Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 83 (2016) 658 – 664

Procedia Computer Science

The 7th International Conference on Ambient Systems, Networks and Technologies (ANT 2016)

On TGG Ability for Transforming UML 2 Sequence Diagrams with Imbricate Combined Fragments to π-Calculus Specifications.

Nabil Messaoudi[a,b], Allaoua Chaoui[b], Mourad Derardja[a], Mohamed Bettaz[c]

[a] *University Abbes Laghrour Khenchela, Algeria*

[b] *MISC Laboratory, Department of Computer Science and its applications Faculty of NTIC University Abdelhamid Mehri Constantine, Algeria*

[c] *Ecole Superieure d'Informatique (ESI) Algiers, Algeria*

Abstract

This paper describes a way based on operational semantics of π-calculus and uses TGG tool to formalize sequence diagrams to establish formal verification through model transformations. Our transformation uses basic interactions and combined fragments with the operator *Alt*, *Opt* and *Par*. We argue that TGG rules can be more easily used and they become more understandable. The transformation feasibility is illustrated on a scenario of phone system.

Peer-review under responsibility of the Conference Program Chairs

Keywords: Triple Graph Grammar; Bidirectional model Transformation; UML Verification; UML Validation; π-calculus.

1. Introduction

The lack of UML formal semantics pushed us to rely our work to several others in order to establish a formal verification for a specification based on UML2SD(Unified Modeling Language 2 Sequence Diagrams). The challenge facing our approach is to extract π-calculus processes from UML2SD specifications using TGG (Triple Graph Grammar) model transformations. The final goal is to have a design environment in which the user interacts only with UML2SD to carry out a formal analysis for his specifications. This paper is written for one simple reason is to bridge the gap between theoretic studies on formal semantics and practical studies to implement languages.

In spite of endowing UML with OCL(Object Constraint Language) notations, OCL constructs dont allow to create, to destroy or to modify objects in a model, and constraint checking is done without effect i.e. the instances are not modified by the constraints. So, the transformation to a model formal is more than necessary. Many researches have proposed the semantics to deal with the lack of UML formal semantics and many others have developed to transform UML notations into formal model to launch the verification process[5,14,15].

* Corresponding author. Tel.: +213-07-73-13-56-33 ; fax.: +213-032-72-51-60

E-mail address: messaoudi@misc-umc.org

Peer-review under responsibility of the Conference Program Chairs

doi:10.1016/j.procs.2016.04.147

The main advantage of TGG[2] is used mainly to define a bidirectional transformation and also to define a relationship between two types of models. It allows an incremental propagations change between two models, and it can be used in checking consistency between two models; this phenomenon is generally known in UML specifications.

A TGG rule consists of three graph transformation rules[1]. TGG rule= *pleft*, *pmap*, *pright* specified in one diagram rule. *pleft* which represents the source model objects, *pright* represents the target model objects and *pmap* represents the corresponding model objects which drawn in the middle of the diagram rule. The imbricate combined fragments give UML2SD a rich syntax and allow for users to express their requirements in an easier way. So, The use of TGG rules to transform an UML2SD is an important task due to the advantages presented above. To carry out the transformation, we propose a metamodel for UML2SD based on OMG[3] specifications and a metamodel for π-calculus specification. The choice of TGG tool is justified by the development of a bidirectional transformation (from UML2SD to π-calculus and from π-calculus to UML2SD). The rest of this paper is structured as follows: In the section 2, we situate our work among others and we mention some works related to the achievement of model transformations. The section 3 is destined for the presentation of the transformation elements. The section 4 describes the solution to the TGG example and ultimately, we closure with conclusion and main outcomes.

2. Related Work

We quote some works in relation with our TGG specifications and some others which allows us to transform UML2SD. In particular, the papers which transform UML2SD to formal model such as Büchi automata and Petri-nets.

In their paper[10], the authors carry out a transformation from UML activity diagram to CSP (Communicating Sequential Processes); they define a relation between two models in an intuitive way. We suggest a formal semantics to perform the relation between the input and the target models, and we ensure that the development of TGG rules is not intuitive but they reflect an idea based on operational semantics.

In their paper[11], the authors specify a transformation from state machine to Petri nets that is performed with TGG, AGG[12] tool is used to implement the transformation. In our approach, the management of combined fragments and the choice of π-calculus specifications automata for the model checker create many differences in the implementation.

In the paper[13], the authors discuss TGGs as a technology for model transformations, integration, and synchronization. Different ways of implementing TGGs are also presented.

The paper[15] illustrates a classical way to generate automata from UML sequence diagrams where combined fragments imbrications are not presented, the authors explain only how to translate a simple operator to their corresponding automata.

In their paper[14], the authors illustrate how to transform UML sequence diagrams to PROMELA (Protocol Meta Language) code with ATOM3[16] tool using imbricate combined fragments, the lack of corresponding metamodel presents an inconvenient because it increases the risk of losing information.

In their paper[9], the authors illustrate the transformation from MSD (Modal Sequence Diagrams) specification to networks of TGA (Timed Game Automata). They extend TGG by OCL constraints, application conditions, and custom attributes. We converge with this paper in the use of OCL to express constraints in our TGG rules.

In this paper, we have presented a manner to transform the complex operator applied to combined fragments, and then we can capture the equivalent semantics of sequence diagrams with imbricate combined fragments and π-calculus specifications.

3. From UML2SD to π-calculus language

In this section, we present the metamodels and the transformation rules that allowed us to transform UML2SD to π-calculus.

3.1. UML2SD Metamodel

The metamodel depicted in the Fig.1 is composed of 15 metaclass connected by 34 associations, which is illustrated by *Named Element*. The metaclass *Diag_sequence* is the concept of UML2SD which is no more than a set of messages,

interaction fragments and lifelines. The metaclass *lifeline* is the concept of the lifeline which is defined as a set of events of occurrence specifications where these latter are represented with the metaclass *Occu_sepcf*. The metaclass *message* is the generalization of two metaclass *send_occ* and *Receive_occ*, it represents a connectable point on the lifeline in the case of sending or receiving message. *Interaction_fragments* is an abstract metaclass that represents a unit of interaction. *Combined_fragment* is defined by an *interaction_operand* and it is a generalization of the operator *Alt*, *Par* and *Opt*. *Interaction_constraint* consists of Value-specification.

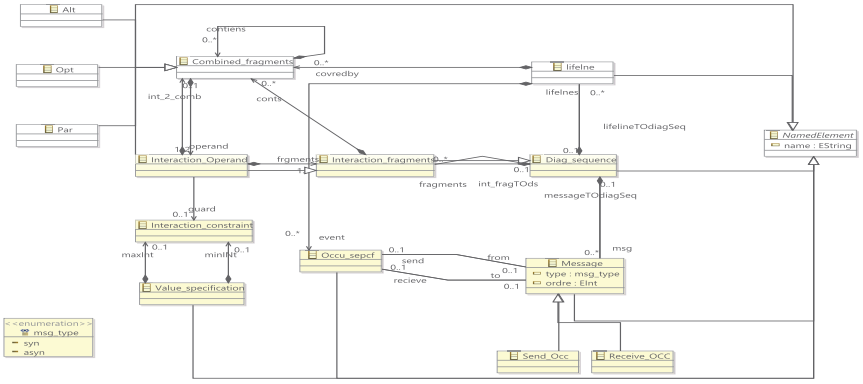

Fig. 1. UML2SD Metamodel based on OMG Specifications.

3.2. Metamodel π-calculus language

π-calculus defined by the metaclass *Pi_calculus* (see Fig.2.) contains a set of process and channels where the metaclass *Process* represents the process and the metaclass *Channel* represents the Channels. The left side of the instruction refers to the metaclass *process_expression* that is a generalization of the metaclasses *Prefix*, *Condition*, *Operateur*, *Parallel*, *Choice*, *skip*, *Stop* and *Empty_process*. The metaclass *Prefix* is composed of events which transit belong the channel. The channel is identified by the inputs represented with the metaclass *Input* and the outputs represented by the metaclass *Output* and these are the two ports of the corresponding channel.

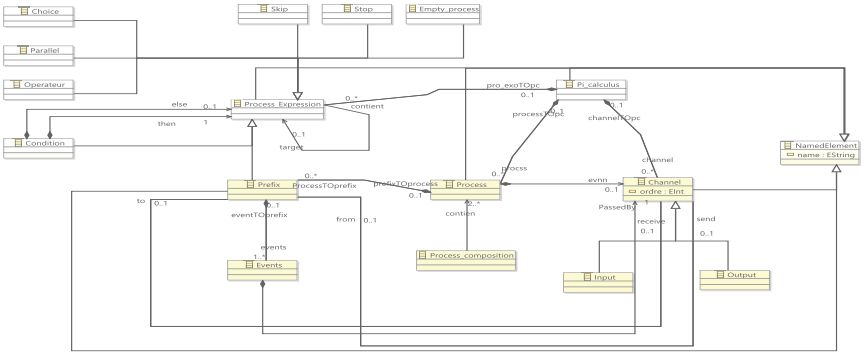

Fig. 2. π-Calculus Metamodel.

3.3. Corresponding Metamodel

The corresponding metamodel generated by TGG is composed of 27 metaclass. Each metaclass represents a transformation rule as a graph grammar. In the section below, we show only some rules that we have considered as

the most important due to paper lenght limit. We present in particular the rules of combined fragments imbrications. The corresponding metamodel is defined by the generalizable metaclass *AbstractContainerCorrespondenceNode*.

3.4. Transformation of UML2SD

The development of the basic TGG rules presented in this paper is based on the semantics presented in[7]. Each object supported by an UML2SD lifeline is transformed to π-calculus process. A private channel represents the communication between two π-calculus processes. This private channel is the transformation result of each message used in the UML2SD. So, it is natural and logical to identify each message with an integer in the metamodel (see Fig.1) owing to use a unique identity for each message. As well known that the semantics of a message is the sent denoted (!) and receive denoted (?). Beside transformation, we creat two objects signal input and signal output that represents respectively the sent and receive synchronization message via the corresponding private channel.

For two π-calculus processes connected by a message *m*, a TGG transformation rule transforms this message as an input on the private channel *m* in the object sending the message, and an other one transforms it as an output on the same private channel in the object sending the message.

The transformation of the combined fragments imbrications presented in UML2SD is carried out according to the transformation rules assigned to each operator. To realize this task, it is more than necessary to write the π-calculus specifications of each rule that allows the following operations:

- *Opt* may contains one or more *Opt*.
- *Par* may contains one or more *Par*.
- *Alt* may contains one or more *Alt*.

Next, each combined fragment may contains one or more combined fragments; for instance *Alt* may contains more *Par* and more *Opt*. Each node of new combined fragments is placed in the inner of an existent node. The transformation of each node is inherited from the rule interaction fragment to process expression. To take into consideration this phenomen, we have defined the relations *Contiens* and *Cont* at UML2SD metamodel and π-calculus metamodel. The relations *Contiens* and *Conts* are affected to pleft diagram rule where *Contiens* ensures the combined fragment imbrications and *Conts* covers the set of interaction fragments attached to the corresponding operator. In the right diagram rule, the relation *contient* permits to cover the condition affected to the operator via the process expression. The coverage of the full elements contained in a π-calculus process is ensured with the metaclass interfragm2process, *process_exp*, *nameofoperator2condition* and *nameofoperator2nameofoperator* of the corresponding metamodel and the metaclass *process_exp*, *alors_sinon*, *nameofoperator* of the right diagram rule (the libel *nameofoperator* presents in each time the operators concerned by the transformation process). Some rules are presented in Fig. 4, Fig .5, and Fig .6 to clarify our strategy. However, the whole imbrications are generated gradually and recursively belong TGG interpreter[4]. Ultimately, the whole π-calculus specifications is obtained via parallel composition. This task is achieved with a rule transformation *par2par* which is not presented in this paper.

3.4.1. Axiom

Each sequence diagram is transformed to π-calculus specifications where the name of diag_sequence is affected to p_icalculus. This is also the starting point for all transformations.

3.4.2. Life line to Process

Each life line is transformed to π-calculus process where the name of lifeline identification is affected to pi_calculus process.

3.4.3. Message to channel

Each message is transformed to a private channel. We produce sequentially for each sent message an input signal and receive message an output signal along the corresponding channel. The constraints with yellow color affect each signal name. See Fig.3.

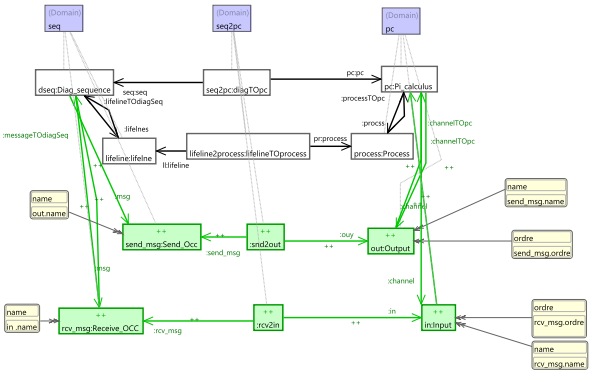

Fig. 3. Message to Channel.

3.4.4. Alt contain Alt and Alt contain Opt

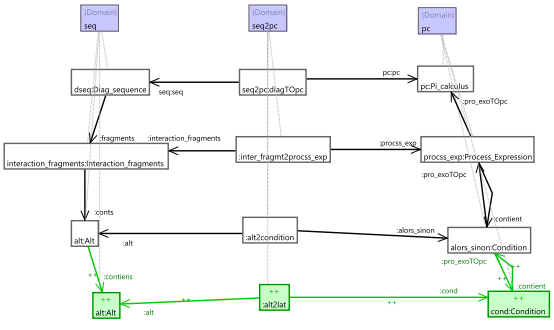

Fig. 4. Alt contain Alt.

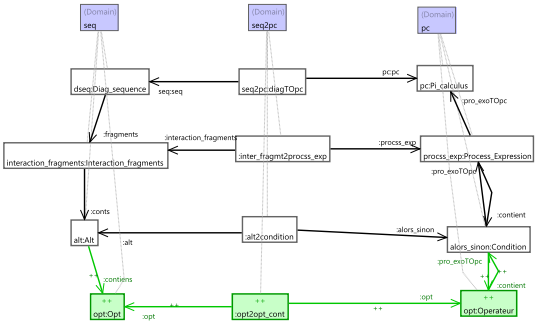

Fig. 5. Alt contain Opt.

3.4.5. Alt contain Par

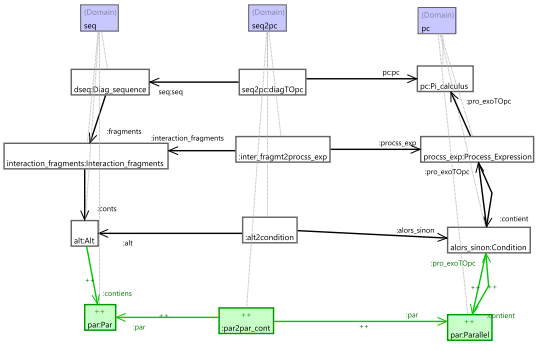

Fig. 6. Alt contain Par.

4. Example

To show the applicability of our environment, we use a scenario in the context of phone system. It consists in three objects Caller, Phone and Receiver. We use the operator ALT to imbricate the combined fragment with basic

interactions for representing the scenario when the caller is busy and the scenario when the called dont answer. The Fig.7 represents the concrete syntax of our example. We mention that the example should be edited in the EMF(Eclipse Modelling Framework) editor and should be represented in abstract syntax. The figure is not shown due to the limited number of pages.

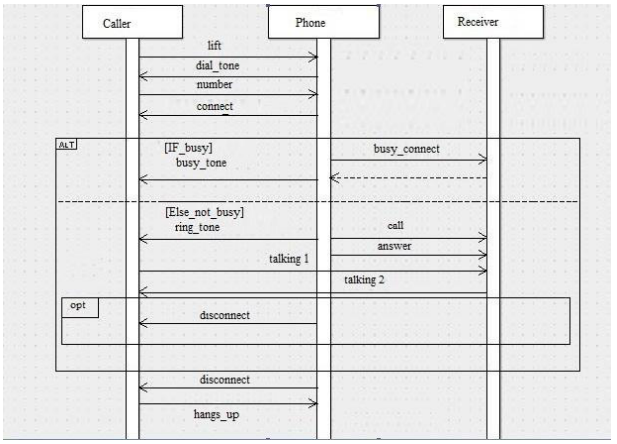

Fig. 7. Concret Syntax of Phone System Caller.

The application of our environment on the example is illustrated in the Fig.8. There were 40 rules, requiring 2977 edge explorations and 1085 unsuccessful rule application attempts requiring 9489541 edge explorations in 566110 milliseconds which illustrate the TGG interpreter power.

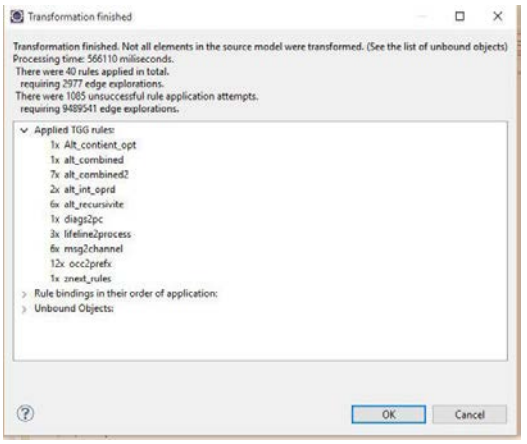

Fig. 8. TGG interpreter Result .

The TGG Interpreter is used to obtain the result of our transformation; the π-calculus result is displayed in two frames. The frame on the right follows the left frame. The interpretation of the ransformation results shown in Fig.9 reveals the existence of the π-calculus processes *Caller*, *Phone*, and *Receiver*. The communication via private channels even exists with input and output objects.

Fig. 9. π-calculus Specifications for Phone System Caller.

5. Conclusion and future works

We have achieved an environment for the equivalent of sequence diagrams in π-calculus. This environment is bidirectional, and it allows using π-calculus for the formal verification. It permits to bridge the gap between theoretic studies of formal semantics and the implementation of π-calculus specifications.

The imbrications of combined fragments with the operators *Alt*, *Par* and *Opt* are also achieved gradually and recursively with TGG interpreter by the application of our transformation rules.

However, the combined fragments imbrications with the operators *Neg* and *Loop* will be interesting as future work. We also plan to use GMF[7] to develop a graphical editor to consider the sequence diagrams in concrete syntax. . We envisage to use Xpand[6] to generate text from the obtained result to launch verification with a model checker; the result is in an XMI(XML Metadata Interchange) file. We are currently planning such study with student teams to implement these tasks.

References

1. Schürr, A.: Specification of Graph Translators with Triple Graph Grammars. In WG '94 Proceedings of the 20th International Workshop on Graph-Theoretic Concepts in Computer Science, Springer LNCS, vol. 903, pp. 151-163, 1995.
2. Grunske, L., Geiger, L., Lawley, M.: A Graphical Specification of Model Transformations with Triple Graph Grammars.In Proceedings ECMDA-FA, Springer-Verlag Berlin Heidelberg, LNCS vol. 3748, pp. 284-298, 2005.
3. OMG-PTC/03-08-02. UML2.0 Superstructure Specification, 2003.
4. TGG Home page: http://www.informatik.uni-marburg.de.
5. Messaoudi, N., Chaoui, A., Bettaz, M.: An Operational Semantics for UML2 Sequence Diagrams Supported by Model Transformations. In Procedia Computer Science vol. 56, pp. 604-611, 2015.
6. Markus, V., Patrick, S., Moritz, E., Steven, R., Darius, J., Andre, A.: Xpand documentation. Available from http://www.researchgate.net/publication/265083790_Xpand_Documentation, 2010.
7. Katerina Pokozy-Korenblat Pokozy-Korenblat, K., Corrado, K.: Toward Extracting π-calculus from UML Sequence and State Diagrams. Electronic Notes in Theoretical Computer Science, ELSEVIER vol. 101, pp. 51-72, 2004.
8. GMF site available from https://www.eclipse.org/gmf
9. Greenyer, J., Rieke, J.: Applying Advanced TGG Concepts for a Complex Transformation of Sequence Diagram Specifications to Timed Game Automata. In 4th International Symposium, AGTIVE 2011, Budapest, Hungary, Springer, LNCS, vol. 7233, pp. 222-237, 2012.
10. Greenyer, J., Rieke, J., Travkin, O:. TGGs for Transforming UML to CSP: Contribution to the ACTIVE 2007 Graph Transformation Tools Contest. Tech-rep tr-ri-08-287, Software Engineering Group, Dept of Computer Science, Univ of Paderbon, 2008.
11. Liu, S., Liu,Y. , Andre, E., Choppy, Sun, J.: A Formal Semantics for the Complete Syntax of UML State Machines with Communications. Proceedings of the 10th International Conference on Integrated Formal Methods, IFM, vol.38613, pp. 8-23, 2013.
12. AGG official Site : http://tfs.cs.tu-berlin.de/agg.
13. Triple Graph Grammars : Concepts, Extensions, Implementations, and Application Scenarios. Tech. Rep.-ri-08-287, Software Engineering Group, Dept of Computer Science, University of Paderborn, Germany, 2007.
14. Amirat, A., Menasria, A., Ait Oubelli, M., Younsi, N.: Automatic Generation of PROMELA code from sequence diagram with imbricate combined fragments. Innovative Computing Technology (INTECH), IEEE, pp. 111-116, 2012.
15. Cengarle, M.V,. Knapp, A.: Model Checking of UML2.0 Interactions. Available from http:citeseer.uark.edu:8080/citeseerx/viewdoc /summary? doi=10.1.1.69.8798, 2006.
16. AToM3 home page: http://atom3.cs.mcgill.ca