Model Repair and Transformation with Echo

Nuno Macedo, Tiago Guimarães, Alcino Cunha
HASLAB — High Assurance Software Laboratory
INESC TEC & Universidade do Minho, Braga, Portugal
{nfmmacedo,tguimaraes,alcino}@di.uminho.pt

***Abstract*—Models are paramount in model-driven engineering. In a software project many models may coexist, capturing different views of the system or different levels of abstraction. A key and arduous task in this development method is to keep all such models consistent, both with their meta-models (and the respective constraints) and among themselves. This paper describes Echo, a tool that aims at simplifying this task by automating inconsistency detection and repair using a solver based engine. Consistency between different models can be specified by bidirectional model transformations, and is guaranteed to be recovered by minimal updates on the inconsistent models. The tool is freely available as an Eclipse plugin, developed on top of the popular EMF framework, and supports constraints and transformations specified in the OMG standard languages OCL and QVT-R, respectively.**

I. Introduction

Model-Driven Engineering (MDE) is an approach to software development that focuses on models as the primary development artifact. In MDE, different models, conforming to different meta-models, capture different views of the same system (typically different models are used to specify structural and dynamic features) or may be used at different levels of abstraction (code is obtained by refining platform-independent models to platform-specific ones).

During the development process, user updates will undoubtedly produce inconsistencies which must eventually be repaired. Manually performing such repairs is in most cases unfeasible, due to model size and the complexity of meta-model constraints and inter-model consistency rules. Tool support for this task is essential, to automate as much as possible model repair and update propagation from one model to the remaining, to recover consistency. In order to be effective and predictable, such repairs cannot simply generate new consistent models from scratch, but must instead produce models that are as close as possible to the original ones.

To support MDE, the *Object Management Group* (OMG) has launched the *Model-Driven Architecture* (MDA) initiative, which prescribed the usage of the *Meta-Object Facility* (MOF) and *Object Constraint Language* (OCL) for the specification of (object oriented) meta-models and constraints over them. To specify inter-model consistency, the OMG prescribed the usage of bidirectional model transformations, and proposed the *Query/View/Transformation Relations* (QVT-R) [1] declarative language to specify them. Unlike MOF and OCL, QVT-R has yet to be widely accepted, much due to the unpredictable and ambiguous semantic defined in the standard, which has hindered effective tool development.

Inconsistencies can be repaired by two main techniques. One is to derive repair plans by syntactic analysis of the constraints and model instances. The other is through model finding, by using a solver to calculate a new model that satisfies the constraints. While the former is usually more efficient and scales better, it is less expressive and flexible than the latter. For example, it is not as well suited to deal with multiple inconsistencies, nor inconsistencies that affect a large portion of the model (likely to occur when using closures to express reachability properties). Moreover, they usually rely on a fixed set of abstract edit operations to specify the repairs, which the user must manually instantiate and is not able to parametrize. Most existing intra-model repair tools [2]–[4] fall into the first class of repair techniques. To achieve high efficiency, they typically limit the expressiveness of repair updates, do not require the generation of fully consistent models, or require constraints to be manually annotated with inconsistency resolution hints. The few existing tools based on model finding [5], [6] do not abide to the desirable *principle of least-change* [7], that requires repaired models to be as close as possible to the original. Effective support for inter-model consistency repair (namely via bidirectional QVT-R transformations) is even scarcer. Most tools plainly ignore meta-model constraints and may output inconsistent models [8]–[10], or are not bidirectional, allowing only the creation of fresh consistent models, ignoring the existing (inconsistent) ones.

This paper presents Echo, a tool for model repair and transformation based on model finding. While less scalable that some of the existing tools, Echo is: more *expressive*, allowing the annotation of meta-models with rich OCL constraints and the specification of QVT-R bidirectional model transformations; more *flexible*, being able to check and repair both intra- and inter-model consistency, and providing control over repairs by letting the user specify the allowed edit operations; *correct*, in the sense that resulting models are always fully consistent; and *minimal*, in the sense that it follows the clear and predictable *principle of least-change*.

Echo is deployed as an Eclipse plugin, developed on top of the popular *Eclipse Modeling Framework* (EMF), with meta-models being specified in ECore with embedded OCL constraints. Its engine works by translating both meta-models (annotated with OCL) and QVT-R transformations to Alloy [11], a lightweight formal specification language with support for automatic model finding via SAT solving. Even with such model finding approach, Echo is already effective at handling realistic medium sized models, and is particularly

978-1-4799-0215-6/13 © 2013 IEEE

ASE 2013, Palo Alto, USA
Tool Demonstrations

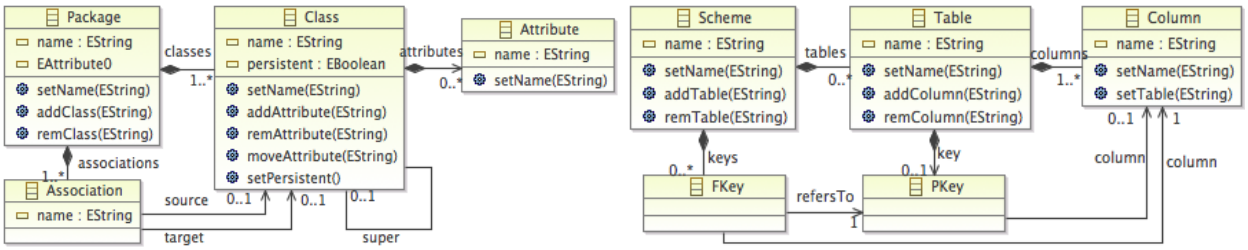

Fig. 1. Class diagrams of the UML and RDBMS meta-models.

useful for exploring and debugging meta-model constraints and inter-model consistency rules. In particular, it already helped us unveil several errors in the well-known object-relational mapping that illustrates the QVT-R standard [12].

II. The Echo Framework

Echo's environment consists of a set of meta-models with internal constraints $(A, B, C, \dots)$ and a set of QVT-R bidirectional transformations $(R, S, T, \dots)$ that specify inter-model consistency. The focus of the tool is to help manage the evolution of a set of models $(a, b, c, \dots)$, keeping them consistent with the respective meta-models (the fact that a conforms to all the constraints of A will be denoted by $a :: A$) and the inter-model consistency rules specified in the QVT-R transformations (the fact that a and b are R-consistent will be denoted by $(a, b) :: R$). Two models can be kept consistent by multiple QVT-R transformations simultaneously, and the same transformation can relate multiple pairs of models.

Echo was designed to be used in an *online setting*, in the sense that the consistency tests are automatically applied as the user is editing the models, flagging faulty models whenever inconsistencies are found. The user can then ask the tool to repair the models – Echo produces repairs that follow the *principle of least change*, so that the suggested repaired models will be as close as possible to the original inconsistent ones. This process is inherently non-deterministic, as there may be more than one consistent model at minimal distance. Echo presents all possible repaired models by increasing distance, allowing the user to choose the desired one, at which time the update is effectively applied to the original model.

As a running example, we rely on a simplified version of the classic object-relational mapping transformation that illustrates the QVT-R specification [1]. Fig. 1 depicts a simplified version of the object and relational meta-models, including possible edit operations. Over those meta-models, a QVT-R transformation `Uml2Rdbms` is defined, whose goal is to map every `persistent Class` in a `Package` to a `Table` in a `Scheme` with the same `name`. Each `Table` should contain a `Column` for each `Attribute` (including inherited ones) of the corresponding `Class`. As for `Associations`, each one is mapped into a foreign key `FKey` in the relational scheme. A constraint of the `UML` meta-model that cannot be captured by class diagrams alone is the requirement that the `super` association must be acyclic. One must resort to OCL to express it, for example by adding the invariant:

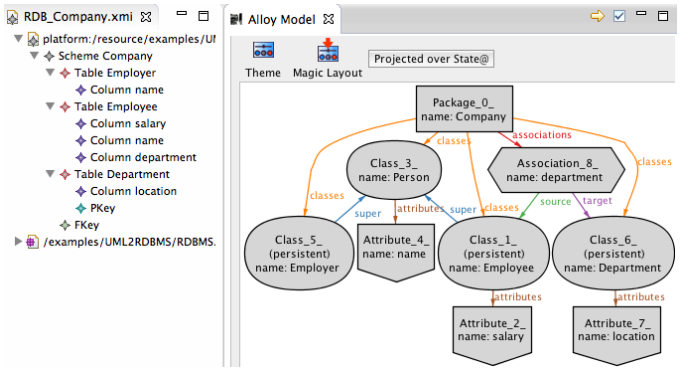

Fig. 2. A snapshot of Echo, with RDBMS and UML models depicted in EMF and in the embedded Alloy visualizer, respectively.

```
context Class inv:
    not self.closure(super)->includes(self)
```

A pair of (the respective) meta-model and `Uml2Rdbms`-consistent instances is depicted in Fig. 2. The class diagram captures a very simple company model, where there are `Employees` and `Employers`, which are `Persons`, the former connected to a `Department` via an association. Classes `Employee`, `Employer` and `Department` are flagged as persistent, hence the relational scheme on the left-hand-side.

A. Features

Echo currently supports the following features:

1) Model Visualization: Models are presented using the Alloy visualizer, as seen in Fig. 2 where an `UML` model is depicted. For better readability, an Alloy theme is automatically inferred from the meta-models. A user-defined theme can also be provided if desired.

2) Model Generation: Given a meta-model A and user-specified size, Echo can generate a new model a such that $a :: A$. This is useful, for example, in the creation of new models to kick-start model development: due to meta-model constraints models may not be allowed to be empty, and Echo generates the smallest model that satisfies such constraints. It is also useful to verify meta-model consistency and scenario exploration, allowing the user to verify if the meta-model can be inhabited by models satisfying given constraints.

3) Consistency Check: Given a model a, Echo can check if $a :: A$. As the models evolve, inconsistencies will indubitably occur which may break the meta-model implicit and explicit

OCL constraints. For instance, imagine the user inserts a new `super` association from `Person` to `Employee` in the example of Fig. 2, breaking the OCL constraint specified above. Echo will immediately flag this model as inconsistent.

4) Model Repair: Given a model a that does not conform to A, Echo can find a *minimal* repair that produces an a' such that $a' :: A$. While inconsistencies should temporarily be tolerated during development, they will eventually need fixing. Echo can enumerate all repaired models a' at increasing distances from a, letting the user choose the desired one. Proceeding with our example, if Echo is asked to repair the now inconsistent model it would present two alternatives at minimal distance: either remove the newly inserted `super` association from `Person` to `Employee` or remove the previously existing one from `Employee` to `Person`.

5) Inter-model Consistency Check: Given a QVT-R transformation R and models $a :: A$ and $b :: B$ that are supposed to be consistent via R, Echo can check if $(a, b) :: R$. The checking semantics follows exactly that specified in the QVT standard [1]. For instance, inserting a new `Column Location` on the `Table Department` from Fig. 2 would flag both models as inconsistent.

6) Inter-model Consistency Repair: QVT-R specifications are interpreted as bidirectional transformations, and, given a transformation R and models $a :: A$ and $b :: B$ such that $(a, b) :: R$ does not hold, Echo is able to repair either a or b to recover consistency. For example, Echo can find a *minimal* repair that produces a b' such that $b' :: B$ and $(a, b') :: R$. Once again, the user is able to choose the desired model from all minimal consistent models. In our running example, Echo would present a single minimal repair, namely insert a new `Attribute Location` in the `Class Department`.

7) Inter-model Generation: Given a QVT-R transformation R and a model $a :: A$, Echo can find the minimal $b :: B$ such that $(a, b) :: R$. This is achieved by combining the previously presented model generation and inter-model consistency recovery features: first a b' that conforms to B is produced, which is then updated to a b that is R-consistent with a.

B. Implementation

Echo's engine is built on top of Alloy [11], a lightweight formal specification language with support for both model checking and automatic model finding via SAT solving. MDE artifacts are translated into Alloy, and then we rely on these features to check consistency and find repairs, respectively. The technical details of such translations have been presented in previous work [12], [13]. Alloy also has an instance visualizer, to which we resort to depict models to the user.

At the core of Echo we have the following translations:

- ECore $\rightarrow$ Alloy — embeds ECore meta-models in Alloy [13];
- OCL $\rightarrow$ Alloy — translates the OCL constraints over the meta-model to Alloy [13];
- XMI $\leftrightarrow$ Alloy — translates XMI models to and from Alloy instances [12];
- QVT-R $\rightarrow$ Alloy — embeds QVT-R transformations, as well as the underlying semantics, in Alloy [12].

The key component of our tool is a target oriented model finding procedure, built on top of Alloy's solver, that finds minimal updates to an Alloy instance that satisfy a given set of constraints, details of which can be found in [12]. Essentially, minimal updates are found by asking the solver to find consistent instances at increasing distances from the original inconsistent one. Since models can be seen as graphs, the standard *graph edit distance* (GED), that just counts additions and deletions of nodes and edges, can be used to measure such distance. Echo automatically infers this measure for any given meta-model. A user-parametrized model distance measure is also available, that requires the user to specify the allowed edit operations in the meta-model, enabling a finer degree of control over valid repairs.

To better discern the impact of the two metrics, consider that a new `Column` named `salary` is inserted in the `Employer Table`. The minimal repairs on the `UML` model according to GED are either setting `Employee` as a super-`Class` of `Employer` or moving the `Attribute salary` from `Employee` up to `Person`. If none of these are desirable repairs, the user can ask for the next closest solution, which in this case consists of the introduction of a new `Attribute salary` in `Employer`. Suppose the user wants to rule out all repairs that change the `Class` hierarchy or assign the same cost to either create a new attribute or move an attribute from one class to another. To do so, he can specify (using OCL) which are the valid edit operations that can be performed to repair a model (as depicted in Fig. 1). Notice that there are no edit operations that modify the hierarchy, and both creation and moving of an attribute are now atomic edit operations. In this case, the Echo engine tries to find the minimal sequence of edit operations that repairs the model. In our running example, there will now be two minimal repairs, namely insert the new `Attribute salary` in `Employer`, or moving the existing one from `Employee` to the common super-`Class` `Person`.

C. Deployment

Echo is deployed as an Eclipse plugin[1], developed in accordance to the EMF. It resorts to the *Model Development Tools* (MDT) and *Model-to-Model Transformation* (MMT) components to parse the OCL constraints and QVT-R specifications, respectively. EMF prescribes ECore for the specification of meta-models, while model instances are presented as XMI objects. To enhance the meta-models with additional constraints, we follow the technique proposed by MDT, of embedding the OCL constraints as meta-model annotations.

The plugin automates the features described in Section II-A according to the architecture described in the previous section. Thus, every time the user updates a model, the system automatically checks its consistency in relation to the other artifacts. If a model is flagged as inconsistent, the user can then ask Echo to repair it. The user is then able to iterate through the minimal repaired models and choose the desired one through the interface presented in Fig. 2.

[1]Download and more information about Echo available at http://haslab.github.io/echo/, including a video demonstrating the tool being used.

III. Related Work

As discussed in Section I, due to scalability issues, most existing model repair tools use some kind of syntactic analysis to generate repair plans. For example, *xlinkit* [14] derives all possible sequences of repair actions for elements breaking a model constraint. However, it only considers one inconsistency at a time, not taking into consideration possible negative side-effects. To alleviate this problem, *Model/Analyzer* [2] uses an efficient incremental consistency checker to withdraw repair operations that have negative side-effects, yielding more refined repair plans, presented as a repair tree. *Badger* [3], a tool built with Praxis (a Prolog-based toolset of model-independent Eclipse plugins to generate and check model consistency), generates repair plans using automated regression planning, given concrete, previously detected, inconsistencies. The order in which repair plans are suggested can be controlled by the user by assigning different costs to edit operations. The main problem of all these tools is that repair plans consist of sequences of abstract edit operations, which the user must instantiate in order to effectively repair the models. Instead, Echo can compute such repaired models automatically.

Very few model repair tools rely on solvers and model finders, and no existing tools abide to the principle of least-change, meaning that repaired models are not guaranteed to be as close as possible to the original. For example, in [5] the inconsistent model is just used as a lower bound during solving, meaning that the repairs can only be performed by adding new nodes and edges to the model graph. In [6] a repaired model is found by relaxing the bounds on some entities and associations: nodes and edges suspected of causing the inconsistencies are removed from the lower-bound, and the upper-bound is augmented to allow additions. Such potentially guilty model elements must be identified by an (unspecified) external tool, and again there are no guarantees the repairs will be minimal. Unlike Echo, none of these tools allow control over repairs via parametrized edit operations. [15] describes a technique for generating quick fixes for DSMLs using the CSP(M) solver. The technique guarantees that the number of inconsistencies on the model decreases, even if side-effects occur. This is achieved by applying every candidate fix to the inconsistent model and detecting and counting the inconsistencies in the resulting model.

A different approach is implemented in the *Beanbag* language [4]: together with the specification of consistency rules, this language allows the user (with a small overhead) to specify how models breaking such rules should be fixed. Rules can then be run either in checking or fixing mode to repair models.

Regarding tool support for QVT-R, ModelMorf [8] and Medini [9] are the main existing functional tools. Medini is an Eclipse plugin for a subset of the QVT-R language. Although popular, its (not formally specified) semantics admittedly disregards the semantics from the QVT standard (it does not support the *checkonly mode* for instance). ModelMorf allegedly follows the QVT standard more closely (although, once again, the exact semantics is unknown), since its development team was involved in the specification of the standard. However, the development of the tool seems to have stopped.

Like Echo, the JTL tool [10] is solver based, although it does not support QVT-R, but rather a restricted QVT-like language. Repaired models are generated by resorting to the DLV solver, which is able to retrieve information from the original inconsistent model. However, it is not clear how the repaired model is generated, namely how the solver chooses which model elements to retain. It also forces the totality of the transformation, returning inconsistent models in the case that there is no consistent one. Unlike Echo, it does not allow for user-parametrized edit operations to control repairs. Some other prototype tools for QVT-R exist, but they are not bidirectional, in the sense that they only allow the execution of the transformation in one direction when generating new fresh models, completely ignoring the previously existing inconsistent ones. No existing QVT-R tools have support for OCL constraints on the meta-models, meaning that repaired models may be inconsistent.

Acknowledgments

This work is funded by ERDF - European Regional Development Fund through the COMPETE Programme (operational programme for competitiveness) and by national funds through the FCT - Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within project FCOMP-01-0124-FEDER-020532. The first author is also sponsored by FCT grant SFRH/BD/69585/2010.

References

[1] OMG, "MOF 2.0 Query/View/Transformation specification (QVT), version 1.1," January 2011, http://www.omg.org/spec/QVT/1.1/.
[2] A. Reder and A. Egyed, "Computing repair trees for resolving inconsistencies in design models," in *ASE*. ACM, 2012, pp. 220–229.
[3] J. Puissant, R. Straeten, and T. Mens, "Resolving model inconsistencies using automated regression planning," *SoSyM*, pp. 1–21, 2013.
[4] Y. Xiong, Z. Hu, H. Zhao, H. Song, M. Takeichi, and H. Mei, "Supporting automatic model inconsistency fixing," in *FSE*. ACM, 2009, pp. 315–324.
[5] M. Kleiner, M. Fabro, and P. Albert, "Model search: Formalizing and automating constraint solving in MDE platforms," in *ECMFA*, ser. LNCS, vol. 6138. Springer, 2010, pp. 173–188.
[6] R. Straeten, J. Puissant, and T. Mens, "Assessing the Kodkod model finder for resolving model inconsistencies," in *ECMFA*, ser. LNCS, vol. 6698. Springer, 2011, pp. 69–84.
[7] L. Meertens, "Designing constraint maintainers for user interaction," 1998, available at http://www.kestrel.edu/home/people/meertens.
[8] Tata Research Development and Design Centre, "ModelMorf," http://www.tcs-trddc.com/trddc_website/ModelMorf/ModelMorf.htm.
[9] ikv++ technologies ag, "Medini QVT," http://projects.ikv.de/qvt/.
[10] A. Cicchetti, D. Ruscio, R. Eramo, and A. Pierantonio, "JTL: a bidirectional and change propagating transformation language," in *SLE*, ser. LNCS, vol. 6563. Springer, 2010, pp. 183–202.
[11] D. Jackson, *Software Abstractions: Logic, Language, and Analysis*, revised ed. MIT Press, 2012.
[12] N. Macedo and A. Cunha, "Implementing QVT-R bidirectional model transformations using Alloy," in *FASE*, ser. LNCS, vol. 7793. Springer, 2013, pp. 297 – 311.
[13] A. Cunha, A. Garis, and D. Riesco, "Translating between Alloy specifications and UML class diagrams annotated with OCL," *SoSyM*, 2013.
[14] C. Nentwich, W. Emmerich, and A. Finkelstein, "Consistency management with repair actions," in *ICSE*. IEEE, 2003, pp. 455–464.
[15] Á. Hegedüs, Á. Horváth, I. Ráth, M. Branco, and D. Varró, "Quick fix generation for DSMLs," in *VL/HCC*. IEEE, 2011, pp. 17–24.