Available online at www.sciencedirect.com
SCIENCE DIRECT®
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 152 (2006) 111–123
www.elsevier.com/locate/entcs

Metamodel-Based Model Transformation with Aspect-Oriented Constraints

László Lengyel, Tihamér Levendovszky, Gergely Mezei, Bertalan Forstner, Hassan Charaf[1]

Department of Automation and Applied Informatics
Budapest University of Technology and Economics
Goldmann György tér 3., 1111 Budapest, Hungary

Abstract

Model transformation means converting an input model available at the beginning of the transformation process to an output model. A widely used approach to model transformation uses graph rewriting as the underlying transformation technique. In case of diagrammatic languages, such as the Unified Modeling Language (UML), the exclusive topological matching is found to be not enough. To define precisely the transformation steps beyond the topology of the visual models, additional constraints must be specified which ensures the correctness of the attributes, or other properties to be enforced. Dealing with OCL constraints provides a solution for these unsolved issues, because topological and attribute transformation methods cannot perform and express the problems which can be addressed by constraint validation. The use of OCL as a constraint and query language in modeling is essential. We have shown that it can be applied to model transformations as well. Often, the same constraint is repetitiously applied in many different places in a transformation. It would be beneficial to describe a common constraint in a modular manner, and to designate the places where it is to be applied. This paper presents the problem of crosscutting constraints in transformation rules, and provides an aspect-oriented solution for it. Our approach makes it possible to define constraints separately from the transformation steps, and facilitates specifying their propagation assignment to graph transformation rules. To illustrate the conceptual results, a case study is also provided, which introduces (i) how our approach generates user interface handler source code for mobile platform from a resource model and a statechart diagram, and (ii) how it validates specific properties during the transformation steps using aspect-oriented constraints.

Keywords: Metamodel-Based Model Transformation Rules, Crosscutting Constraints, Aspect-Oriented Constraints, Constraint Weaving, OCL, VMTS

[1] Emails: {lengyel, tihamer, gmezei, cyberci, hassan}@aut.bme.hu

doi:10.1016/j.entcs.2005.10.020

1 Introduction

OMG's Model Driven Architecture [16] offers a standardized framework to separate the essential, platform-independent information from the platform-dependent constructs and assumptions. A complete MDA application consists of a definitive platform-independent model (PIM), one or more platform-specific models (PSM) and complete implementations, one on each platform that the application developer decides to support. The platform-independent artifacts are mainly UML and other software models containing enough specification to generate the platform-dependent artifacts automatically by so-called model compilers. Hence, software model transformation provides a basis for model compilers, which plays a central role in the MDA architecture.

The increasing demand for visual languages (VL) in software engineering (e.g., Unified Modeling Language - UML; Domain-Specific Languages - DSLs) requires more sophisticated transformation mechanisms for diagrammatic languages. Although these VLs can often be modeled with labeled, directed graphs, the complex attribute dependencies peculiar to the individual software engineering models cannot be treated with this general model. Consequently, often it is not enough to transform graphs based on the topological information only, we want to restrict the desired match by other properties, e.g. we want to match a node with a special integer type property whose value is between 4 and 12.

Previous work [15] has shown that the rules can be made more relevant to software engineering models if the metamodel-based specification of the transformations allows assigning OCL [17] constraints to the individual transformation steps. Because these constraints are bound to the rewriting rules, they are able to express constraints local to the host graph area affected by the rules. This approach is inherently a local construct, because the elements not appearing in graph transformations cannot be directly included in the OCL statements. Although the specification has this local nature, it does not mean that validating them does not involve checking other graph elements in the input graph: constraint propagation needs to be taken into account by both the algorithms and the user of the transformation. The OCL constraints enlisted in the transformation steps affect the matched instances of the rules.

Often, the same constraint is repetitiously applied in many different places in a transformation, therefore crosscuts the transformation steps. Aspect-Oriented Software Development (AOSD) [2] provides a new technique to address an emerging separation of concerns (SoC) that is focused on crosscutting. The methods of AOSD facilitate the modularization of crosscutting concerns within a system. Aspects may appear in any stage of the software development lifecycle (e.g. requirements, specification, design and implementation).

Crosscutting concerns can range from high-level notions of security to low-level notions, like caching. Furthermore, functional requirements such as business rules and non-functional requirements, like transactions can also be expressed by aspects. AOSD has started on the programming level of the software development life-cycle, and over the last decade several aspect-oriented programming languages have been introduced (e.g. AspectJ [3]). Aspect-oriented programming eliminates the crosscutting concerns on the programming language level, but the aspect-oriented techniques must also be applicable on a higher abstraction level. It would be beneficial to describe a common constraint in a modular manner, and propagate it automatically to the adequate places [12] [13]. This work presents an aspect-oriented method to propose a solution to eliminate the crosscutting concerns from the constraints of the transformation rules. Our approach provides the possibility to specify constraints separately, and to assign them to model transformation rules.

2 Background and Related Work

Graph rewriting [21] is a powerful tool for graph transformation with a strong mathematical background. The atoms of the graph transformation are rewriting rules, each rewriting rule consists of a left-hand-side graph (LHS) and a right-hand-side graph (RHS). Applying a graph rewriting rule means finding an isomorphic occurrence (match) of LHS in the graph to which the rule is applied (host graph), and replacing this subgraph with RHS. Replacing means removing the elements that are in LHS but not in RHS, and gluing the elements that are in RHS but not in LHS.

Models can be considered special graphs, which contain nodes and edges between them. This mathematical background makes it possible to treat models as labeled graphs and to apply graph transformation algorithms to models using graph rewriting [15]. Previous work [15] has introduced an approach, where the LHS and RHS of the rules are built from metamodel elements. This means that an instantiation of the LHS must be found in the host graph instead of the isomorphic subgraph of the LHS. The LHS and RHS nodes are referred to as pattern rule nodes (PRNs).

The Object Constraint Language (OCL) [17] is a formal language for the analysis and design of software systems. It is the subset of the UML standard [18] that allows software developers to write constraints and queries over object models. A constraint is a restriction on one or more values of an object-oriented model or system: A precondition to an operation is a restriction that must be true at the moment that the operation is going to be executed. Similarly, a postcondition to an operation is a restriction that must be true

at the moment that the operation has just ended its execution.

Aspect-oriented programming (AOP) [6] is based on the idea that computer systems are programmed in a better way by separately specifying the various crosscutting concerns of a system. Then the program relies on the mechanisms of the underlying AOP environment, which weaves or composes the separated pieces into a coherent program. AOP regards scattered concerns as first-class elements, and eject them horizontally from the object structure [6]. A concern whose code becomes tangled into other structural elements becomes a mess. To ameliorate this problem, AOP offers the notion of aspects: mechanisms beyond subroutines and inheritance for localizing the expression of a crosscutting concern. AOP systems also provide some mechanism for weaving the aspects and the base code into a coherent system.

An aspect-oriented approach is introduced in [7] for software models containing constraints, where the dominant decomposition is based upon the functional hierarchy of a physical system. This approach provides a separate module for specifying constraints and their propagation. A new type of aspect is used to provide the weaver with the necessary information to perform the propagation: the strategy aspect. Strategy aspect provides a hook that the weaver may call in order to process the node-specific constraint propagations.

Visual Modeling and Transformation System (VMTS) [26] is an n-layer multipurpose modeling and metamodel-based transformation system. Using this environment it is easy to edit metamodels, to design models according to their metamodels, and to transform models using graph rewriting [15]. Moreover, VMTS facilitates checking constraints contained by metamodels during the metamodel instantiation, and to validate the constraints enlisted in the rewriting rules during the graph transformation process [11].

The metamodel-based constraint checking method of the VMTS benefits from the results of the mathematical background of formal languages, graph rewriting and research related to the metamodel-based software model transformation. It also incorporates several ideas from the following environments.

The GReAT framework [8] [9] [23] is a transformation system for domain-specific languages (DSL) built on metamodeling and graph rewriting concepts. Its attribute transformation is specified by a proprietary attribute mapping language. The LHS of the GReAT rules can contain OCL constraint to refine the pattern.

PROGRES [19] [20] [22] is a visual programming language in the sense that it has a graph-oriented data model and a graphical syntax for its most important language constructs. In PROGRES the precondition of a transaction is a query, which should never fail, applied to the input graph of the surrounding transaction. Similarly the postcondition of a transaction is a query, which

should never fail applied to the output graph of the surrounding transaction. It is also allowed to access the in- and out-parameters of its transaction.

The Attributed Graph Grammar (AGG) [1] [5] [24] system is a Java-based visual programming environment. It also applies textual elements in the AGG language. The "visual programming" is performed by graph rewriting. Constraints can be specified either visually or textually. The textual constraint expressions are provided in Java.

3 Problem Statement

A *precondition* assigned to a rewriting rule is a Boolean expression that must be true at the moment when the rewriting rule is fired, and a *postcondition* assigned to a rewriting rule is a Boolean expression that must be true after the completion of a rewriting rule. If a precondition of a rewriting rule is not true then the rewriting rule fails without being fired. If a postcondition of a rewriting rule is not true after the execution of the rewriting rule, then the rewriting rule fails. A direct corollary of this is that an OCL expression in the LHS is a precondition to the rewriting rule, and an OCL expression in the RHS is a postcondition to the rewriting rule.

Using a simplified case study, this paper discusses the concept of the cross-cutting constraints in metamodel-based rewriting rules and provides a solution applying aspect-oriented technologies.

With the help of the case study we introduce how VMTS generates source code for a mobile platform from a resource model and a statechart diagram, applying graph-rewriting-based transformation methods. Furthermore we present how it validates specific properties using the OCL constraints enlisted in the rewriting rules. The aim of this method is that if the statechart is specified in detail, then the generated code will handle the user interface described by the resource model.

Fig. 1 introduces the block diagram of the case study. The input of the case study is a resource model which describes the user interface and a statechart diagram which represents the operation of the resource model. Input models are prepared with VMTS Adaptive Modeler using the VMTS Resource and Statechart modeling languages [26]. The models represent the resource and the operation of the Cinema Ticket application user interface, which is used on mobile platform to order cinema tickets using a cellular phone. The resource model (Fig. 2) - created based on the resource metamodel [26], which is a Domain Specific Model - contains a form with the necessary controls (*Page*, *ButtonBar*, *Menu*, *MenuItem*, *RadioButtonList* and *Slider*). The incomplete statechart diagram of the form is presented also in Fig. 2, where only five events

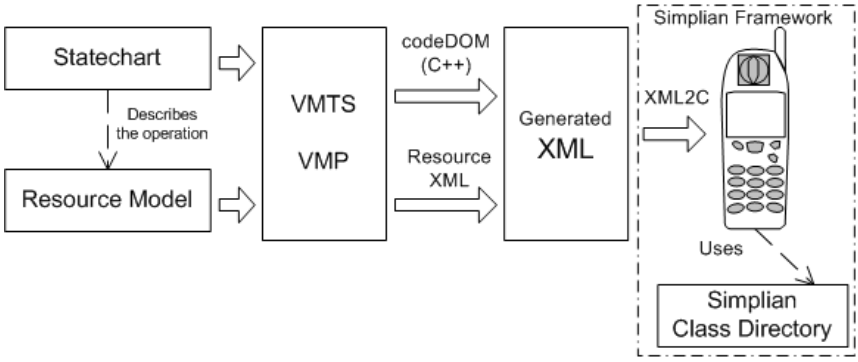


Fig. 1. Block diagram of the case study

are modeled: *On_rblCinema_SelectedItemChanged*, *On_miAddOrder_OnClick*, *On_miSendOrders_OnClick*, *On_miEditOrder_OnClick* and *On_rblFilmTitle_SelectedItemChanged*. The complete statechart diagram is too large to present here, but further details can be found in [26]. In the case study the required operations of the events are simple ones and can be modeled with one-one handler state, but often we have to use several states to model the behavior of an event [14].

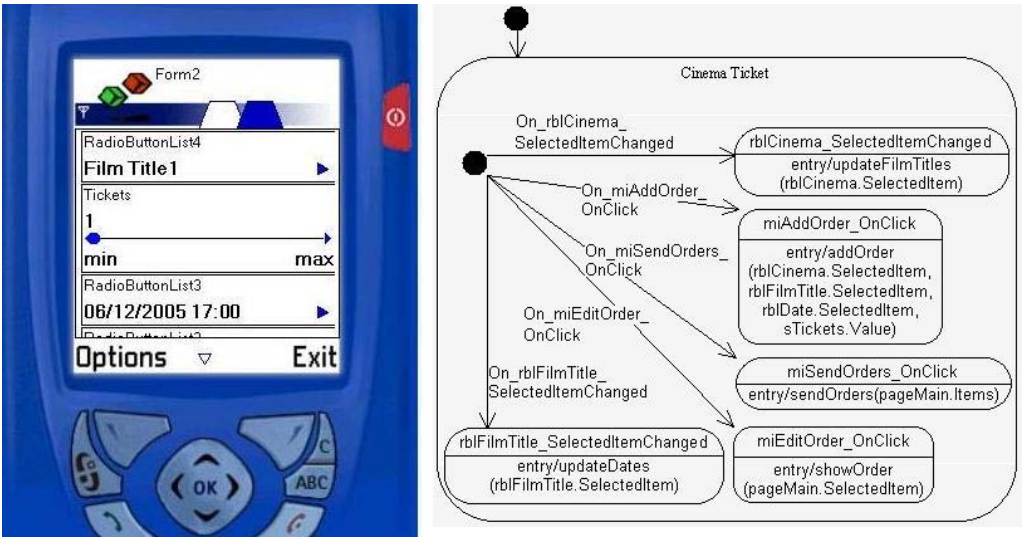


Fig. 2. Resource model and statechart diagram of the case study in VMTS

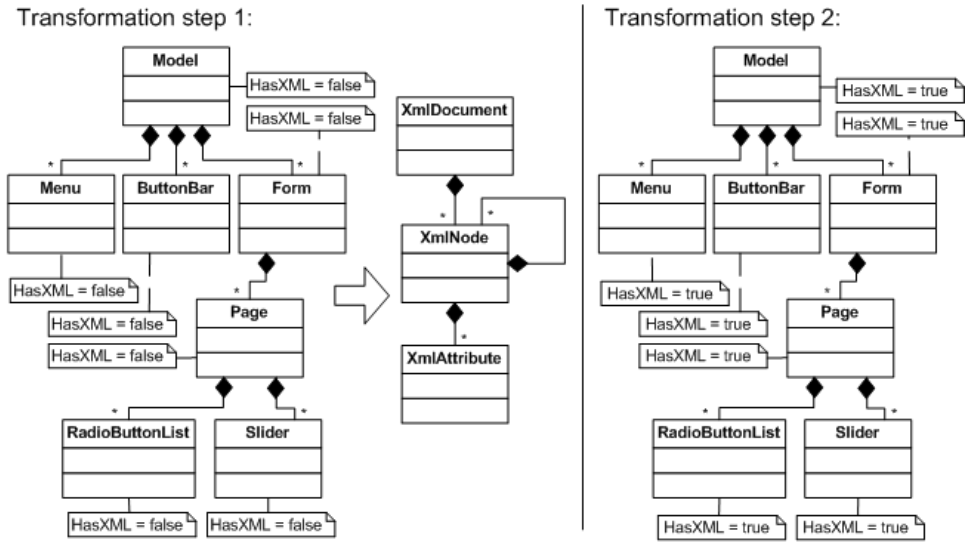
VMTS Visual Model Processor (VMTS VMP) processes the input models, using graph rewriting: (i) From the statechart diagram VMTS generates a CodeDOM tree [15] for each event handler. The CodeDOM tree is a model representation of the source code. It means that the code generation is a syntax tree composition, from which the .NET Framework [25] generates the

source code using the System.CodeDom namespace. (ii) The resource model is transformed into another model which represents an XML file and contains all information related to the user interface. In the XML file there are tags also for the source code snippets of the event handler functions. Therefore the last step of the transformation is that the generated source code is merged into the generated XML file. The result of this transformation and merging is also an XML file which is the input of the XML2C [26] application we use to generate the C++ source code for Symbian platform [4]. The generated source code uses the Simplian Class Directory and runs on a Symbian OS based cellular phone (Fig. 1). The XML2C tool and the Simplian Class Directory are part of the Simplian framework, which makes easier the programming for Symbian platform.

In general a transformation consists of several steps, often not only a transformation rule but a whole transformation is required to validate, preserve or guarantee a certain property. To meet this expectation all the transformation steps have to be taken into consideration. If one defines a constraint for more transformation steps or for a whole transformation, then the same constraint can appear in each transformation step several times, therefore the constraint crosscuts the whole transformation. Its modification and deletion is not consistent, because such an operation has to be performed on all occurrences of the constraint. Moreover, it is often difficult to estimate the effects of a complex constraint when it is scattered across the numerous PRNs of the transformation rules [12] [13].

Fig. 3 depicts the transformation with two rewriting steps that we used to transform the resource model into XML. We require the first rewriting rule to match only those controls which do not have a generated XML model ($HasXML = false$). If the first step finishes its transformation successfully, then the other node sets the value of the *HasXML* property to true. The second transformation contains only the LHS graph whose goal is to check whether the *HasXML* property of the controls affected by the first step is true. If the second step can match the required subgraph then the transformation was successful and it means that (i) each matched control has a generated XML and (ii) the transformation produced the valid and desired result. Although, two rules are similar the purpose of the first one is transformation but the aim of the second one is validation with constraint checking.

The second corollary is very important. A transformation contains steps specified properly using constraints, and the transformation has been executed successfully for the input model. Therefore the generated output model is in accordance with the expected result described by the steps of the transformation refined with the constraints. It means that the modeler's task is to



Model	Create	XmlDocument
Menu	Create	XmlNode
Menu	Create	XmlAttribute
ButtonBar	Create	XmlNode
ButtonBar	Create	XmlAttribute
Form	Create	XmlNode
Form	Create	XmlAttribute
Page	Create	XmlNode
Page	Create	XmlAttribute
RadioButtonList	Create	XmlNode
RadioButtonList	Create	XmlAttribute
Slider	Create	XmlNode
Slider	Create	XmlAttribute

Fig. 3. The rewriting rules of the *resource to XML* transformation with scattered constraints

create proper transformation steps and fully specify them with constraints (pre- and postconditions) then if the execution of the transformation finishes successfully. Thus, it produces a valid result.

In a rewriting rule we can connect the LHS elements to the RHS elements. This relation between the LHS and RHS elements is called causality [8], which facilitates assigning an operation to this connection. Causalities can express modification or removal of an LHS element, and creation of an RHS element. In Fig. 3 the causalities are enlisted in the table. The create operation and the attribute transformation are accomplished by XSL scripts. The XSL scripts can access the attributes of the object matched to the LHS elements, and produce a set of attributes for the RHS element to which the causality point to. VMTS stores models as labeled graphs, and each node and each edge have a property XML, which contains the attributes of the model element. In the current case study the VMTS rewriting engine concatenates the property XMLs of the matched states and edges, and it uses the result as the input of the XSL script [15].

The open issue with respect to the transformation presented in Fig. 3 is

that the same constraints appear several times and crosscut the rules.

4 Aspectifying Transformation Constraints

The disadvantage of assigning the constraints to the rules directly [11] can be seen in many tangling constraints throughout the rewriting rules. We need a mechanism to separate this concern. After having separated the constraints from the PRNs, a weaver method is needed to facilitate the propagation (link) of constraints to PRNs (elements of the transformation steps). The Global Constraint Weaver (GCW) algorithm [12] is passed a transformation with an optional number of transformation steps and a constraint list. The GCW algorithm propagates the constraints to the PRNs contained by the transformation steps.

This method means that our novel approach manages constraints using AO techniques [12]. Similarly to aspects, the constraints are specified and stored independently of any graph rewriting rule or PRN and are linked to the PRNs by the Global Constraint Weaver. The GCW selects the PRNs based on their types. The output of the weaver is not stored as a new rewriting rule; the result is represented as a link between the constraints and a transformation rule. This link is referred to as the Weaving Configuration. Fig. 4 presents our transformation with aspect-oriented constraints. The constraint is not directly linked to the PRNs, dashed lines denote the constraint assignment created by the constraint weaver automatically.

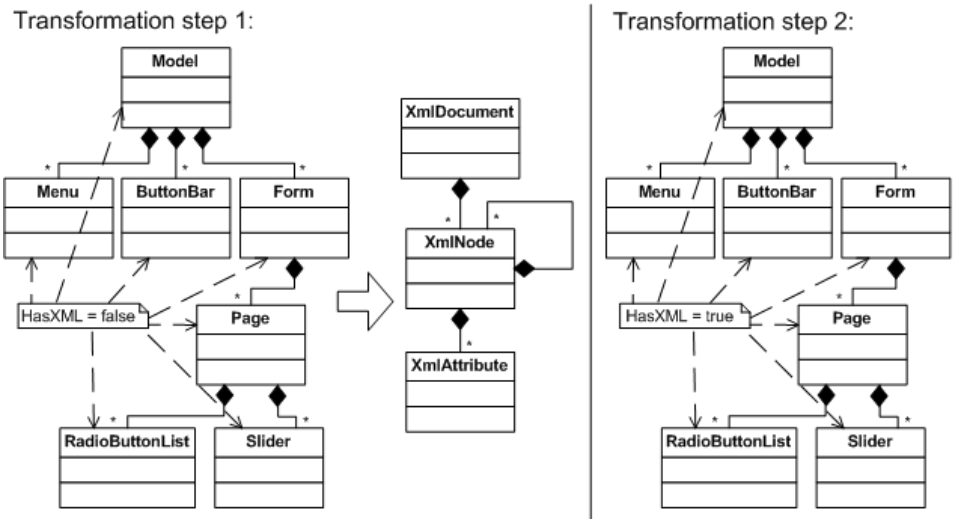


Fig. 4. The rewriting rules of the *resource to XML* transformation with aspect-oriented constraints

To summarize the main idea of the AO constraints, we can say that one

can create the constraints and the rewriting rules separately - and with the help of a weaver - constraints can be propagated to the PRNs contained by the transformation steps based on their type information.

The second transformation of the case study uses the statechart model (Fig. 2) as an input model and applies the rewriting rule depicted in Fig. 5. In the rewriting rule the LHS graph uses the meta-elements of the Statechart metamodel [18] [26] and the RHS graph uses the meta-elements of the CodeDOM metamodel. On the left hand side of the rewriting rule there are two states which correspond to the statechart state, and there is a transition between them with a 0..* multiplicity on the side of the target state. This means that applying this rewriting rule exhaustively to a statechart model, it matches all the states with their target adjacent states. The rule has to match the accessible adjacent states, because we need them to generate the state-transitions into the source code. Of course, it is possible that a state has no outgoing transitions, and the reason why we enable the 0 in the multiplicity is that we want to match states having only incoming transitions in order to generate CodeDOM tree for them as well. As it is mentioned above the code generation means a syntax tree generation (CodeDOM tree) from which the framework generates the C++ source code.

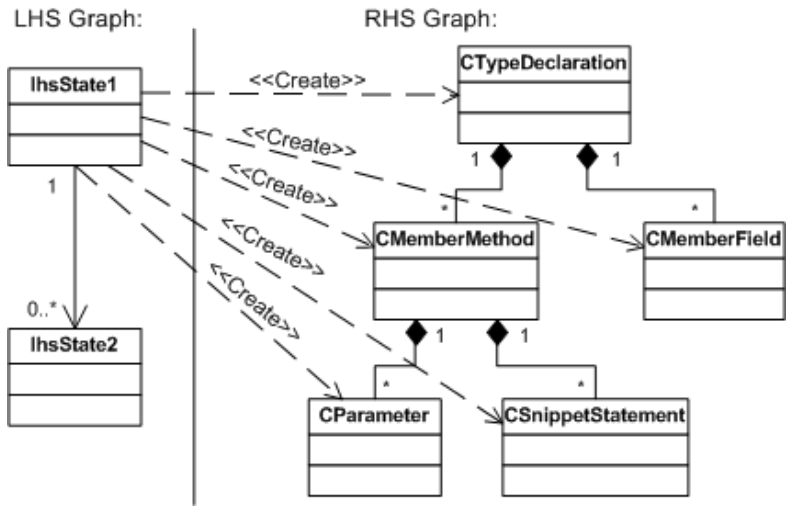


Fig. 5. The rewriting rule of the *statechart to XML* transformation

For the complete case study please refer to [26].

In the case of this example the constraints are moderately scattered across the model, but the situation could be worse. Assume the case that we have a transformation with several (10 or more) transformation steps, which modify the properties of *Person* type objects and we would like the transformation to

validate that the *age* property of a *Person* is always under 200 ($Person.age < 200$). It is certain that the transformation preserves this property if the constraint is defined for all PRN of type *Person*. This means that the constraint can appear in each transformation step several times, therefore the constraint crosscuts the whole transformation.

5 Summary

Extending the metamodel-based transformation steps with OCL constraints makes the transformations sophisticated enough to parse diagrammatic graphs with optional conditions.

We have found that the source of our rewriting problems is often related to the lack of support for modularizing crosscutting concerns. As we have adopted an aspect-oriented approach to our metamodel-based transformation process, it was observed that the maintainability and understandability of our transformation steps have been increased along with the attached constraints.

In this paper the problem of the crosscutting constraints in the rewriting rules is presented with the help of a case study, and an aspect-oriented solution is provided. Constraints are specified and stored independently of any graph rewriting rule, and they are linked to the pattern rule nodes by a weaver algorithm.

The most important benefits of the aspect-oriented constraint management in metamodel-based model transformations are the following: (i) the same constraint does not appear repetitiously in many different places, (ii) consistent constraint modification, simple constraint removal. (iii) Moreover, it is not necessary for the rewriting rules to be aware of the constraints, nor for the modeler who creates the rewriting rules. Rewriting rules are applicable with or without constraints, in accordance with the AOSD principles.

The aspect-oriented constraint management does not replace the classical constraint assignment; it extends the possibilities of constraint handling in metamodel-based model transformation frameworks.

The main limitation of the aspect-oriented constraint management is that it requires more preprocessing steps than the general approach. We have to define the constraints and rewriting rules separately and then specify the propagation of the constraints to the transformation steps.

References

[1] The Attributed Graph Grammar System (AGG) Homepage, http://tfs.cs.tu-berlin.de/agg

[2] AOSD Homepage, http://www.aosd.net/

[3] The AspectJ Programming Guide, http://www.aspectj.org

[4] Leigh Edwards et al: Developing Series60 Applications. Addison-Wesley, 2004.

[5] H. Ehrig, G. Engels, H-J. Kreowski, G. Rozemberg (ed.), "Handbook on Graph Grammars and Computing by Graph Transformation: Application, Languages and Tools", Vol.2. World Scientific, Singapore, 1999.

[6] Robert E. Filman, Tzilla Elrad, Siobhan Clarke, Mehmet Aksit, "Aspect-Oriented Software Developement", Addison-Wesley, 2004.

[7] Jeff Gray, Ted Bapty, Sandeep Neema, and James Tuck, "Handling Crosscutting Constraints in Domain-Specific Modeling", Communications of the ACM, October 2001, pp. 87-93.

[8] Gabor Karsai, Aditya Agrawal, Feng Shi, Jonathan Sprinkle, *On the Use of Graph Transformation in the Formal Specification of Model Interpreters*, Journal of Universal Computer Science, Special issue on Formal Specification of CBS, 2003.

[9] Gabor Karsai and Aditya Agrawal, "Graph Transformations in OMG's Model-Driven", Applications of Graph Transformations with Industrial Relevance, To be published in the Lecture Notes of Computer Science, Charlotsville, VA, December, 2003.

[10] Gabor Karsai, Janos Sztipanovits, Akos Ledeczi and Ted Bapty, "Model-integrated development of embedded software", Proceedings of the IEEE, 91(1):145-164, 2003.

[11] L. Lengyel, T.Levendovszky, P. Kozma, H. Charaf, "Compiling and validating OCL constraints in metamodeling environments and visual model compilers", IASTED on SE, February 15-17, 2005, Innsbruck, Austria, pp. 48-54.

[12] L. Lengyel, T. Levendovszky, H. Charaf, "Weaving Crosscutting Constraints in Metamodel-Based Transformation Rules", 8th International Conference on Information Systems Implementation and Modeling, ISIM '05, April 19-20, 2005, Hradec nad Moravic, Czech Republic, pp. 119-126.

[13] L. Lengyel, T. Levendovszky, H. Charaf, "Managing Crosscutting Constraints in Metamodel-Based Model Transformation Frameworks", Proceedings of 6th International Carpathian Control Conference, ICCC'2005, Volume II, Miskolc-Lillafred, Hungary, May 24-27, 2005, pp. 41-46.

[14] L. Lengyel, T. Levendovszky, H. Charaf, "Implementing an OCL Compiler for .NET", In Proceedings of the 3rd International Conference on .NET Technologies, Pilsen, Czech Republic, May-June 2005, pp. 121-130.

[15] T. Levendovszky, L. Lengyel, G. Mezei, H. Charaf, "A Systematic Approach to Metamodeling Environments and Model Transformation Systems in VMTS", Electronic Notes in Theoretical Computer Science, International Workshop on Graph-Based Tools (GraBaTs) Rome, 2004.

[16] OMG MDA Guide Version 1.0.1, OMG, doc. number: omg/2003-06-01, 12th June 2003, http://www.omg.org/docs/omg/03-06-01.pdf

[17] OMG Object Constraint Language Specification (OCL), http://www.omg.org

[18] OMG UML 2.0 Specifications, http://www.omg.org/uml/

[19] PROGRES http://mozart.informatik.rwth-aachen.de/research/projects/progres/main.html

[20] J. Reekers, A. Schürr, "Defining and Parsing Visual Languages", Journal of Visual Languages and Computing, 8, Academic Press, 1997, pp. 27-55.

[21] G. Rozenberg (ed.), "Handbook on Graph Grammars and Computing by Graph Transformation: Foundations", Vol.1 World Scientific, Singapore, 1997.

[22] A. Schürr , "Specification of Graph Translators with Triple Graph Grammars", Proceedings of the 20th International Workshop on Graph-Theoretic Concepts in Computer Science, LNCS 903, Berlin: Springer-Verlag; June 1994; 151-163.

[23] Jonathan Sprinkle, Gabor Karsai, *A Domain-Specific Visual Language For Domain Model Evolution*, Journal of Visual Languages and Computing, vol. 15, no. 2, April, 2004.

[24] Gabriele Taentzer, "AGG: A Graph Transformation Environment for Modeling and Validation of Software", In J. Pfaltz, M. Nagl, and B. Boehlen (eds.), Application of Graph Transformations with Industrial Relevance (AGTIVE'03), volume 3062. Springer LNCS, 2004.

[25] Thuan Thai and Hoang Lam, ".NET Framework Essentials", O'Reilly, 2003.

[26] VMTS Web Site, http://avalon.aut.bme.hu/∼tihamer/research/vmts