Trade-offs for Model Inconsistency Resolution

Anne Keller, Hans Schippers and Serge Demeyer

University of Antwerp
Middelheimlaan 1, 2020 Antwerp, Belgium

Abstract. With the advent of model-driven software development, models gain importance in all steps of the software development process. This makes the problem of consistency between models an important challenge. Models in large projects typically contain many inconsistencies, where each of them can be resolved in multiple ways with different effects on the system, thus making the decision on a resolution strategy a hard one. In this paper, we present metrics that assess the impact of a resolution operation on the model and the computational cost of a resolution operation. We propose to use these metrics to compare different resolution strategies.

1 Introduction

In a model-driven process, as for example Model-Driven Architecture (MDA), models are the main software development entities and artefacts. On the one hand, models change in time, e.g., model elements are added or removed. On the other hand, high-level platform-independent models are transformed into low-level platform-specific models that are the basis for code generation. In a software development process that so heavily relies on models, consistency management of these models is crucial.

Inconsistencies between models occur when a prior defined consistency rule is checked and violated. The cause for inconsistencies ranges from distributed work on the models, over incomplete specifications, to evolution of models.

For each inconsistency there exist a number of resolution operations that resolve the inconsistency completely. However, the resolution operations are not equivalent. Additional to their differences in semantics (i.e., how they actually resolve the inconsistency), they can differ in the impact they have on the model, as well as in the computational cost of their application. Choosing a resolution operation that implies the least changes to the overall model (i.e., low impact) can be an as relevant decision as choosing one that is applied in few steps (i.e., low computational cost). The difficulty lies in assessing the effects of a resolution operation and incorporating this knowledge into the choice of resolution operation.

In this paper we introduce metrics to judge the impact and computational cost of a resolution operation and propose to use these metrics to choose a resolution operation with a favourable trade-off point.

The paper is structured as follows, in Section 2 we introduce the metrics, Section 3 contains the related work, and Section 4 concludes the paper.

2 Metrics for Inconsistency Resolution

Evaluating Impact. Our starting observation is that applying changes to model elements which are heavily used by other model elements should be avoided. Thus, to avoid changing heavily used model elements the resolution operation with the lowest impact would be preferred. The rationale behind this idea is that changing such model element is likely to cause changes in the other model elements as well. Such changes include modifications to a semantically important class ('god class') and changes to a referenced model element. Changing a semantically important class will also modify the expected semantics for the related model elements (e.g., for all classes connected by association). Changes to a referenced model element result in having to check, and possibly modify, the existing references. In Table 1 we present a preliminary collection of *impact relationships* that capture a number of these relevant connections between model elements in UML Class and Sequence Diagrams. To calculate the impact we check how often each model element involved in a resolution operation is taking part in such a impact relationship. In order to put the amount of connections in relation with the design of the overall model, we divide the number of occurrences of a specific impact relationship by the maximum number of occurrences in the model. For resolution operations consisting of several operations to different model elements the impact values are added. The more impact relationships a model element takes part in, the higher the impact of this model element in the given model.

Computational Cost. Whereas the impact metric allows for determining model elements which we would like to avoid applying changes to, it fails to take into account how hard it is, computationally, to actually carry out a certain resolution operation. In other words, we would like to express the idea that a resolution operation which involves a small number of computational steps is preferable over a more expensive alternative. To this end, we introduce the computational cost as a second metric. This is especially relevant when dealing with large and complex models that contain many inconsistencies. In such models an often applied, computationally expensive resolution will lead high execution time and thus to poor performance.

Computational cost considers modification and navigation operations on the model as a data structure. The computational steps which we take into account are on the one hand operations on the data structure, i.e., modifications, creations, deletions of model elements and on the other hand navigational steps in the model. Thus, the steps needed to execute a resolution depend on the UML meta-model.

Example. We explain the usage of the metrics on a short example. Figure 2 shows an example of the 'Dangling Connectable Feature Reference' inconsistency as specified in [5]. This inconsistency occurs when a message in a sequence diagram references an operation not found in the corresponding class. In the example the `AddEventRequest` operation is not found in the `Calendar` class. The available options to resolve this inconsistency are to *1)* add a new operation to the `Calendar` class in the class diagram (*AddOp*), *2)* remove the message `AddEventRequest` from the sequence diagram (*RmvMsg*), *3)* change an existing operation in the class diagram to match the existing `AddEventRequest` message (*ChngOp*) and *4)* change the `AddEventRequest` message in the sequence diagram to refer to an existing operation in the `Calendar`

Class	· is type of an attribute · has instance · has association to other class · contains an operation · contains an attribute · is generalisation
Operation	· is called by a message · is inherited
Attribute	· is parameter of an operation · is inherited
Object	· is instance of a class · has message calls to it
Message	· references an operation · is create or delete message · has reply message

Table 1. Impact Relationships.

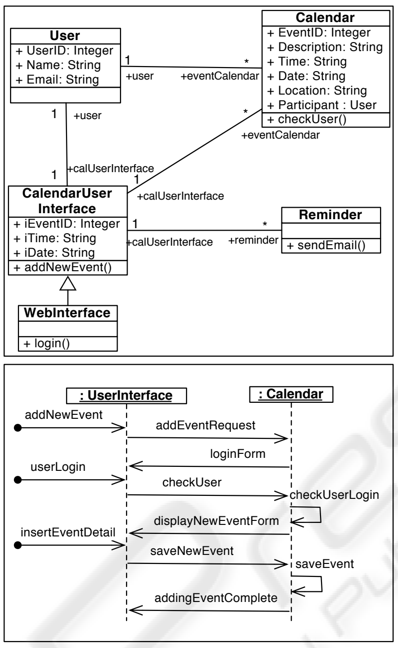

Table 2. Inconsistent UML Class and Sequence Diagram.

class (*ChngMsg*). Each resolution operation consists of several steps. For example, *AddOp* consists of the steps: *a)* create a new operation, *b)* name the operation, *c)* assign the operation to a class.

Calculating impact for the example, we obtain *AddOp* (1.1), *RmvMsg* (0), *ChngOp* (1.1) and *ChngMsg* (0). The computational cost is *AddOp* (5), *RmvMsg* (11), *ChngOp* (6) and *ChngMsg* (4). Comparing the results for impact and computational cost we see that each metric values the application of the shown resolution operations differently. For example, while according to the impact *RmvMsg* and *ChngMsg* are preferred, the computational cost metric ranks *RmvMsg* as the operation with the highest cost. That is, while the `AddEventRequest` message is a model element whose modification is judged to have a low impact, it is defined in such a way that its removal is a costly operation.

3 Related Work

Kuester et al. [1] evaluate inconsistency resolutions based on side-effect expressions and the resolution's overall cost reduction. Different to this approach, we do not relate the resolution operations to a set of defined inconsistencies. Instead, we compare the resolutions based on the changes they pose to the whole model.

In [2], [3] Mens et al. describe how they use critical pair analysis of graph transformation rules in order to evaluate dependencies between inconsistency resolutions such as mutual exclusion and sequential dependency. Different to their work, we do not relate inconsistency resolutions of different consistency rules with each other. Instead, we evaluate each resolution independent of other possible inconsistencies.

Spanoudakis et al. [4] use object-oriented metrics to evaluate the significance of inconsistencies. Spanoudakis et al. analyse the significance of inconsistencies rather than single resolution operations. A similarity exists in the premise that significance of inconsistencies depends on the significance of the involved model elements.

4 Conclusions and Future Work

In this paper, we introduced two metrics to support the process of inconsistency resolution. The impact metric determines to which extent a resolution operation will cause a model to change. The computational cost of a resolution operation expresses how hard it is, technically, to actually perform the operation. We consider these metrics an important step towards reifying the trade-offs involved in the complex process of deciding which resolution operation to apply in a given situation of inconsistency.

There are two main parts of future work. In a first phase, the process of calculating our metrics will be automated by means of a tool which operates on a model repository. In a second phase, using this tool, we want to show the relevance of our metrics for inconsistency resolution. This validation will need to take place on large scale case study.

Once the metrics are automated and validated, there is room to refine the metric calculation functions. For example, the impact calculation can be refined by assigning a weight to each impact relationship and thus respecting the fact that different impact relationships have different semantics. (For example, compare the semantics of a UML composition and aggregation.) Also, adding weights to the different kinds of resolution operations (e.g., add, remove) can prevent that one kind of resolution operation (e.g., an addition) is always preferred over another type of resolution operation. For the computational cost, weights should be assigned to each operation, for example, deletion computational costs more than navigation. All of this remains to be investigated and is future work.

References

1. J. M. Küster and K. Ryndina. Improving inconsistency resolution with side-effect evaluation and costs. In *MoDELS*, pages 136–150, 2007.
2. T. Mens and R. Van Der Straeten. Incremental resolution of model inconsistencies. In *WADT 2006*, volume 4409 of *Lecture Notes in Computer Science*, pages 111–126. Springer-Verlag, 2007.
3. T. Mens, R. Van Der Straeten, and M. D'Hondt. Detecting and resolving model inconsistencies using transformation dependency analysis. In *MoDELS*, pages 200–214, 2006.
4. G. Spanoudakis and H. Kim. Diagnosis of the significance of inconsistencies in object-oriented designs: a framework and its experimental evaluation. *J. Syst. Softw.*, 64(1):3–22, 2002.
5. Ragnhild Van Der Straeten. *Inconsistency Management in Model-Driven Engineering: An Approach using Description Logics*. PhD thesis, Vrije Universiteit Brussel, 2005.