text{emb}, \text{cond})$ , where

- L: left-side graph;
- R: right-side graph;
- K: subgraph of L;
- glue: occurrence of K (subgraph of L) in R (right-side graph);
- emb: embedding relationship emb that connects vertices of L and those of R;
- cond: set of conditions for applying rule r.

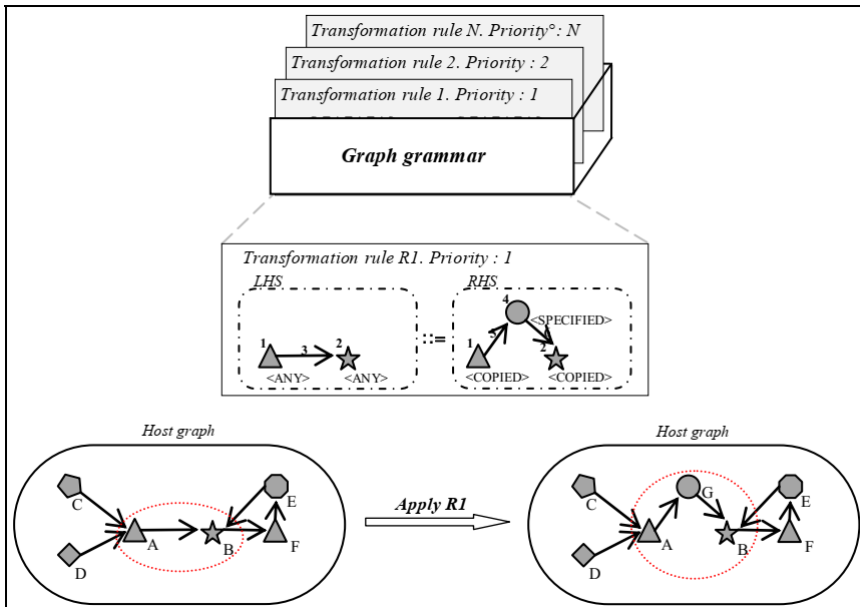
### 3.4. Application of rules

Applying rule  $r = (L, R, K, \text{glue}, \text{emb}, \text{cond})$  to GRAPH  $G$  produces resulting GRAPH  $H$ . The provided graph  $H$  can be obtained from the original graph  $G$  through the following five steps:

1. choose occurrence of left-side graph  $L$  in  $G$ ;
2. check application conditions according to  $\text{cond}$ ;
3. remove occurrence of  $L$  (up to  $K$ ) from  $G$  as well as dangling arcs, which are all arcs that have lost their sources and/or destinations – this provides context graph  $D$  of  $L$ ;
4. paste context graph  $D$  and right-side graph  $R$  by following occurrence of  $K$  in  $D$  and  $R$  – this is construction of disjunction union of  $D$  and  $R$  and, for each point in  $K$ , identify corresponding point in  $D$  with corresponding point in  $R$ ;
5. embed right-side graph into context graph of  $L$  following embedding relationship  $\text{emb}$ .

The application of  $r$  on a graph  $G$  to provide a graph  $H$  is called a direct derivation from  $G$  to  $H$  through  $r$ ; this is denoted by  $G \Rightarrow H$ .

The main idea of graph transformations is the rule-based modification of the graphs. Figure 1 shows the general principle of applying a rule to a graph.



**Figure 1.** Principle of application of rule [25]

The use of graph transformations has some advantages over an implicit representation [13]; it is an abstract, declarative, high-level representation. This enables the

exchange, re-use, and symbolic analysis of the transformation model. The theoretical foundations of graph rewriting systems may assist in proving the correctness and convergence properties of the transformation tool.

### 3.5. AToM<sup>3</sup>

In AToM<sup>3</sup>, we can use either the entity-relationship or the UML class diagram meta-formalisms for meta-modeling. A meta-formalism can be used to define formalisms as well as other meta-formalisms; in addition, meta-models can be provided with textual constraints that are expressed as OCL or Python code [13].

Additionally, AToM<sup>3</sup> allows us to define visual languages by means of meta-models. Model manipulation can be expressed as either Python programs or by means of attribute graph grammars. AToM<sup>3</sup>'s graph rewriting processor can be configured to work in the single or double pushout approaches [40].




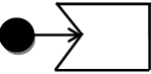
In our contribution, we have realized the transformation of BPMN models to UML-AD models by a combination of meta-modeling and graph grammar using the AToM<sup>3</sup> tool.




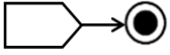



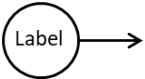
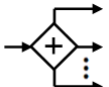
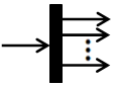
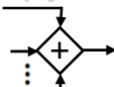
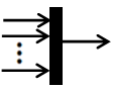
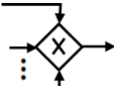
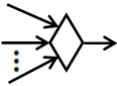
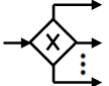
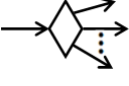
## 4. Proposed Approach

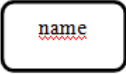


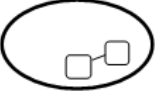

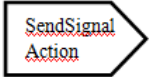

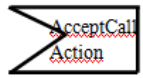



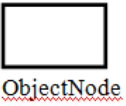

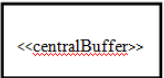
In this section, we present our automatic approach as well as the visual tool that is proposed for transforming the business process models that are created by BPMN into their equivalent UML activity diagrams. This transformation enables us to create a bridge between these two standards.

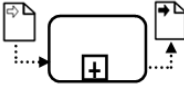


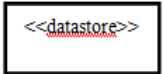
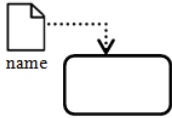

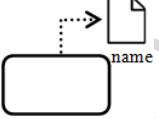
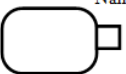
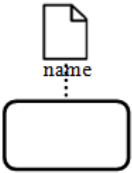
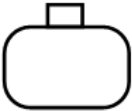
We start our contribution by proposing the correspondences between the most elements of BPMN2.0.2 and UML Activity Diagram 2.5 (as shown in Table 1).

**Table 1**  
Correspondences between BPMN and UML-AD

<i>Description in BPMN</i>	<i>Graphical representation</i>	<i>Description in UML-AD</i>	<i>Graphical representation</i>
<b>Start Event</b> indicates where particular process will start		<b>Initial Node</b> is control node that acts as starting point for executing activity	
<b>Message Start Event</b> – message arrives from participant and triggers start of process		Following group (Initial Node, Control Flow, and Accept Call Action)	

<p><b>End Event</b> indicates where process will end</p>		<p><b>Final Node</b> is control node at which flow in activity stops (activity final)</p>	
<p><b>Message End Event</b> finishes process and sends message to participant</p>		<p>Following group (Send Signal Action, Control Flow, and Flow Final Node)</p>	
<p><b>Link Intermediate Events (throw)</b> is mechanism for connecting two sections of process – it is used for throwing</p>		<p><b>Connector</b> is used for connecting two sections of activity diagram – it is employed for throwing</p>	
<p><b>Link Intermediate Events (catch)</b> is mechanism for connecting two sections of process – it is used for catching</p>		<p><b>Connector</b> is used for connecting two sections of activity diagram – it is employed for catching</p>	
<p><b>Parallel Split gateway (AND-Split)</b></p>		<p><b>Fork Node</b> is control node that splits flow into multiple concurrent flows</p>	
<p><b>Synchronization gateway (AND-Join)</b></p>		<p><b>Join Node</b> is control node that synchronizes multiple flows</p>	
<p><b>Simple Merge gateway (OR-Join)</b></p>		<p><b>Merge Node</b> is control node that brings multiple flows together without synchronization</p>	
<p><b>Exclusion Choice gateway (OR-Split)</b></p>		<p><b>Decision Node</b> is control node that chooses between outgoing flows</p>	

<p><b>Activity</b> is generic term for work that company performs in process – activity can be atomic</p>		<p><b>Action</b> is executable activity node that is fundamental unit of executable functionality in activity</p>	
<p><b>Sub-Process</b> is compound activity that is included within process or choreography</p>		<p><b>Sub-Activity</b> is set of actions and control node using control and data flow</p>	
<p><b>Intermediate Event</b> «Throwing» signal and message</p>		<p><b>SendSignalAction</b></p>	
<p><b>Intermediate Event</b> «Catching» signal and message</p>		<p><b>AcceptCallAction</b></p>	
<p><b>Timer Intermediate Event</b></p>		<p><b>Time Event generating</b></p>	
<p><b>Data Object</b> provides information about which activities required to be performed and/or what they produce</p>		<p><b>Object Node</b> is used to hold value-containing object tokens during course of execution of activity</p>	
<p><b>Data Object Collection</b> can represent collection of objects</p>		<p><b>Central Buffer Node</b> acts as buffer between incoming object flows and outgoing object flows</p>	

<p><b>Data Input and Data Output</b> provide same information for processes</p>		<p><b>Activity Parameter Node</b> accepts input to activity or provides output from activity</p>	
<p><b>Data Store</b> provides mechanism for activities to retrieve or update stored information that will persist beyond scope of process</p>		<p><b>Data Store Node</b> is central buffer node that holds its object tokens persistently while its activity is executing</p>	
<p>Data Object is associated with inputs of activities (tasks) – data association is used to move data between them</p>		<p>Action Pin is used to define data values that are passed out of and into action – <b>input pin</b> provides value to action</p>	<p>Name</p>  <p>Input pin</p>
<p>Data Object is associated with outputs of activities (tasks) – data association is used to move data between them</p>		<p>Action Pin is used to define data values that are passed out of and into action – <b>output pin</b> contains results from this action</p>	<p>Name</p>  <p>Output pin</p>
<p>Data Object is associated directly with activity (task)</p>		<p><b>Value Pin</b> provides value by evaluating value specification; e.g., this may be used as simple way to specify constant inputs to action</p>	<p>Name</p> 

#### 4.1. Overview of Proposed Approach

The proposed approach is based on a combination of meta-modeling and graph grammars. First, we were inspired by the OMG meta-models and redefined two meta-models for the basic category of business process models (source models) and UML activity diagrams (target models). We present these last meta-models in UML class diagrams by using the AToM<sup>3</sup> meta-tool, which allows us to generate a visual-modeling tool for each proposed formalism.

Second, we propose a graph grammar that contains 58 rules that automatically perform transformations of business process models to UML-ADs. This graph grammar is the core of our work. Furthermore, we define for each rule:

- the left and right sides;
- initialization;
- pre and post-conditions;
- actions.

These last three items are expressed in Python [17].

In this study, we have only focused on the transformations of BPMs to UML-ADs at the meta-model level and used model-to-model transformation, as the two parts of our rules are graphical models. We will describe these in detail and show some rules later. Figure 2 provides an overview of the proposed approach.

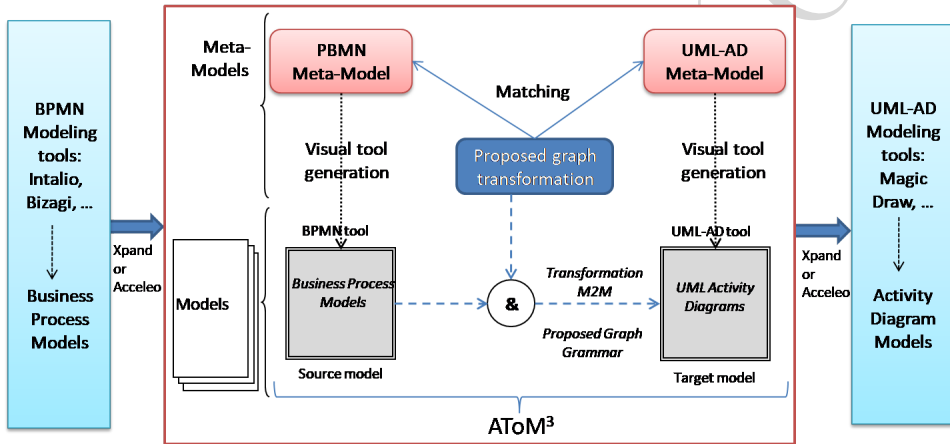


Figure 2. Overview of proposed approach

## 4.2. Business Process Meta-Model

We have proposed the meta-model that is shown in Figure 3, which contains 28 classes that are linked by 3 association and inheritance relationships. In this meta-model, we have taken the most used elements into account.

From this meta-model, we used AToM<sup>3</sup> to automatically generate a visual-modeling environment for manipulating BPMs (as shown in Figure 4). It contains a set of buttons that allow the user to manipulate (create, edit, etc.) business process models that conform to the above-presented meta-model in a graphical manner.

## 4.3. Activity Diagram Meta-Model

We have proposed the meta-model presented in Figure 5, which contains 17 classes that are linked by 6 association and inheritance relationships.



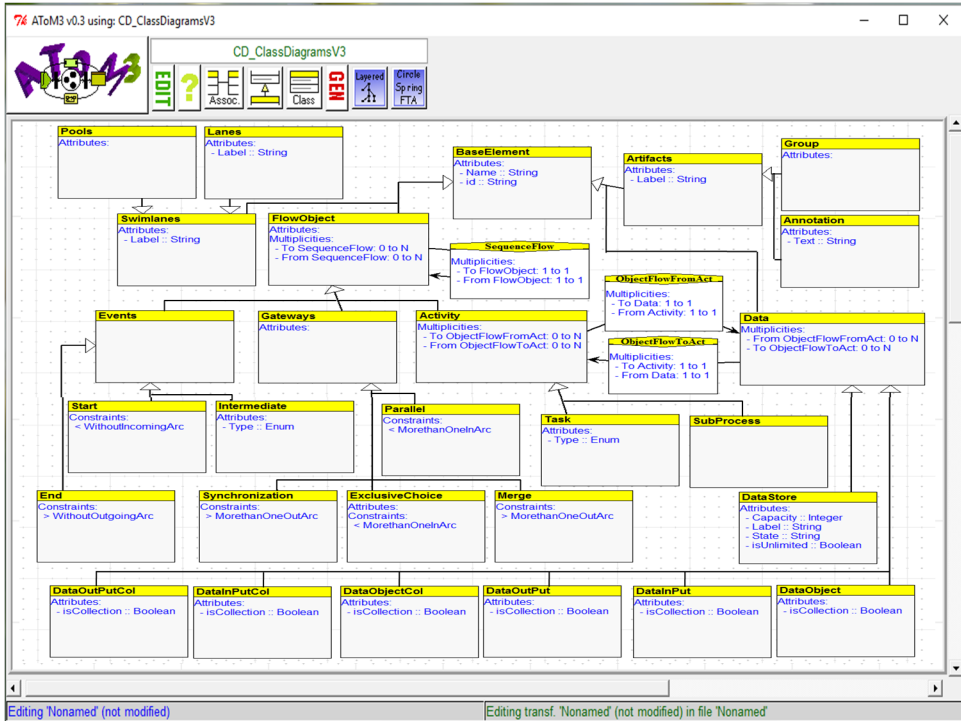


Figure 3. Proposed business process meta-model

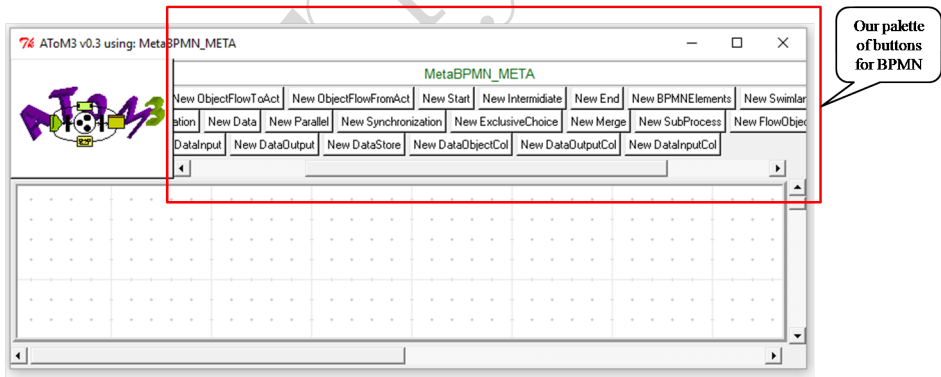


Figure 4. Generated tool for business process models

From this meta-model, we used AToM<sup>3</sup> to automatically generate a visual-modeling environment for manipulating UML-ADs (as shown in Figure 6). This contains a set of buttons that allow the user to manipulate (create, modify, etc.) the

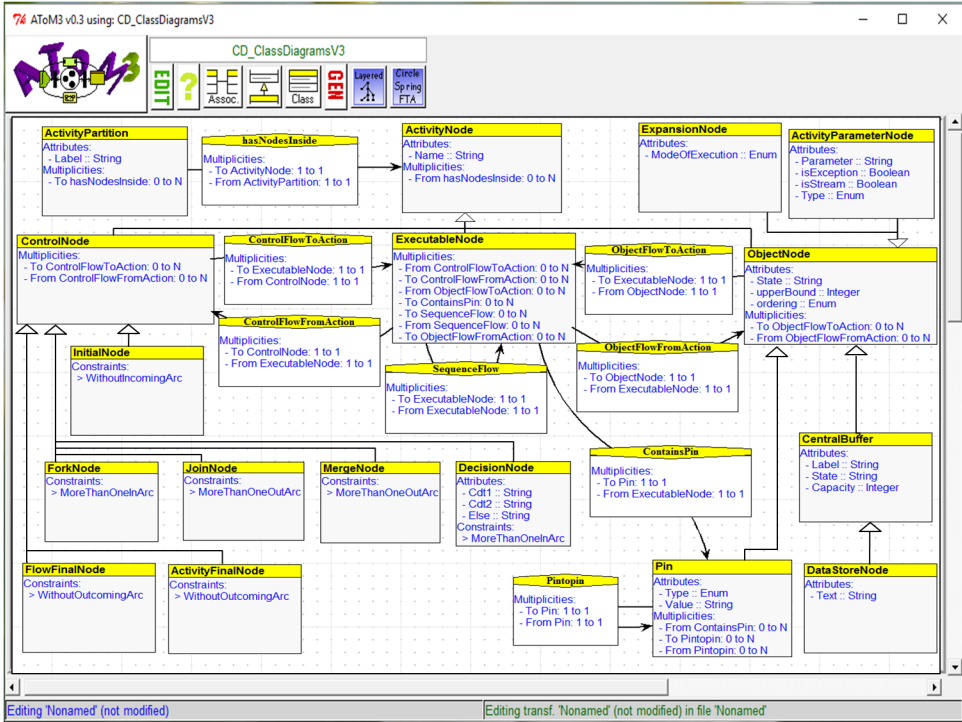


Figure 5. Proposed activity diagram meta-model

activity diagrams that conform to the above-presented meta-model in a graphical manner.

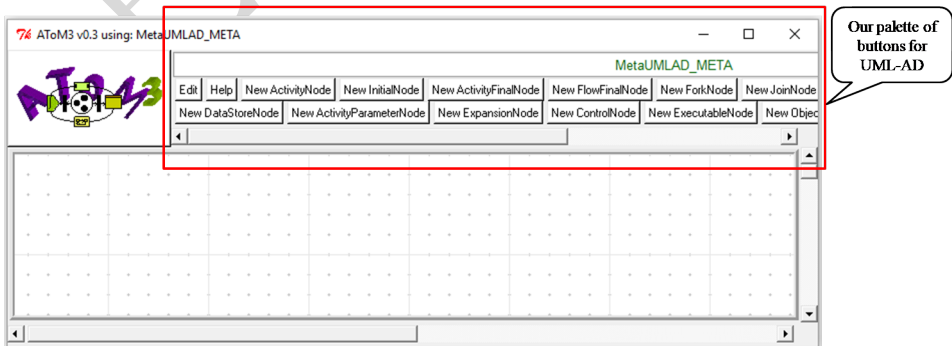


Figure 6. Generated tool for UML activity diagrams

After creating these two meta-models and generating their visual environments with AToM<sup>3</sup>, we propose a graph grammar, which contains a set of rules for accomplishing this transformation.

#### 4.4. Proposed Graph Grammar

Graph grammars have a mathematical foundation and are a generalization of Chomsky grammars for graphs [40]. One of the advantages of a graph grammar is that it facilitates the graphical manipulation of model transformations.

There are many definitions of a graph grammar; in [5], it was defined by a triplet –  $GG = (P, S, T)$ , where

$$\left\{ \begin{array}{l} P: \text{set of rules (our proposed graph grammar);} \\ S: \text{initial graph (BPM source model);} \\ T: \text{set of symbols (all elements of BPMN and UML-AD that are defined in both of our} \\ \quad \text{meta-models).} \end{array} \right.$$

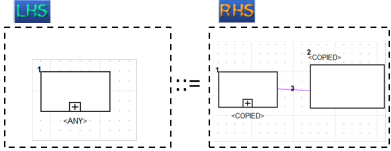
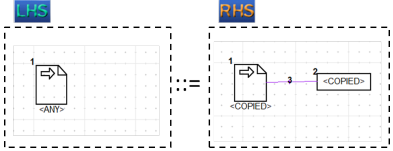
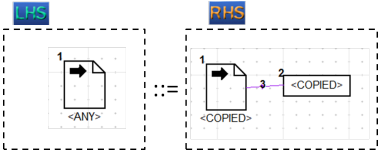
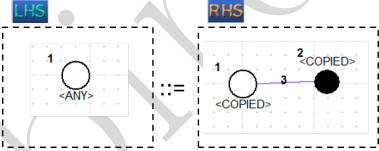
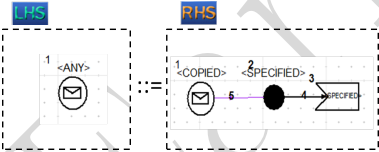
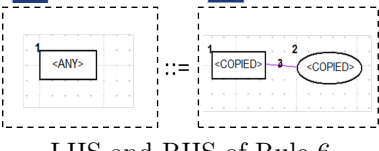
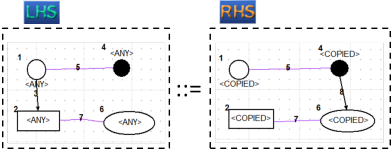
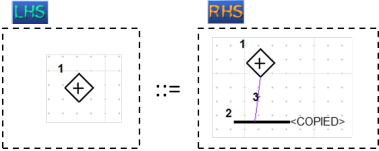
Each  $r$  rule ( $r \in P$ ) has graphs on the left-hand and right-hand sides (LHS and RHS). In the transformation process, the LHS rules are evaluated against an input graph that is called the host graph. If a matching is found between the LHS of a rule and a subgraph of the host graph, the rule can then be applied.

In this respect, we have proposed a graph grammar that enables us to transform the most frequently used elements of BPMN. The graph grammar includes a set of rules that cover flow objects (events, activities, and gateways) and data (data objects, data inputs, data outputs, etc.). These rules are applied in ascending order, where each rule has a priority (number). Giving a priority for each rule and coordinating all of the proposed rules is considered to be the main challenge when proposing a graph grammar.

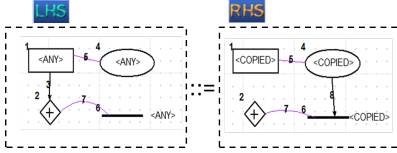
In addition, each rule in our graph grammar may have an initialization, condition, and action; these instructions are described in the Python language. The initialization, the condition and the action of Rule 1 will be presented later in Table 6. Table 2 contains our proposed graph grammar. For each rule, we have presented its LHS, RHS, and priority.

**Table 2**  
Our proposed graph grammar (BPMN2UML-AD)

---

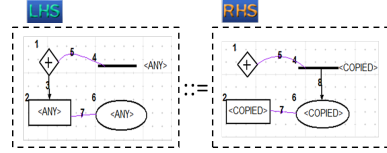
<p><i>Rule 1: Sub-Process2Sub-Activity (Priority 1):</i></p>  <p>LHS and RHS of Rule 1</p>	<p><i>Rule 2: DataInPut2ActParameterNode (Priority 2):</i></p>  <p>LHS and RHS of Rule 2</p>
<p><i>Rule 3: DataOutPut2ActParameterNode (Priority 3):</i></p>  <p>LHS and RHS of Rule 3</p>	<p><i>Rule 4: Start2InitialNodee (Priority 4):</i></p>  <p>LHS and RHS of Rule 4</p>
<p><i>Rule 5: StartMsg2Initial/AcceptCallAction (Priority 5):</i></p>  <p>LHS and RHS of Rule 5</p>	<p><i>Rule 6: Activity2Action (Priority 6):</i></p>  <p>LHS and RHS of Rule 6</p>
<p><i>Rule 7: Start2ActivityLink (Priority 7):</i></p>  <p>LHS and RHS of Rule 7</p>	<p><i>Rule 8: Parallel(And-Split)2ForkNode (Priority 8):</i></p>  <p>LHS and RHS of Rule 8</p>

**Rule 9: Act2ParallelLink (Priority 9):**



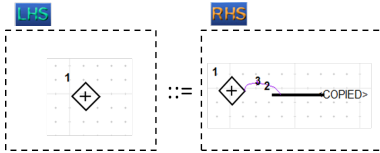
LHS and RHS of Rule 9

**Rule 10: Parallel2ActLink (Priority 10):**



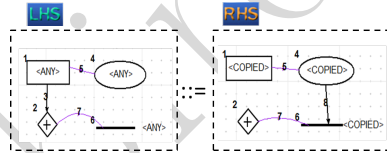
LHS and RHS of Rule 10

**Rule 11: Synchronization(And-Join)2JoinNode (Priority 11):**



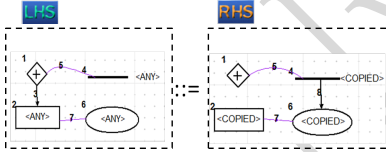
LHS and RHS of Rule 11

**Rule 12: Act2SynchLink (Priority 12):**



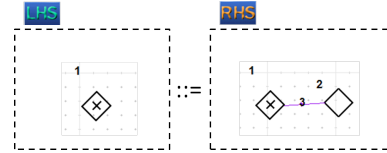
LHS and RHS of Rule 12

**Rule 13: Synch2ActLink (Priority 13):**



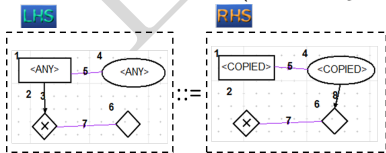
LHS and RHS of Rule 13

**Rule 14: Xor(OR-Split)2DecisionNode (Priority 14):**



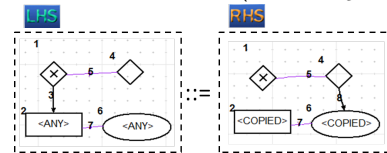
LHS and RHS of Rule 14

**Rule 15: Act2XorLink (Priority 15):**



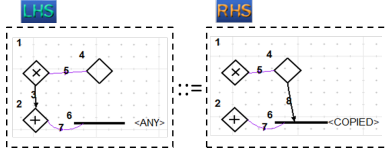
LHS and RHS of Rule 15

**Rule 16: Xor2ActLink (Priority 16):**



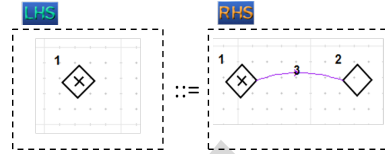
LHS and RHS of Rule 16

**Rule 17: Xor2SynchLink (Priority 17):**



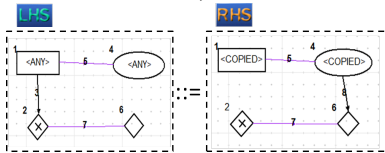
LHS and RHS of Rule 17

**Rule 18: SimpleMerge(OR-Join)2MergeNode (Priority 18):**



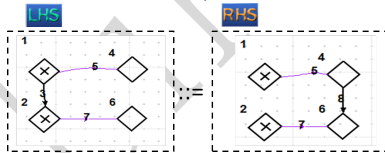
LHS and RHS of Rule 18

**Rule 19: Act2MergeLink (Priority 19):**



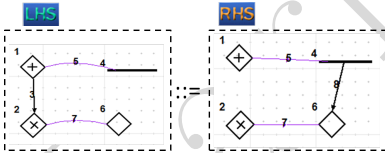
LHS and RHS of Rule 19

**Rule 20: Xor2MergeLink (Priority 20):**



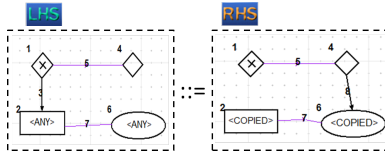
LHS and RHS of Rule 20

**Rule 21: Synch2MergeLink (Priority 21):**



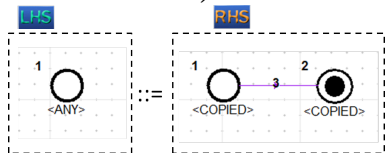
LHS and RHS of Rule 21

**Rule 22: Merge2ActLink (Priority 22):**



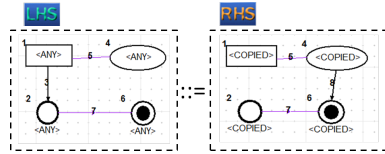
LHS and RHS of Rule 22

**Rule 23: End2FinalNode (Priority 23):**

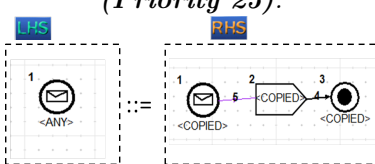
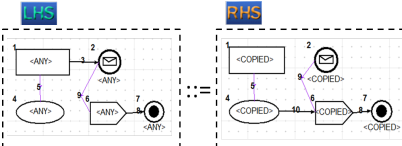
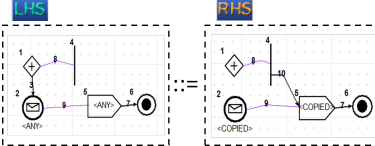
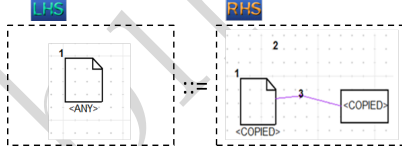
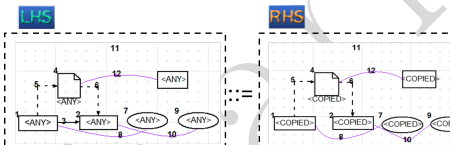
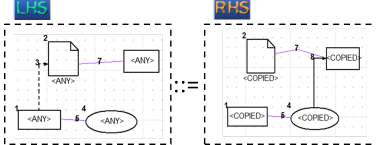
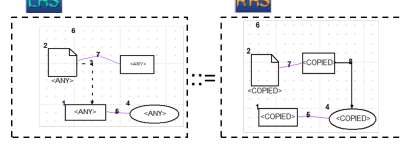
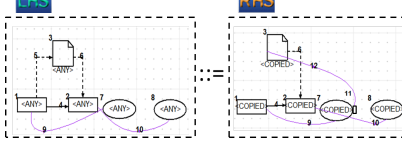


LHS and RHS of Rule 23

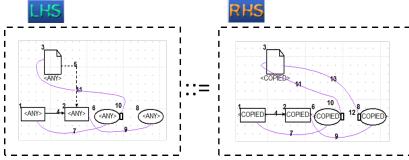
**Rule 24: Act2EndLink (Priority 24):**



LHS and RHS of Rule 24

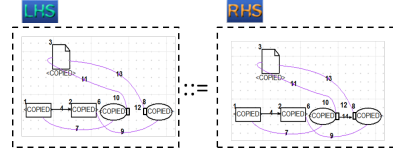
<p><i>Rule 25: EndMsg-gEvent2SendSignalAction&amp;FinalNode (Priority 25):</i></p>  <p>LHS and RHS of Rule 25</p>	<p><i>Rule 26: Act2EndMsgEventLink (Priority 26):</i></p>  <p>LHS and RHS of Rule 26</p>
<p><i>Rule 27: Synch2EndMsgEventLink (Priority 27):</i></p>  <p>LHS and RHS of Rule 27</p>	<p><i>Rule 28: DataObject2ObjectNode (Priority 28):</i></p>  <p>LHS and RHS of Rule 28</p>
<p><i>Rule 29: Delete Act2Act linked by DataObject (Priority 29):</i></p>  <p>LHS and RHS of Rule 29</p>	<p><i>Rule 30: Activity2DataObjectLink (Priority 30):</i></p>  <p>LHS and RHS of Rule 30</p>
<p><i>Rule 31: DataObject2ActivityLink (Priority 31):</i></p>  <p>LHS and RHS of Rule 31</p>	<p><i>Rule 32: DataObject2PinOut (Priority 32):</i></p>  <p>LHS and RHS of Rule 32</p>

*Rule 33: DataObject2PinIn (Priority 33):*



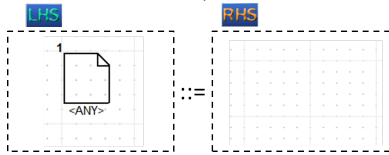
LHS and RHS of Rule 33

*Rule 34: PinOut2PinInLink (Priority 34):*



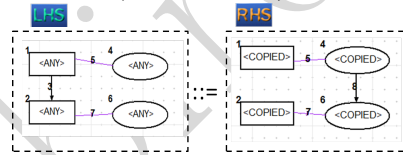
LHS and RHS of Rule 34

*Rule 35: DeleteDataObject (Priority 35):*



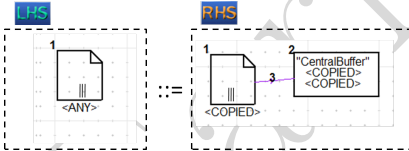
LHS and RHS of Rule 35

*Rule 36: Activity2ActivityLink (Priority 36):*



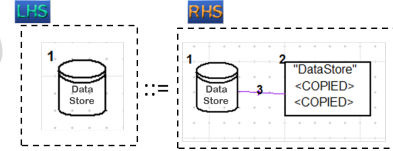
LHS and RHS of Rule 36

*Rule 37: CollectionDataObject2CentralBuffer (Priority 37):*



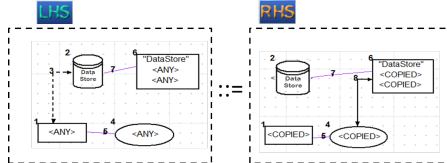
LHS and RHS of Rule 37

*Rule 38: DataStore2DataStoreNode (Priority 38):*



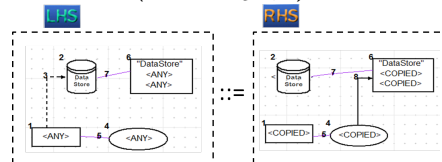
LHS and RHS of Rule 38

*Rule 39: Activity/DataStoreBidirectionalLink (Priority 39):*



LHS and RHS of Rule 39

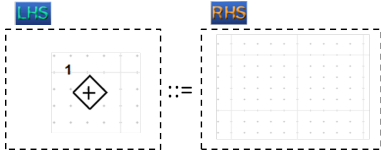
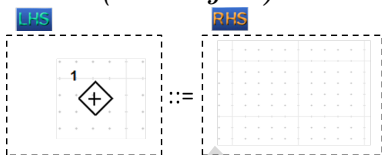
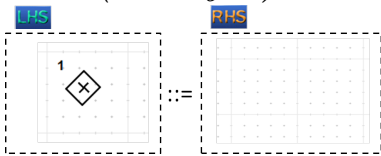
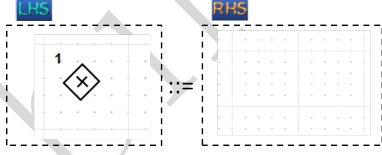
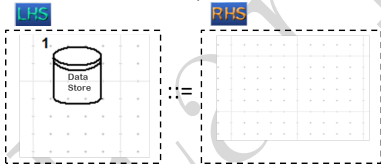
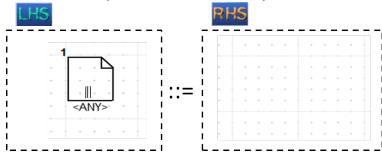
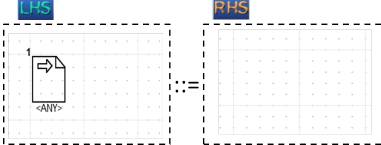
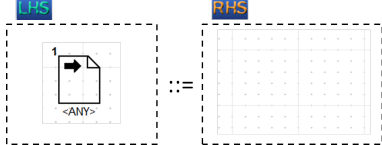
*Rule 40: Activity2DataStoreLink (Priority 40):*

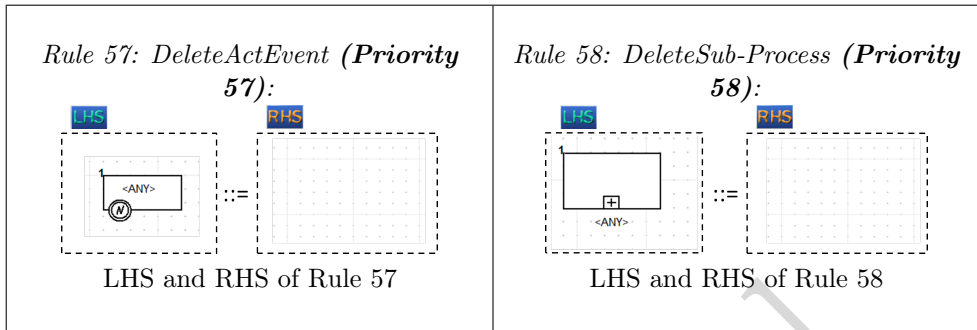


LHS and RHS of Rule 40

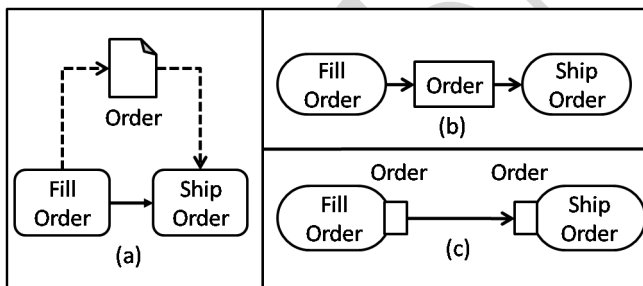


<p><i>Rule 41: DataStore2ActivityLink (Priority 41):</i></p> <p>LHS and RHS of Rule 41</p>	<p><i>Rule 42: DataInPut2ActivityLink (Priority 42):</i></p> <p>LHS and RHS of Rule 42</p>
<p><i>Rule 43: Activity2DataOutPutLink (Priority 43):</i></p> <p>LHS and RHS of Rule 43</p>	<p><i>Rule 44: DeleteStart (Priority 44):</i></p> <p>LHS and RHS of Rule 44</p>
<p><i>Rule 45: DeleteStartMsgEvent (Priority 45):</i></p> <p>LHS and RHS of Rule 45</p>	<p><i>Rule 46: DeleteActivity (Priority 46):</i></p> <p>LHS and RHS of Rule 46</p>
<p><i>Rule 47: DeleteEnd (Priority 47):</i></p> <p>LHS and RHS of Rule 47</p>	<p><i>Rule 48: DeleteEndMsgEvent (Priority 48):</i></p> <p>LHS and RHS of Rule 48</p>

<p><i>Rule 49: DeleteParallel(And-Split)</i> <b>(Priority 49):</b></p>  <p>LHS and RHS of Rule 49</p>	<p><i>Rule 50: DeleteSynchronization(And-Join)</i> <b>(Priority 50):</b></p>  <p>LHS and RHS of Rule 50</p>
<p><i>Rule 51: DeleteExclusiveChoice(OR-Split)</i> <b>(Priority 51):</b></p>  <p>LHS and RHS of Rule 51</p>	<p><i>Rule 52: DeleteSimpleMerge(OR-Join)</i> <b>(Priority 52):</b></p>  <p>LHS and RHS of Rule 52</p>
<p><i>Rule 53: DeleteDataStore</i> <b>(Priority 53):</b></p>  <p>LHS and RHS of Rule 53</p>	<p><i>Rule 54: DeleteCollectionDataObject</i> <b>(Priority 54):</b></p>  <p>LHS and RHS of Rule 54</p>
<p><i>Rule 55: DeleteDataInPut</i> <b>(Priority 55):</b></p>  <p>LHS and RHS of Rule 55</p>	<p><i>Rule 56: DeleteDataOutPut</i> <b>(Priority 56):</b></p>  <p>LHS and RHS of Rule 56</p>



The main challenge that is encountered in the part of a data object transformation is the existence of two representations of a single piece of data between the activities in UML-AD. The result of the transformation of the example that is shown in Part (a) of Figure 7 is provided in Parts (b) and (c) of Figure 7.



**Figure 7.** (a) data object in BPMN; (b) object node; and (c) data input and output pin in UML-AD

In our graph grammar, we have dealt with all of the element representations in UML-AD such as the object node, data input, and output pin. So, we have allowed the user to choose the desired notation in the target model.

- **First case:** if the user desires to use a data input pin and a data output pin notations (Figure 7[c]), he/she can click the “DataPin” button. As a result, Rules 29 and 30 are automatically deactivated.
- **Second case:** if the user desires to use an object node between activities (Figure 7[b]), he/she must click the “ObjectNode” button. As a result, Rules 31 through 35 are automatically deactivated.

**Discussion:** ‘Expressive power’ refers to the language ability to present different kinds of process constructs, patterns, and situations that appear in business processes. This aspect should be considered when analyzing the representation power of BPMN and UML AD (in particular, the complexity of the graphical symbols that are used to represent the real business processes of an organization). In many cases, BPMN

and UML-AD use similar symbols to describe business processes; however, there are elements of business processes that can be modeled in BPMN that use only one symbol, where their representations in UML-AD require groups of symbols.

### 5. Case Studies

The goal of this section is to illustrate the transformation process and the execution of our proposed graph grammar that transforms source business process models to target UML-AD models.

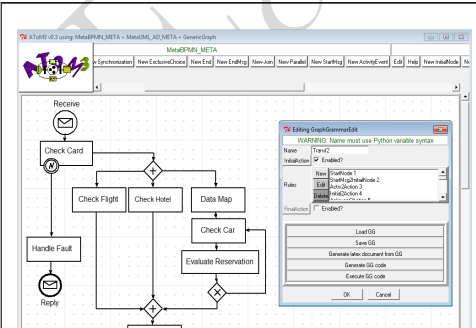
In order to present and test the proposed rules, we chose three examples that contained most of the elements that are studied in our contribution. The first example contained only flow object elements, while the second example included certain data elements such as *Data Inputs*, *Data Outputs*, and *Data Store*. The third model was comprised other data elements like *Message Start Event*, *Message End Event*, and *Data Object*.

To test our transformation, we followed the three well-known levels of testing: unit testing, integration testing, and system testing. In addition to the testing, we performed the verification of the transformation itself in the following section.

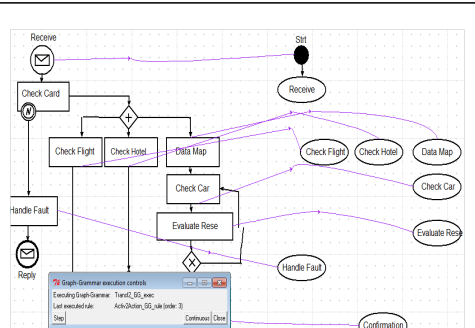
#### 5.1. First case study

We focused on the transformation of flow object elements (events, activities, and gateways) and their corresponding elements in the control part of UML-AD. Table 3 shows some steps of the transformation of the travel-booking process to an activity diagram. This was the first version of our tool that contained only control flow (in the first version of our tool, we proposed only 44 rules).

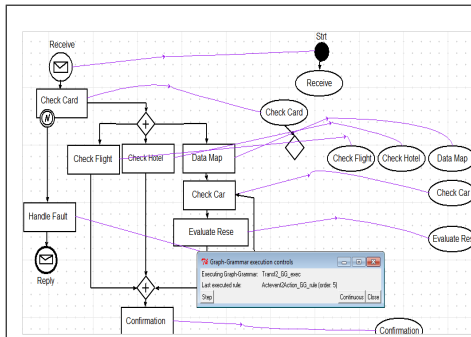
**Table 3**  
Some steps of transformation of travel booking process to UML-AD



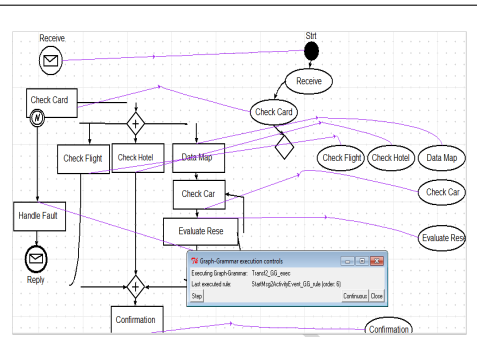
Invoking transformation rule



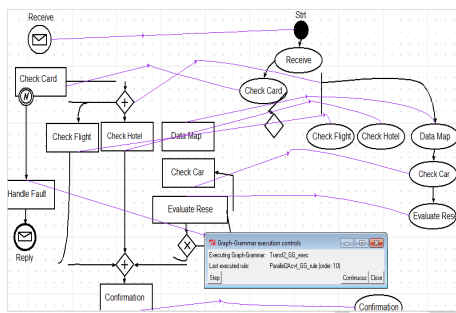
After execution of Rule 3 seven times



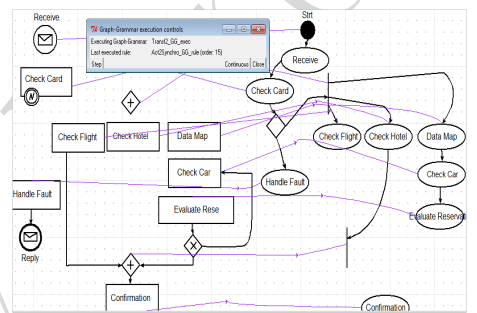
After execution of Rule 5



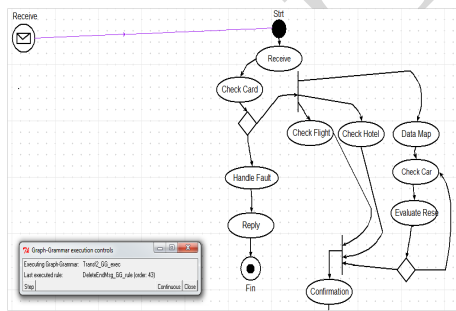
After execution of Rule 6



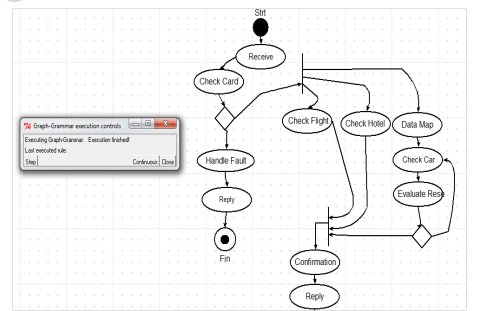
After execution of Rule 10



After execution of Rule 15



After execution of Rule 43 two times



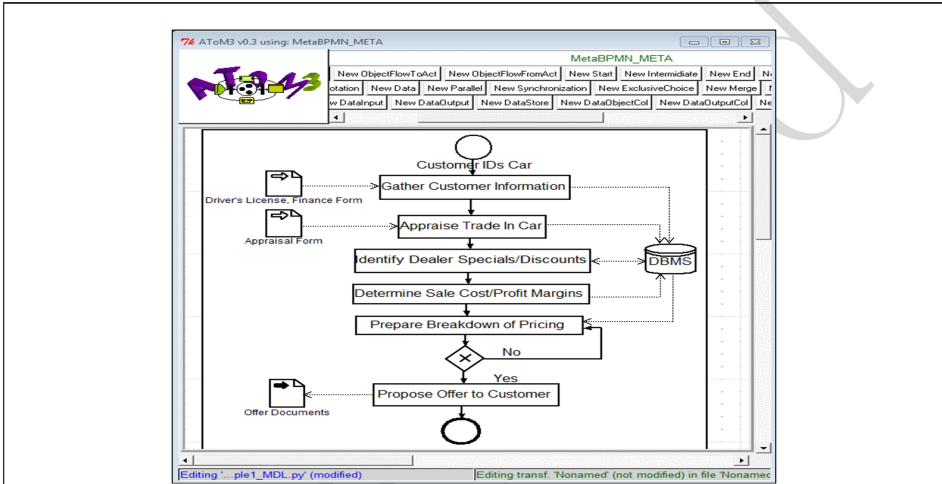
After execution of last step

In the following case studies, we focused on the transformation of data elements (data object, data input, data output, database, etc.) and their corresponding elements in a UML activity diagram. After enriching and enhancing our visual-modeling environment and the graph grammar that was proposed in the first case study, we have illustrated the execution of the added transformation rules.

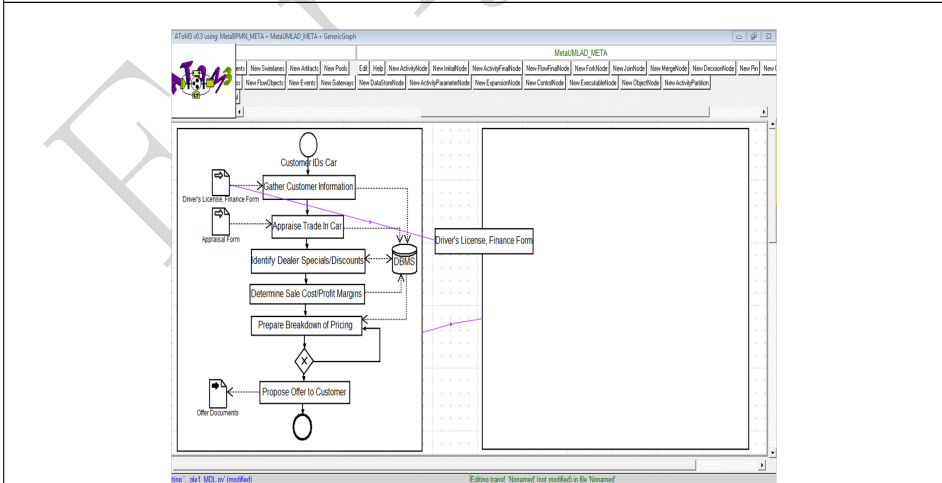
### 5.2. Second case study

We chose a simple example of a so-called *create-offer process* ; this was a sub-process of *the car-purchasing process* [20]. In Table 4, we present the transformation of the preceding process to a corresponding UML activity diagram by our tool step by step. We only selected some figures of this transformation, as it would occupy a large amount of space.

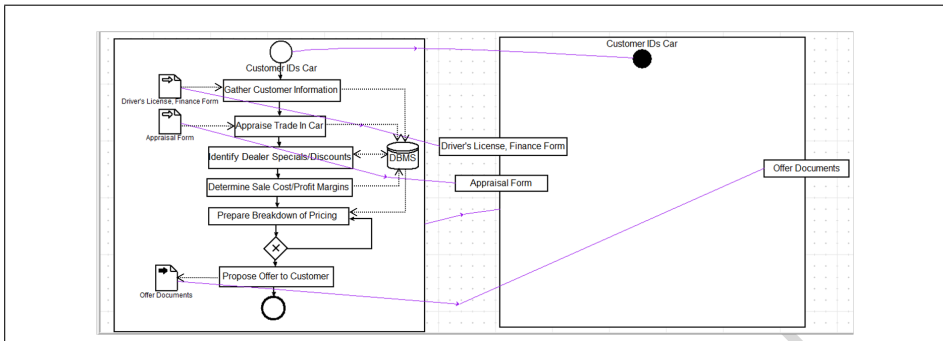
**Table 4**  
Some steps of transformation of Create Offer process to UML-AD



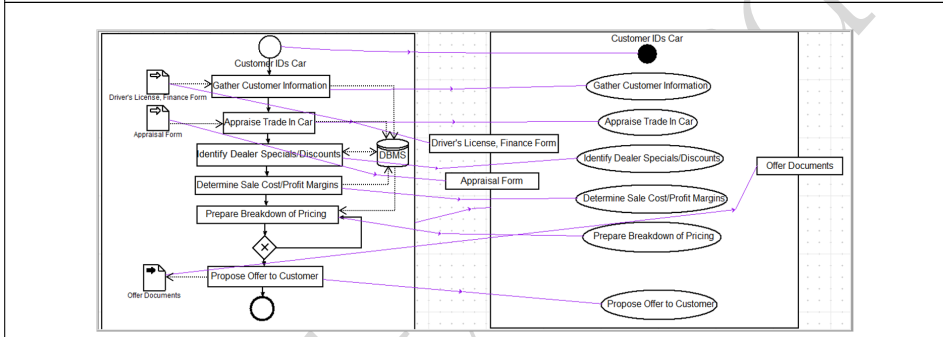
Create Offer process in BPMN created by our proposed tool



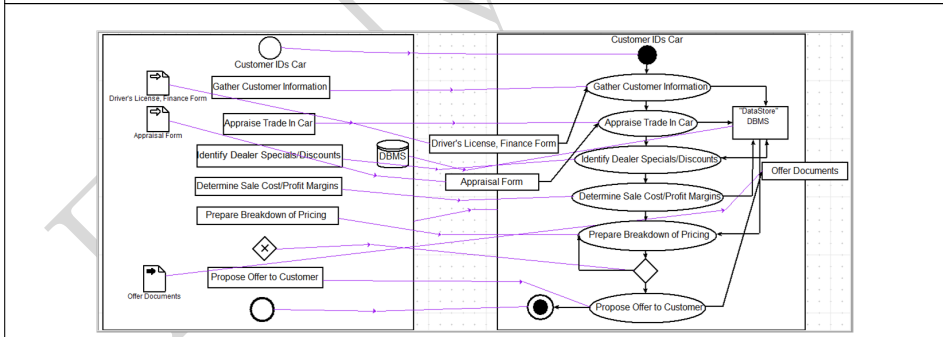
After execution of Rule 2 one time



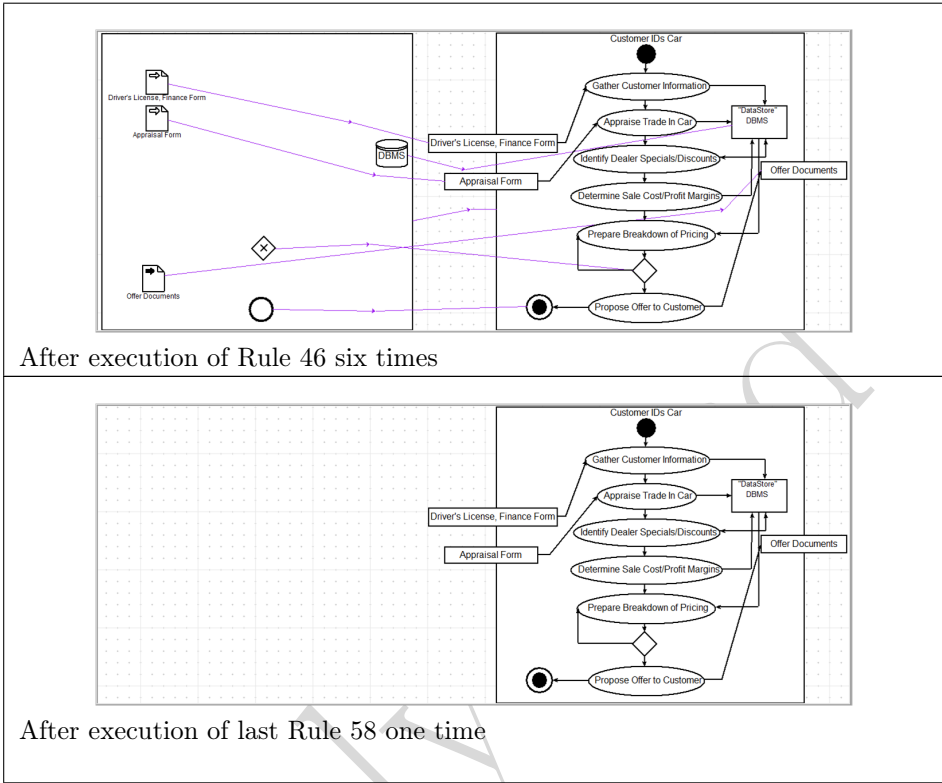
After execution of Rule 4 one time



After execution of Rule 6 six times



After execution of Rule 43 one time



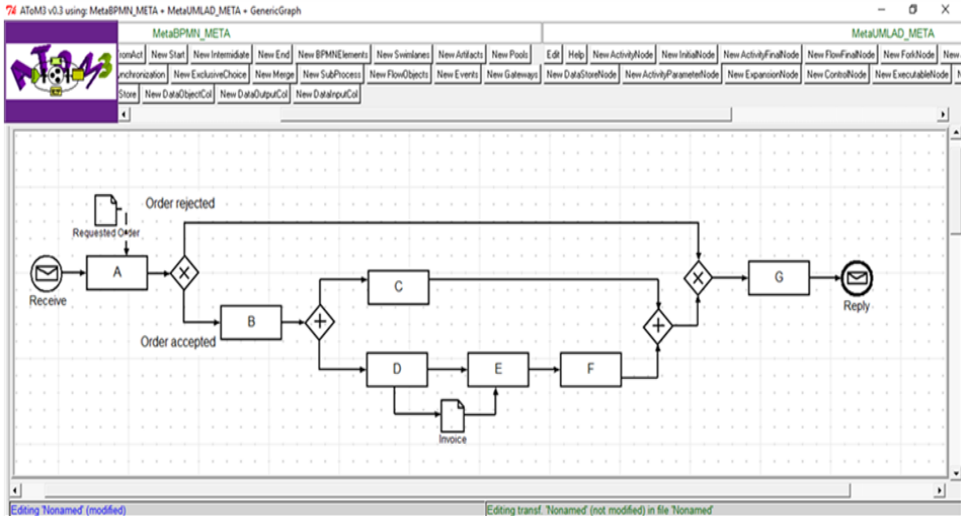
In this case study, we have illustrated the transformation of certain elements such as data inputs and data outputs to an activity parameter node in UML-AD and have specified the name as well as the type of data (input or output). For this reason, Rule 2 was executed two times for the data input (Driver’s License, Finance Form, and Appraisal Form). Thereafter, Rule 3 was executed one time for the data output (offer documents). Moreover, the inputs and outputs served as place holders for the data requirements that indicated important information. We also presented the transformation of a data store and its name in this example.

### 5.3. Third case study

We chose a *purchase-order process*; we selected this process because it contains some important elements that have not been studied previously (such as Message Start Event, Message End Event, and Data Object). This process was composed of several activities. We used abbreviations in order to reduce our source model during the transformation: A – Receive Order; B – Fill Order; C – Ship Order; D – Send Invoice; E – Make Payment; F – Accept Payment; and G – Close Order. In this process model, the first activity was to receive a requested order. If the order was accepted and all of



the required information was filled out, the payment was accepted, and the order was shipped. Figure 8 illustrates the purchase-order-process example that was created by our tool.

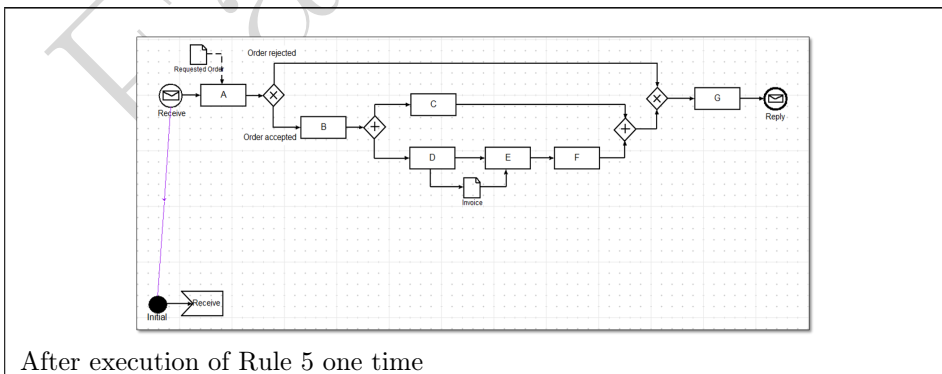


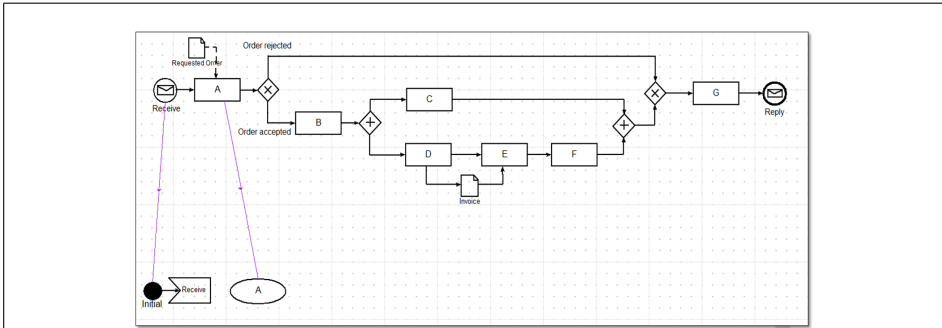
**Figure 8.** Purchase-order process example

In Table 5, we present the transformation of this process to a corresponding UML activity diagram by our tool step by step. We have selected some figures of this transformation.

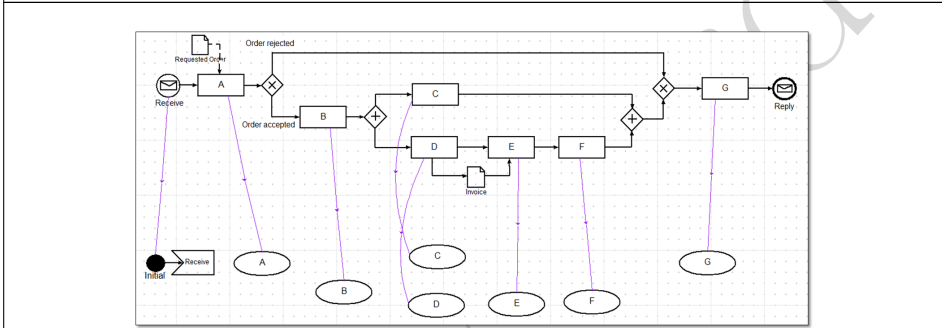
**Table 5**

Some steps of transformation of purchase-order process to UML-AD

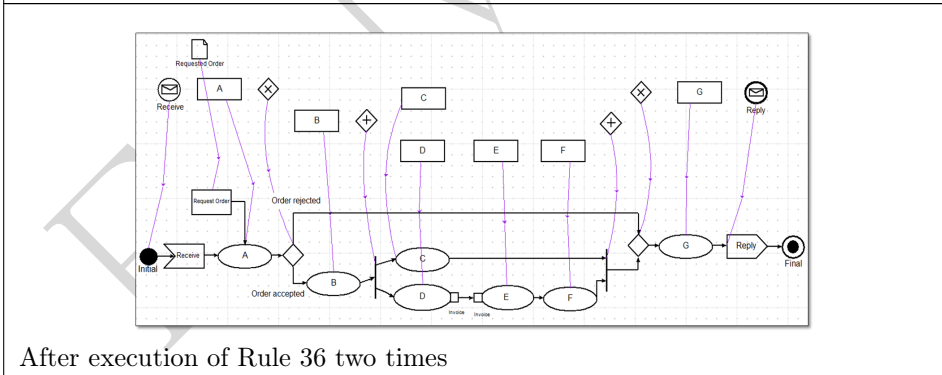




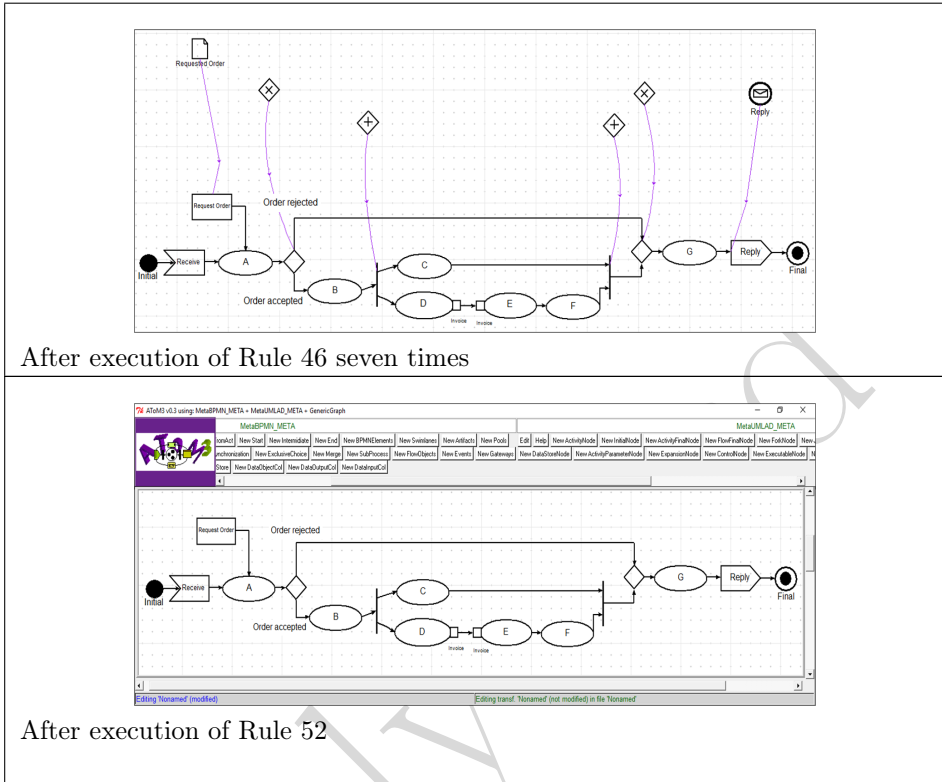
After execution of Rule 6 one time



After execution of Rule 6 seven times



After execution of Rule 36 two times



## 6. Verification and Testing of Proposed Approach

To prove the transformation approaches, there are several methods: a theoretical case study, a practical case, or an automatic tool that ensures the model's transformation. In our approach, we have proven our rules by a developed tool that was tested on several cases studies.

Recently, several works have focused on the verification of model transformation ([1, 3, 4, 28, 32]). Actually, model transformations have a variety of properties that are needed to ensure their correctness, such as termination, confluence, syntactic correctness, and others [31]. Termination and confluence properties are important requirements for practical applications of model transformations; they guarantee that a model transformation always terminates and produces a unique result [28].

In order to check our proposed approach, we have discussed and established the verification of the important properties.

## 6.1. Termination

The termination property refers to Turing’s halting problem; it guarantees the existence of target model(s) or must ensure that a model transformation will end.

In our transformation approach, we used a global variable named “*Visited*” in the AToM<sup>3</sup> tool. This is a Boolean variable; its initial value is false (*Visited* = 0). The use of this variable allows us to avoid infinite loops during the transformation (the execution of the same rule on the same element).

For more detail, each rule in the grammar may feature initialization, condition, and action. The following instructions (described in the Python language) give the initialization, condition, and action of Rule 1.

**Table 6**  
Basic instructions used in AToM<sup>3</sup>

Initialization	for node in graph.listNodes[‘SubProcess’]:node.Visited = 0
Condition	node = self.getMatched(graphID,self.LHS.nodeWithLabel(1)) return node.Visited == 0
Action	node = self.getMatched(graphID,self.LHS.nodeWithLabel(1)) node.Visited = 1

After the execution of each rule of the source model, all of the visited elements change their values to “true” (*Visited* = 1 – see Table 6). At the end of the transformation, no rule can be executed.

The use of the “*Visited*” variable ensures the visit and transformation of all of the elements of the source model. In addition, our rules ensure the transformation of all elements (studied elements) and their links to the source model. At the end of the transformation process of the model source, all of the elements have been deleted (and only the target model remains).

Consequently, we have deduced that the termination property is verified.

## 6.2. Confluence (Determinism)

The determinism property refers to the notion of confluence; it ensures that the transformations always produce the same result (target model).

In our transformation, we have proposed a graph grammar that contains 58 rules. Each rule has a priority (see Table 2); this priority is invariant and very important in the transformation process, as these rules will be applied in ascending order until no more rules are applied. This is one of the reasons for using a graph grammar in AToM<sup>3</sup>.

The graph grammar in AToM<sup>3</sup> consists of an initial action, a set of rules, and a final action:

- The initial action specifies the actions to be executed before the rules.

- The rules are ranked according to a priority in order to guide the choice of the rule to apply. For any iteration, all of the rules are tested in the ascending order of their priorities. Each rule may also have additional conditions of application and the actions to perform (for example, for Rule 1 – see Table 6).
- The final action specifies the actions to be carried after the application of the rules.

Each time we execute the transformation of the source model, the transformation follows the same predefined rule order (priority); this ensures repeating the same steps and generating the same target model. Hence, the use of priorities ensures the confluence property in our proposed graph transformation.

### 6.3. Syntactic Correctness

For the syntactic correctness of the process of transformation, we were inspired by the definitions and propositions that were proposed in [28]. This property can be guaranteed by the theoretical foundation of graph rewriting systems. These systems try to iteratively apply rules that are defined in the graph grammar. These rules specify the visual correspondence between the elements in the source and target models at the meta-model level. Thus, the target models are syntactically correct; all of the new elements that are created by the RHS (right-hand sides) of the rules conform to the target meta-model (the syntactic correctness property is verified).

## 7. Conclusion

Model transformation is one of the most important advantages of the MDE approach. This allows for exchanges between different communities and can be used to transform the models of one community to equivalent models that can be used by the other one. In this paper, we have proposed a transformation approach for creating a bridge between the BPMN and UML-AD communities. To this end, we have proposed an approach and a visual-modeling tool that are based on graph transformation, and we have used the AToM<sup>3</sup> tool. This approach allows for the transformation of business process models to UML activity diagrams; it is based on the transformation of flow objects and data elements (data object and data flow).

We used UML class diagram formalism as a meta-formalism for the two proposed meta-models: the first one was for BPMN (source model), and the second for UML-AD (target model). Then, we proposed a graph grammar that contained 58 rules using the AToM<sup>3</sup> tool and illustrated this transformation with three examples.

To make this practical, we plan to develop exporters and importers for loading actual BPMN models that are developed by BPMN editors, transform them, and store their equivalent UML-AD versions in a syntax that is accepted by UML tools.

We also plan to transform the rest of the elements of BPMN to UML-AD and enrich the target model semantically by using a UML profile approach. In addition, we will check our proposed graph grammar by using one of the techniques for verifying

model transformation (such as the GROOVE tool – GRaphs for Object-Oriented VERification) [19].

## References

- [1] Ab Rahim L., Whittle J.: A survey of approaches for verifying model transformations, *Software & Systems Modeling*, vol. 14(2), pp. 1003–1028, 2015. doi: 10.1007/s10270-013-0358-0.
- [2] Amjad A., Haq S.U., Abbas M., Arif M.H.: UML Profile for Business Process Modeling Notation. In: *2021 International Bhurban Conference on Applied Sciences and Technologies (IBCAST)*, pp. 389–394, IEEE, 2021.
- [3] Amrani M., Combemale B., Lúcio L., Selim G.M., Dingel J., Le Traon Y., Vangheluwe H., Cordy J.R.: Formal verification techniques for model transformations: A tridimensional classification, *Journal of Object Technology*, vol. 14(3), pp. 1–43, 2015. doi: 10.5381/jot.2015.14.1.a3.
- [4] Amrani M., Syriani E., Wimmer M., Bill R., Gogolla M., Hermann F., Lano K.: Report on the Third Workshop on Verification of Model Transformations (VOLT 2014). In: *VOLT@ STAF*, pp. 1–9, 2014.
- [5] Andries M., Engels G., Habel A., Hoffmann B., Kreowski H.J., Kuske S., Plump D., Schürr A., Taentzer G.: Graph transformation for specification and programming, *Science of Computer programming*, vol. 34(1), pp. 1–54, 1999.
- [6] Bao N.Q.: A proposal for a method to translate BPMN model into UML activity diagram. In: *13th International Conference on Business Information Systems*, 2010.
- [7] Birkmeier D.Q., Klöckner S., Overhage S.: An Empirical Comparison of the Usability of BPMN and UML Activity Diagrams for Business Users. In: P.M. Alexander, M. Turpin, J.P. van Deventer (Eds.), *18th European Conference on Information Systems, ECIS 2010, Pretoria, South Africa, June 7–9, 2010*, 2010. <http://aisel.aisnet.org/ecis2010/51>.
- [8] Bouzidi A., Haddar N., Abdallah M.B., Haddar K.: Deriving use case models from BPMN models. In: *2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA)*, pp. 238–243, IEEE, 2017.
- [9] Cibran M.A.: Translating BPMN models into UML activities. In: *International Conference on Business Process Management*, pp. 236–247, Springer, 2008. doi: 10.1007/978-3-642-00328-8\_23.
- [10] De Lara J.: A Tool for Multi-formalism and Meta-Modeling, Home page, 2003. <http://http://atom3.cs.mcgill.ca/>.
- [11] De Lara J., Guerra E.: Towards the uniform manipulation of visual and textual languages in AToM3. In: *Proceedings of III Jornadas de Programación y Lenguajes. Universidad de Alicante, Alicante, Noviembre 12-14, 2003*, 2003.

- [12] De Lara J., Vangheluwe H.: AToM3: A Tool for Multi-Formalism Modeling and Meta-Modeling, *LNCS 2306, Presented at Fundamental Approaches to Software Engineering-FASE*, vol. 2, 2002.
- [13] De Lara J., Vangheluwe H.: Using atom3 as a meta-case tool. In: *ICEIS*, vol. 2, 2002.
- [14] Dörr H.: *Efficient graph rewriting and its implementation*, vol. 922, Springer Science & Business Media, 1995.
- [15] Elmansouri R., Hamrouche H., Chaoui A.: From uml activity diagrams to csp expressions: A graph transformation approach using atom 3 tool, *IJCSI International Journal of Computer Science Issues*, vol. 8(2), pp. 368—374, 2011.
- [16] Elmansouri R., Meghzili S., Chaoui A.: A UML 2.0 Activity Diagrams/CSP Integrated Approach for Modeling and Verification of Software Systems, *Computer Science*, vol. 22(2), 2021.
- [17] Foundation P.S.: Python (python language) Home page. <http://www.python.org>.
- [18] Geambaşu C.V.: BPMN vs. UML activity diagram for business process modeling, *Accounting and Management Information Systems*, vol. 11(4), pp. 934–945, 2012.
- [19] GROOVE: GRaphs for Object-Oriented Verification, Home page, 2015. <http://groove.cs.utwente.nl/>.
- [20] Guidebook B.A.: The car purchasing process. [https://en.wikibooks.org/wiki/Business\\_Analysis\\_Guidebook/Requirement\\_Gathering\\_Tools](https://en.wikibooks.org/wiki/Business_Analysis_Guidebook/Requirement_Gathering_Tools).
- [21] H. E. E., M. P.: Business Modeling with UML: Business Patterns at Work, 2000.
- [22] Hettab A., Kerkouche E., Chaoui A.: A graph transformation approach for automatic test cases generation from UML activity diagrams. In: *Proceedings of the Eighth International C\* Conference on Computer Science & Software Engineering*, pp. 88–97, 2015.
- [23] Kerkouche E., Chaoui A., Bourennane E.B., Labbani O.: A UML and Colored Petri Nets Integrated Modeling and Analysis Approach using Graph Transformation, *Journal of Object Technology*, vol. 9(4), pp. 25—43, 2010. doi: 10.5381/jot.2010.9.4.a2.
- [24] Kerkouche E., Elmansouri R., Chaoui A., Khalfaoui K.: An Automatic approach to verify business process models using INA petri nets analyzer, *International Journal of Computer and Information Technology (ISSN)*, vol. 3(4), pp. 706–711, 2014.
- [25] Kerkouche E., Khalfaoui K., Chaoui A.: A rewriting logic-based semantics and analysis of UML activity diagrams: a graph transformation approach, *International Journal of Computer Aided Engineering and Technology*, vol. 12(2), pp. 237–262, 2020.
- [26] Kerkouche E., Khalfaoui K., Chaoui A., Aldahoud A.: UML Activity Diagrams and Maude Integrated Modeling and Analysis Approach Using Graph Transformation. In: *7th International Conference on Information Technology*, pp. 515–521, 2015. doi: 10.15849/icit.2015.0093.

- [27] Korherr B., List B.: A UML 2 profile for event driven process chains. In: *Research and Practical Issues of Enterprise Information Systems*, pp. 161–172, Springer, 2006.
- [28] Küster J.M.: Definition and validation of model transformations, *Software & Systems Modeling*, vol. 5(3), pp. 233–259, 2006. doi: 10.1007/s10270-006-0018-8.
- [29] List B., Korherr B.: A uml 2 profile for business process modelling. In: *International Conference on Conceptual Modeling*, pp. 85–96, Springer, 2005.
- [30] Macek O., Richta K.: The BPM to UML activity diagram transformation using XSLT. In: *Dateso*, vol. 9, pp. 119–129, 2009.
- [31] Meghzili S., Chaoui A., Strecker M., Kerkouche E.: On the Verification of UML State Machine Diagrams to Colored Petri Nets Transformation Using Isabelle/HOL. In: *International Conference on Information Reuse and Integration (IRI)*, pp. 419–426, IEEE, 2017. doi: 10.1109/IRI.2017.63.
- [32] Meghzili S., Chaoui A., Strecker M., Kerkouche E.: Verification of Model Transformations Using Isabelle/HOL and Scala, *Information Systems Frontiers*, vol. 21(1), pp. 45–65, 2019.
- [33] Meghzili S., Chaoui A., Strecker M., Kerkouche E.: An approach for the transformation and verification of BPMN models to colored petri nets models, *International Journal of Software Innovation (IJSI)*, vol. 8(1), pp. 17–49, 2020.
- [34] OMG: Business Process Model and Notation, 2013. <http://www.omg.org/spec/BPMN/2.0.2/>.
- [35] OMG: Unified Modeling Language, 2015. <http://www.omg.org/spec/UML/2.5/>.
- [36] Rahmoune Y., Chaoui A., Kerkouche E.: A framework for modeling and analysis UML Activity diagram using graph transformation, *Procedia Computer Science*, vol. 56, pp. 612–617, 2015. doi: 10.1016/j.procs.2015.07.261.
- [37] Recker J., Zur Muehlen M., Siau K., Erickson J., Indulska M.: Measuring method complexity: UML versus BPMN. In: *Proceedings of the Fifteenth Americas Conference on Information Systems*, pp. 1–9, Association for Information Systems, 2009.
- [38] Rodríguez A., Fernández-Medina E., Piattini M.: Analysis-level classes from secure business processes through model transformations. In: *International Conference on Trust, Privacy and Security in Digital Business*, pp. 104–114, Springer, 2007. doi: 10.1007/978-3-540-74409-2\_13.
- [39] Rodríguez A., Fernández-Medina E., Piattini M.: CIM to PIM transformation: A reality. In: *Research and Practical Issues of Enterprise Information Systems II*, pp. 1239–1249, Springer, 2008. doi: 10.1007/978-0-387-76312-5\_50.
- [40] Rozenberg G.: *Handbook of graph grammars and computing by graph transformation*, vol. 1, World Scientific, 1997.
- [41] Russell N., van der Aalst W.M., Ter Hofstede A.H., Wohed P.: On the suitability of UML 2.0 activity diagrams for business process modelling. In: *Proceedings of the 3rd Asia-Pacific conference on Conceptual modelling*, vol. 53, pp. 95–104, Australian Computer Society, Inc., 2006.



- [42] Suchenia A., Lopata P., Wiśniewski P., Stachura-Terlecka B.: Towards UML representation for BPMN and DMN models. In: *MATEC Web of Conferences*, vol. 252, p. 02007, EDP Sciences, 2019.
- [43] Swain R.K., Panthi V., Behera P.K.: Generation of test cases using activity diagram, *International journal of computer science and informatics*, vol. 3(2), pp. 1–10, 2013.
- [44] White S.A.: Process modeling notations and workflow patterns, *Workflow handbook*, vol. 2004, pp. 265–294, 2004.
- [45] Wohed P., van der Aalst W.M., Dumas M., ter Hofstede A.H., Russell N.: On the suitability of BPMN for business process modelling. In: *International conference on business process management*, pp. 161–176, Springer, 2006. doi: 10.1007/11841760\_12.

## Affiliations

### Yasmina Rahmoune

Department of Computer Science, Assia Djebar Teacher Training School of Constantine, Constantine, Algeria, MISC Laboratory, Abdelhamid Mehri Constantine2 University, Constantine, Algeria, yasmina.rahmoune@univ-constantine2.dz

### Allaoua Chaoui

University Constantine 2-Abdelhamid Mehri, MISC Laboratory, Department of Computer Science and Its Applications, Faculty of Ntic, Constantine, Algeria, allaoua.chaoui@univ-constantine2.dz

**Received:** 28.06.2021

**Revised:** 14.10.2021

**Accepted:** 04.04.2022