TOWARDS A SEMANTIC FOR UML ACTIVITY DIAGRAM BASED ON INSTITUTION THEORY FOR IT'S TRANSFORMATION TO EVENT-B MODEL

Aymen Achouri

Faculty of science of Tunis - University of Tunis
aymen_achouri@hotmail.fr

ABSTRACT

In this article, we define an approach for model transformation. We use the example of UML Activity Diagram (UML AD) and Event-B as a source and a target formalism. Before doing the transformation, a formal semantic is given to the source formalism. We use the institution theory to define the intended semantic. With this theory, we gain a algebraic specification for this formalism. Thus, the source formalism will be defined in its own natural semantic meaning without any intermediate semantic. Model transformation will be performed by a set of transformation schema which preserve the semantic expressed in the source model during the transformation process. The generated model expressed in Event-B language will be used for the formal verification of the source model. As a result, some model expressed in a precise formalism, the verification of this model can be seen as the verification of the Event-B model semantically equivalent to the source model. Then, in the present work we combine the institution theory, Event-Bmethod and graph grammar to develop an approach supporting the specification, the transformation and the verification of UML AD.

KEYWORDS

Model transformation; Institution; UML Activity Diagram Institution; Transformation rules; Formal semantic

1. INTRODUCTION

Model Transformation is a critical process in software construction and development. As increasingly larger software systems are being developed, there is a tendency to have solid and effective tools to automatize the software development. The specification of a software can be formal and (or) graphical. As for graphical formalisms, we can mention as example the UML models, UML class diagram, UML activity diagram and interaction diagram as examples. For the formal ones, logic are increasingly used due to their mathematical basis. For example, Petri-net is used as a graphical and a formal specification formalism. Logic is the language of formalmethods like theoremproving and model checking. To facilitate and to make jointly graphical and formal language, there is a massive need to make generic techniques for the transformation of graphical models to formal notations. Also the use of the logic is difficult for non-familiar with logical concepts and specification. As a result, it is necessary to provide the possibility to make specification in a modeling level. The result is to use transformation engine so that we can to transform the specification (defined as a model conform to a specific formalism).

Stakeholders can begin with a graphical model (possibly with many system views), and with an automatic and correct transformation they cant to produce a specification in a formal logic. In the context of logic, institution theory is emerged as a cadre to study the multitude of logics and different relationships between it.

In our previous work [13], we used graph grammar to define an automatic transformation between UML AD and Event-B. Thanks to the notion of graph grammar, the automation aspect is given to the transformation. As for the semantic equivalence between source and target model is not proved. The reason is the absence of formal semantic for the source and the target formalism. As a first step, we use the institution theory in this paper to make the semantic needed for only the source formalism. In this paper, we take the example of UML Activity Diagram as a source formalism and Event-B as a target one.

The first contribution aims to give institutional presentation of UML AD. We can view this institutional presentation as a formal semantics of UML AD. This algebraic presentation of the source formalism will be a meta-level to study possible transformation to Event-B models. Also, it can be used to study some proprieties like model amalgamation, pushout and pullback of this formalism Those notions are important for the specification, composition specifications, and heterogeneous specification approaches. In addition, we define a set of rules that automatizes the generation of Event-B model from the defined semantic of UML AD. With the proposed solution, UML AD models are formalized in Institutions which make easier the translation into Event-B also the proof of correctness ans completeness of translation process.

The paper is organized as follows: in section 2 we present model transformation in general. Then in section 3, we define institution and we recall some important definitions. Section 4 shows how to prove that UML AD establish an institution. Section 5 defines the language of Event B, the target formalism. Section 6 defines a set of rules for the transformation of UML AD to Event-B. Finally, the last section concludes our work.

2. RELATED WORKS

In the literature, the institution theory is well used and studied. We have three categories of works based on the institution theory.

The first category is interested in the use of institution theory and it’s known concepts in the development of an heterogeneous specification approaches. This category consists of global theoretic logical based works. We begin with the approach of the heterogeneous specification in the tool cafeOBJ. This approach is based on a cube on eight logic and twelve projections (defined as a set of institution morphism and institution comorphism)[2]. This approach is based on the semantic based on Diaconescu’s notion of Grothendieck institution [1]. Another approach is developed in the work of Till Mossakowski [15]. The heterogeneous logical environment developed by the author is formed by a number of logical systems formalized as institutions linked with the concepts of institution morphism and comorphism.

The second category of works are specific to the use of institution theory in the specification of graphical formalism such UML diagrams. In this category, we mention the work present in [10] [17] [18] [19]. The approach defined by Cengarle et al. aims to define a semantic for UML class diagram, UML interactions diagram and OCL. Each diagram is described in it's natural semantic since the use of the algebraic formalization of each formalism. In addition to that relations between diagrams are expressed via institution morphism and comorphism. We note here that this approach is inspired by Mossakowski works in the heterogeneous institution setting.

The third category of works use this theory for an specific intention and precise cases of study. The work in [7] is a good candidate in this category where authors define a heterogeneous framework of services oriented system, using the institution theory. Authors (in [7]) aims to define a heterogeneous specification approach for service-oriented architecture (SOA). The developed framework consists of a number of specification of individual services written in a local logic and where the specification of their interactions is written in a global logic. The two logic are described via institution theory and an institution comorphism is used to link the two defined institution. This approach is inspired by the work of Mossakowski. Another work is developed in [11]where the authors propose to use institution to represent the logics underling OWL and Z. Then, they propose a formal semantic foundation to the transformation of OWL to Z via the use of institution comorphism. The foundations of this approach is also based on the works in [15].

Our proposed approach aims at first to give a semantic of UML AD via its representation as an institution. So that to consolidate our approach given [13]. Thus, with the defined semantic the transformation of UML AD model to an Event-B model can be semantically proven so that after the transformation we will have the two model semantically equivalent. It's clear that the approach we propose don't tackle the problematic of heterogeneous specification environment like [10] and in [15]. But we think that our work is a first attemps to support the use of a formal method like Event-B as target specification formalism since the Heterogeneous Tool Set [15] among the logic used Event-B is'nt present . The use of Event-B is argued with the following reasons:

- Event-B as a formal method support an interactive and an automatic theorem proving so that the resulted specification after the transformation process can be proved automatically. Event-B as a theorem prover is seeing a continuous improvement by industrial society.

- With the notion of refinement, we can to perform successive refinement to the Event-B model in order to arrive to a pseudo code written in language such Ada.

- Thanks to the notion of composition supported in Event-B, we can to define heterogeneous specification environment with different graphical formalism. And with the notion of composition system described with heterogeneous specification can be composed and so proved formally.

Our work is an inspired form [15]. We are devoted to use UML AD as a formalism for applications modelling. This formalismwill be represented as an institution. Which means that due the algebraic categorical presentation we gain to formal semantic of UML AD.

The version of UML AD used in this paper is 2.0. In the literature, many approaches are proposed for the development of a formal semantic of UML AD. Recent works which addresses the newest version are the work of Harald Storrle in [5] [4] [12]. Storrle provide a formal definitions of the semantics of control-flow, procedure call, dataflow, and exceptions in UML 2.0 Activities. The defined semantic is inspired from Petri-net semantic. The choice of petri-net semantic by the authors is argued by the following reasons.

- The standard claims that in the version 2.0 of UML AD *Activities are redesigned to use a Petri-like semantics instead of state machines.*
- Thanks to the formal foundation adequateness of Petri-net to give a formal semantic for UML AD
- In addition, in [4] St¨orrle have shown how standard Petri-net tools may be applied to verify properties of UML 2.0 activity diagrams, using a Petri-net semantics.

In our paper, we will not used any intermediate semantic for UML AD. We provide a formal semantic of UML AD with mathematical notions in term of categorical abstract presentation. Thus UML AD will be defined in it's own semantic. We gain from this categorical presentation the next benefit:

- From this categorical presentation of syntax and semantic of UML AD, we can to prove that UML AD can be written as an institution
- we can to use the defined institution for a heterogeneous specification tools like [10]
- Because we use Event-B as formal method for the verification of the UML AD we can to use the concepts of institution comorphism and institution morphism to transform UML AD to Event-B

3. LOGIC AS AN INSTITUTION

Institution is an abstract concept invented by Joseph Goguen and Rod Brustall because of the important variety of logics [3]. It offers an abstract theoretic presentation of logic in a mathematical way. An institution consists of notions of signatures, models, sentences, with a technical requirement, called the 'Satisfaction Condition', which can be paraphrased as the statement that 'truth is invariant under change of notation' [16]. The Satisfaction Condition is essential for reuse of specifications: it states that all properties that are true of a specification remain true in the context of another specification which imports that specification.

Definition 1:

An institution $I = (\mathbb{S}ig^I, \mathbb{S}en^I, \mathbb{M}od^I, \models^I)$ *consists of:*

- *A category* $\mathbb{S}ig^I$ *whose objects are called signatures and the arrow are signature morphism.*

- *A functor* $\mathbb{S}en^I : \mathbb{S}ig^I \rightarrow \mathbb{S}et$, *this functor map each signature* Σ *to the set whose elements are called sentences constructed over that signature. Also Sen map each signature morphism to function between sentences.*

- *A functor* $\mathbb{M}od^I : (\mathbb{S}ig^I)^{op} \rightarrow \mathbb{C}at$, *this functor map each signature* Σ *to the category of models of this signature. Also* $\mathbb{M}od$ *map each signature morphism to model homomorphism between models.*

- *A relation* $\models^I_\Sigma$ *giving for each sentences of a signature* Σ *the models in which the sentences are true.*

The relation $\models^I_\Sigma$ *is called the satisfaction condition which can be interpreted like follows:*

Given a signature morphism $\varphi : \Sigma \longrightarrow \Sigma'$ *in the institution I.*

For each model $M' \in \mid \mathbb{M}od(\Sigma') \mid$ *and* $e \in \mathbb{S}en(\Sigma)$

$$\mathbb{M}od^I(\varphi)(M') \models^I_\Sigma e \Rightarrow M' \models^I_{\Sigma'} \mathbb{S}en^I(\varphi)(e)$$

In the next section, we will prove that UML AD can be represented by an institution. The institution of UML AD will be used as a formal semantic for this formalism.

4. USING INSTITUTION FOR THE DESCRIPTION OF SOURCE FORMALISM

4.1 Graphical Formalism

UML activity diagrams (UML AD) are graphical notation developed by OMG. It is used for the specification of workflow applications and to give details for an operation in software development. UML AD serve many purposes, during many phases of the software life cycle [4]. They are intended for being used for describing all process-like structures, (business processes), software processes, use case behaviors, web services, and algorithmic structures of programs. UML AD are thus applicable throughout the whole software life cycle, which means during business modeling, acquisition, analysis, design, testing, and operation, and in fact in many other activities. Thus, they are intended for usage not just by Software-Architects and Software-Engineers, but also by domain specialists, programmers, administrators and so on. The presentation of UML AD as an institution is inspired by the works in [10] [17] [18] [19]. The later work is devoted to define three institution for respectively UML Class diagram, UML Interactions Diagram and OCL. In our paper the semantic and the syntax of UML AD will be based on the works of H. Storrle. As we say in the previous section, the considered work is the more recent and relevant work in this context conformed with the standard.

With the version 2.0 of UML AD, the meta model for Activities has been redesigned from scratch. The main concept underlying Activity Diagrams is now called Activity [12]. The meta model defines six levels increasing expressiveness. The first level (Basic Activities) already includes control flow and procedurally calling of subordinate Activities by Activity Nodes that are in fact Actions. This paper is restricted to Basic Activities. Readers may refer to [4] [5] [12] for more details about the syntax and the semantic of UML AD.

To perform a model transformation between UML AD and Event-B we will use institution theory. Firstly, we represent UML AD as an institution. After we give the structure of Event-B model. Third, we determine a structure preserving mapping from the first institution to Event- B model.

4.2 UML AD as an institution

4.2.1 The syntax of UML AD

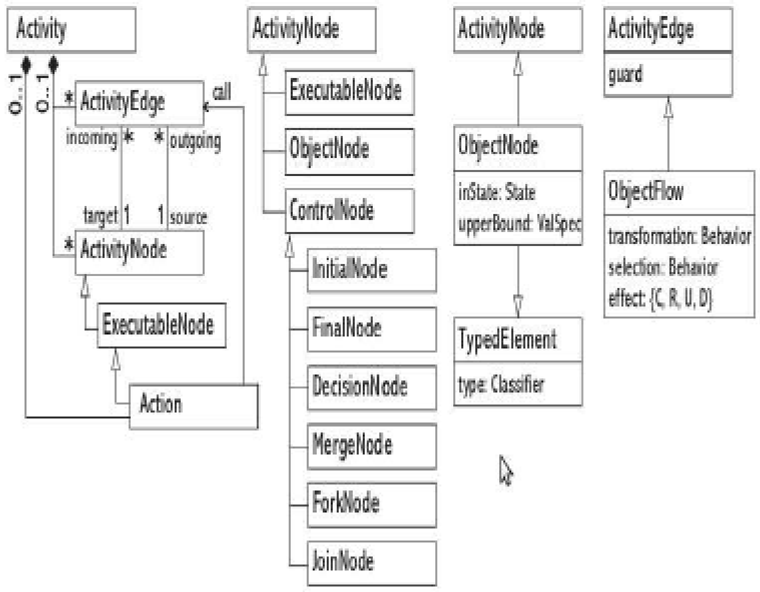


Figure 1. The diagram of the category of model and model homomorphism

Activity as defined in [5] is the coordination of elementary actions or it consists of one atomic action. Besides, given a class diagram, methods are functions that uses attributes of the considered class. Then, class diagram methods are functions or operations that changes the state of an object (defined as an instance of the considered class). In this two cases, we consider an activity as a method of a class in UML class diagram or we consider an activity as a coordination of one action or more. As result, we can to define an relation of hierarchy. This relation is defined between two activities Activity A and Activity B.

An activity hierarchy $\mathbb{A}$ written as $\mathbb{A} = (A, \preccurlyeq_A)$ is a partial order with a set of activity names A and a subclass relation $\preccurlyeq_A \subseteq A \times A$.
Given an activity hierarchy $\mathbb{A} = (A, \preccurlyeq_A)$, a A-activity domain is a A-indexed family $\mathbb{N} = (Na)_{a \in A}$ of sets of activity with $Na \subseteq N_{a'}$ if $a \preccurlyeq_A a'$. we aim to prove that the Activity hierarchies can be formalized as a category which can be done via it's formalization as a Grothendieck construction and also as a monad. The two presentations of Activity hierarchies as Grothendieck construction and as a monad are shows in [19] with replacing class hierarchies with Activity hierarchies.

An UML AD signature consists of a pair $\Sigma = (\mathbb{A}, E)$ where $\mathbb{A}$ is the activity hierarchy and E is the set of Activity Edges.
Given a signature $\Sigma = (\mathbb{A}, E)$ with $\mathbb{A} = (A, \preccurlyeq_A)$, we define a set T of atomic formulas over Σ by:
T :=**skip** | **seq(C,e,D)** $\wedge$ T

with e $\in E$ and C, D $\in A$,

Given UML AD signature $\Sigma_1 = (\mathbb{A}_1, E_1)$ and $\Sigma_2 =$ $(\mathbb{A}_2, E_2)$. we define a UML AD signature morphism $\varphi : \Sigma_1 \longrightarrow \Sigma_2$ as a morphism that maps Activity node names to Activity node names and maps Activity Edges to Activity Edges. We note here that Activity node can be one of the following node:

- EN: The set of Executable Nodes (i.e. elementary Actions);
- IN or FN : The Initial Nodes or the Final Nodes
- BN: the set of branch nodes, including both Merge Nodes and Decision Nodes
- CN: the set of concurrency nodes, subsuming Fork Nodes and Join Nodes;
- ON: the set of Object Nodes;

As for Activity Edges may be a pair AE, OF , where:

- AE: the set of plain Activity Edges between Executable Nodes and Control Nodes;
- OF: the set of Object Flows between Executable Nodes and Control Nodes on the one hand, and Object Nodes on the other.

Signature morphism extend to atomic formulas over $\sum_1$ as follows:
$\varphi(skip) = skip$

$\varphi(\mathbf{seq}(C_1, e_1, D_1) \wedge \quad T_1) = \quad \mathbf{seq}(\varphi(C_1), \varphi(e_1), \varphi(D_1)) \wedge$
$\varphi(T_1)$

4.3 The semantic of UML AD

In Standard, The semantic of UML AD is determined by a path expressing the trace of the execution. For the execution, a token will move from initial Activity Node To final Activity Node [4]. Each AD Activity has its role in AD execution [12]. First of all, a token in The Initial Nodes means the beginning of the execution of UML AD. Then, the trace of the token will be defined by the outgoing edges of the initial AD. When a token arrive to an Executable Node, it will trigger the Action or the operation in this node. For Join Node, if there is a token offered on all incoming edges, then tokens are offered on the outgoing edge. A Fork Node means that, when an offered token is accepted on all the outgoing edges, duplicates of the token are made and one copy traverses each edge. In the case of Merge Node and Decision Node, every edge is associated to a condition determining the condition of the activation of the edge. For Merge Node, if there is a token offered to only one of the incoming edges where the condition is true (it is a sufficient condition), then token are offered on the outgoing edge of the Merge Node. A Decision Node means that, In the outgoing edge where the condition is true, an offered token will traverses this edge. A token that traverses a Object Node means the availability of the object (variable) needed to the execution of the coming activity.

Given a UML signature $\Sigma = (\mathbb{A}, E)$ with $\mathbb{A} = (A, \preccurlyeq_A)$, a structure I for Σ is a triple I=$(\mathbb{N}, \mathbb{E}, \mu)$ where $\mathbb{N}$=$(N^a)_{a\in A}$ is an Activity domain for $\mathbb{A}$, $\mathbb{E}$ a domain of edges and $\mu : E \longrightarrow \mathbb{E}$ is an interpretation function for edges. Given a variable C a valuation β for C in I assigns values to variables. This means:
$\beta : C \longrightarrow N^a$

A sub-signature $\Sigma' = (\mathbb{A}', E') \subseteq \Sigma$ with $\mathbb{A}' = (A', \preccurlyeq_{A'})$ induces a set of traces $T(\Sigma', I)$ defined as follows:

$T(\Sigma', I)$=$\{e_1.e_2..e_n \mid i \in \{1, ..., n\}, e_i = seq(C_i, e_i, D_i), C_i, D_i \in \mathbb{A}' and e_i \in E'\}$

The set of $\mathbb{T}(I)$of all traces is defined as :

$\mathbb{T}(I)$=$\{e_1.e_2..e_n \mid i \in \{1, ..., n\}, e_i = seq(C_i, e_i, D_i) and C_i, D_i, e_i \in I\}$

The set $\Theta(T, \beta)$ of traces of an atomic formula T over Σ in the structure I under the valuation β are inductively defined as follows:

T:=skip $\Longrightarrow \Theta(T, \beta)$=$\{\varepsilon\}$

T:=seq(C,e,D) $\Longrightarrow \Theta(T, \beta) = \{seq(\beta(C), \mu(e), \beta(D))\}$

T :=skip | seq(C,e,D) $\wedge$ T

with e $\in E$ and C, D $\in A$,

4.4 The satisfaction condition under the UML AD institution

Let $\Sigma_1 = (\mathbb{A}_1, E_1)$ and $\Sigma_2 = (\mathbb{A}_2, E_2)$ be two UML AD signatures, an UML AD signature morphism $\varphi : \Sigma_1 \longrightarrow \Sigma_2$, two structure I_1 a Σ_1-structure and I_2 a Σ_2-structure defined as $I_1 = (\mathbb{N}_1, \mathbb{E}_1, \mu_1)$ and $I_2 = (\mathbb{N}_2, \mathbb{E}_2, \mu_2)$. Semantic invariance under the change of notation is formulated as $\Theta_{I_2}(\varphi(T_1), \beta_2) = \Theta_{I_1}(T_1, \beta_1)$ for any atomic formula T_1 over Σ_1. This can be shown by induction on the structure of T_1.

$\Theta_{I_2}(\varphi(skip), \beta_2) = \{\varepsilon\} = \Theta_{I_1}(\varphi(skip), \beta_1)$

$\Theta_{I_2}(\varphi(seq(C, e, D)), \beta_2) =$
$\Theta_{I_2}(seq(\varphi(C), \mu(e), \varphi(D))(skip), \beta_2) =$
$\{seq(\beta_2(\varphi(C)), \beta_2(\mu(e)), \beta_2(\varphi(D))\} =$
$\{seq(\beta_1(C), \beta_1(e), \beta_1(D)\} = \Theta_{I_1}(T_1, \beta_1)$

Also we have $T(\varphi(\Sigma_1), I_2) = T(\Sigma_1, I_1)$

4.5 Signatures

As shown in the example(fig??), the following elements or components can be captured visually. We have Noeud Initiale, A0, A1, A2, A3, A4 and Noeud Finale seven activity nodes. O is object node. We have also two control node AND SPLIT node and OR JOIN node. In addition to that two node one initial node and the other final node. Then, an itinerary between the activities showing a sequence of execution.

Institution in this context will be used to represent the concrete syntax thus the formal semantic of UML AD. To describe as an institution an UML AD, we identify the four elements of the mathematical definition of institution from the UML AD formalism.

We are conducted to define five predicates that represent different interconnections between UML AD objects. Those predicates are defined as follows:

- seq(a,b): A predicate that declare a sequence execution of a and b and an order means a before b.
- AND SPLIT(a,b,c): A predicate that declare a control node which is an and split node between the the activity node a, b and c.
- OR SPLIT(a,b,c): A predicate that declare a control node which is an or split node between the the activity node a, b and c.
- AND JOIN(a,b,c): A predicate that declare a control node which is an and join node between the the activity node a, b and c.
- OR JOIN(a,b,c): A predicate that declare a control node which is an or join node between the the activity node a, b and c.

Let $Sig^{UML\ AD}$ be the category of UML AD signatures. $Sig^{UML\ AD}$ is denoted by (AN,ON,CN), where AN is the set of Activity Node, ON is the set of Object Node and CN is the set of Control Node. The set CN represent also the list of predicates used in UML AD. For the example shown in figure 1, A0, A1, A2, A3, A4, O, Noeud Initiale, Noeud Fianle, and split, or join are the signatures.

4.6 Sentences

As for the sentences of the UML AD institution, we will define the concrete syntax. The abstract syntax can be studied by the notion of graph grammar [?]. The functor Sen is defined as follows :

$Sen^{UML\ AD} : Sig^{UML\ AD} \rightarrow Set$

Set is the set of sentences. It is constructed over the category of signatures. It expresses the existing information present in a graphical model of UML AD. Using the previous predicates, the sentences associated with a signature declares the clauses describing the execution of the activities over the elements of the signature. The five predicates defined previously represent (CN) terms. which mean a syntactic structure $\sigma(t_1,\ldots..t_n)$ where $\sigma \in CN_{8\rightarrow 8}$ is an predicates symbol and $t_1,\ldots.,t_n$ are (CN) term. The set of sentences of UML AD contains the conjunction of the set of (CN) terms.

The UML AD of ?? can be viewed by the following clauses(sentences):

$$\begin{cases} seq(NoeudInitiale, A0)\wedge \\ ANDSPLIT(A0, A1, A2)\wedge \\ ORJOIN(A1, A2, A3)\wedge \\ seq(A3, O, A4)\wedge \\ seq(A4, NoeudFinale) \end{cases}$$

Given a signature of UML AD signatures φ :(AN,ON,CN) $\longrightarrow$ (AN',ON',CN').
The translation $Sen^{UMLAD}(\varphi)$ is defined as follows:

$Sen^{UMLAD}(\varphi)(\rho_1 \wedge \rho_2)$=$Sen^{UMLAD}(\varphi)(\rho_1)\wedge$ $Sen^{UMLAD}(\varphi)(\rho_2)$

Proposition 1:
Sen^{UMLAD} is a functor defined as follows: $Sen^{UMLAD} : Sig^{UMLAD} \rightarrow Set$

As for the example (fig ??), Set = $\{seq(NoeudInitiale, A0), ANDSPLIT(A0, A1, A2),$ $ORJOIN(A1, A2, A3), seq(A3, O, A4),$ $seq(A4, NoeudFinale)\}$

4.7 Models

Given an UML AD signature(AN,ON,CN), a model M interprets:

- Each Activity Node as a name of concrete activity for example A1 as the addition between a and b,
- Each Object Node as a name of concrete object for example read real a,
- Each Control Node $\sigma \in CN_w$ as a function M_σ:$M_\sigma \rightarrow \{True\}$ where w is the arity of the control node M_w stand for $M_{s_1} \times \ldots \times M_{s_n}$ for w=$s_1 \times \ldots \times s_n$.

An (AN,ON,CN) model homomorphism h:$M \rightarrow M'$ is a triple of functions such that:

- h is an CN algebra homomorphism $M \rightarrow M'$ with h(M_σ(m))=$M'_\sigma(h_w(m))$ for each $\sigma \in CN_{w \rightarrow \{True\}}$ and for each m $\in M_w$
- $h_w(m) \in M'_w$ for each m $\in M_w$

$$\begin{tikzcd}
M_w \arrow[r, "M_\sigma"] \arrow[d, "h_w"'] & M_s \arrow[d, "h_s"] \\
M'_w \arrow[r, "M'_\sigma"'] & M'_s
\end{tikzcd}$$

Figure 2. The diagram of the category of model and model homomorphism

Proposition 2:

(AN,ON,CN) models with (AN,ON,CN) model homomorphism for a category.

Proposition 3:

UML Activity Diagram form an Institution presented as below:

- *Signatures are activity nodes names, control node names, objects node names.*

- *Sentences are the conjunction of the predicates identifying the itinerary of the execution of activities. The predicates are seq(*,*), AND SPLIT(*,*,*), OR SPLIT(*,*,*), AND JOIN(*,*,*), OR JOIN(*,*,*).*

- *Model interprets each signature as follows:*

  – *Each activity as a name of concrete activity for example A1 as the addition between a and b,*

  – *Each object as a name of concrete objects for example read a,*

  – *Each Control Node $\sigma \in CN_w$ as a function M_σ:$M_\sigma \rightarrow \{True\}$ where w is the arity of the control node M_w stand for $M_{s_1} \times \ldots \times M_{s_n}$ for w=$s_1 \times \ldots \times s_n$.*

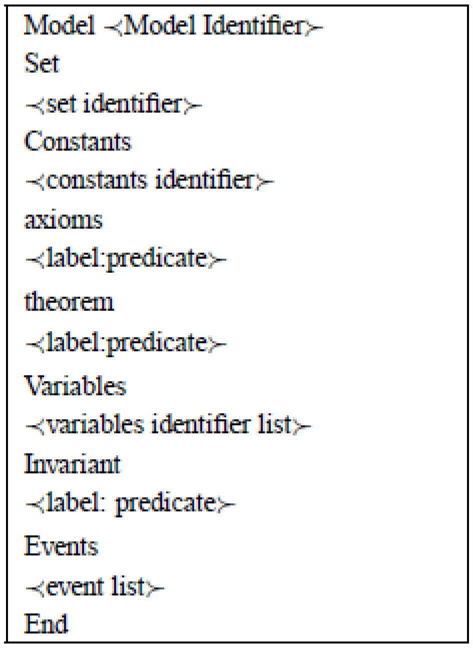
In this section, we have presented a UML AD as an institution with signatures, sentences and models. This institution will be the kernel of our transformation to the target formalism which is Event-B. From this institution we can to conclude that the description is composed of a static part and a dynamic one. The static part appear in signatures and the dynamic part is the sentences of the institution. We mean by dynamic part the order of the execution of the activities.

5. EVENT-B

Event-B is the variant of the method B. It is a formal method used for the verification of reactive system developed by Jean-Raymond Abrial Event-B model is formed by a static part named contexts and dynamic part named machines. Like it mentioned in the example of an Event-B model(fig 3), the static part are carrier sets, constants, axioms, and theorems. As for the dynamic part, it consist of variables, invariants, theorems, variants, and events Here we recall some definition in order to construct the Event-B institution.

5.0.1 Event-B language

We give the structure of Event-B model as it is defined in Now we will describe the contents of each clause:



Model ≺Model Identifier≻
Set
≺set identifier≻
Constants
≺constants identifier≻
axioms
≺label:predicate≻
theorem
≺label:predicate≻
Variables
≺variables identifier list≻
Invariant
≺label: predicate≻
Events
≺event list≻
End

Figure 3. The structure of an Event-B model.

The clause Model stand for the name of the model.
This name will replace ≺Model Identifier≻.
-The clause Set mean the set used in the model. The list of sets will be written in the model.
-The clause Constants list the list of the constants.
-The clause axioms list the predicates used in the model.
-The clause theorem lists the various theorems which have to be proved within the static part.
-The clause Variables contains the list of the variables used in the dynamic part of the model.
-The clause Invariant list the invariants that must be proved by the list of events of the model.
-The clause Events contains the definition of event of the model.

6. The Transformation of an UML AD Institution to an Event-B Model

The transformation of UML AD to Event-B model will be ensured by a set of transformation rules. In fact, this transformation will rules will behave on the algebraic definition of UML AD. To perform the transformation, we will begin by the transformation of the dynamic part of UML AD to to the dynamic part of Event-B. The dynamic part of UML AD is the sentences of UML AD institution. As for Event-B, the dynamic part is the set of events presents in the Event-B model.

6.1 Sentences of UML AD to set of Event-B model

We present here a set of transformation rules between the set of sentences of UML AD and the set of Event-B events. The table below give the set of the transformation rules.

For more details of how to make the transformations rules, readers are invited to read our previous work.

Table 1. Transformation rules of the dynamic part.

UML AD sentences	Event-B event
seq(a,b)	Eventseq1=**SELECT** stateN1=a & G0a **THEN** stateN1:=b || S0b **END**
AND SPLIT(a,b,c)	Eventand_split1=**IF** G0b & stateN1B1=a **THEN** stateN1B1:=b || S0b **ELSIF** G0c & stateN1B2=a **THEN** stateN1B2:=c || S0c **END** ;
OR SPLIT(a,b,c)	Eventor_join2=**IF** G0c & stateN1B1=a **THEN** S0c || stateN2:=c **ELSIF** G0c & stateN1B2=b **THEN** S0c || stateN2:=c **END** ;
AND JOIN(a,b,c)	Eventand_join1=**IF** G0c & stateN1B1=a & stateN1B2=b **THEN** stateN1B1:=c || stateN1B1:=c || S0c **END** ;
OR JOIN(a,b,c)	Eventor_join2=**IF** G0c & stateN1B1=a **THEN** S0c || stateN2:=c **ELSIF** G0c & stateN1B2=b **THEN** S0c || stateN2:=c **END** ;

6.2 Sentences and Signatures of UML AD to the static part of Event-B model

After obtaining the dynamic part of the Event-B model, we will, in this section, define how to get the static part of Event-B model from the sentences and the signatures of UML AD institution. Below we give an overview of how to get the static part:

• Model ≺Model Identifier≻
The name of the model can to be given arbitrary.

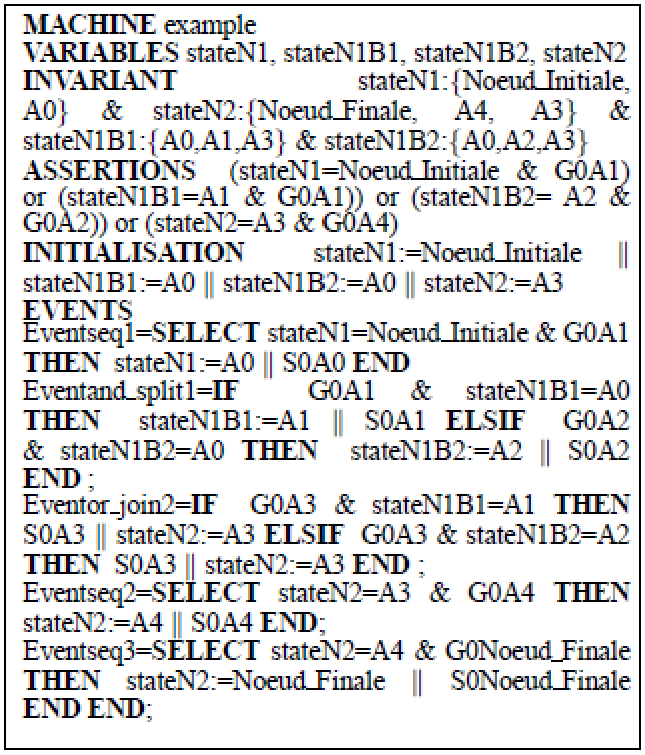
• Variables≺Variables list ≻
The list of the variables can be identified from the event defined in the previous sections.

• Invariant ≺label: predicate≻
Invariant show the set of each variable. It can be also identified from the set of event defined in the previous section.

• Assertions ≺ Assertions list≻
The assertions clause is the conjunction of the guard of the event of the Event-B model. The guard will be identified fromthe set of events defined in the previous section.

• Initialization <Initialization list>

As an example, we take the UML ad model of the figure ?? and perform the transformation rules. Transformation rules will acts on the institution of UML AD. 4 represent the transformation of UML AD of fig ??.



```
MACHINE example
VARIABLES stateN1, stateN1B1, stateN1B2, stateN2
INVARIANT stateN1:{Noeud_Initiale,
A0} & stateN2:{Noeud_Finale, A4, A3} &
stateN1B1:{A0,A1,A3} & stateN1B2:{A0,A2,A3}
ASSERTIONS (stateN1=Noeud_Initiale & G0A1)
or (stateN1B1=A1 & G0A1)) or (stateN1B2= A2 &
G0A2)) or (stateN2=A3 & G0A4)
INITIALISATION stateN1:=Noeud_Initiale ||
stateN1B1:=A0 || stateN1B2:=A0 || stateN2:=A3
EVENTS
Eventseq1=SELECT stateN1=Noeud_Initiale & G0A1
THEN stateN1:=A0 || S0A0 END
Eventand_split1=IF G0A1 & stateN1B1=A0
THEN stateN1B1:=A1 || S0A1 ELSIF G0A2
& stateN1B2=A0 THEN stateN1B2:=A2 || S0A2
END ;
Eventor_join2=IF G0A3 & stateN1B1=A1 THEN
S0A3 || stateN2:=A3 ELSIF G0A3 & stateN1B2=A2
THEN S0A3 || stateN2:=A3 END ;
Eventseq2=SELECT stateN2=A3 & G0A4 THEN
stateN2:=A4 || S0A4 END;
Eventseq3=SELECT stateN2=A4 & G0Noeud_Finale
THEN stateN2:=Noeud_Finale || S0Noeud_Finale
END END;
```

Figure 4. Event-B model: the result of the UML AD transformation.

7. CONCLUSIONS

We have presented in this paper a formal approach for the transformation of UML AD to Event-B model. We use institution as a language for the description of UML AD. Thus, we give an institutional definition of UML AD. The institution of UML AD is constructed under the institution of First Order Logic. As for the target formalism which is Event-B, we define properly the syntax of an Event-B model adapted to UML AD models. For the transformation, we give a set of transformation rules of UML AD sentences to Event-B models. As for the future work, we aim to prove that Event-B can be represented as an institution. As a consequence, we will have two different institution describing each formalism. Institution morphism and comorphism will be a solid and effective notions to link and make transformation from one formalism to the other. Also refinement can be another future goal to move from one institution to the other. In our works, institution is considered as meta level to fulfill model transformation. As for the development of an engine of model transformation between the UML AD and Event-B, we aims to enrich our previous developed tool mentioned in [?]. We will add the formal semantic of UML AD developed in this work to our tool.

REFERENCES

[1] R. Diaconescu, Grothendieck institutions, Applied Categorical Structures, 10, 1994, 167-174.

[2] R. Diaconescu and K. Futatsugi, Logical Foundations of CafeOBJ, Theoretical Computer Science, 285, 289–318.

[3] J. Goguen and G. Rosu, Institution Morphisms, Formal Aspects Computing, 13, 2002, 274-307.

[4] H. Storrle, Structured nodes in UML 2.0 activities, Nordic J. of Computing, 11, 2004, 279–302.

[5] H. Storrle, Semantics and Verification of Data Flow in UML 2.0 Activities, Electronic Notes Theoretic Computer Science, 127, 2005, 35-52.

[6] J.-R.Abrial, Modeling in Event-B: System and Software Engineering (Cambridge University Press, 2010).

[7] A.Knapp, G.Marczynski, M.Wirsing and A.Zawlocki, A heterogeneous approach to service-oriented systems specification, SAC, 2010, 2477-2484.

[8] M. Codescu, F. Horozal,M. Kohlhase, T.Mossakowski, F. Rabe and K. Sojakova, Towards Logical Frameworks in the Heterogeneous Tool Set Hets, WADT, 2010, 139-159.

[9] M. A. Martins, A. Madeira, R. Diaconescu and L. S. Barbosa, Hybridization of Institutions, CALCO, 2011, 283-297.

[10] M. V. Cengarle, A. Knapp, A. Tarlecki and M. Wirsing, A Heterogeneous Approach to UML Semantics, Concurrency, Graphs and Models, 2008, 383-402.

[11] D. Lucanu, Institution Morphisms for Relating OWL and Z, The 17th International Conference on Software Engineering and Knowledge Engineering, 2005.

[12] H. St¨orrle and J. H. Hausmann and U. Paderborn, Towards a formal semantics of UML 2.0 activities, German Software Engineering Conference, 2005, 117–128.

[13] L. J. Ben Ayed, A. Ben Younes and A. Achouri, Using AToM3 for the Verification of Workflow Applications, ICSOFT (2), 2010, 32-39.

[14] Vitolins, Valdis and Kalnins, Audris, Semantics of UML 2.0 Activity Diagram for Business Modeling by Means of Virtual Machine, the Ninth IEEE International EDOC Enterprise Computing Conference, 2005, 181–194.

[15] T. Mossakowski , Heterogeneous Specification and the Heterogeneous Tool Set, Habilitation thesis, Universitaet Bremen, 2005.

[16] R. Diaconescu, Institution-independentModel Theory (Birkh¨auser Basel, 2008).

[17] M.V.Cengarle , UML 2.0 Interactions: Semantics and Refinement, Technical report, Institut fr Informatik, Technische Universitt Mnchen, 2004.

[18] M.V.Cengarle and A.Knapp , An Institution for UML 2.0 Interactions, Technical report, Institut fr Informatik, Technische Universitt Mnchen, 2008.

[19] M.V.Cengarle , An Institution for UML 2.0 Static Structures, Technical report, Institut fr Informatik, Technische Universitt Mnchen, 2008.