



HAL
open science

SAC-OCL: A Tool for A Semi Automatic Co-evolution of OCL Constraints

Djamel Eddine Khelladi, Reda Bendraou, Marie-Pierre Gervais

► **To cite this version:**

Djamel Eddine Khelladi, Reda Bendraou, Marie-Pierre Gervais. SAC-OCL: A Tool for A Semi Automatic Co-evolution of OCL Constraints. D&P@MoDELS 2016, Oct 2016, Saint-Malo, France. pp.40-47. hal-01475253

HAL Id: hal-01475253

<https://hal.sorbonne-universite.fr/hal-01475253v1>

Submitted on 15 Jun 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

SAC-OCL: A Tool for A Semi Automatic Co-evolution of OCL Constraints

Djamel Eddine Khelladi¹, Reda Bendraou^{1,2}, Marie-Pierre Gervais^{1,2}

¹ Sorbonne Universités, UPMC Univ Paris 06, UMR 7606, F-75005, Paris, France.

² Université Paris Ouest Nanterre La Defense, F-92001, Nanterre, France.

{firstname.lastname}@lip6.fr

Abstract. Object-Oriented Models (OOM) are widely used in Model-Driven Engineering (MDE). As a complement, OCL constraints are used to specify business rules and detailed aspects of the business domain. Automatically co-evolving OCL constraints w.r.t. the evolved OOM is challenging since alternative resolutions can be applied but also a resolution cannot be applied to any arbitrary constraint. In this paper, we present SAC-OCL an Eclipse-based tool to semi automatically co-evolve OCL constraints when an OOM evolves. In contrast to existing tools, SAC-OCL proposes alternative resolutions for an impacted OCL constraint and proposes only the appropriate resolutions that can be applied on the impacted constraint. Our evaluation on six case studies shows an average correct co-evolution rate of 92%.

1 Introduction

In *Model-Driven Engineering*, Object-Oriented Models (OOM) play an essential role to capture the different aspects of a system [5]. This covers the system's architecture, its data structure or its design and GUI classes. A OOM alone is insufficient to capture all the relevant aspects and information of a domain specification [11]. To overcome this limitation, the Object Constraint Language (OCL) [12] is used to define constraints on top of OOMs. For instance, the Unified Modeling Language (UML) [14] metamodel (an OOM) in version 2.4.1 contains more than 750 OCL constraints expressing well-formedness rules.

Whenever an OOM is evolved, its OCL constraints may become invalid and need to be co-evolved accordingly. However, existing tools have so far overlooked two main issues: 1) *the existence of multiple and semantically different resolutions*, and 2) *a resolution can be applicable only to a subset of OCL constraints*.

1) An impacted OCL constraint by an OOM evolution can be resolved with syntactically and/or semantically different resolutions. For instance, the change "*multiplicity generalization of a property p from a single value to multiple values*" requires the OCL constraints to work on a collection of values, e.g. by introducing an iterator. Alternative resolutions must be considered since multiple iterators with *different semantics* exist, e.g. `forAll()`, `exists()` etc.

2) A given resolution strategy is not always applicable for all OCL constraints. Figure 1 illustrates this issue. It depicts an evolution of an OOM where the property *score* is *pushed* from the superclass `Course` to the subclass `Project`. The

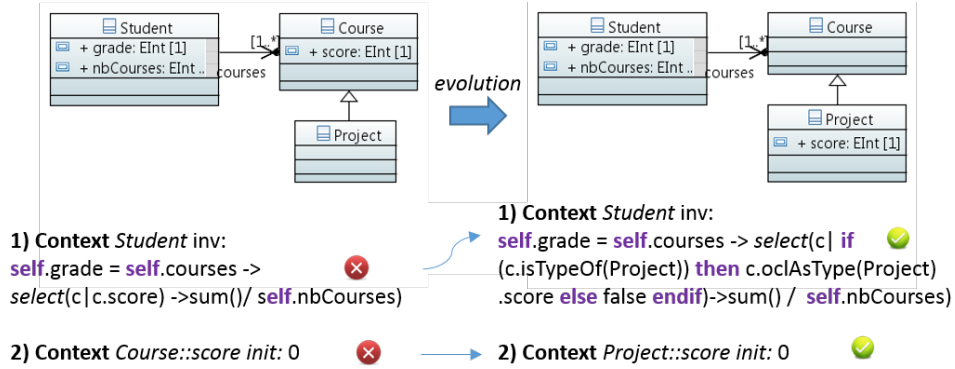


Fig. 1: An evolution and co-evolution example.

two constraints become invalid because *score* is no more accessible in *Course*. The first constraint that uses the pushed property *score* through the reference *courses*, can be co-evolved by introducing an If expression that first checks whether *courses* references an instance of the subclass *Project* so that *score* is accessible. In contrast, the second OCL constraint whose context (i.e. the element on which an OCL constraint is specified) is defined on the pushed property *score*, is co-evolved differently by updating its context. Clearly, a resolution strategy cannot be applied for all impacted OCL constraints by the same change. Existing tools [3, 2, 10, 9, 1] neither consider the two above issues, nor interact with the user.

In [8] we addressed these issues by presenting an efficient approach for a semi automatic co-evolution of OCL constraints. We proposed a set of resolutions while considering alternative resolutions per impacted OCL constraint. We identified with a systematic analysis three influencing factors that we rely on to propose the appropriate resolutions that can be applied to an impacted OCL constraint. To detect the OOM's evolution we use our previous tool [6, 7].

This paper presents SAC-OCL a tool that supports our previous work with a graphical user interface in the form of an Eclipse plugin. It allows to co-evolve OCL constraints that are impacted by the OOM's evolution. To demonstrate the usefulness of SAC-OCL, we validate it on six open-source case studies. The results show that we are able to co-evolve the impacted OCL constraints near to the user intent by reaching an average correct co-evolution rate of 92%.

2 Tool's Description

The conceptual framework of SAC-OCL has been already published in [8]. In this section we present an overview of its main aspects. Figure 2 shows the workflow phases of the tool: [1] detecting OOM's evolution, [2] identifying impacted OCL constraints, [3] proposing alternative and the appropriate resolutions for each impacted OCL constraint, and [4] applying automatically the resolutions.

The entry point of our tool is the detected OOM's changes during evolution [1]. We start by loading the evolution trace and the specified OCL constraints

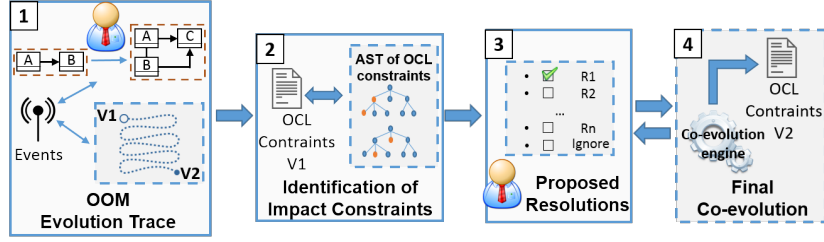


Fig. 2: Overall approach and the SAC-OCL workflow.

Table 1: Impact Identification on OCL constraints.

OOM Elements	OCL Constraints	References To AST Nodes
e_1	OCL_1, \dots	ref_1, ref_2, \dots

to be parsed. Phase [2] runs an impact analysis to identify the impacted OCL constraints that need to be co-evolved. After that, the appropriate resolutions are proposed for each impacted constraint [3]. The user confirms the resolution to be applied for each impacted constraint among the proposed alternative ones. Finally, the selected resolutions are automatically performed in phase [4].

2.1 OOM's Evolution

During evolution two types of changes are considered in phase [1]: a) *Atomic changes* that are additions, removals, and updates of a metamodel element, and b) *Complex changes* that consist in a sequence of atomic changes combined together [4]. Our previous tool AD-ROOM [6] considers all atomic changes and seven complex changes: *move property*, *pull property*, *push property*, *extract super class*, *flatten hierarchy*, *extract class*, and *inline class* [4].

2.2 Identification of Impacted OCL Constraints

To identify the impacted OCL constraints we run an impact analysis at the level of the Abstract Syntax Tree (AST) of the OCL constraints (phase [2]). Before identifying where an AST is impacted, we compute the Table 1 that lists for each metamodel element e , all OCL constraints using e with references to the AST nodes using e . Those references will be further used in the resolution step.

Thus, for each change on an OOM element e_i , we access the impacted OCL constraints and we access exactly the impacted AST nodes. Note that during the co-evolution, Table 1 is updated accordingly w.r.t. the applied resolutions. For instance, when a rename element e occurs, it is also renamed in the table.

2.3 Resolutions and Selection Process of the Appropriate Ones

In [8] we first conducted a systematic analysis that allowed us to identify three influencing factors that must be considered to propose the appropriate resolu-

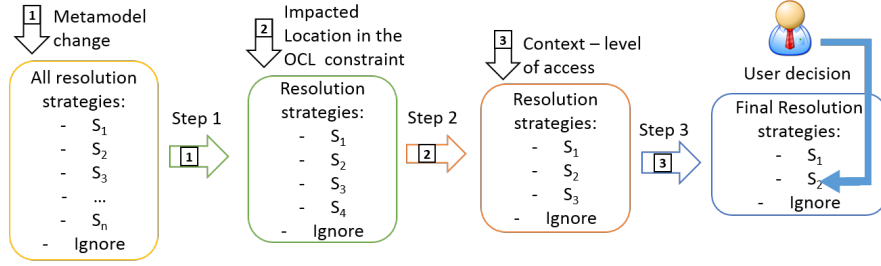


Fig. 3: Process of selecting the appropriate resolution strategies per impacted part of a constraint and per metamodel change.

tions for an impacted OCL constraints. The three factors are highlighted in the framebox here under.

1. The OOM change of an element e .
2. Location of the impacted element e in an OCL constraint:
 - (a) e is used in the context of an OCL constraint
 - (b) e is used in the OCL expression through a navigation path.
3. The context of the impacted constraint, to know from which class the impacted element is accessed.

We then proposed a set of resolutions to co-evolve the impacted OCL constraints (can be found in [8]). When defining each resolution we already specify under which influencing factors each resolution will be proposed. Each resolution is implemented as a transformation function that updates the AST by adding, removing, or updating nodes.

When an OCL constraint is impacted by an OOM change, the process of Figure 3 is used to select a subset of the appropriate resolutions to be proposed to the user (phase [3]). It starts with all the available resolutions from Table ?? and excludes a subset of resolutions based on the influencing factors. The first factor we consider is the metamodel change that excludes resolutions that are not associated to the given change (step 1). After that the two last factors are similarly considered (step 2 and 3) to further reduce the set of appropriate resolutions. This process is followed for each impacted part of an OCL constraint.

3 Implementation

The tool is implemented as an Eclipse IDE plugin to ease its use by software engineers who are already familiar with development IDEs like Eclipse. It supports the co-evolution of OCL constraints defined in an OCL file on both Ecore [15] and UML CD [14] models. The core functionalities are implemented with Java (4568 LoC) interfacing with our AD-ROOM [6] plugin.

Figure 4 part 1 shows the OOM to be evolved. Figure 4 part 2 shows the OCL constraints to be co-evolved. In a first step, the OOM evolution (atomic and complex changes) are taken as input with the OCL constraints. After performing the impact analysis, the impacted OCL constraints are highlighted to the user

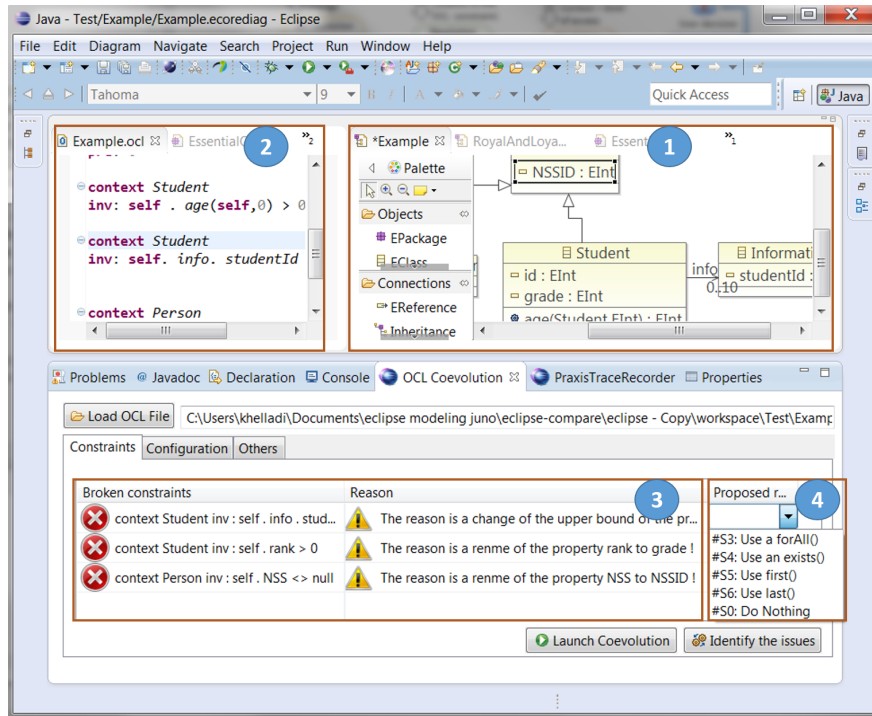


Fig. 4: Screenshot of the Eclipse plugin Tool SAC-OCL.

as well as the impacting reason (i.e. the OOM change) for a prompt and a better understanding, as shown in Figure 4 part 3. Figure 4 part 4 shows the subset of the proposed resolutions that the user chooses from in a dropdown menu for each impacted OCL constraint. The user's effort is to select the resolutions to be applied automatically afterwards (phase 4). See the demonstration video: <https://youtu.be/qLn2u0J2Wow>.

4 Evaluation

We evaluated our tool on six open-source case studies that are Ecore metamodels (OOMs) with their associated OCL constraints from the OMG specification, namely: the UML Class Diagram (UML CD), the Structured Metrics metamodel (SMM), the Unified Profile for DoDAF/MoDAF (UPDM), the National Information Exchange Model (NIEM), the EXPRESS Information Modeling Language (EXPRESS), and the Requirements Interchange Format (ReqIF).

The evaluation process consisted in comparing how the OCL constraints were manually co-evolved in practice against how the same OCL constraints are co-evolved with our tool. In the experiment, the authors play the user role. We aimed at co-evolving the OCL constraints *as close as possible* to the co-evolution in practice, for an objective comparison. We highlight the obtained results.

Table 2: Correct Co-evolution Percentage (%).

	UML	CD	SMM	UPDM	NIEM	EXPRESS	ReqIF
Syntactically correct co-evolution %	72%	83%	100%	100%	100%	100%	100%
Semantically correct co-evolution %	80%	83%	100%	100%	100%	100%	100%

Table 2 presents the results of the measured co-evolution percentage. For four case studies, we reached 100% co-evolution percentage, whereas in UML CD and SMM, we had respectively 72%, 80% and 83% of co-evolution percentage. An average of 92% (syntactically) and 93% (semantically) correct co-evolution is reached in our evaluation. These results show that our approach can efficiently support the user with a co-evolution near to her intent and needs.

However, some constraints were co-evolved differently in our approach than in practice. We further investigated these cases in particular for UML CD and SMM case studies. We noticed that those cases were due to a refactoring that changed their semantic either by removing/adding a part in the constraint. Or by replacing some elements in the original constraint with other elements that are differently typed. For example, constraint 1 is impacted by the renaming of `GeneralizableElement` to `PackageableElement`. It is co-evolved by our approach as follows from (1) to (2) by propagating the rename. However, in practice the context of constraint (1) was changed to the subclass `Classifier` instead of applying a rename. Therefore, the semantic is slightly different since the applicability scope of the new constraint is reduced to elements of type `Classifier`.

```
context GeneralizableElement inv: not self.allParents()-> includes(self) (1)
context PackageableElement inv: not self.allParents()-> includes(self) (2)
```

5 Related Work

Here we give a succinct related work to OCL co-evolution. Demuth et al. [2] proposed an approach for OCL co-evolution based on templates. They provided 11 templates that define a fixed structure for OCL constraints. Hassam et al. [3] proposed to co-evolve OCL constraints using QVT [13] a transformation language. Similarly, Markovic et al. [10] proposed to refactor, based on QVT, OCL constraints annotated on UML class diagrams when these last evolve. Kusel et al. [9] discussed the impact of metamodel evolution on OCL expressions and proposed to resolve impacted expressions. Cabot et al. [1] focused on the metamodel change delete element. In particular, they aimed at removing only a sub part of the OCL constraint that is using the deleted element.

All existing approaches [2, 3, 10, 1, 9] consider only the metamodel change as a factor to propose a resolution. Thus, they define for each metamodel change only a unique resolution. In contrast, we propose alternative resolutions while considering three influencing factors instead of one, i.e. the OOM change.

6 Conclusion

We presented SAC-OCL a tool for a semi automatic co-evolution of OCL constraints defined on OOMs. SAC-OCL is designed to consider alternative res-

olutions and to propose the appropriate resolutions for each impacted OCL constraint. It is implemented as a plugin for the Eclipse IDE a wide-spread development environment for software developers. The evaluation illustrates our tool's ability to co-evolve OCL constraints near to the user intent while releasing the burden of manual co-evolution from the user.

Acknowledgment. The research leading to these results has received funding from the industrial innovation Project MoNoGe under grant FUI - AAP no. 15.

References

1. J. Cabot and J. Conesa. Automatic integrity constraint evolution due to model subtract operations. In *Conceptual Modeling for Advanced Application Domains*, pages 350–362. Springer, 2004.
2. A. Demuth, R. E. Lopez-Herrejon, and A. Egyed. Supporting the co-evolution of metamodels and constraints through incremental constraint management. In *MODELS*, pages 287–303. Jan. 2013.
3. K. Hassam, S. Sadou, V. L. Gloahec, and R. Fleurquin. Assistance system for ocl constraints adaptation during metamodel evolution. In *CSMR*, pages 151–160. IEEE, 2011.
4. M. Herrmannsdoerfer, S. D. Vermolen, and G. Wachsmuth. An extensive catalog of operators for the coupled evolution of metamodels and models. In B. Malloy, S. Staab, and M. v. d. Brand, editors, *SLE*, pages 163–182. Jan. 2011.
5. J. Hutchinson, J. Whittle, M. Rouncefield, and S. Kristoffersen. Empirical assessment of mde in industry. In *ICSE'11*, pages 471–480. ACM, 2011.
6. D. E. Khelladi, R. Bendraou, and M.-P. Gervais. Ad-room : a tool for automatic detection of refactorings in object-oriented models. In *The 38th ICSE*, 2016.
7. D. E. Khelladi, R. Hebig, R. Bendraou, J. Robin, and M.-P. Gervais. Detecting complex changes during metamodel evolution. In *CAISE'15*, pages 264–278, 2015.
8. D. E. Khelladi, R. Hebig, R. Bendraou, J. Robin, and M.-P. Gervais. Metamodel and constraints co-evolution: A semi automatic maintenance of ocl constraints. In *International Conference on Software Reuse*, pages 333–349. Springer, 2016.
9. A. Kusel, J. Etlstorfer, E. Kapsammer, W. Retschitzegger, J. Schoenboeck, W. Schwinger, and M. Wimmer. Systematic co-evolution of ocl expressions. In *11th APCCM 2015*, volume 27, page 30, 2015.
10. S. Markovic and T. Baar. Refactoring OCL annotated UML class diagrams. *Softw Syst Model*, 7(1):25–47, Feb. 2008.
11. G. Mezei, T. Levendovszky, and H. Charaf. An optimizing ocl compiler for meta-modeling and model transformation environments. In *Software Engineering Techniques: Design for Quality*, pages 61–71. Springer, 2006.
12. OMG. Object constraints language (ocl). www.omg.org/spec/OCL/, 2015.
13. OMG. Query/views/transformations (qvt). www.omg.org/spec/QVT/, 2015.
14. OMG. Unified modeling language (uml). www.omg.org/spec/UML/, 2015.
15. D. Steinberg, F. Budinsky, E. Merks, and M. Paternostro. *EMF: eclipse modeling framework*. Pearson Education, 2008.