A Combined Graph-based Approach for Systems Design And Verification

Najet Zoubeir and Adel Khalfallah
Département du Génie Logiciel et Systèmes d'Information, Institut Supérieur d'Informatique, Ariana, Tunisia

Keywords: Graph Transformation Systems, Software Architecture Analysis, Syntax, Semantics, Verification.

Abstract: Software architecture's interoperability faces many problems when it comes to integrating different components or formalisms in describing the architecture. Even within the same modeling language such as UML, the diversity of notations and the lack of semantic information make the interoperability between models difficult. In this paper, we propose semantic foundations that unify the notations of classes, interactions and constraints, and hence provide a robust basis for models interoperability. We rely on graphs and graph transformations to describe systems structure and behavior, semantics and constraints in a combined form within an integrated framework, which constitutes a robust basis for automated software architecture analysis.

1 INTRODUCTION

Describing the software architecture of a given system is a challenging task, defined as "*the set of design decisions which, if made incorrectly, may cause your project to be canceled*" (E. Woods, 2010). Yet, describing software architecture should not be a purpose on itself. In fact, designers and architects, while and after describing architecture, should be able to enhance their decisions by analyzing and optimizing the architecture.

The Unified Modeling Language has known a wide success in describing systems architecture, due to the flexibility and universality of its graphical notations. The diversity of notations in UML diagrams enriches the architecture description, however, it creates distances between the diagrams, and hence, obstacles for models interoperability. This lack of interoperability would make it difficult to analyze results from different models.

Through our work, we try to enhance models interoperability by unifying the notations describing systems structure and behavior. We assume that the semantic information is an essential factor for refining architecture analysis, as well as constraints verification. In this paper, we propose a framework for software architecture description, verification and execution using the unique notation of graphs and graph transformations. Our proposed framework offers the following features:

- Models syntax and semantics description: Both syntactic and semantic information are necessary for architecture analysis, such as for design patterns or architectural styles recognition. In fact, the more information the architecture provides, the more correct and refined the patterns utilization will be. This description is homogenous since the syntax and semantics are described in a combined form using the formalism of graphs;
- Constraints and properties expression and verification on models: There are two levels of constraints on models: the meta-level constraints, restricting the number of models satisfying a given meta-model, and model-level constraints describing the model properties and constraints. A software architecture description has to permit to express and verify the different levels of constraints, allowing designers to analyze architecture, especially regarding quality. These constraints, from a verification point of view, are a means of test case generation, which creates with graphs an adjacent state space permitting to search for states violating these constraints;
- Identify systems behavior: which permits to trace the execution of the systems, and consequently identify its desirable and undesirable behaviors.

The robust theoretical foundations of the formalism of graphs permit to automate a set of analysis on the created architectures, such as design optimization and quality evaluation. Figure 1 resumes the features provided by our graph-based framework for software architecture description: models structure and behavior expression, model-level and meta-level constraints

Zoubeir N. and Khalfallah A..
A Combined Graph-based Approach for Systems Design And Verification.
DOI: 10.5220/0005272603460353
In *Proceedings of the 3rd International Conference on Model-Driven Engineering and Software Development* (MODELSWARD-2015), pages 346-353
ISBN: 978-989-758-083-3

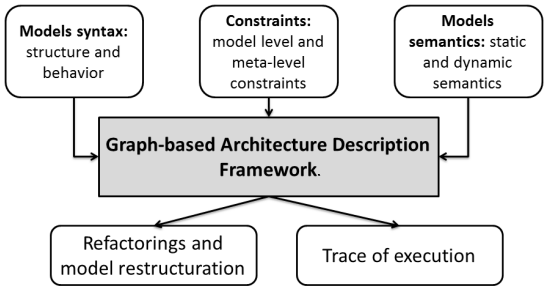

Figure 1: Architecture Description Framework Features.

definition and verification, and models static and dynamic semantics description. The framework has to offer also the possibility to execute models and to trace this execution.

2 RELATED WORK

Graph Transformation Systems (GTSs) are broadly used in software engineering: models transformation, tools implementation, testing (Chan et al., 2009), visual languages semantics description, etc. In this particular last area, diverse graph-based semantics description techniques have been proposed. Describing the semantics of UML diagrams belonging to different views has been the focus of many works, and graph systems has known a wide success in this domain, due to the formal and universal nature of this formalism. For example, graph-based semantics for statechart diagrams were proposed in (Varro, 2002; Engels et al., 2000; Gogolla and Presicce, 1998), sequence diagrams in (Hausmann et al., 2004), activity diagrams in (Hausmann, 2005) and class diagrams in (Gogolla and Richters, 1998). Integrated semantics that combine subsets of UML diagrams were proposed in diverse works, such as the works in (Kuske et al., 2002; Gogolla et al., 2003) that propose a graph-based integrated semantics for UML class, object and statechart diagrams, and the work in (Holscher et al., 2006) which considers a larger subset of UML diagrams, including further the use cases and interaction diagrams. Among these works, the Dynamic Meta-Modeling approach (DMM) (Engels et al., 2000; Hausmann et al., 2004; Hausmann, 2005) proved to be valuable, since it does not target a specific application purpose or specific notations of UML, and focuses on the semantics definition on a meta level. Although our work was inspired by some of the DMM approach assets, such as dynamic semantics definition and semantic mappings, it is distinguished by a more complete perspective of the architectural activity, which combines structure, behavior and constraints.

Graph-based constraints semantics are proposed in different works (Bauer, 2008; Rutle et al., 2012; Dang and Gogolla, 2008; Rensink and Kleppe, 2008; Bottoni et al., 2002; Winkelmann et al., 2008; Dang et al., 2010). The work in (Bauer, 2008) treats constraints as additional refinements of the matching in Graph Transformation Rules (GTRs). Works in (Rutle et al., 2012; Rensink and Kleppe, 2008) manipulate OCL expressions whether in their textual form (Rensink and Kleppe, 2008) or using First Order Logic (Rutle et al., 2012), which are notations external to graphs, and hence the proposed semantics are heterogeneous. Works in (Bottoni et al., 2002; Winkelmann et al., 2008) propose graphical description for constraints using graph grammars (Winkelmann et al., 2008) and collaboration diagrams (Bottoni et al., 2002). In (Bottoni et al., 2002) the proposed graph visualization for OCL constraints semantics is quite complex, especially for advanced OCL constraints. Besides, the transformation to graphs uses a hybrid notation based on textual syntax. The work in (Winkelmann et al., 2008) addresses a restricted subset of OCL expressions, but does not study the diverse placements of constraints. The works in (Dang and Gogolla, 2008; Dang et al., 2010) combine OCL in its textual form with Triple Graph Grammars in order to check conformance between models. However, these multi-leveled and multi-viewed graph-based semantics are proposed separately, i.e. no combined graph-based semantics was proposed to represent systems structure, behavior and constraints through different modeling levels.

The approach proposed in this paper tries to take full benefits of the formalism of graph systems, and proposes a graph-based framework for software architecture description that integrates models structure, dynamics and constraints within a unified formalism that provides robust basis for exploiting models syntactic and semantic information. We assume that our framework will facilitate and automate a considerable set of software architecture analysis.

The remainder of this paper is organized as follows: In section 3 we will highlight the importance and the use of the semantic information in software architecture analysis. In section 4 we will describe the structure of our proposed framework by presenting a set of architectural meta-models for systems syntax and semantics. This section will also specify the semantic mappings that will associate to every syntactic and semantic element in a model, a corresponding graph in a GTS. Section 5 will be devoted to introduce our proposed graph-based verification. We will present the meta-level constraints as a set of well-

formedness rules of the interactions semantics. In this section, we will show the transformation of these well-formedness rules to GTRs, and how they can be checked on models. The last section will be devoted to a conclusion and a presentation of our further work.

3 ARCHITECTURE ANALYSIS AND SEMANTICS

In order to perform diverse analyses on software architecture, all kinds of information are required. The first kind is the syntactical information. Such information constitutes the basic level of information provided by any architecture description language. The second kind of information is the semantic one, which may concern diverse aspects, such as model elements semantics, operation semantics, constraints semantics, etc.

Great amount of researches have been conducted in the description and expression of systems semantics. In this section, we will try to point out two specific matters: How exactly semantic information is useful in analyzing software architecture, and what approaches of semantics permit to perform these analyses. We consider the set of architecture analyses we presented in the introduction: software architecture optimization, quality evaluation and architecture simulation. Semantic information can have a great use in refining and optimizing software architecture. In fact, architecture description that provides information such as attributes values (in a denotational way), methods operational semantics, interactions nature (synchronous or asynchronous) and sequencing, etc. can easily be implemented using a programming language. Semantics also can be helpful in optimizing software architecture. In fact, through a previous work based on models syntax in order to characterize systems architecture (Zoubeir and Khalfallah, 2010), we noticed that it was insufficient, especially when it comes to behavior recognition or injection. Adding semantic information such as the operational semantics of methods, will definitely lead to better algorithms for these tasks.

Semantic information is also needful for systems simulation. In fact, in order to trace the execution of a system it is necessary to identify its diverse possible behaviors. This task cannot be performed without semantic information, and especially operational semantics. The operational semantics approach is indeed particularly useful in performing this purpose, since it interprets systems as state transition systems (Plotkin, 1981).

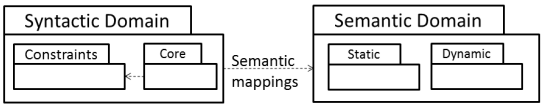

Figure 2: The Framework packages structure.

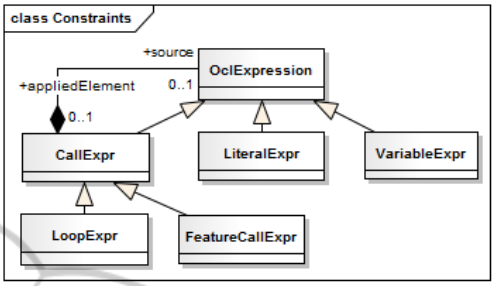

Figure 3: Class diagram for the ”Constraints” package.

4 ARCHITECTURAL META-MODELS

We propose a framework for describing software architecture. Our framework combines systems syntax, semantics and constraints using graph notations. In this section, we will present the structure of our framework through a set of meta-models describing a graph-based combined semantics for UML classes and interactions, accompanied by OCL constraints. This structure is described using the package diagram of figure 2.

4.1 Syntactic Domain

The syntactic domain is expressed using a meta-model that combines structure, behavior and constraints. Our combined meta-model is composed by two packages: the package ”Core” and the package ”Constraints”, such as the former one uses the last one, as illustrated in figure 2. The package ”Constraints” was directly inspired from the abstract syntax of OCL expressions specified in the OCL specification document (OCL, 2001). Indeed, it is composed by the subset of OCL constraints that have been covered by the work in (Zoubeir et al., 2013) on the expression of OCL constraints using graphs. In that work, a set of Graph Constraint Patterns (GCP) was proposed, representing the transformation of a subset of OCL constraints into graphs. The class diagram corresponding to the package ”Constraints” is depicted in figure 3.

The package ”Core” constitutes a simplified version of the UML class diagram meta-model, in which we introduced the behavioral notions of interactions. The class diagram corresponding to the ”Core” pack-

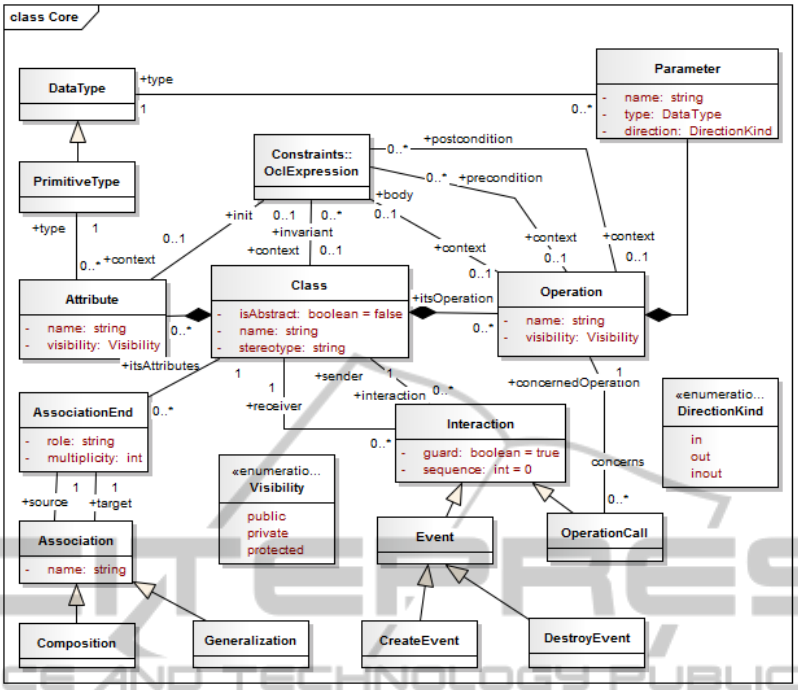

Figure 4: Class diagram for the ”Core” package.

age is illustrated in figure 4. An interaction is identified by a sequence number and a Boolean expression named guard. Interactions have two forms: synchronous interactions represented by event sending, and asynchronous interactions corresponding to operation calls. An interaction takes place between a sender and a receiver, presented as classes in the combine meta-model. Working with GTSs, the sender and receiver of an interaction will be represented in instance graphs by instances of classes, which corresponds to objects in UML.

In this combined meta-model, constraints can be defined on classes, attributes and operations, the elements constituting the possible contexts of a constraints.

4.2 Semantic Domain

The semantic domain is composed by two packages for the static and dynamic semantics. The static semantics can be represented by meta-modeling. In figure 5, we present the static semantics of interactions. The sequencer can be regarded as a global variable whose responsibility is to order the sequence of interactions, and consequently the invocation of the GTRs corresponding to the model operations. Concerning the dynamic semantics, we did not describe it by meta-modeling. Inspired by the DMM approach and most works in the field, we chose to describe the dynamic semantics and particularly the operational

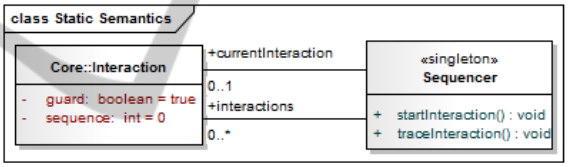

Figure 5: Class diagram for the Static Semantic package.

semantics using GTRs. Indeed, these transformation units permit to describe by a transition system what an object does through a given method.

4.3 Semantic Mappings

The semantic mappings permit to assign a meaning for each syntactic element. Defining the semantics of a visual language cannot be complete without a precise description of the relations that map every syntactic element to a set of semantic elements. The main semantic mappings in our proposal are depicted in figure 6. In our semantics, a class is described by a typed node, an attribute by a valued node and an operation by a GTR.

Our contribution manifests essentially in the semantic mappings that assign GTRs to constraints, and scenarios to interactions.

The first mapping is described in details with an illustrating example in (Zoubeir et al., 2013). It proposes a set of Graph Constraints Patterns (GCP) that represents the transformation of OCL expressions to

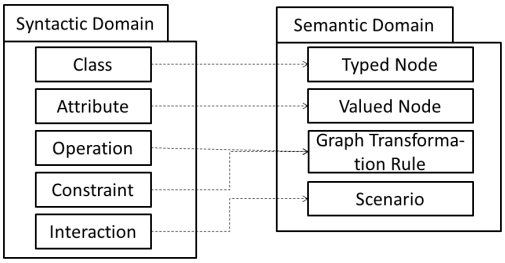

Figure 6: Semantic mappings.

graphs, and then explains how to express OCL constraints using GCP within GTRs. The second mapping assigns scenarios to interactions. As an interaction is defined as "*a unit of behavior that focuses on the observable exchange of information between (...) elements*" (UML, 2000), we assume that the execution of a sequence of interactions can be interpreted as a possible scenario of the modeling system behavior. In the context of GTSs, a scenario is the ordered execution of a sequence of GTRs, scheduled using the mechanism of controls, which constitute a sort of control programs that permit to schedule the execution of different rules.

We propose to use typed attributed graphs in order to describe models syntax and semantics. Figure 7, which is a refinement of figure 1, illustrates the correspondences between syntactic and semantics elements, and GTSs. Syntactic elements are represented in GTSs using typed graphs. Constraints, however, are described by GTRs, whether as part of methods operational semantics transformation rules, or as empty-side effect transformation rules that represent invariants. In the semantic domain, while static elements are also described in type graphs, the dynamic ones will be represented by GTRs, which, scheduled as scenarios using controls, can be simulated and interpreted within GTS. Simulation constitutes an important feature offered by GTSs that allows the designer to analyze in depth the obtained results. In fact, the execution of GTRs on a given graph instance can be regarded as a labeled transition system that describes the system's different states during the simulation.

4.4 Transformation to Graphs

The transformation of a combined model to a GTS is obtained through three major steps. These steps differ by their implementation and their possibility of automation, as detailed in the following paragraph and depicted in figure 7:

1. Step 1: the first step consists of the transformation of the combined meta-model of the package "Core" (cf. figure 4) into a corresponding type graph. Similar transformations were defined for class diagrams (Gogolla and Richters, 1998), and some of them are supported by tools, such as the tool "ecore2groove" from the toolset GROOVE[1]. Our transformation is implemented using the OMG model transformation standard Query/View/Transformation (QVT) (QVT, 2008), such as the source meta-model is the package "Core" (without representing the constraints, and containing the semantic class "Sequencer"), and the target one is a meta-model for type graphs.
2. Step 2: the second step correspond to the transformation of the constraints defined on the model into graph transformation systems. This step is carried out using the Graph Constraint Patterns (GCP) (Zoubeir et al., 2013).
3. Step 3: the third step corresponds to the expression of the operational semantics of the model operations using GTRs. This kind of operational semantics expression is widely used in graph-based software engineering (Hausmann, 2005; Kuske et al., 2002; Gogolla et al., 2003; Holscher et al., 2006). This is a manual step that depends on the designer and the designed system.

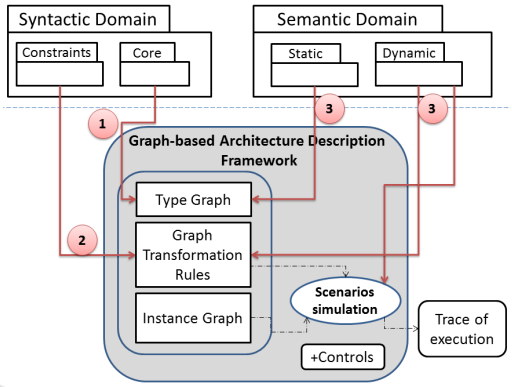

Figure 7: Correspondence between GTS components and the syntactic and semantic domains.

Once transformed and in order to be simulated, instances of the model have to be created, in the form of instance graphs. On these instance graphs, GTRs can be applied according to different scenarios, and constraints can be defined and checked on the generated state space. In the next section we will explain our proposal for graph-based constraints verification.

[1]GRaphs for Object-Oriented VErification, url: http://groove.cs.utwente.nl

5 GRAPH-BASED VERIFICATION

One important feature of a software architecture description framework is to offer the possibility of defining and checking constraints and properties of the modeled systems. In this paper we focus on two levels of constraints: the model level constraints and the meta-level constraints. The verification of the first level of constraints consists on the check of a set of user constraints defined on the designed system in order to monitor its structure and behavior. Usually these constraints are defined on UML model using the constraints language OCL. As we mentioned earlier, the definition of OCL expression semantics using graphs and their transformation into graph constraints patterns used within GTSs is studied and detailed in (Zoubeir et al., 2013). In this section, we will focus on the meta-level verification, which consists generally on the set of well-formedness rules ensuring that a given model is satisfying its meta-model. For that, we will propose semantics for interactions, and express it using graphs. Then we will define some well-formedness rules for interactions and express them as graph transformation units. For the representation of our GTRs we used the GROOVE notation, which combines the rules left hand side, right hand side and negative application conditions in one view, and distinguishes them using colors. We also relied on GROOVE model checker in order to check our graph constraints.

5.1 Interaction Semantics

The semantics of interactions corresponds to the ordered execution of the different operation calls and event sends, according to the order given by the designer for a particular scenario. We decompose this semantics into two dependent parts: the interaction sequencing and the interaction execution.

- Interaction sequencing: The execution of a scenario starts when the current interaction of the sequencer points to the interaction with the sequence 1, if it is available (method ”startInteraction()” in class ”Sequencer”, figure 5). The following interactions are traced by pointing the sequencer to the next available interaction (method ”traceInteraction()” in class ”Sequencer”). An interaction is available when its guard, whenever it exists, is true. The guards are evaluated to true if they are not explicitly represented. The behavior of these methods can be expressed using OCL pre and post-conditions as follows:

```
1. context Sequencer::startInteraction() :
    OclVoid
    pre: let i:Interaction = interactions->
        select(j|j.sequence=1)-> first() in
        i.guard = true
    post: currentInteraction.sequence=1

2. context Sequencer::traceInteraction() :
    OclVoid
    pre: let i:Interaction = interactions->
        select(j|j.sequence =
        currentInteraction.sequence+1)
        ->first() in i.guard = true
    post: currentInteraction.sequence =
        currentInteraction@pre.sequence + 1
```

  The first constraint expresses that in order to start the interactions, the interaction with sequence 1 should have a guard evaluated to true. Once the interactions started, the current interaction of the Sequencer will point on this particular interaction. The second constraint concerns the rest of interactions. It also monitors the verification of the guard of the next interaction before pointing the Sequencer on it.

  The operational semantics of these two methods can be formalized using GTRs, and the OCL pre and post-conditions defined on them will be expressed within these. The corresponding GTRs are depicted respectively in figures 8 and 9. In these rules we chose to represent guards in a negative form, since they are considered true if they do not exist. So the rule of figure 8 consists on pointing the sequencer current interaction on the interaction with the sequence 1, if its guard is not false. And the rule of figure 9 deletes the current interaction of the sequencer and pointes it on the interaction with the next sequence, if its guard is not false.

- Interaction execution: The execution of an interaction corresponds to the invocation of an operation in the case of operation calls, or the sending of a signal in the other cases. The mechanism of invocation can be expressed within GTSs using controls. Indeed, the execution of interactions in a given order constitutes a scenario of a possible behavior of the modeled system.

Executing interactions scenario will be performed by alternating interactions sequencing and execution.

5.2 Well-formedness Verification

We assume that the use of GTSs to express well-formedness rules has many advantages. The most important are: (1) Rules are expressed using the same formalism as models: uniformity and semantic integration, hence there will be no need to use heterogeneous formalism to express models and constraints

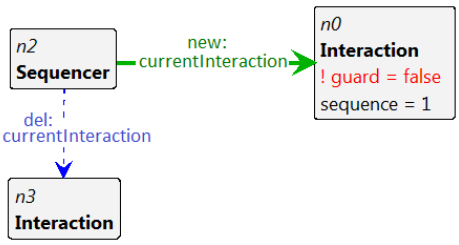

Figure 8: GTR for method ”startInteraction()”.

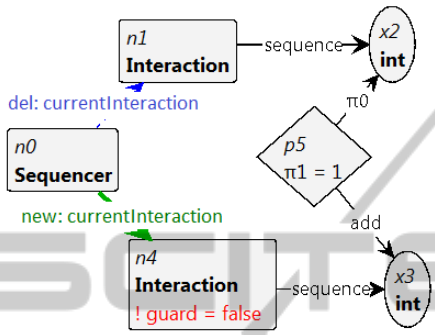

Figure 9: GTR for method ”traceInteraction()”.

(such as the graphical UML and the textual OCL), (2) Rules can be formally checked using model checking, and (3) Rules can be simulated in order to parse the states violating the given rules, in the state space adjacent to the instance graph.

The well-formedness rules defined on the meta-model of package ”Core” can be expressed using OCL. For example, we present the following well-formedness rules, expressing that an operation call concerns an operation belonging to its receiving object, and that an object cannot be the source or target of an interaction before it is created or after its destruction:

```
1. context OperationCall
    inv: self.receiver.itsOperation->includes
        (self.concernedOperation)

2. context CreateEvt
    inv: self.receiver.interaction->forAll
    (i:Interaction |self.sequence < i.sequence)

3. context DestroyEvt
    inv: self.receiver.interaction->forAll
    (i:Interaction |self.sequence > i.sequence)
```

The well-formedness rules are presented as invariants attached to the meta-model elements. They can be transformed into empty side-effect GTRs, that should be checked using model checking in order to verify whether the corresponding invariant holds for all the states while the whole simulation. These GTRs are depicted in figures 10, 11 and 12. An expression can be presented in its negative form as undesirable properties, and verified to be never true on the state space, as for the two latter rules.

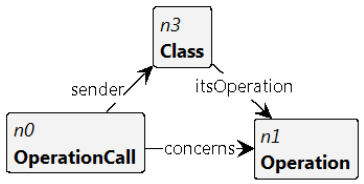

Figure 10: GTR corresponding to the first well-formedness rule.

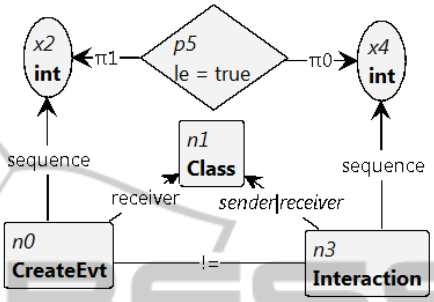

Figure 11: GTR corresponding to the second well-formedness rule.

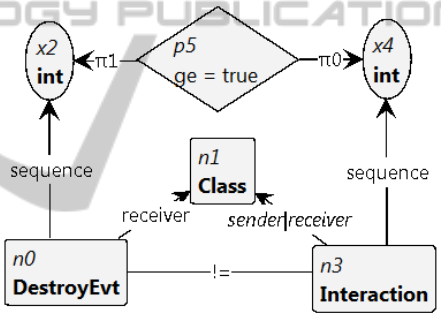

Figure 12: GTR corresponding to the third well-formedness rule.

Verification can be seen as a fonction of invariants for graph instances generation, i.e. given a meta-model, a set of graph transformations expressing its semantics and a set of invariants, a verification algorithm should be able to generate a set of graph instances as test cases by analyzing the invariants.

6 CONCLUSION

In this paper we defined a robust and flexible framework for software architecture description that permits to facilitate, improve and in certain situations automate analysis on this architecture. In our proposal, we relied on Graph Transformation Systems to describe systems syntax, static semantics and dynamic semantics. These GTSs are also used to carry

out meta-level and model-level constraints definition and verification, based on the notion of graph constraints. We assume that our framework is fully integrated within graphs, which guarantees compatibility, homogeneity and robustness.

In our future work, we aim to exploit the proposed framework in the analysis of systems architecture, in micro and macro levels, especially those conducted by transformations. Indeed, we exploited an aspect of this framework to define a graph-based design patterns decomposition (Zoubeir et al., 2014), and we aim to evaluate and validate the overall approach using experimental validation based on a repository of models.

REFERENCES

Bauer, E. (2008). *Enhancing the Dynamic Meta Modeling Formalism and its Eclipse-based Tool Support with Attributes*. PhD thesis, University of Paderborn, Germany.

Bottoni, P., Koch, M., Parisi-Presicci, F., and Taentzer, G. (2002). Working on ocl with graph transformations. In *APPLIGRAPH Workshop on Applied Graph Transformation*, pages 1–10.

Chan, W. K., Mei, L., and Zhang, Z. (2009). Modeling and testing of cloud applications. In Kirchberg, M., Hung, P. C. K., Carminati, B., Chi, C.-H., Kanagasabai, R., Valle, E. D., Lan, K.-C., and Chen, L.-J., editors, *APSCC*, pages 111–118. IEEE.

Dang, D.-H. and Gogolla, M. (2008). On integrating ocl and triple graph grammars. In Chaudron, M. R. V., editor, *MoDELS Workshops*, volume 5421 of *Lecture Notes in Computer Science*, pages 124–137. Springer.

Dang, D.-H., Truong, A.-H., and Gogolla, M. (2010). Checking the conformance between models based on scenario synchronization. *J. UCS*, 16(17):2293–2312.

Engels, G., Hausmann, J. H., Heckel, R., and Sauer, S. (2000). Dynamic meta modeling: A graphical approach to the operational semantics of behavioral diagrams in uml. In *UML*, pages 323–337.

Gogolla, M. and Presicce, F. P. (1998). State diagrams in uml: A formal semantics using graph transformations - or diagrams are nice, but graphs are worth their price. In *University of Munich*, pages 55–72.

Gogolla, M. and Richters, M. (1998). Transformation rules for uml class diagrams. In Bzivin, J. and Muller, P.-A., editors, *UML*, volume 1618 of *Lecture Notes in Computer Science*, pages 92–106. Springer.

Gogolla, M., Ziemann, P., and Kuske, S. (2003). Towards an integrated graph based semantics for uml. *Electr. Notes Theor. Comput. Sci.*, 72(3):160–175.

Hausmann, J. H. (2005). *Dynamic META modeling: a semantics description technique for visual modeling languages.* PhD thesis, University of Paderborn.

Hausmann, J. H., Heckel, R., and Sauer, S. (2004). Dynamic meta modeling with time: Specifying the semantics of multimedia sequence diagrams. *Software and System Modeling*, 3(3):181–193.

Holscher, K., Ziemann, P., and Gogolla, M. (2006). On translating uml models into graph transformation systems. *Journal of Visual Languages & Computing*, 17(1):78–105.

Kuske, S., Gogolla, M., Kollmann, R., and Kreowski, H.-J. (2002). An integrated semantics for uml class, object and state diagrams based on graph transformation. In *IFM*, pages 11–28.

OCL (2001). Object constraint language v 2.3.1. URL : http://www.omg.org/spec/OCL/2.3.1, visited 10/07/2014.

Plotkin, G. D. (1981). A structural approach to operational semantics. Technical report, University of Aarhus.

QVT (2008). Meta object facility (mof) 2.0 query/view/transformation (qvt). URL : http://www.omg.org/spec/QVT/, visited 10/07/2014.

Rensink, A. and Kleppe, A. (2008). On a graph-based semantics for uml class and object diagrams. *ECEASST*, 10.

Rutle, A., Rossini, A., Lamo, Y., and Wolter, U. (2012). A formal approach to the specification and transformation of constraints in mde. *J. Log. Algebr. Program.*, 81(4):422–457.

UML (2000). Unified modelling language superstructure v 2.4.1. OMG Document Number: formal/11-08-06, Standard document URL: http://www.uml.org/, visited 10/07/2014.

Varro, D. (2002). A formal semantics of uml statecharts by model transition systems. In *Graph Transformation*, pages 378–392.

Winkelmann, J., Taentzer, G., Ehrig, K., and K"uster, J. M. (2008). Translation of restricted ocl constraints into graph constraints for generating meta model instances by graph grammars. *Electronic Notes in Theoretical Computer Science*, 211:159–170.

Zoubeir, N. and Khalfallah, A. (2010). Synchronization of the static and dynamic views in rts modeling. In *Workshop en Informatique et Applications WIA 2010*, pages 81–85, Tunisia.

Zoubeir, N., Khalfallah, A., and Ahmed, S. B. (2014). Graph-based decomposition of design patterns. *International Journal of Software Engineering and Its Applications (IJSEIA)*, 8(2):391–408.

Zoubeir, N., Khalfallah, A., and Benahmed, S. (2013). *Embedded Computing Systems: Applications, Optimization and Advanced Design*, chapter Expressing and Validation OCL Constraints using Graphs, pages 93–107. IGI Global.