

Modeling UML2 activity diagram by using Graph Transformation Systems and Abstract State Machine

Somayeh Azizi

Computer science department

Azad Sama Technical and

vocational Training School,

Islamic University, Arak

Branch

s.azizi2011@gmail.com

Vahid Panahi

v.p1386@gmail.com

Abstract

Graphs and diagrams provide a simple and powerful approach variety of problems that are typical to computer science, for example for activities. In software development visual notation are used for modeling that including activity diagram, class diagram, control flow, graphs and some another diagrams. Models based on these notation can be seen a graph and graph transformation are involved.

So Abstract State Machine (ASM) is a modern computation model. ASM based tools are used to academia and industry, albeit on a modest scale. They allow we to give high-level operational semantics to computer artifacts and to write executable specifications of software and hardware at the desired abstraction level. The token flow semantics of UML2 activity diagrams is formally defined using Abstract State Machines and Graph Transformation System. The state of the art in semantics for UML2 activity diagrams covers three distinct approaches: mapping to Petri-nets, using graph transformation rules, or providing pseudo-code. ASM using pseudo- code and graph transformation system using graph transformation rules for determining semantics. A major goal of this paper is ability to determine the correctness behavior and formal semantics of UML2 activity diagram by Graph Transformation System and Abstract state machine.

Key words: *Graph Transformation, Abstract state machine, Activity Diagram, Semantics, Verification and Validation*

1.Introduction

Recently modeling has been developing and it was a proper method to descript semantic of activities. Selection a proper model is based of modeling, There are some method for modeling, such as:

- Unified modeling language (UML)
- Graph Transformation system (GTS)
- Abstract State Machine (ASM)
- Petri Nets
- Process Algebra
- State Diagrams

Unified modeling language will increase the already substantial collection of notations supported by UML 1.x. UML 2 promises to answer a number of new language constructs that enable a broader range of specifications to be constructed. A key issue in successfully using UML2 is understanding the semantics of the augmented language. For example, what is the meaning of a class diagram in terms of a component diagram, or what is the meaning of a state machine in terms of an activity diagram. These are not easy questions to answer and involve understanding the semantics of each individual construct.

The UML diagrams become a common work product developers use to discuss all phases of software development from requirements analysis, design, implementation and maintenance. The goal here is to model the software system before you build it. UML 2 specification defines two major kinds of UML diagram: structure diagrams and behavior diagrams. Structure diagrams show the static structure of the system and its parts on different abstraction and implementation levels and how they are related to each other. The elements in a structure diagram represent the meaningful concepts of a system, and may include abstract, real world and implementation concepts. Behavior diagrams show the dynamic behavior of the objects in a system, which can be described as a series of changes to the system over time. An activity diagram is used to display the sequence of activities. Activity diagrams show the workflow from a start point to the finish point detailing the many decision paths that exist in the progression of events contained in the activity. They may be used to detail situations where parallel processing may occur in the execution of some activities. Activity diagrams are useful for business modeling where they used to formal description of workflows. They show dynamic state systems and ordinal between activities. Workflow is based tools for modeling, description its formally have advantage such as:

- It express workflows clearly and transparently
- It is based support tools which possible automatic verification and validation. Model Checker is public method for this work
- It is graphical
- It can analyze of model
- It can support verification and validation
- It has formal concepts

Therefore, selecting a proper model and formal method can be determined behavior activity diagrams intelligible.

2. Related Work

There are many research done about definition formal semantics of UML activity diagram by using different formal language.

HAREL defines the state diagram to model activities behavior in STATEMENT structured analysis notation.

WODTKE and WEIKUM uses the STATEMENT state diagrams to model distributed workflow. In this method state diagrams are determined control flow between the activities.

In this way ESHUIS uses of this method to describe behavior of UML1.5 activity diagram. He defines concept of strong fairness that based on, a model should not be indefinite loops. He models the concept in two levels:

- Requirement-level semantics: This level is easy for analysis.
- Implementation-level semantics: This level is hard for analysis but it provides a real vision of systems

GEHREKE defines a concept based Petri nets for activity diagrams. He maps node to place and edge to transition. In this method, transitions determine by tokens as input events and a place provide for each input event, If this place fills tokens, the event will occur. In this method a problem is activation of more one transition by similar event. Therefore it is not enough for modeling.

Vander ALAST uses Colored Petri Nets (CPN). He uses circle as place and rectangle as transitions. States represent by dispatch tokens and at place it shows active state. In describing the concept, there is a global clock which measures current time. Per token has a timestamp which is equal when it can be fired. A single transition will fire immediately when the current time is equal maximum timestamp of input tokens. He

provides as symbolic event but he could not determine concept it, therefore Colored Petri Nets failed for workflow modeling.

Bogor have be used Abstract State Machine (ASM) to describe UML2 activity diagrams. In this method, each state is algebra and *if-then* rule states what will have done by transformation one state to other state. It is based on event, therefore if condition then will be done operations. In this approach concept description are implement level and it will reject model based concept. In UML2, also semantic description the level of Implementation is difficult to analyze and understand.

HAUSMANN have be compared some approach and developed Dynamic Meta Modeling (DMM). He was to descript static concept by meta modeling symbolic and dynamic concept determined with usable rules set. In DMM this useful rules defined as Graph Transformation Rules. We will use it in our approach and it will have helped us. It could not verification and validation by model checker automatically.

Engels had been used DMM for modeling and verification of workflows. For verification, he used to GROOVE tools but GROOVE could not support typed graphs and rule invocation, they change the rules to be verifiable by GROOVE. They check deadlock freeness and action readability properties on the modeled workflows. In contrast to this work, our approach has more flexibility to support user defined properties. Furthermore, event and exception modeling can be supported by this approach. Additionally, the extension defined by HAUSMANN (small/big step rules and rule invocation) could not model directly in existing graph transformation tools, hence it is not so easy for designers to use this approach.

3. Activity Diagram

An activity is operation sequence from start to end the system done and per activity can be transformed on data or process. A data flow diagram shows what kinds of data will be input to and output from the system, where the data will come from and go to, and where the data will be stored. It does not show information about the timing of processes, or information about whether processes will operate in sequence or parallel.

A control flow graph (CFG) in computer science is a representation, using graph notation, of all paths that might be traversed through a program during its execution.

An activity specifies the coordination of executions of subordinate behaviors, using a control and data flow model. The subordinate behaviors coordinated by these models may be initiated because other behaviors in the model finish executing, because objects and data become available, or because events occur external to the flow. The flow of execution is modeled as activity nodes connected by activity edges. A node can be the execution of a subordinate behavior, such as an arithmetic computation, a call to an operation, or manipulation of object contents. Activity nodes also include flow of control constructs, such as synchronization, decision, and concurrency control. Activities may form invocation hierarchies invoking other activities, ultimately resolving to individual actions. In an object-oriented model, activities are usually invoked indirectly as methods bound to operations that are directly invoked.

3.1. Goals of the Activity Diagram

- Model the logic of a use case
- Model procedural logic, algorithms, business process and work flow concentrating on the state of the transitions.
- Model parallel logic.
- Show the activities that make up an action.
- Describe workflow and process modeling.
- A useful diagram especially if you are using UML as a programming language.

3.2. Architecture of activity diagrams

- **Action nodes** operate on control and data values that they receive, and provide control and data to other actions.
- **Control nodes:** They route control and data tokens through the graph. These include constructs for choosing between alternative flows, for proceeding along multiple flows in parallel (forks), and so on.
- **Object nodes:** They hold data tokens temporarily as they wait to move through the graph. Below, the notation for some of the activity nodes to be discussed. Contrary to the names, control nodes coordinate both data flow and control flow in the graph, and object nodes can hold both objects and data.

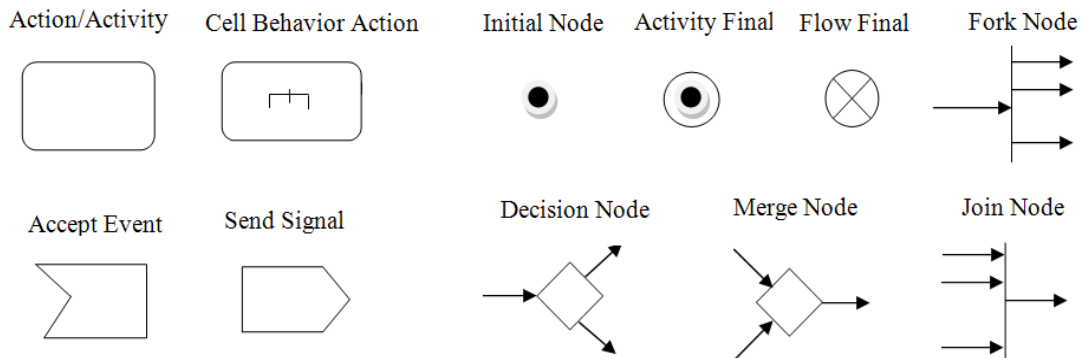


Fig.1. Kind of activity nodes

3.3. Advanced activity model

Unified modeling language (UML2) have be presented some modeling approach for activity diagrams which display complex behavior.

These are Activity Partition, Gerent Exception and Structured Node.

- **Activity Partition:** They have been divided an activity diagram on based subject of operation.
- **Gerent Exception:** They have been determined what must be done if exception condition occur.
- **Structured Node:** They have been set of activities which can execute parallel or serial together.

4. Graph Transformation

Graph transformation, or Graph rewriting, concerns the technique of creating a new graph out of an original graph using some automatic machine. It has numerous applications, ranging from software verification to layout algorithms.

Graph transformations can be used as a computation abstraction. The basic idea is that the state of a computation can be represented as a graph, further steps in that computation can then be represented as transformation rules on that graph. Such rules consist of an original graph, which is to be matched to a sub graph in the complete state, and a replacing graph, which will replace the matched sub graph.

Formally, a graph rewriting system consists of a set of graph rewrite rules of the form $L \rightarrow R$ with L being called pattern graph or Left-Hand Side

(LHS) and R being called replacement graph or Right-Hand Side (RHS). A graph rewrite rule is applied to the host graph by searching for an occurrence of the pattern graph and by replacing the found occurrence by an instance of the replacement graph.

Graphs are an expressive, visual and mathematically precise formalism for modeling of objects (entities) linked by relations, objects are represented by nodes and relations between them by edges. Nodes and edges are commonly typed and attributed. Computations are described in this model by changes in the relations between the entities or by attribute changes of the graph elements. They are encoded in graph rewrite/graph transformation rules and executed by graph rewrite systems/graph transformation tools.

Sometimes graph grammar is used as a synonym for graph rewriting system, especially in the context of formal

languages; the different wording is used to emphasize the goal of enumerating all graphs from some starting graph, i.e. describing a graph language - instead of transforming a given state (host graph) into a new state. The application of a graph transformation rule transforms a graph G , the source graph, into a graph H , the target graph, by looking for an occurrence of L in G and then replacing that occurrence of L with R , resulting in H . The role of the Negative Application Condition (NAC) is that they can still prevent application of the rule when an occurrence of the LHS has been found, namely if there is an occurrence of some $N \in N$ in G that extends the candidate occurrence of L . [3,4].

In fig. 2 this process is shown.

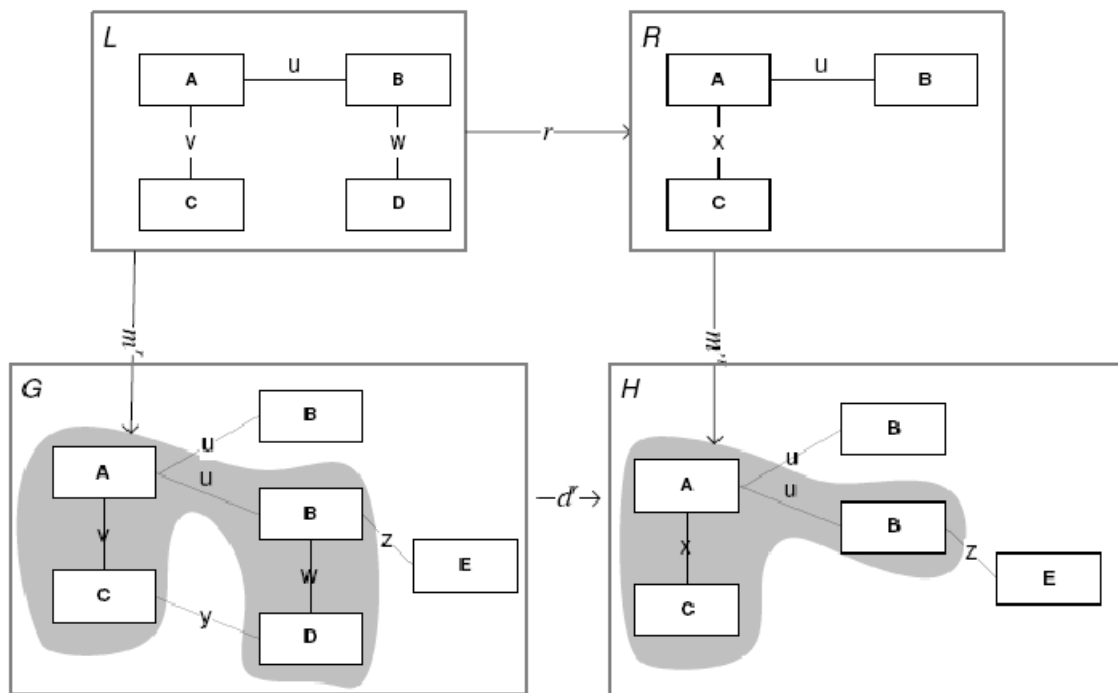


Fig.2. Graph transformation with adaptation L to G and H derived from G [3]

5. Abstract State Machine

The Abstract State Machine (ASM) Project (formerly known as the Evolving Algebras Project) was started by Yuri GUREVICH as an attempt to bridge the gap between formal models of computation and practical specification methods.

The method built around the notion of Abstract State Machine has been proved to be a scientifically well founded and an industrially viable method for the design and analysis of complex systems, which has been applied successfully to programming languages, protocols, embedded systems, architectures, requirements engineering, etc. The analysis covers both verification and validation, using mathematical reasoning (possibly) or experimental simulation (by running the executable models). Their major scientific and industrial applications are surveyed, covering the period from 1990 to 2000. Then an outstanding case study is presented, namely the ASM definition and Analysis (Verification and Validation) of Java and the Java virtual machine. An ASM programming system making ASM models executable by programming the external functions in Haskell. The ASM Gofer introduction includes a demo of the executable versions of the Java/JVM ASM models.

A sequential ASM is defined as a set of transition rules of form: { If condition then updates } which transform first-order structures, where the guard condition, which has to be satisfied for a rule to be applicable, is a variable free first-order formula, and updates is a finite set of function updates of form: $f(t_1, \dots, t_n) := t$. The execution of these rules is understood as updating, in the given state and in the

indicated way, the value of the function f indicated parameters, leaving everything else unchanged. In every state, all the rules which are applicable are simultaneously applied to produce the next state. If desired or useful, declarative features can be built into an ASM by integrity constraints and by assumptions on the state, on the environment, and on the applicability of rules.

For distributed ASMs, the notion of run, which is defined for sequential systems as sequence of computation steps of a single agent, is replaced by the notion of a partial order of moves of finitely many agents, such that the following three conditions are satisfied: (i) Co-finiteness: each move has only finitely many predecessors, (ii) Sequential of single agents, the moves of every agent are linearly ordered, (iii) Coherence: each finite initial segment X corresponds to a state $s(X)$, interpreted as the result of executing all moves in X , which for every maximal element x of X is obtainable by applying move x in states $(X - \{x\})$. The moves of the single agents can be atomic or durative, but for simplicity the preceding definition of distributed runs assumes actions to be atomic.

This definition of distributed ASMs, which are also called multi-agent ASMs, provides a theoretical basis for a coherent global system view for concurrent sequential computations of single agents, each executing its own sequential ASM, at its own pace and with atomic actions applied in its own local states, including input from the environment as monitored functions. The definition guarantees that given any finite initial segment of a distributed run, all linearization of this segment, also called interleaving, yield the same global view of the state resulting from the computation. In other words, if moves of different agents are independent of each other in a given run, they can be scheduled relatively to each other in an arbitrary manner, without influencing the overall view of the state resulting from the run.

The notion of distributed ASM runs comes with no recipe for constructing such runs, which in applications may turn out to be a challenging problem. However, it is this generality of the concept that provides the freedom to design and analysis models for distributed systems without any a priori commitment to special synchronization or communication concepts, an important consequence for the practice of modeling. This includes the abstraction from any particular conditions on the timing of moves of the single agents. The ordering of moves reflects only their causal dependency, which is a before-after-relation without further details on the precise timing of the moves. The ASM is formal and can therefore serve as a foundation for the implementation of tools. Finally, it helps to ensure that the specified behavior meets the intuition of the modeler. Abstract State Machine has a main rule that computation transition and uses from it for determine token flow semantics. The semantics of activity diagram determine based token flow. When token available in initial, object and action nodes then calling transition of rule. if guards evaluation true then token move toward destination nodes. [2]

Activities may describe procedural computation. In this context, they are the methods corresponding to operations on classes. Activities may be applied to organizational modeling for business process engineering and workflow. In this context, events often originate from inside the system, such as the finishing of a task, but also from outside the system, such as a customer call. Activities can also be used for information system modeling to specify system level processes. An activity diagram have be studied two aspect which to consist of Abstract syntax and concrete syntax.

6. Abstract Syntax

Abstract syntax is associated with a concrete syntax that defines the rendering of the notations to the user, as illustrated in figure3. This approach is also followed in the dominant submission to UML2 superstructure [4]. Because every concrete syntax construct has a construct in the abstract syntax, there are as many semantic units to understand as there are concrete constructs. Consequently, comprehending the semantics of UML is non-trivial.

Classes Concrete Syntax	Component Concrete Syntax	Use Case Concrete Syntax	Collaboration Concrete Syntax	State Machine Concrete Syntax	Activity Concrete Syntax	Sequence Concrete Syntax
Classes Abstract Syntax	Component Abstract Syntax	Use Case Abstract Syntax	Collaboration Abstract Syntax	State Machine Abstract Syntax	Activity Abstract Syntax	Sequence Abstract Syntax

Fig.3. The current approach to defining UML [12]

6.1. Structure abstract syntax

Currently UML enables the definition of structural models using class diagrams. UML 2 augments this with the capability of modeling component diagrams. Component diagrams are used to express architectural level definitions of systems and are particularly used within the context of systems engineering. In the following subsections we demonstrate how unification can be leveraged between class diagrams and component diagrams. This unification is enabled by the treatment of components as classes and a component’s ports as attributes of a class. This process shows in figure4.

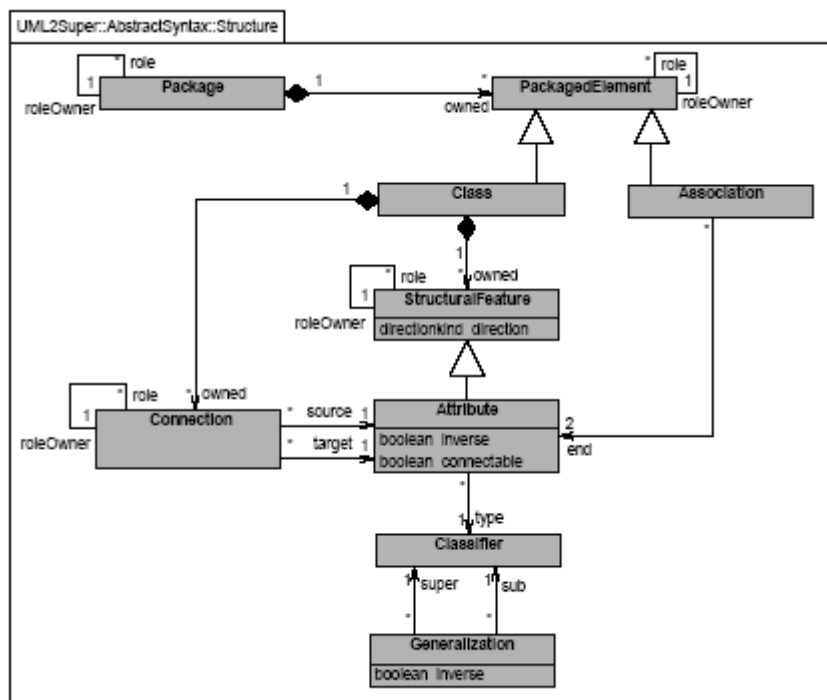


Fig.4. Structure abstract syntax [12]

6.2. Behavior abstract syntax

In definition of behavior, a single abstract syntax model is used as a basis for workflow, state machines, activity diagrams and we deal only with state machines, Graph transformation and activity diagrams.

The abstract syntax of behavior illustrated in figure5 can be broadly considered in two parts. The first part is a hierarchy of actions some of which have sub actions. The second part is relations between actions such as Flow and Transition.

In UML2, state machine states have dynamic properties. For example, a state specifies an entry action, exit

action and continuously execute an action until the exit action is invoked. Thus, it is natural and convenient to treat a state as an action.

State machines and activity diagrams share many concepts the superstructure proposal provides a common abstract syntax model for both these and other behavioral notations. This unification is leveraged by treating state machine states as actions.

The formal definition of the semantics of behavior will have explained at end of this paper.

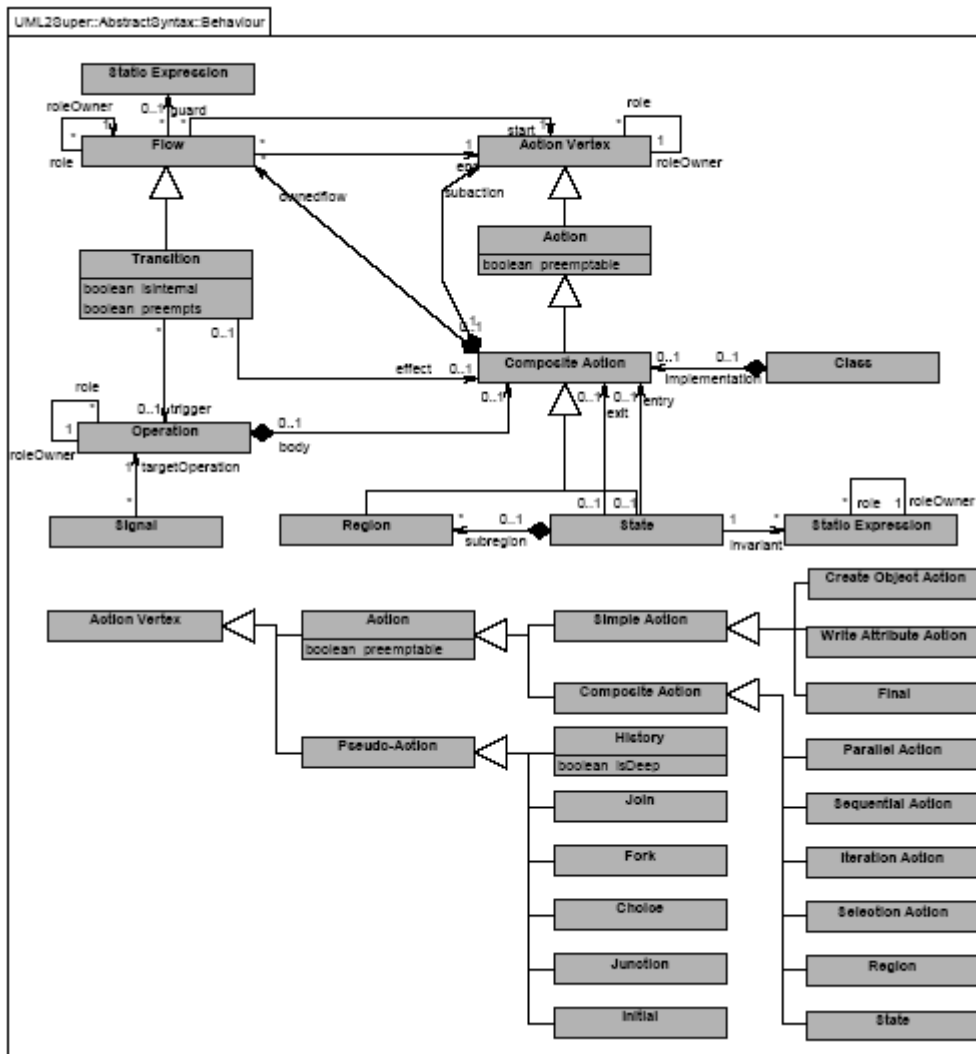


Fig.5. Behavior abstract syntax [12]

7. Workflow Modeling

In this section we will explain our approach for activities modeling, In it any node in activity diagram is equivalent class. It contains three parts, which to consist of class name, properties and operations. class name is as node name, properties are specifications of node such as offer, token,...and operations are functions of node such as Include(), Exclude(),Paths() and Data Tokens().Often these operations determined by pseudo codes of Abstract State Machine. But properties are mixture particular Graph Transformation Systems and Abstract State Machine.

Formal description activities is based token flow, therefore token must be entered within objects and control nodes, it will have reminded in that place awhile. In an activity diagram, it is possible be different paths to

transmit. on the other hand, the token cannot enter all nodes, therefore it will have stopped and created objects which called offers. These will flow in different paths on edges.

Data Tokens () in this method will be available current token of node. If more one of token available, only one of them will respect based first come first serve (FCFS) algorithm. When token have be arriving to other node, must have stopped again and then fixed token at prior control node will have moved and put in current control node. Token offers will have destroyed by input token to it. This process does by Exclude function, again token creates new offers and it will continue till all existing nodes being traversed and the token would be to final node. Token attribute uses to offer token flow in control node. It is type of Boolean, namely if token enter to control node, it will be true else it becomes false.

For all of nodes used Include function, it is contains offers which cooperate with current offers token.

Paths function represents start path from initial node till current position.

In end of section we will have our method by example.

Decision node: pseudo-code of decision node

FORALL I with $1 < I < |\text{accepting Edges}|$ do

T (I):= new (Token Offer)

SEQ

FORALL I with $1 < I < |\text{accepting Edges}|$ do t(i).offered Token := t. offered Token

T(I). Paths() := {p \dashv element At(accepting Edges, i) | p \in t. paths}

T(I). Exclude() := t. exclude \cup {t(j) | $1 \leq j$

$\leq |\text{Accepting Edges}| \wedge I \neq j$ }

T (I).Include() := t. include \cup {t}

Add T (I) to offers (element at (accepting Edges, I))

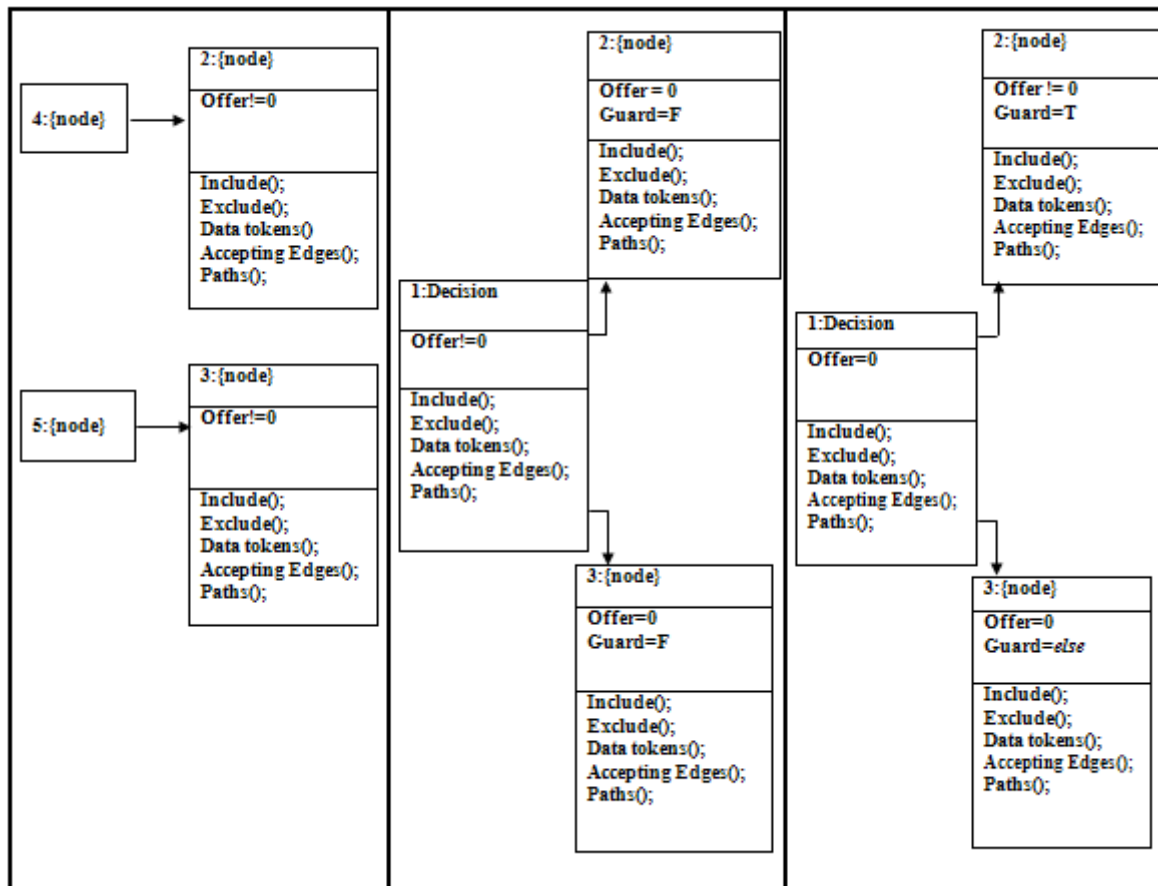


Fig.6.Rules to show semantics of decision node

Fig .6 shows diagrammatically decision node by this method.

Decision node has one input and more one output, token will have flowed initial node, when it have be put in this node, token offers will create. One of them will enter action node if conditions being correct, property for this aim is Guard, therefore output patch will determine. In this node, on the basis of input will have selected only one output path. Accepting edge is containing set of edges by true guard.

In this node rules are:

- The NAC is to palace merge and join node after it, because these nodes have more of one output, moreover it will have traveled one of them. This constraint implement by the Exclude ().
- The LHS shows precondition of rule, namely token offers will have entered at this node.
- The RHS shows which token offers will have come out of this node.

In our approach also we have be implemented semantics of other nodes.

We will have implemented this approach by case study which shows processing of operations in a company.

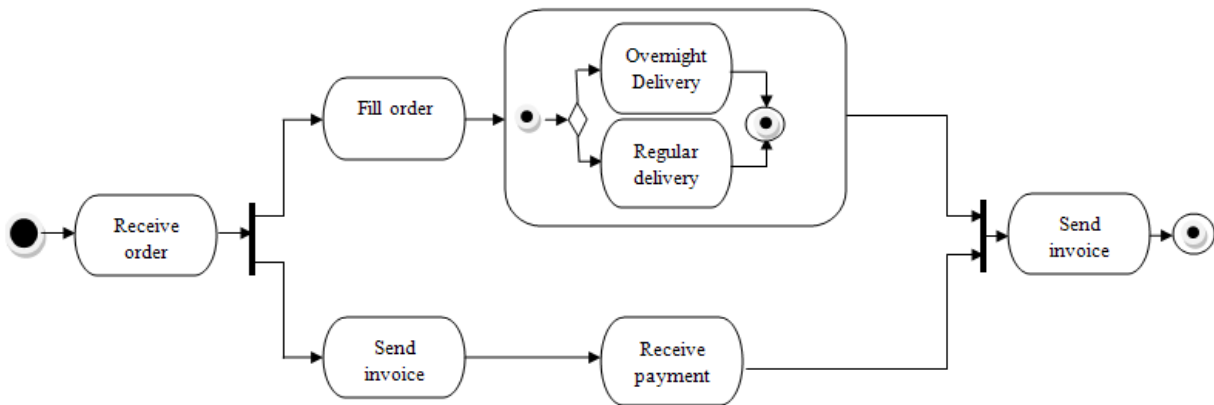


Fig.7. A sample of activity diagram

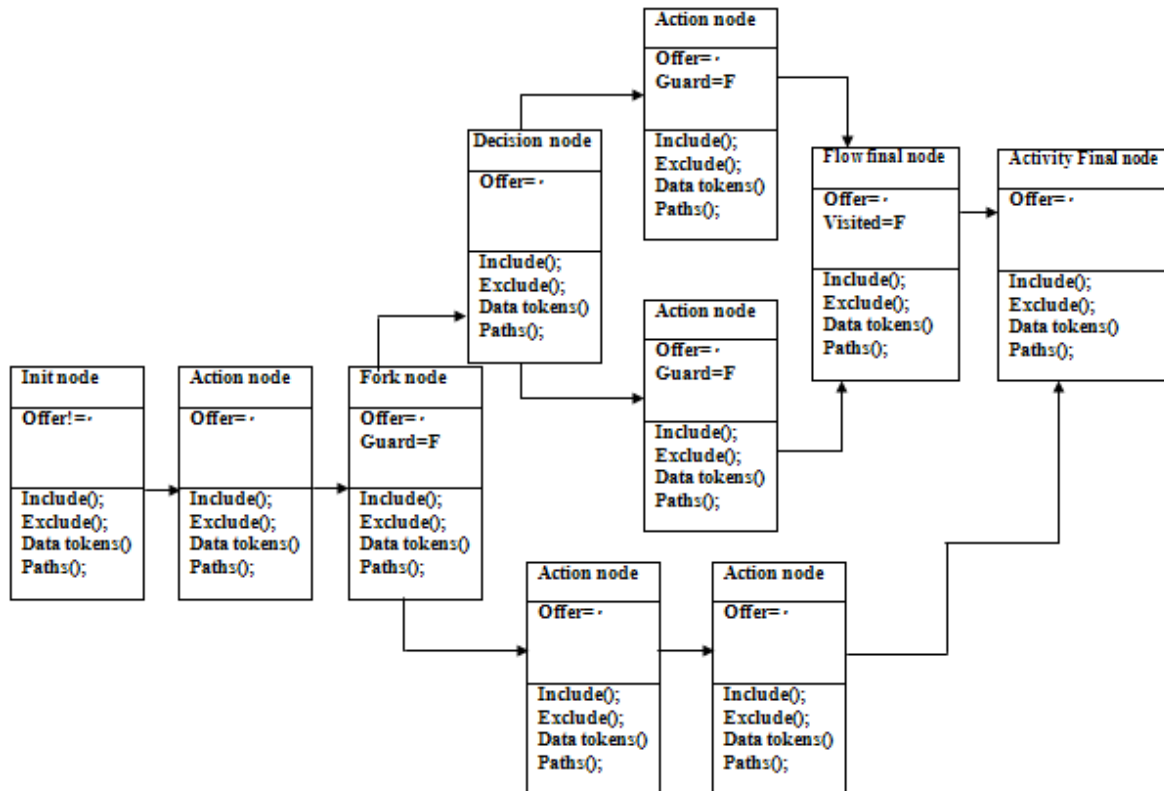


Fig.8.The sample activity diagram in fig3 as host graph with ASM and GTS

Figure8 shows diagrammatically a workflow which we have be modeled by this approach.

5. Verification and validation

One central goal of model based development is to enable analysis of the system, thus ensuring the quality of the system already on the model level. That is, we want to reason about certain properties of the system prior to the construction of the implementation. Examples for such properties are deadlock freedom, timing consistency and limited memory resources. When developing a concurrent object oriented application, deadlock freedom of the interaction is often a major requirement. Timing consistency is of importance for real-time systems.

The use of formal verification methods is essential in the design process of dependable computer controlled systems. The efficiency of applying these formal methods will be highly increased if the underlying mathematical background is hidden from the designer. In such an integrated system effective techniques are needed to transform the system model to different sort of mathematical models supporting the assessment of system characteristics.

For verification of properties, first a suitable formal verification tool (a model checker) has to be chosen capable of verifying the aspects associated to the property. For example, for the property of deadlock freedom, the model checker has to support the aspects of concurrency, communication and interaction of processes. After identification of such a verification tool, conditions need to be established for the property. These conditions are specified within the specification language understood.

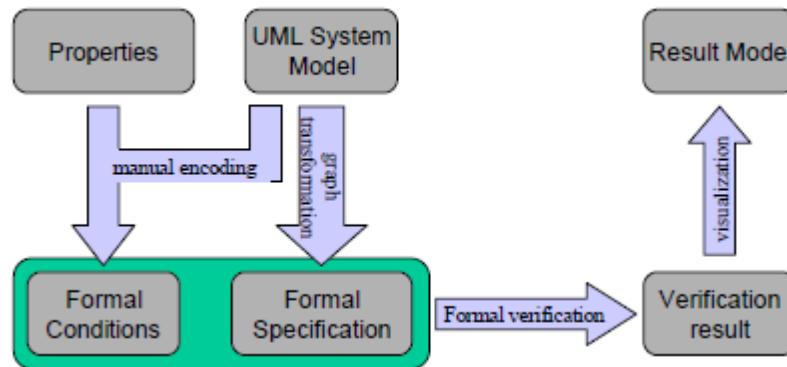


Fig.9. Steps verification of properties [17]

For validation of properties, tests check conformance of the system under test to a specification by simulating the model or testing an implementation derived from the model. Tests must be designed as a sufficient representative of the checked property of the system under test. As a consequence, software testing relies on the execution of code using different test inputs for one property.

For testing such a specific property, a test case specifies the pre-test state of the implementation under test, the test inputs and the expected results. A test suite is a collection of test cases, typically related by a common goal [17].

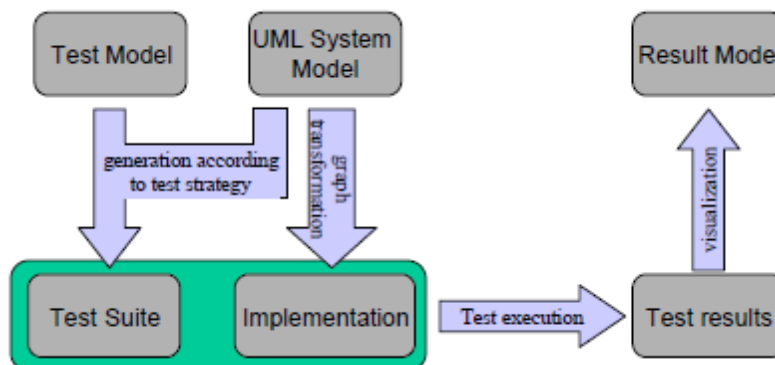


Fig.10. Steps validation of properties [17]

For verification and validation of activities must be designed an approach. Theory and application of visual languages is also based on the strong paradigm of graph transformation. Therefore for analyze designed activities , graph transformation system (type graph, host graph, rules) and properties will have get as input of verification approach. Graph transformation system must be designed in AGG (A Graph Transformation Environment for Modeling and Validation of Software), Also properties define by special rules. If designers are expert in graph transformation, they can directly to model workflows by using graph transformation system. In this approach designers do not need to learn of formal method. In other case, they can model workflows by UML2 activity diagram, then by transformer, designed activities in UML transformed to graph transformation.

In this approach verification will have done automatically and designers do not need to define rules of verification.

VIATRA (Visual Automated Transformations) is a model transformation framework developed mainly for the formal dependability analysis of UML models. In VIATRA, Meta modeling is conceived specially, the instantiation is based on mathematical formalisms and called visual precise meta modeling. The attribute transformation is performed by abstract state machine statements, and there is built-in support for attributes of basic Java types. The model constraints can be expressed by graph patterns with arbitrary levels of negation. The rule constraints are also specified by graph patterns. VIATRA uses abstract state machines to define the

control flow of the system.[1,3]

6. Conclusions

In this paper, we proposed a formal approach based composition Graph Transformation Systems (GTS) and Abstract State Machine (ASM). This approach has be determined behavior of UML2 activity diagrams based token flow. Rules of ASM which have defined with *pseudo-code*, display by fundamental element of Graph Transformation System (LHS, RHS, NAC).

6.1. Results

Table 1

Some properties have be compared together such as control flow, date flow, exceptions, structured nodes, rigor and result of them have almost Summarized.

Authors, references	UML version	Semantic domain	Control flow	Data flow	exceptions	Structured nodes	rigor
BOGER	1.x	ASM ¹	² NWF	–	–	–	medium
ESHUIS & WIERINGA	1.x	algorithm	³ WF , NWF	–	–	–	high
GEHREK	1.0	PN ⁴	WF , NWF	–	–	–	medium
RODRIGUES	1.x	FSP ⁵	WF	–	–	–	low
STORRLE	2.0	CPN ⁶	WF, NWF	√	–	–	high
VAHID RAFEH	2.0	GTS ⁷	WF	√	√	√	high
SARSTEDT & GUTTMANN	2.0	ASM	WF	√	√	√	high
SOMAYEH AZIZI	2.0	GTS & ASM	WF	√	√	√	high

References

- [1] Rafeh, V., Rahmani, A. T.: Formal Analysis of Workflows Using UML 2.0 Activities and Graph Transformation Systems, ICTAC 2008, LNCS 5160, pp.305--318. (2008).
 [2] Sarstedt, S., Guttmann, W. : An ASM Semantics of Token Flow in UML2.0 Activity Diagrams,(2008).

¹ Abstract State Machines

² Non Well Form

³ Well Form

⁴ Simple P/T-Petri-Nets

⁵ Finite State Processes

⁶ Colored Petri-Nets

⁷ Graph Transformation Systems

- [3] Rafeh, V., Rafeh, R., Azizi, S., Miralvand, M.R.Z.: Verification and Validation of Activity Diagrams Using Graph Transformation, ICCTD '09, Computer Technology and Development, (2009).
- [4] Lengyel, L., Levendovszky, T., Mezei, G., Charaf, H.: Model Transformation with a Visual Control Flow Language, International Journal of Computer Science Vol. 1, Number1,(2005).
- [5] Perez, P.P.: Matrix Graph Grammars: An Algebraic Approach to Graph Dynamics, VDM Verlag, ISBN 978-3639212556, (2009).
- [6] Engels, G., Soltenborn, C., Wehrheim, H.: Analysis UML Activities Using Dynamic Meta Modeling, FMOODS 2007. LNCS, vol. 4468, pp. 76–90. Springer, Heidelberg (2007)
- [7] Bock, C.: UML 2 Activity and Action Models, in Journal of Object Technology, vol. 2, pp. 43--53, July-August (2003).
- [8] Kastenbergh, H., Kleppe, A., Rensink, A.: Defining Object- oriented Execution Semantics Using Graph Transformations, Computer Science Enschede, The Netherlands, LNCS 4037, pp 186--201.(2006).
- [9] Störrle, H.: Semantics and Verification of Data Flow in UML 2.0 Activities, Electronic Notes in Theoretical Computer Science, pp. 35--55.(2004).
- [10] Broy, M., Crane, M. L., Dingel, J., Hartman, A., Rumpe, B., Selic, B.: Formal Semantics for UML, MODELS 2006 Workshops, LNCS 4364, pp. 318--323. (2007).
- [11] Russell, van der Aalst, W., ter Hofstede, H. M., Wohed, P.: On the suitability of UML 2.0 activity diagrams for business process modeling, APCCM 2006, Vol. 53, pp. 95--104.(2006).
- [12] 2U. 2nd revised submission to OMG RFP unified modeling language superstructure version 2.0. <http://www.2uworks.org/uml2submission/.ad/02-12-23>.
- [13] 2U. 3rd revised submission to OMG RFP unified modeling language infrastructure version 2.0. <http://www.2uworks.org/uml2submission/.ad/03-01-08>.
- [14] Object Management Group, Unified modeling language specification version 1.4. <http://www.omg.org.ad/01-09-67>.
- [15] U2 Partners. 2nd revised submission to OMG RFP unified modeling language superstructure version 2.0. <http://www.u2-partners.org/artifacts.htm.ad/03-01-01>.
- [16] Rumbaugh, J., Jacobson, I., Booch, G.: The Unified Modeling Language Reference Manual, Addison Wesley, first edition, (1999).
- [17] Robert, V.: Testing Object-Oriented Systems: Models, Patterns, and Tools, Object Technology Series. Addison Wesley, (1999).
- [18] Object Management Group: UML 2.0 Superstructure Specification, October (2004), <http://www.omg.org/cgi-bin/doc?ptc/2004-10-02>.
- [19] Bock, C.:UML 2 Activity and Action Models, in Journal of Object Technology, vol. 2, pp. 43--53, July-August (2003).
- [20] Bhattacharjee, A. K., Shyamasundar, R. K.: Activity Diagrams: A Formal Framework to Model Business Processes and Code Generation, in Journal of Object Technology, vol. 8, no.1, pp. 189--220, January-February (2009).
- [21] Meena, H.K., Saha, I., Mondal, K., Prabhakar, T.V.: An Approach to Workflow Modeling and Analysis, Dep of Computer Science & Engineering, IIT Kanpur, India, pp. 85--89.(2005).
- [22] Kastenbergh, H., Kleppe, A., Rensink, A.: Defining Object- oriented Execution Semantics Using Graph Transformations, Computer Science Enschede, The Netherlands, LNCS 4037, pp. 186--201.(2006).
- [23] Störrle, H., Hausmann, J.H.: Towards a Formal Semantics of UML 2.0 Activities, Software Engineering. LNI., GI, vol. 64, pp. 117--128.(2005).
- [24] Soltenborn, C.: Analysis of UML Workflow diagrams with Dynamic Meta Modeling techniques, Master's Thesis, University of Paderborn, Germany (2006).
- [25] Baresi, L., Ehrig, K., Heckel, R.: Verification of Model Transformations: A Case Study with BPEL, TGC 2006, pp. 183--199.(2006).
- [26] Andries, M., Engels, G., Habel, A., Homann, B., Kreowski, H. J., Kuske, S., Plump, D., Schurr, A., Taentzery, G.: Graph Transformation for Specification and Programming, Elsevier Science of Computer, pp. 1--54. (1999).

- [27] Broy, M., Crane, M. L., Dingel, J., Hartman, A., Rumpe, B., Selic, B.: Formal Semantics for UML, MoDELS 2006 Workshops, LNCS 4364, pp. 318--323.(2007).
- [28] Vitolins, V., Kalnins, A.: Semantics of UML 2.0 Activity Diagram for Business Modeling by Means of Virtual Machine, Proceedings Ninth IEEE International EDOC Enterprise Computing Conference, IEEE, pp. 181--192.(2005).
- [29] Eshuis, R. Wieringa, H.: A Formal Semantics for UML Activity Diagrams Formalizing Workflow Models, CTIT technical reports, pp. 1--44.(2001).
- [30] Eshuis, R., Wieringa, R.: Comparing Petri Net and Activity Diagram Variants for Workflow Modeling– A Quest for Reactive Petri Nets, Lecture Notes in Computer Science 2472, pp. 321--351.(2003).
- [31] Eshuis, H.: Semantics and Verification of UML Activity Diagrams For Workflow Modeling, Ph.D. Thesis, University of Twente, Netherlands (2005).
- [32] Russell, N., Van der Aalst, W. P., Ter Hofstede, H. M., Wohed, P.: On the suitability of UML 2.0 activity diagrams for business process modeling, APCCM 2006, Vol. 53, pp 95—104.(2006).
- [33] Karsai, G., Agrawa, A., Shi, F., Sprinkle, J.: On the Use of Graph Transformation in the Formal Specification of Model Interpreters, Journal of Universal Computer Science, vol. 9, no. 11, pp. 1296--1321, (2003).
- [34] Störrle, H.: Structured Nodes in UML 2.0 Activities, Nord. J. Compute. 11(3), pp. 279--302.(2004).
- [35] Bruni, R., Melgratti, H.C.: Dynamic Graph Transformation Systems, ICGT 2006, pp. 230--244.(2006).
- [36] Börger, E., Cavarra, A., Riccobene, E.: An ASM Semantics for UML Activity Diagrams, AMAST 2000, LNCS, vol. 1816, pp. 287--302.(2000).
- [37] Engels, G., Kuster, J.M., Heckel, R., Lohmann, M.: Model-Based Verification and Validation of Properties, Electronic Notes in Theoretical Computer Science 82, vol. 82, no. 7, pp.1--18.(2003).
- [38]Chapurlat, V., Kamsu_Foguem, B., Prunet, F.: Enterprise Model Verification And Validation, Journal reference: Annual Review in Control, ISSN 1367-5788, vol. 27, no. 2, pp. 185--197, November (2003).
- [39] Sundari, M. H., Sen, A. , Bagchi, A. K.: Workflows as UML Activity Diagrams: Analytical Method for Control Flow Verification, Working Paper Series. WPS, no. 628, September (2008).