International Journal of Information and Computation Technology.
ISSN 0974-2239 Volume 3, Number 10 (2013), pp. 1107-1114
© International Research Publications House
http://www. irphouse.com /ijict.htm

Identify Crosscutting Concern for Constraint Based Model Transformation

Vishal Verma[1] and Anita Handa[2]

*[1]Department of Computer Science, Kurukshetra University
P.G Regional Centre, Jind, India.
[2]Department of Computer Science, Dronacharya Institute of Management & Technology, Kurukshetra, India.*

Abstract

Efficient synthesis of application based software is possible from their respective software model, when used with domain specific model processor. Very often, model compilers are implemented by transformation model which are based on graph rewriting. Visual Modeling and Transformation Systems provide facility to rewrite the meta-model based rules so that they can be used in Object Based Constraints Languages to model transformation rule. It supports validated model transformation. Validations whenever occur introduces new concern that may crosscut the functional concern of transformation rule. This type of concerns can be separated by applying on aspect oriented solution as constraint management. In this paper we introduce the method for identification of crosscutting constraints with respect to meta-model based model transformation rule. The discussed algorithms also support the reusability of the constraints and rewriting rules by improving understandability of model transformations.

Key-Component: Aspect –Oriented Programming, Aspect – Oriented Constraints. Constraint Weaving, Identifying Cross- Cutting Constraint, Transformation modeling.

1. Introduction

With the passage of time like other branches of software development, Aspect-Oriented Software Development (AOSD) also starts emphasis on the use of model-driven development approaches, examples are Model Integrated Computing (MIC) (Sztipanovits,Karsai, 1997) and Model Driven Architecture (Metzger, 2005). In AOSD it is emphasized that models must be used in all stages of system development. In the method suggested in MIC (Sztipanovits, Karsai, 1997)emphasis is given on the use of domain specific concepts for design of system and then further used for synthesis of executable system, along-with performing analysis to drive simulations. The benefit of same is increase in productivity, easy maintenance, easy implementation of development cycle. In contrast to method discussed in MIC (Sztipanovits, Karsai, 1997), MDA (Metzger, 2005) suggest to separate essential platform independent information from platform dependent information constructs and assumptions. Application developed by using MDA, on completion consist of a well-defined platform independent model, and one or more platform specific model (PSM); which include complete implementation for each and every platform as decided by the developer. All the platform independent variations are UML and other software models containing the necessary information about how to develop their respective platform dependent variations by using model compilers.

All the model driven approaches (Metzger, 2005)(Sztipanovits, 2002) works by considering any model transformation as the basic building block. Model Driven Software Development, uses the set of requirements and form a domain to specify how the system is to be modeled for a specific modeling paradigm. Every time the modeling paradigm is taken in the form of modeling language specifications, known as meta-model. After creation of meta-model for a set of requirements, modeling environment allow the modeler to create domain specific model, synthesis in any artifact latter on.

Model based development support any type of transformation as i) Aspect weaving, ii) Analysis and Verification(Assmann, 1996), iii) Design to Implementation. Availability of number of alternates for transformations enforces the availability of reusable transformation tools to validate model transformation. Testing of implementation of a model is easy and same can be achieved by using any of the traditional method like white box testing, mutation testing, unit testing etc. the hardest part is to do the testing of specification of transformation. Facility for off line transformation artifacts are not granted to give good results but online variants (Lengyel et al, 2005)(Lengyel 2006) are accepted to produce the tested results/artifacts. For example, transformation from class to relational data base management system class2rdbms must convert a class to a table with proper definition of table, fields, primary key, foreign key (if required) etc.

Object Constraint Language (OCL) (OMG OCL 2006) helps to achieve above discussed requirements by applying constraints on transformation rule. But in this case too, if same constraint is required to be applied on many transformations, then it start crosscut the transformation rule and becomes difficult to manage. Further, solution of this problem was suggested in (OMG QVT) with case study of class2rdbms, which said that transformation can be done by 9 transformation rule and 2 constraints, out of which first one appears 30 times in 9 transformation rule and second one 16 times in 6 rules. This all reflects that it is very difficult to manage it manually. With the use of Aspect-Oriented Constraints Management, a solution was suggested in (Lengyel et al, 2005)(Lengyel 2006), which suggest to proceed by considering constraints as aspect, Aspect Oriented Constraints are designed separately from transformation rule, and are woven back by using weaver method, before execution. This type of implementation supports the proper constraint management in terms of modification, deletion and propagation operations.

The work discussed in this paper is a method which can identify the cross-cutting constraints in model transformation. This method gets facilitated by the idea that designers prefer to design transformation rules with constraints. Hence, the emphasis is given on extracting constraints from rules. The discussed method is formulated by same algorithms which are applicable to many other problems of same nature too. Reusability of the designed and identified constrains is also possible.

2. Backgrounds

In this section of paper we bring the solution of problem discussed in visual modeling and transformation system and how it is applied to transformation constraints as aspects. Graph rewriting (Rozenberg, 1997) is also a powerful technique for graph transformation. Basic units of solution in this case is rewriting rule consisting of Left Hand Side Graph, Right Hand Side Graph. Steps of algorithm for this method mainly include searching for a position in graph on which LHS graph rule can be applied to obtain the RHS graph.

2.1 Visual Modeling and Transformation System

The system described in (Levendovszky et al, 2004) and (VMTS website) permit to edit a model as discussed in its meta model and include the constraints specified in Object Constraint Language (OMG OCL 2006). Directed labeled graph (part of its vocabulary) are used to formalize the models. Graph rewriting technique can also be used to in VMTS if required. Rules as part of algorithm helps to verify the constraints specified on transformation rule/process. It is the basic construct of VMTS that LHS and RHS graph are developed from the elements of the domain of the meta-model. Fig. 1 shows an example of how to develop a data base table from UML class along-with propagated rules like cons_r1, cons_r2, cons_rh1, cons_rt1. It is set of rules by using which properties of transformation rule can be validated(Lengyel 2006).

Context class invnonabstract;
Not self. abstract

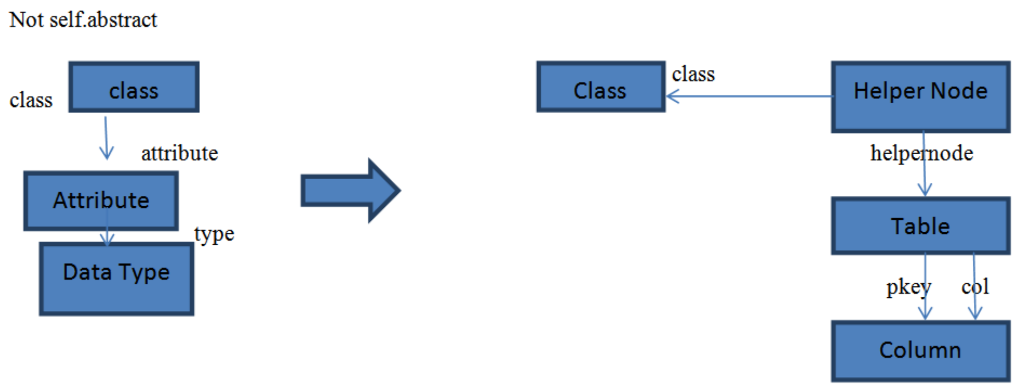

Fig. 1: Transformation Rule classtotable.

2.2 Description of Constraints:

Cons_r1: As clear from the fig.1 this constraint is applied on pattern rule node class in LHS of crtable rule, that in LHS it makes the pre-condition for the rule (process only abstract classes).

Contecttbaleinvprkey;

Self. columns→ exist (c| c.datatype = 'int' and c.is,prkey)

Cons_t1: This primary key constraint behave like post condition of the rule crtable, and is applied to rule node table. It is applied and suggested to enforce each table has primary key only of int type.

Context Atom invclassattandtablecol;

Self.class.attr→ for all

(self.table.column→ exists c| c.colname.class.attname))

Cons_h1: This class attribute and table column constraint is linked to the rule node table helper node. It is applied and suggested to enforce that each class attribute should have column with the same name in resultant table.

The overall purpose of all of the above constraints is to guarantee our requirement. We know it better that after execution of a rule with constraints we can have an valid acceptable output. Constructs which are supported by VMTS are as follows.

a. Sequencing with Transformation rule, b) Branching with OCL constraints, c) Hierarchical rule, d) Parallel Execution of rule, e) Iteration.

Above discussed transformation rule can be executed in two different ways as: Repeated mode, in which LHS keeps on match with the impact model and Multiple match model, in which all matched occurrences in the input model has been matched in one go and RHS execution is done for all the matched site on LHS. The matching criteria can be implemented in chained process also i.e. the RHS of a rule may match with LHS rule of next input (rule). Technically said RHS matching at level n may also match with LHS of level n+1. This model executionalways helpful to speedup the execution process and to control the complexity of execution.VMTS do support the aspect oriented technique based constraint management (Lengyel 2006), this is considered as unique since all decision are based on system variables along-with actual size of input model. VMTS support a very well defined method for model transformation (validated), control flow definition and constraint management.

3. Algorithm and Identification of AOP Constraints

In this section we discuss various algorithms for detecting cross-cutting constraints. The constraints are identified from transformation rule and control flow model with following idea of implementation.

a. Prepare list of constraints appearing in transformation rule.
b. Detect constraints.
c. Among all detected constraints, filter out the real repetitive constraints which is cross cutting for transformation.
d. Represent the identified constraints as aspect.

By following the basic principal of identification of concern, concerns must be identified having separate scope w.r.t complexity, adaptability, maintainability, robustness and especially reusability. All of the concerns then must be in one to one correspondence with the module through which they are implemented. In some complex situations it is not possible to implement above said idea for concern and single concern is spreaded over many other modules, giving cross cutting concern. It should not be considered as bad design of algorithm but should be treated as a constraint which cannot be over-come. This may lead to some serious disadvantages that it becomes hard to understand, extend, maintain and adapt.

We divide and discuss our approach of cross – cutting constraints identification as coded (sec 3.1) and extract (sec 3.2). Each of the user defined or system defined constraints (to be implemented) is assigned a separate code, if any two concerns appears with same code in different modules, they are considered as crossed, and required to be extracted (as aspect oriented constraint). Constraint decomposition is used for extraction(Lengyel 2006).

3.1 Coding Algorithm:

It starts with model transformation having propagated constraints and give output as code representing concern. The transformation designer also required to specify the related concerns. Constraint is used to represent an optional property of a concern which is used to represent the code e.g the value of an attribute. For example:

```
Context class invnonabstract
Not self. Abstract
```

Here, nonabstract is treated as constraint on a class with its name as its meaning.

```
Context table invsr_class
Self. helpnode.class→ exists (c| c.name = self.name)
```

Sr_class constraint enforce that generated table has source class with same name. Along with two above alternates we may face different situations if there are more than one concerns in one constraint.

Context class invnonabstractandprocess;

Not self.abstract and not selfisprocessed

The above constraint is designed to handle the complex situation where the matched class is non-abstract and non-processed. The considered concern are represented by meta OCL constarints from a SET, where element of set represent ID and evaluated for model transformation constraints. The steps of algorithm (pseudo code) are as follows

Algorithm 1

```
1. Coding (Transformation T, OCL set meta OCL): coded table
2. Coding table = new coding table();
3. For all constraints C in T do
4. For all constraints meta contraints in metaOCL do
5. If check_contraint_relevance (c,metaconstraint) then
6. Update_coding_table(coding table, c, metaconstraint.code)
7. End if
8. End for
9. End for
10. Return
```

As shown in algorithm meta OCL constraints are defined with attributes as (context, contraints,code) e.g for Boolean type (class, abstract, code) (class, isprocessed, code). Example of adjacent node is (class, self. helpnode, table$\rightarrow$sizeof()$\rightarrow$0,code N). By implementation all different constraints must have unique code.

Algorithm 1 is shown in the form of pseudo code of coding algorithm. Arguments transformation and meta OCL are passed for initialization and same iterates on the constraints propagated to the rule node of transformation (T). During various iterations meta OCL constraints are iterated and evaluation is done for meta OCL constraint with actual constraint. If it matches then code of meta constraint is assigned to constraint C. algorithm return the final code.

3.2 Extract Algorithm

Input to this algorithm, are model transformation as in code algorithm and output of code algorithm. This gives the output constraints extracted into aspects.If above algorithm gives unambiguous coding, then with reference to only one assigned code to each constraint, respective aspect can be created. In case of multi-coding scenario refactoring can be done to dissolve the design constraints. While working on domain of meta – model based model transformation, constraint reallocation and decomposition is applied to dissolve the cross cutting concerns.Algorithm 2 is the pseudo code of second algorithm, it uses constraint relocation and decomposition provided by constraint decomposition method (Lengyel 2006).

```
Algorithm 2
1. Extract (transformation T, coding table coding table): list of aspect
2. List of aspect aspect_list = new List of aspect (coding table.code.size)
3. Decompose transformation = decompose_constraint (T)
4. For all coding item coding_item in coding table do
5. For all code code in coding item do
6. Update List of aspect (aspect_list, code, decompose.getdecomposedconstraint
   (Codeitem.Getconstraint by code (code))
7. End for
8. End for
9. Return
```

Transformation T and coding table is used as argument to extract algorithm and decompose constraint of transformation (Lengyel 2006). Algorithm iterate on coding items, provided by coding and iterates on for each item on code assigned to actual coding.

4. Conclusion

Main problem dealt in this paper is the identification of crosscutting constraints in model transformation with reference to input. We proposed the solution by design of some algorithms which are able to identify the constraints with reference to the concerns and helps to represent them into aspects. This solution can further be improved by supporting the system which can do this task in automated way.

References

[1] A.Metzger, (2005) systematic look at model transformations, In Model Driven Software Development.Vol II of Research and Practice in Software Engineering.

[2] G. Rozenberg (ed.), (1997) Handbook on Graph Grammars and Computing by Graph Transformation: Foundation,vol I Word Scientific, Singapore.

[3] J. Sztipanovits,and G. Karsai (1997) Model – Integrated Computing,IEEE Computer, pp. 110-112

[4] J. Sztipanovits, (2002) Generative programming for embedded systems. In Generative Programming and Component Engineering (GPCE0, Vol 2487 of LNCS, pp. 32-49

[5] L. Lengyel, T. Levendovszky,H. Charaf, (2005)Eliminating Crosscutting Constraints from Visual Model Transformation Rules,ACM/IEEE 7th Int. workshop on AOM,MaontegoBay,Jamaica

[6] L.Lengyel, T. Levendovszky,H. Charaf (2005) Constraint Validation Support in Visual Model Transformation Systems, ActaCybermetica,ISSN 0324-721X,vol 17(2), pp. 339-357

[7] L.Lengyel (2006) Online Validation of Visual Model Transformation,Ph.D thesis, Budpest University of Technology and Economics, Department of Automation and Applied Informatics

[8] OMG OCL Spec,Version 2.0, (2006) http://www.omg.org/

[9] OMG QVT,MOF 2.0 Query/Views/Transformation Specification, http://www.omg.org/cgi-bin/apps/doc?ad/05-03-02.pdf

[10] T. Levendovszky,L.Lengyel,G.Mezei,H.Charaf, (2004)A Systematic Approach to Metamodeling Environments and Model Transformation System in VMTS, ENTCS

[11] U.Assmann, (1996) How to Uniformly Specify Program Analysis and Transformation, Int. Conference on Compiler Construction (CC) 96, LNCS 1060,Springer

[12] VMTS Website, http://www.vmts.aut.bme.hu