The International Journal of Soft Computing and Software Engineering [JSCSE], Vol. 3, No. 3, Special Issue:
The Proceeding of International Conference on Soft Computing and Software Engineering 2013 [SCSE'13],
San Francisco State University, CA, U.S.A., March 2013
Doi: 10.7321/jscse.v3.n3.30 e-ISSN: 2251-7545

A Formal Semantic for UML 2.0 Activity Diagram based on Institution Theory

Amine Achouri
Laboratory LaTICE
ESSTT
University of Tunis, TUNISIA
Amine.Achouri@fst.rnu.tn

Leila Jemni ben Ayed
Laboratory LaTICE
ESSTT
University of Tunis, TUNISIA
Leila.Jemni@fsegt.rnu.tn

***Abstract*—Giving a formal semantic to an UML Activity diagram (UML AD) is a hard task. The reason of this difficulty is the ambiguity and the absence of a precise formal semantic of such semi-formal formalism. A variety of semantics exist in the literature having tackled the aspects covered by this language. We can give as example denotational, functional and compositional semantics. To cope with the recent tendency which gave a heterogeneous semantic to UML diagrams, we aim to define an algebraic presentation of the semantic of UML AD. In this work, we define a formal semantic of UML 2.0 AD based on institution theory. For UML AD formalism, which is a graphical language, no precise formal semantic is given to it. We use the institution theory to define the intended semantic. Thus, the UML AD formalism will be defined in its own natural semantic.**

***Keywords*-Institution theory; UML 2.0 Activity Diagram; Formal semantic;**

I. INTRODUCTION

Model transformation is a critical process in software construction and development. As increasingly larger software systems are being developed, there is tendency to have solid and effective tools to automatize the software development. The specification of a software can be formal and (or) graphical. For graphical formalisms, we can mention as example UML models, UML class diagram, UML activity diagram and interaction diagram. For the formal ones, logic are increasingly used due to their mathematical background. For example, Petri-net is used as a graphical and a formal specification formalism. Logic is the language of formal methods such that theorem proving and model checking. To facilitate and to link graphical and formal language, there is a massive need to make generic techniques for the transformation of graphical models to formal notations. The use of logic is difficult for non familiar with logical concepts and specification. As a result, there is a need to provide the possibility to make specifications in a modeling level.

Stakeholders can begin with a graphical model (possibly with many system views). Then, with an automatic and correct transformation they can to produce a specification in a formal logic. In the context of logic, institution theory has emerged as a framework allowing their study and the different relation between them.

In our previous work [8], 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. 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. To overcome this drawback, we use institution theory to make the required semantic for the source formalism which is UML AD.

The first contribution aims to give institutional presentation of UML AD. In our knowledge, in the literature, no proven institution for UML AD exists. This institutional presentation define a formal semantic of UML AD. In addition, this algebraic presentation of the source formalism will be a meta-level to study possible transformation to Event-B models [8]. Thus, the study of some proprieties like model amalgamation and theory co-limits of this formalism will be enhanced [**?**]. Those notions play a key role in heterogeneous specification approaches. The UML AD institution may be used in a heterogeneous modeling language such UML diagram like in [13].

The paper is organized as follows: in section 2 we present the related works. Then in section 3, we recall institution definition. Section 4 shows how to prove that UML AD establish an institution. Section 5 give an example of UML AD model and make focus in its institution. Finally, the last section concludes our work.

II. RELATED WORKS

In literature, institution theory is largely used and studied. We have three category of works based on institution theory.

The first category is interested on the use of institution theory and its known concepts in the development of an heterogeneous specification approaches. We mention the approach of the heterogeneous specification in the tool cafeOBJ [5]. This approach is based on a cube on eight logic and twelve projections (defined as a set of institution morphism and institution comorphism) [5]. It's inspired by the semantic based on Diaconescu's notion of Grothendieck institution [4]. Another approach is developed in the work of Mossakowski [9] [2]. The heterogeneous logical environment developed by the author is formed by a number

The International Journal of Soft Computing and Software Engineering [JSCSE], Vol. 3, No. 3, Special Issue:
The Proceeding of International Conference on Soft Computing and Software Engineering 2013 [SCSE'13],
San Francisco State University, CA, U.S.A., March 2013
Doi: 10.7321/jscse.v3.n3.30 e-ISSN: 2251-7545

of logical systems. These logical systems are formalized as institutions linked with the concepts of institution morphism and comorphism.

The second category of works focus on the use of institution theory in the specification of graphical formalism such as UML diagrams. In this category, we mention the work present in [13] [10] [11] [12]. 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 its natural semantic because of the use of the algebraic formalization of each formalism. In addition, 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 uses this theory for a specific intention and a precise case study. The work in [1] is a good candidate in this category where authors defined a heterogeneous framework of services oriented system, using institution theory. Authors (in [1]) aims to define a heterogeneous specification approach for service-oriented architecture (SOA). The developed framework consists of a several individual services specification written in a local logic. The specification of their interactions is written in a global logic. The two defined logics 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 [6] where the authors propose to use institution to represent the logics underling OWL and Z. Then, they propose a formal semantic for the transformation of OWL to Z specification via the use of institution comorphism.

Our proposed approach aims at first to give a semantic for UML AD via its representation as an institution. As a result, we propose to consolidate our approach given [8]. Thus, with the defined semantic the transformation of UML AD model to an Event-B model can be semantically proven which means that the two model will be semantically equivalent. It's clear that the approach we propose do not tackle the problematic of heterogeneous specification environment like [13] and in [9]. The use of Event-B is argued with the following reasons:

- Event-B is a formal method that supports interactive and automatic theorem proving. 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 refinements to the Event-B model in order to obtain a pseudo code written in declarative language.
- Thanks to the notion of composition supported in Event-B, we can define heterogeneous specification environment with different graphical formalism. With the notion of composition, system described with heterogeneous specification can be composed and then proved formally.

Our work is inspired form [9]. We are devoted to use UML AD as a formalism for applications modeling. This formalism will be represented as an institution. We intend to gain a formal semantic of UML AD thanks to its algebraic categorical presentation.

The version of UML AD used in this paper is 2.0. In literature, many approaches are proposed for the development of UML AD formal semantic. Recent works which treated the newest version are the work of Störrle in [16] [15] [17]. Störrle provides a formal definitions for the semantics of control-flow, procedure call, data-flow, and exceptions in UML 2.0 Activities. The defined semantic is inspired by 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 [15] Störrle 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 use any intermediate semantic for UML AD such using Petri-net semantics. We provide a formal semantic of UML AD with mathematical notions in term of categorical abstract presentation. We get profit from this categorical presentation the next benefit:

- From this categorical presentation of the syntax and the 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 an heterogeneous specification tools like [13]
- 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

III. Logic as an Institution

Institution is an abstract concept invented by Joseph Goguen and Rod Brustall because of the important variety of logics. It provide a basis for reasoning about software specifications independent of the choice of the underlying logical system [7].

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' [14]. Modeling the signatures of a logical system as a category, we get the possibility to translate sentences

The International Journal of Soft Computing and Software Engineering [JSCSE], Vol. 3, No. 3, Special Issue:
The Proceeding of International Conference on Soft Computing and Software Engineering 2013 [SCSE'13],
San Francisco State University, CA, U.S.A., March 2013
Doi: 10.7321/jscse.v3.n3.30 e-ISSN: 2251-7545

and models across signature morphisms. 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 $I = (Sig^I, Sen^I, Mod^I, \models^I)$ *consists of:*

- *A category* Sig^I *whose objects are called signatures and the arrow are signature morphism.*
- *A functor* $Sen^I : Sig^I \rightarrow Set$*, 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* $Mod^I : (Sig^I)^{op} \rightarrow Cat$*, this functor map each signature* Σ *to the category of models of this signature. Also Mod 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 $\phi : \Sigma \longrightarrow \Sigma'$ *in the institution I.*

For each model $M^t \in |\ Mod(\Sigma^t)\ |$ *and* $e \in Sen(\Sigma)$

$$Mod^I(\phi)(M^t) \models^I_\Sigma e \Rightarrow M^t \models^I_{\Sigma'} Sen^I(\phi)(e)$$

IV. Using institution for the description of UML AD formalism

A. Graphical Formalism

UML activity diagrams (UML AD) are graphical notation developed by the OMG. It's 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 [15]. 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. Some works in the literature use to define an institution for UML diagrams, we mention [13] [10] [11] [12]. The cited works is devoted to define three institution for respectively UML Class diagram, UML Interactions Diagram and OCL. In our paper, the semantic of UML AD will be based on the works of H. Störrle. 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 (fig 1). The main concept underlying Activity Diagrams is now called Activity [17]. 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 (see fig 1). This paper is restricted to Basic Activities. Readers may refer to [15] [16] [17] for more details about the syntax and the semantic of UML AD.

Next, we will prove that UML AD formalism can be written as an institution.

B. The syntax of UML AD

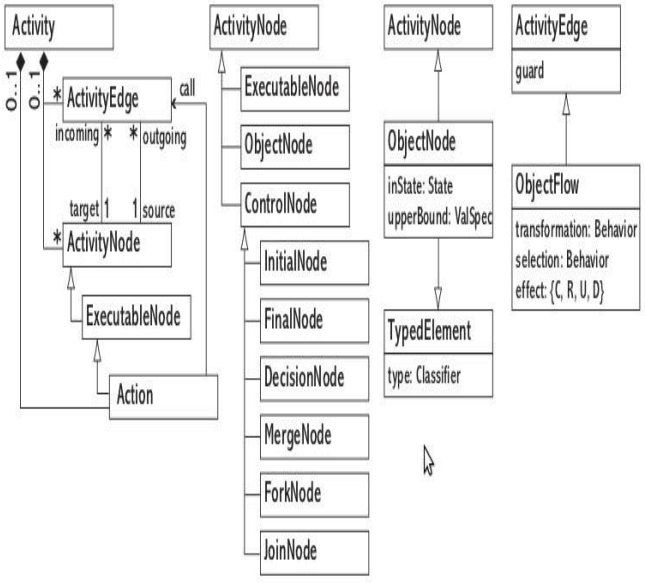

Figure 1. A portion of the meta model of UML AD (as it is defined in the standard).

Activity as defined in [16] 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

The International Journal of Soft Computing and Software Engineering [JSCSE], Vol. 3, No. 3, Special Issue:
The Proceeding of International Conference on Soft Computing and Software Engineering 2013 [SCSE'13],
San Francisco State University, CA, U.S.A., March 2013
Doi: 10.7321/jscse.v3.n3.30 e-ISSN: 2251-7545

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 define a relation of hierarchy. This relation is defined between two activities Activity A and Activity B.

An activity hierarchy A written as A = $(A, =;_A)$ is a partial order with a set of activity names A and a subclass relation $=;_A \subseteq A \times A$.

Given an activity hierarchy A = $(A, =;_A)$, a A-activity domain is a A-indexed family N = $(N_a)_{a \in A}$ of sets of activity with $N_a \subseteq N_{a'}$ if $a =;_A a'$. 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 shown in [12] (with replacing class hierarchies with Activity hierarchies).

An UML AD signature consists of a pair $\Sigma = (A, E)$ where A is the activity hierarchy and E is the set of Activity Edges.

Given a signature $\Sigma = (A, E)$ with A = $(A, =;_A)$, we define a set $\boldsymbol{T}$ of atomic formulas over Σ by:

$\boldsymbol{T}$ **:=skip** | **seq(C,e,D)**with e $\in E$ and C, D $\in A$,

Given UML AD signatures $\Sigma_1 = (A_1, E_1)$ and $\Sigma_2 = (A_2, E_2)$.

We define a UML AD signature morphism $\phi : \Sigma_1 \longrightarrow \Sigma_2$ 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 Σ_1 as follows:

$\phi(skip) = skip$

ϕ**(seq(**C_1**,**e_1**,**D_1**)=seq(**$\phi(C_1)$**,**$\phi(e_1)$**,**$\phi(D_1)$**)**

Let $\Sigma = (A, E)$ be an UML AD signature. X =$(X^a)_{a \in A}$. The language of propositional (Σ,X) formulas has the below form:

$\boldsymbol{T}$ **:=skip** | **seq(C,e,D)**.

The language of first order (Σ,X) formulas has the form: φ**::=**$\boldsymbol{T}$ | $\boldsymbol{T}$**=**$\boldsymbol{T}$ | $\neg\varphi$ | $\varphi \wedge \varphi$ | $\varphi \vee \varphi$ | $\varphi \Longrightarrow \varphi$ | $\varphi \Leftrightarrow \varphi$ | $(\exists x)\varphi$ | $(\forall x)\varphi$.

Σ sentences are closed formulas defined on (Σ,X) formulas.

C. The semantic of UML AD

In the 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 the Initial Activity Node To the Final Activity Node [15]. Each Activity has its role in AD execution [17]. First of all, a token in the Initial Node 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 node. When a token arrive to an Executable Node, it will trigger the Action or the operation in this node. For the Join Node, if there is a token offered on all incoming edges, then a token 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 (s) respectively incoming and outgoing is associated to a condition determining the condition of the activation of this edge. For Merge Node, if there is a token offered to only one of the incoming edges where the condition is true (it's a sufficient condition), then a 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 $\Sigma = (A, E)$ with A = $(A, =;_A)$, a structure I for Σ is a triple I=(N, E, μ) where N=$(N^a)_{a \in A}$ is an Activity domain for A, E a domain of edges and $\mu : E \longrightarrow$ 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 $\Sigma^t = (A^t, E^t) \subseteq \Sigma$ with $A^t = (A^t, =;_{A^t})$ induces a set of traces $T(\Sigma^t, I)$ defined as follows:

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

The set of T(I)of all traces is defined as :

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

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

The International Journal of Soft Computing and Software Engineering [JSCSE], Vol. 3, No. 3, Special Issue:
The Proceeding of International Conference on Soft Computing and Software Engineering 2013 [SCSE'13],
San Francisco State University, CA, U.S.A., March 2013
Doi: 10.7321/jscse.v3.n3.30 e-ISSN: 2251-7545

***T* :=skip | seq(C,e,D)** with e ∈ *E* and C, D ∈ *A*,

D. The satisfaction condition under the UML AD institution

Let $\Sigma_1 = (A_1, E_1)$ and $\Sigma_2 = (A_2, E_2)$ be two UML AD signatures, an UML AD signature morphism $\phi : \Sigma_1 \longrightarrow \Sigma_2$, two structure I_1 a Σ_1-structure and I_2 a Σ_2-structure defined as $I_1 = (N_1, E_1, \mu_1)$ and $I_2 = (N_2, E_2, \mu_2)$. Semantic invariance under the change of notation is formulated as $\Theta_{I_2}(\phi(T_1), \beta_2) = \Theta_{I_1}(T_1, \beta_1)$ for any atomic formula T_1 over Σ_1. This can be shown by induction on the structure of T_1.

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

$\Theta_{I_2}(\phi(seq(C, e, D)), \beta_2) =$

$\Theta_{I_2}(seq(\phi(C), \mu(e), \phi(D))(skip), \beta_2) =$

$\{seq(\beta_2(\phi(C)), \beta_2(\mu(e)), \beta_2(\phi(D))\} =$

$\{seq(\beta_1(C), \beta_1(e), \beta_1(D)\} = \Theta_{I_1}(T_1, \beta_1)$

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

E. The institution of UML AD

After this theoretic study of UML AD, we can to prove that it form an institution. We can immediately observe that institutional presentation rely heavily on the institution of First Order Logic.

Proposition 1:
UML Activity Diagram form an Institution presented as below:

- *Signatures declares Activity Nodes names, Edges Nodes names.*
- *Sentences are closed formulas where well formed formulas combines atomic formulas using the conjunction, negation, universal quantification and equality of variables. The atomic formulas associated to UML AD are UML AD branch (connection between Activity Node names) and it's composition using the operator seq.*
- *Model interprets each signature as follows:*
  - *Each activity node (depending to Activity Node type) as:*
    - *An instance of Executable Nodes if it denote the set EN.*
    - *A truth valuation if it is Initial Nodes or the Final Nodes.*
    - *A valuation to true or false depending to the condition on the branch nodes (including both Merge Nodes and Decision Nodes).*
    - *A valuation to true when it denote a concurrency nodes, subsuming Fork Nodes and Join Nodes.*
    - *An instance of object or an attributes on a Object for Object Nodes.*
  - *As for Activity Edges the interpretation:*
    - *An instance showing the end of execution of the Activity Node (where this edge is defined as the outgoing connection) and the beginning of the execution of another Activity (where this edge is defined as the incoming connection).*

V. EXEMPLE OF UML AD MODEL

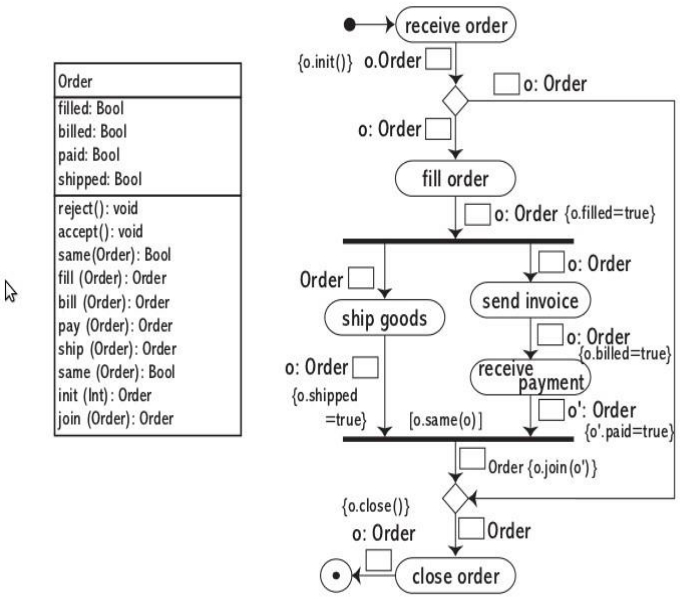

Figure 2. An example of UML AD model([16])

The example of the figure 2 is presented in ([16]). It represent an UML AD model and UML class diagram. The later contain the different action(method) used in the UML AD model. From the categorical theoretic presentation of UML AD in the previous subsection, we can identify the signatures, the sentences and the interpretation of the example 2.

For the example (fig 2) the signatures declares Activity Node names Initial Node, receive order, fill order, ship goods, send invoice receive payment, close payment, Final node, And Split, Or Split, And Join and Or Join. And split denote a subsuming Fork. Or Split denote a Decision Node. And Join denote a Join Nodes. Or Join denote a Decision Node. As for edges, the example declares e1, e2 e3, e4, e5, e6, e7, e8, e9, e10, e11, e12, e13. The sentence presented by the above example is the following closed formulas:
seq(Initial Node,e1,receive order) ∧ seq(receive order,e2,Or Split) ∧

The International Journal of Soft Computing and Software Engineering [JSCSE], Vol. 3, No. 3, Special Issue:
The Proceeding of International Conference on Soft Computing and Software Engineering 2013 [SCSE'13],
San Francisco State University, CA, U.S.A., March 2013
Doi: 10.7321/jscse.v3.n3.30 e-ISSN: 2251-7545

seq(Or Split,e3,Or Join) ∧ seq(Or Split,e4,fill order) ∧ seq(fill order,e5,And Split) ∧ seq(And Split,e6,ship goods) ∧
seq(And Split,e7,send invoice) ∧ seq(ship goods,e8,And Join) ∧
seq(send invoice,e9,receive payment) ∧ seq(receive payment,e10,And Join)∧
seq(And Join,e11,Or Join)∧ seq(Or Join,e12,close payment) ∧
seq(close payment,e13,Final node).

VI. Conclusions

In our paper, we investigated the use of institution theory in a modeling formalism. We are motivated by the fact that we want to borrow the verification of system requirement and UML AD properties to Event-B. In other terms, we aim to verify properties inexpressible in UML AD model with the theorem prover Event-B. The institution of UML AD work as a meta-modelling language for this formalism. In addition, UML AD model conformance with the meta-model (formalism) will be seen as a verification of the syntax correctness in the framework of UML AD institution. The defined syntax for UML AD don't address the whole syntax such it's defined in the standard. As future work, we aim to add more aspects for the UML AD institution. Then, We intend to prove an institution of Event-B and an institution comorphism from UML AD institution to Event-B institution. Thus, the semantic equivalence between source and target model will full preserved.

References

[1] A. Knapp, G. M. Marczynski and A. Zawlocki, A heterogeneous approach to service-oriented systems specification. In *SAC*, pages 2477–2484, 2010.

[2] M. Codescu, F. Horozal, M. Kohlhase, T. Mossakowski, F. Rabe and K. Sojakova, Towards logical frameworks in the heterogeneous tool set hets. In *WADT*, pages 139–159, 2010.

[3] A. Boronat, A. Knapp, J. Meseguer and M. Wirsing, What Is a Multi-modeling Language?. In *WADT*, pages 71–87, 2008.

[4] R. Diaconescu, Grothendieck institutions. *Applied Categorical Structures*, 10, 2002.

[5] R. Diaconescu and K. Futatsugi, Logical foundations of cafeobj. *Theoretical Computer Science*, 285:289–318, 2002.

[6] D. Lucanu, Institution morphisms for relating owl and z. In *The 17th International Conference on Software Engineering and Knowledge Engineering*, 2005.

[7] J. Goguen and G. Rosu, Institution morphisms. *Formal Aspects Computing*, 13(3-5):274–307, 2002.

[8] L. J.Ben Ayed, A. Ben younes and A. Achouri, : Using atom3 for the verification of workflow applications. In *ICSOFT (2)*, pages 32–39, 2010.

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

[10] M. V.Cengarle, Uml 2.0 interactions: Semantics and refinement. Technical report, Institut fr Informatik, Technische Universitt Mnchen, 2004.

[11] M. V.Cengarle and A. Knapp, An institution for uml 2.0 interactions. Technical report, Institut fr Informatik, 2008.

[12] M. V.Cengarle and A. Knapp, An institution for uml 2.0 static structures. Technical report, Institut fr Informatik, 2008.

[13] M. V.Cengarle, A. Knapp and M. Wirsing, A heterogeneous approach to UML semantics. In *Concurrency, Graphs and Models*, pages 383–402, 2008.

[14] R. Diaconescu, *Institution-independent Model Theory*. Birkhuser Basel, 1st edition, 2008.

[15] H. Störrle, Structured nodes in UML 2.0 activities. *Nordic J. of Computing*, 11(3):279–302, 2004.

[16] H. Störrle, Semantics and verification of data flow in uml 2.0 activities. *Electronic Notes Theoretic Computer Science*, 127(4):35–52, 2005.

[17] H. Störrle, J. H.Hausmann and U. Paderborn, Towards a formal semantics of uml 2.0 activities. In *In Proceedings German Software Engineering Conference, volume P-64 of LNI*, pages 117–128, 2005.