International Journal on New Computer Architectures and Their Applications (IJNCAA) 2(1): 236-245
The Society of Digital Information and Wireless Communications, 2012 (ISSN: 2220-9085)

A Meta-Modeling and Graph Grammar Approach for Modeling and analysis of e-entrepreneurship systems

Mouna Bouarioua, Allaoua Chaoui and Raida Elmansouri
MISC Laboratory, Department of Computer Science, University Mentouri Constantine, Algeria
mouna.bouarioua@yahoo.com, a_chaoui2001@yahoo.com, raidaelmansouri@yahoo.fr

ABSTRACT

UML provides standard abstractions to simplify the documentation, understanding, and maintenance of object oriented complex software systems. UML sequence diagrams are used to describe the behaviors of systems. However, UML is a semi-formal language that lacks tools for analysis and validation of user requirements. In other hand, Petri Nets models are tools for the validation and performance analysis of distributed systems. In this paper we present an approach for transforming UML sequence diagrams to labeled generalized stochastic Petri nets. By this transformation we aim to bridge the gap between the semi-formal notation (UML) and the formal notations (Petri nets) for analysis and validation purposes. UML is used for modeling and Petri nets are used for analysis. The approach is based on the combined use of Meta-Modeling and graph grammars. Since the input and the output models are graphs, we use Java based graph transformation to perform this process automatically. The approach is illustrated through an example from e-entrepreneurship domain.

KEYWORDS

UML, Sequence Diagrams, Generalized Petri Nets, LGSPN, graph grammars, Java, e-entrepreneurship.

1 INTRODUCTION

Today's computer systems are confronted to the fast growing of their complexity.

UML provides standard abstractions to *simplify* the documentation, understanding, and maintenance of complex software systems. For the implementation part, Model Driven Architecture (MDA) can be used to transform the UML models to the target environment. The Model Driven Architecture MDA is a software framework defined by the OMG (OMG) which is built around the main artifacts: the *model* and *model transformations*. Models are an abstraction of the reality where only relevant properties are retained. Models can be seen at different levels of details. For example, models with high level of abstraction are suitable for e-entrepreneurship architecture where only components and their relation are represented without any technical consideration. Models with low level of abstraction are suitable to represent e-entrepreneurship communication protocols and synchronization. One of the goals of MDA is the automatic generation of code; it has to make a progressive transformation on PIM (Platform Independent Models) to PSM (Platform Specific Models). Formal verification methods [28] are the most used

techniques to deal with concurrent systems questions since they are able to describe systems behaviors rigorously.

The unified modeling language UML [10] [23] is recognized as a standardized object-oriented language for specifying and visualizing software systems. It contains a large number of graphical notations called diagrams which allow capturing different structural and behavioral aspects of systems. UML sequence diagrams [4] are an interaction diagrams (subset of behavior diagrams) that describes the communication between processes taking into account some timing constraints. These diagrams describe how and in which order objects operate, and the relations between them in a visual and intuitive manner. Sequence diagrams, as their name indicates; order messages sequentially according to their occurrence in time. They are used to explore the logic of methods and services and also for model scenarios.

Our goal being performance analysis and validation, UML lacks a precise formal semantic which hinders the formal evaluation and verification [7] of system structure, and still needs more semantic bases for its notation.

Petri nets [2], Predicate/Transition nets [12] and in particular, labeled generalized stochastic Petri nets (LGSPN) [2] are formal and graphical methods [28] for modeling complex systems in distributed environments. LGSPN capture many aspects of systems. The most important are concurrency, synchronization and conflicts.

To reach our goal, UML sequence diagrams are projected into LGSPN models for managing systems performance, and the result of formal analysis can be back-annotated to the UML models.

To manage the complexity of the system, we need to raise the level of abstraction in the development process by building a tool that generates our needed formalisms. This step is very important since it allows modeling models themselves. So in this case meta-models [24] [27] are the definition of models. The mapping between two models also called models transformation [24] must be semantically well-defined in a meta-model level, the series of formal changes applied to source sub-models is called grammar [13].

In this paper that represents an extended version of our conference paper [7], we propose an approach and a tool for transforming sequence diagram models to their equivalent Labeled Generalized Stochastic Petri Net models using Java. The obtained models will be exploited by verification tools. UML is used modeling and Petri nets for analysis.

We have defined two meta-models for UML sequence diagrams and LGSPN as Java classes, and for each instantiation, the compiler generates a model according to the selected meta-model class. The process of translation is performed by a set of rules that constitute the graph grammar.

The remainder of this paper is organized as follow: Section 2 outlines the major related work and differences between their multiple approaches. In section 3 we recall some basic concepts about UML sequence diagrams, Labeled Generalized Stochastic Petri Nets, graph transformation, and e-entrepreneurship domain. In section 4 we describe our approach of transforming sequence diagrams to Labeled Generalized Stochastic Petri Nets, it consists on

proposing two meta-models associated respectively to the sequence diagram model and LGSPN model and a grammar that deals with the transformation. The meta-models and the grammar are implemented in Java. In section 5, we illustrate our approach through an example; we give a sequence diagram and execute the tool. The final section concludes the paper and gives some perspectives of the work.

2 RELATED WORK

This paper deals with model transformation and especially graph transformations for analysis purposes. Many related works have been proposed in the literature. They used meta-modeling concepts and visual tools like Generic Modeling environment (GME) [17] ATOM^3 [3], and other tools from the Eclipse Generative Modeling tools (GMT) project such as Eclipse Modeling Framework (EMF) [9], Graphical Editing Framework (GEF) [15] and Graphical Modeling Framework (GMF) [16]. In most of this, model transformations have to be described textually. In [8] the authors performed a translation of Act-One to Miranda. There are also similar tools which manipulate models by means of graph grammars, such as PROGRES [24], GreAT [14], FUJABA [11], and AGG [1]. The translation is performed by means of meta-modeling concepts for allowing the reuse of formalisms working on other transformations. However, none of these has its own meta-modeling layer. In [5] a similar work to our paper has been developed for obtaining analyzable Petri Net models from UML sequence diagrams and statecharts. The translation into Petri Nets is based on the abstract syntax of UML collaborations and of the state machines packages. In [19] [22], they treat the transformation between UML statecharts and collaboration diagrams to Colored Petri Net models using ATOM^3 since the approach is based on graph transformation and the input and output models are both graphs. It produces graphical models that facilitate early detection of errors. This work was a step in a project that is exploring means to exploit the INA [19] analyzer tool.

3 BACKGROUND

Our objective is to propose an automatic generation of Labeled Generalized Stochastic Petri Nets from Sequence Diagrams using graph transformation. The abstract syntax of formalism is described by means of a meta-model. The transformation process is performed by a graph grammar that takes the source model (Sequence Diagram) as an input, execute the rules of the grammar, and generate the target model (Labeled Generalized Stochastic Petri Nets) as output.
In the following we recall some basic notions about sequence diagrams, Labeled Generalized Stochastic Petri Nets and graph transformations.

3.1 Sequence Diagrams

UML [10][23] is a visual language for the object oriented analysis and design of systems defined by the Object Management Group (OMG). It is based on diagrams. UML 2.3 is the last version of UML released in may 2010 and it provides thirteen diagrams for specifying and modeling systems from different views as use cases, logical view, implementation, deployment and processes, even if it doesn't justify the strategy of the choices. The behavioral

aspect of a system is defined by how objects communicate together and exchange messages. UML propose multiple views to show interaction between objects, one of them is the sequence diagram [4]. The main role of those diagrams is to illustrate systems use cases and show the temporal execution of communication. A sequence diagram is composed of a set of objects followed by lines that describe their lifetimes in a vertical dimension. Messages between objects sent sequentially are represented as arrows from the sender objects to the receiver ones. Messages may be synchronous or asynchronous. Every message is related to an action performed by its sender. When the message is asynchronous, the termination of the action is validated automatically when the message is sent. But when the message is synchronous, this means that the process has been interrupted and the action will terminate after the resumption of the sender. Messages can be delayed or not.

To illustrate the principle of sequence diagrams, let's consider the example of 4 objects and 4 messages in figure 1.

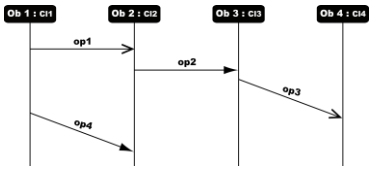

Fig. 1. Sequence diagram

We have 4 objects Ob1, Ob2, Ob3, Ob4. Every object is an instance from the corresponding class Cl1, Cl2, Cl3, Cl4 respectively. To each object frame, an output vertical line describe the lifetime of the object. Messages are call for operations and in the figure they are ordered in time from the top to the bottom op1, op2, op3 and op4. op1 is considered older than op2, and op2 older than op3 and so on.

Every message in the figure has a sender and a receiver, and every message is synchronous or not, delayed or not. Message op1 comes from Ob1:Cl1 (sender) to Ob2:Cl2 (receiver), this message is asynchronous (stick arrowhead) and not delayed. Also op2 isn't delayed but synchronous (solid arrowhead). Op3 is asynchronous but delayed. The last message op4 is synchronous and delayed.

The figure 1 shows the interaction about 4 objects exchanging different messages at different fractions of time, and also giving information on how these messages are send and received, times and delays everything in a graphical manner. This easiness and intuitive aspect is the main power of the unified modeling language (UML).

3.2 Labeled Generalized Stochastic Petri Nets

Petri Nets [2] are mathematical models for the formal description of complex systems. A Petri net is a graph, a set of places and transitions linked by directed arcs. The resulting paradigm from the introduction of temporal concept into classical PN is a sub model called stochastic Petri nets (SPN) [2]. When stochastic timing is mixed with deterministic null delays, we obtain a Generalized stochastic Petri net (GSPN) models where each transition is labeled (LGSPN) [2]. A place in an LGSPN graph represents a system state, a situation or a condition. A transition in an LGSPN graph represent actions that when done, change the state of the system (the transition fires).

Arcs specify the relation between states and actions, they link places to

transitions and vice-versa. Tokens are markers that reside in places used to specify the LGSPN states and different configurations. The place represents a condition; it contains only one token as a Boolean indicator. In addition, when it represents a situation, it contains as tokens a requisite to express the state. Formally, a GSPN model is defined as follow:

LGSPN = (P, T, M_π, I, O, H, W, PAR, P RED, M P)

where :

M_π = (P, T, π, I, O, H, PAR, P RED, M P) is a PN model

P: a set of places.

T: a set of transitions, T∩ P = Φ;

I, O, H: the input, output and inhibition functions, respectively.

P AR: a set of parameters.

PRED: a set of predicates.

M P : parametric initial marking .

W : T→ I is a function defined on the set of transitions that are timed or immediate.

To illustrate the principle of an LGSPN models, let's consider this example (Adapted from [2]).

The LGSPN bellow illustrates timed and immediate transitions, execution of parallel processes, synchronization and free choice conflict.

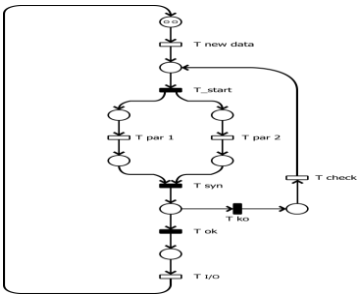

Fig. 2. LGSPN Example

There is 9 transitions: Tnewdata, Tpar1, Tpar2, TI/O, Tcheck are timed and tstart , tsyn , tOK, tKO are immediate.

Every state in this example has a transition in output, and every transition is followed by at least one state.

Tnew_data is the first transition after the initial state that contains two tokens. After validating data, the timed transition Tnewdata fires and the system is automatically switched to a new state. After that, a parallel process starts when the immediate transition T_starts firing. When the parallel processes (Tpar1 and Tpar2) complete; synchronization takes effect at T syn transition. Transitions Tok and Tko describe a free choice conflict case depending on the distribution of weights and tokens in places.

3.3 Graph transformation

The transformation between models is a process that converts a model to another model. This task requires a set of rules that define how the source model has to be analyzed and transformed into other elements of the target model. The transformation engine takes the source model in input; execute the rules of transformation and finally generate the target model in output. Graph grammars [26] are used for model transformation. They are composed of production rules; each having graph in their left and right sides. Rules are compared with an input graph called host graph. If a matching is found between the left hand side of a rule and a subgraph in the host graph, then the rule can be applied and the matching subgraph of the host graph is replaced by the right hand side of the rule. Furthermore, rules may also have a condition that must be satisfied in order for the rule to be applied, as well as actions to be performed when the rule is

executed. A rewriting system iteratively applies matching rules in the grammar to the host graph, until no more rules are applicable. In this work, we use Java.

3.4 E-entrepreneurship

Entrepreneurship is the process of creating something new and assuming the risks and rewards. E-*Entrepreneurshp* is the process of creating owner business activity on Internet in some area. As examples of *e-entrepreneurship* we can cite: Facebook.com, Google.com, eBay.com, yahoo.com, amazon.com, etc … For more details see [18].

4 OUR APPROACH

In order to perform the transformation of sequence diagrams to LSGSPN equivalent models, we propose two meta-models associated respectively to the SD model and the LGSPN model. We note here that the meta-models are described using UML class diagrams. Then we propose a grammar for the transformation. Meta-models and grammar are implemented in Java since the main purpose of UML is to create object oriented models.

4.1 SD meta-model

A sequence diagram consists of objects and messages in time. Our meta-model of SD is composed mainly of three classes (Sequence Diagram, Object, Message).

SequenceDiagram: this class associates objects to messages and build the final model.

Object: this class represents the SD objects that messages send or receive. Every object has a name.

Message: it represents the messages between objects; every message is labeled by an operation, its order in time and Boolean attributes to test if it is asynchronous or delayed.

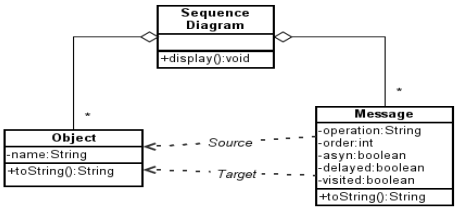

Fig. 3. SD Meta-Model

4.2 LGSPN meta-model

In this section, we propose a meta-model for LGSPN, it consists of states and transitions. So our meta-model of LGSPN is composed mainly of three classes (LGSPN, place, transitions).

Place: this class describes the places of the LGSPN, every place has a name.

Transition: is a superclass of two classes: transition_i for immediate transitions and transition_d for delayed transitions. Every transition is labeled by an action.

LGSPN: it builds the final LGSPN model from places and transitions using arcs.

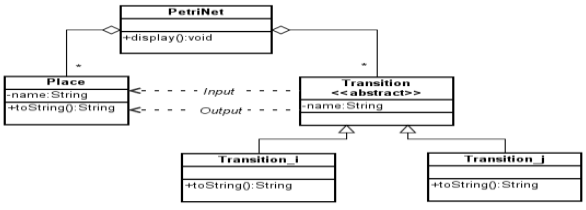

Fig. 4. LGSPN Meta-Model

4.3 Our graph grammar

We have proposed and implemented a graph grammar containing contains 4 rules. Each rule represents one kind of messages (See figure 5).

The *first rule* concerns the asynchronous messages (Asynchronous=True) with no-delay (Delayed=False). When this

rule is applied on this kind of messages, it generates an LGSPN sub diagram of two states and one immediate transition.

The second rule concerns the synchronous messages (Asynchronous=False) with no-delay (Delayed=False). When this rule is applied, it generates two immediate transitions (Start_op and End_op) for the start and the end of the synchronous message.

The third rule concerns the asynchronous messages (Asynchronous=True) with delay (Delayed=True). When applied, this rule processes to the generation of two different types of transitions. The first one for the operation sent with the message and it is immediate. The second one is a timed transition for the exception.

The fourth rule is for synchronous and delayed messages (Asynchronous=False and Delayed=True). This rule creates three transitions. Two immediate transitions for the start and the end of the message's operation respectively, and the other timed transition is for the exception.

We have associated to each rule a priority. The execution of the grammar starts from the left top message of the life-lines of the objects until the last message in the bottom.

In this grammar we apply only one rule on each message.

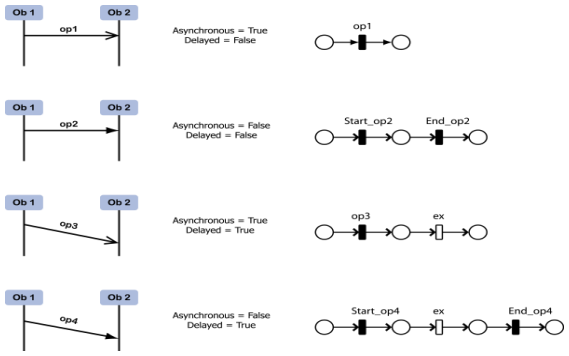

Fig. 5. Graph Grammar rules

5 EXAMPLE

We have tested our approach on several examples. We illustrate it here using the sequence diagram of figure 6 describing the process of an order of a commercial advertisement from a web client and Facebook as an e-entrepreneurship example.

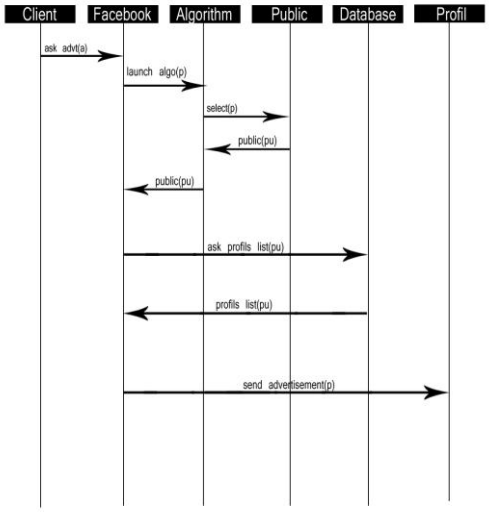

Fig. 6. Sequence diagram of the example

The application of our approach consists of the following steps:

a) We have first described the sequence diagram of figure 6 in java as shown in figure 7.

```
Sequence Diagram :
==================
Objects :
=========
:Client
:Facebook
:Algo
:Public
:Database
:Profil
Messages :
==========
1 :Client---ask_adv(a)-->:Facebook
2 :Facebook---launch_algo(a)-->:Algo
3 :Algo---Select(pu)-->:Public
4 :Public---Selection(pu)-->:Algo
5 :Algo---Public(pu)-->:Facebook
6 :Facebook---Ask_profils_list()-->:Database
7 :Database---Profils_list()-->:Facebook
8 :Facebook---Send_adv()-->:Profil
******************
Running the Grammar
******************
Rule2is applicable
Rule2is applicable
Rule2is applicable
Rule2is applicable
Rule2is applicable
Rule2is applicable
Rule2is applicable
Rule2is applicable
```

Fig. 7. Sequence diagram of the example in Java and the running of the grammar

b) Then, we have applied our grammar (see the bottom part of figure 7: Running the grammar) on the SD of figure 6 (top part: and obtained the LGSPN of figure 8.

```
Petri Net :
***********
Places :
********
(0)
(1)
(2)
(3)
(4)
(5)
(6)
(7)
(8)
(9)
(10)
(11)
(12)
(13)
(14)
(15)
(16)
Transitions :
*************
(0)-->|startask_adv(a)|-->(1)
(1)-->|endask_adv(a)|-->(2)
(2)-->|startlaunch_algo(a)|-->(3)
(3)-->|endlaunch_algo(a)|-->(4)
(4)-->|startSelect(pu)|-->(5)
(5)-->|endSelect(pu)|-->(6)
(6)-->|startSelection(pu)|-->(7)
(7)-->|endSelection(pu)|-->(8)
(8)-->|startPublic(pu)|-->(9)
(9)-->|endPublic(pu)|-->(10)
(10)-->|startAsk_profils_list()|-->(11)
(11)-->|endAsk_profils_list()|-->(12)
(12)-->|startProfils_list()|-->(13)
(13)-->|endProfils_list()|-->(14)
(14)-->|startSend_adv()|-->(15)
(15)-->|endSend_adv()|-->(16)
```

Fig. 8. The obtained LGSPN equivalent to the SD of figure 6

Now, we can perform many analysis and validation of user requirements using LGSPN tools. We can see for example if there is a deadlock or not. We need just to describe the LGSPN of figure 8 in the LGSPN tool syntax and then use the tool.

6 CONCLUSION

In this paper, we have proposed and approach and a tool for transforming sequence diagrams models to labeled generalized stochastic Petri net models using graph grammar since the input and output models are graphs. To achieve this transformation, we have proposed two meta-models: one for the input model (UML sequence diagram) and the other for the output model (LGSPN model). Then based on these two meta-models, we have proposed a graph grammar that deals with the transformation process implemented in java. We have used java as a programming language for implementing our meta-models and grammar instead of all other tools because java is simple, object-oriented, interpreted, robust, secure, architecture neutral, portable, multithreaded and dynamic. This design we made allows some flexibility when readjustment or adding of other classes both in meta-models or grammar are needed. We have illustrated our approach through an example from the e-entrepreneurship domain. This work is concerned only with the transformation of SD models to LGSPN models.

In a future work we plan to integrate the analysis step using an LGSPN Petri net tool. We plan also to implement this transformation using graph transformation visual tools like AGG and transform other UML diagrams. We intend to use the well known reduction techniques on the obtained models before performing the analysis in order to optimize the models, and then use them with INA analyzer. Another way of implementing the transformation is to use the standard exchange format for diagrams of UML.

7 REFERENCES

1. AGG home page, http://tfs.cs.tu-berlin.de/agg/
2. Ajmone M. M., Balbo G., Conte G., Donatelli S., Franceschinis G., "An Introduction to generalized stochastic Petri nets", In International Journal of Microelectronics ans reliability: Special issue on Petri nets and related graph models, Vol. 31(4), pp 699-725, Pergamon Press,1991.
3. ATOM3 home page, http://atom3.cs.mcgill.ca/
4. Barbier F., UML 2 ET MDE Ingénierie des modèles avec études de cas 2005.
5. Bernardi S., Donatelli S., and Merseguer J., "From UML sequence diagrams and statecharts to analyzable petri net models". In Proceedings of the 3rd International workshop on software and performance (WOSP), 2002.
6. Bernot G., Gaudel M. C., and Marre B., "Software testing based on formal specifications: a theory and a tool". In Software Engineering Journal, Vol 6(6):387-405, November 1991.
7. Bouarioua M. ,Chaoui A. , and Elmansouri R. From UML Sequence Diagrams to Labeled Generalized Stochastic Petri Net Models Using Graph TransformationIn J.J. Yonazi et al. (Eds.): ICeND 2011, CCIS 171, pp. 318–328, 2011, Springer-Verlag Berlin Heidelberg 2011.
8. Charles, N., Bowman, H., Thompso, S., "From Act-one to Miranda, a Translation Experiment". In Computer Standards and Interfaces Journal, Vol 19(1), May 1997.
9. EMF home page, http://www.eclipse.org/emf/

10. Fowler M., "UML distilled third edition, a brief guide to the standard object modeling language", 2004.
11. FUJABA home page, http://www.fujaba.de
12. Genrich H.J. and Lautenbach K., "System Modelling with High-Level Petri Nets'". In Theoretical Computer Science Journal, vol. 13, 1981.
13. Graph transformation and graph grammars http://www.gratra.org/
14. GreAT home page, http://www.escherinstitute.org/Plone/tools/
15. GEF home page, http://www.eclipse.org/gef/
16. GMF home page, http://www.eclipse.org/gmf
17. GME home page, http://www.isis.vanderbilt.edu/gme
18. Hissich, R.D. and Peters, M.P. (2002), *Entrepreneurship*, Fifth Edition, International Edition, New York.
19. INA home page, http://www2.informatik.hu-berlin.de/~starke/ina.html
20. Kelly S., Lyytinen K., and Rossi M., "MetaEdit+: A fully configurable Multi-User and Multi-Tool CASE and CAME Environment", Advanced Information System Engineering; LNCS 1080. Berlin, 1996.
21. Kerkouche E., Chaoui A., Khalfaoui K., "Transforming UML models to colored Petri nets models using graph grammars". Proceedings of ISCC 2009: 230-236.
22. Kerkouche E., Chaoui A., Bourennane E., Labbani O., "A UML and Colored Petri Nets Integrated Modeling and Analysis Approach using Graph Transformation". In Journal of Objetc Technology Vol 9(4): pp 25-43 (2010).
23. OMG Unified Modeling Language Specification: version 1.4. Object Management Group Inc., Sept. 2001 http://www.omg.org
24. OMG. (2003, 06 12). MDA Guide Version 1.0.1. Retrieved from Object Management Group: http://www.omg.org/mda
25. PROGRES Applications of Graph Transformations with Industrial Relevance, LNCS, Vol. 5088, 2008.
26. Rozengerg G., "Handbook of Graph Grammar and computing Graph Transformation", World Scientific, 1999.
27. Varró D. and Pataricza A.,"Vpm : A visual, precise and multilevel metamodeling framework for describing mathematical domains and uml (the mathematics of metamodeling is metamodeling mathematics)". In Software and System Modeling Journal, Vol. 2(3) :187-210, 2003.
28. Wing J.M. 1990 "A Specifier's Introduction to Formal Methods", Computer Journal, vol. 23(9):8-23.