Logical constraints for managing non-determinism in bidirectional model transformations

Antonio Cicchetti[1], Davide Di Ruscio[2], Romina Eramo[2], and Alfonso Pierantonio[2]

[1] School of Innovation, Design and Engineering
Mälardalen University,
SE-721 23, Västerås, Sweden
antonio.cicchetti@mdh.se

[2] Dipartimento di Informatica
Università degli Studi dell'Aquila
Via Vetoio, Coppito I-67010, L'Aquila, Italy
{davide.diruscio|romina.eramo|alfonso.pierantonio}@univaq.it

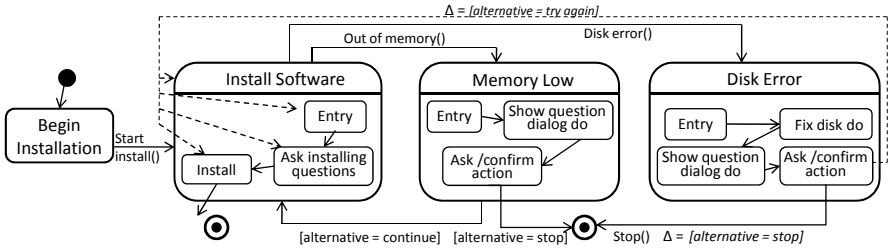
Abstract. In Model Driven Engineering bidirectional transformations are of crucial relevance for round-tripping and consistency checking. Depending on abstraction misalignments between source and target metamodels, such transformations are most of the times non-bijective and give place to *non-determinism*. In practice, each time a target model undergoes modifications a number of corresponding source models are identified although depending on the application scenario not all of them are of interest.
This paper discusses the adoption of a constraint-based semantics for non-bijective bidirectional transformations in order to consistently deal with their intrinsic non-determinism. In particular, transformations can be mapped into logical rules which are constrained by context information which consistently *narrow* the solution space to only those models which are relevant.

1 Introduction

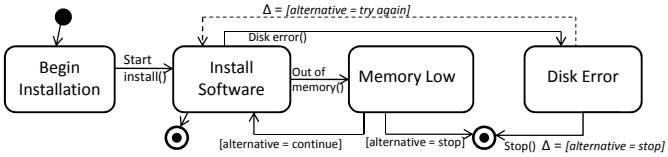
In Model Driven Engineering bidirectional transformations are considered a core ingredient for managing both round-tripping between and consistency of two or more related models. Bidirectionality in model transformations raises not obvious issues mainly related to intrinsic characteristics of model transformations, which are in general *not bijective* [1]: each time a change occurs there is no guarantee to obtain a unique reverse result because of the abstraction misalignment between source and target metamodels, i.e., the source metamodel and the image of the (forward) transformation might not be isomorphic. Existing languages fail in many respect when dealing with non-bijectivity as in many cases its semantic implications are only partially explored, as for instance in bidirectional QVT-R transformations [2].

To better illustrate such difficulties, let us consider the *Collapse/Expand State Diagrams* benchmark defined in the *GRACE International Meeting on Bidirectional Transformations* [3]: starting from a hierarchical state diagram (involving some one-level nesting) as the one reported in Figure 1.a, a flat view has to be provided as in Figure 1.b. Furthermore, any manual modifications on the (target) flat view must be back propagated and eventually reflected in the (source) hierarchical view. For instance, the flattened view reported in Figure 1.b can be extended by adding the alternative `try`

`again` from the state `Disk Error` to `Install software` (see Δ change in Figure 1.b). This gives place to an interesting situation: the new transition can be equally mapped to each one of the nested states within `Install Software` as well as to the container state itself. Consequently, more than one source model propagating the changes exists (see dotted changes in Figure 1.a).



a) A sample Hierarchical State Machine (HSM).



b) The corresponding Non-Hierarchical State Machine (NHSM).

Fig. 1. Sample models for the *Collapse/Expand State Diagrams* benchmark.

This paper describes our experience with the Janus Transformation Language (JTL) [4], a relational and declarative language based on the *Answer Set Programming* (ASP) [5], that is a form of declarative programming oriented towards difficult (primarily NP-hard) search problems and based on the stable model (answer set) semantics of logic programming. JTL provides support to non-bijectivity and its constraint-based semantics assures that all the models are computed at once independently whether they represent the outcome of the backward or forward execution of the bidirectional transformation. Interestingly, a wide range of contextual information can be represented as logical constraints in ASP including the metamodel definition and tracing data in order to eliminate from the solution space those models which are not considered *relevant* (e.g., non conformant models). The next section summarizes the Janus Transformation Languages and its ASP-based semantics, whereas Section 3 illustrates how constraints play an important role in the transformation process by referring to the HSM2NHSM example.

2 Bidirectional transformations with JTL

The JTL environment has been implemented as a set of plug-ins of the Eclipse framework and mainly exploits EMF [6] and AMMA [7]. The JTL transformations are written in a QVT-like syntax and applied on models and metamodels defined as EMF entities. Such components are automatically encoded in ASP and managed by the *transformation engine* during the execution of the considered transformation and to gen-

erate the output models[3]. The computational process is performed by the JTL engine which is based on an ASP bidirectional transformation program executed by means of an ASP solver called DLV [8]. It exploits the benefits of logic programming that enables the specification of relations between source and target types by means of predicates, and intrinsically supports bidirectionality [9] in terms of unification-based matching, searching, and backtracking facilities.

After the encoding of models and metamodels, are defined in a declarative manner by means of a set of logic assertions, the deduction of the target model is performed according to the ASP transformation rules. In particular, the transformation engine is composed of *i) relations* which describe correspondences among element types of the source and target metamodels, *ii) constraints* which specify restrictions on the given relations that must be satisfied in order to execute the corresponding mappings, and an *iii) execution engine* (described in [4]) consisting of bidirectional rules implementing the specified relations as executable mappings.

In general, the transformation engine may generate a huge number of solutions according to the relations specified in the transformation (as such models are generated by the solver outside the EMF environment which does not allow to load non-conformant models). In the next section we show how constraints play an important role in the transformation process.

3 Constraining the solution space

The number of alternatives that we may obtain depends on the intrinsic characteristics of the transformation and on the model elements which are matched by the non-bijective rules. In other words, the number of alternatives depends on the degree of non-determinism of the involved model transformations.

In order to better understand how JTL deal with non-deterministic transformation, let us to consider Listing 3 which contains a fragment of the ASP code implementing the HSM2NHSM transformation. As encoded in lines `1-17`, metamodels are considered as graphs composed of nodes, edges and properties that qualify them (represented by the predicate symbols `metanode`, `metaedge`, and `metaprop`, respectively). Whereas the terms induced by a certain metamodel are exploited for encoding models (represented by the predicate symbols `node`, `edge`, and `prop`, respectively) conforming to it.

```
1 metanode(HSM, state).
2 metanode(HSM, transition).
3 metaprop(HSM, name, state).
4 metaprop(HSM, trigger, transition).
5 metaprop(HSM, effect, transition).
6 metaedge(HSM, association, source, transition, state).
7 metaedge(HSM, association, target, transition, state).
8 [...]
9 node(HSM, "s1", state).
10 node(HSM, "s2", state).
11 node(HSM, "t1", transition).
12 prop(HSM,"s1.1","s1",name,"begin␣installation").
13 prop(HSM,"s2.1","s2",name,"install␣software").
14 prop(HSM,"t1.1","t1",trigger,"install␣software").
15 prop(HSM,"t1.2","t1",effect,"start␣install").
16 edge(HSM,"tr1",association,source, "s1","t1").
```

[3] Details about the JTL environment are available on [4] and at http://jtl.di.univaq.it

```
17 edge(HSM,"tr1",association,target, "s2","t1").
18 [...]
19 relation ("r1", HSM, state).
20 relation ("r1", NHSM, state).
21 :- node(HSM,S1,state), not edge(HSM,OWN1,owningCompositeState,S1,CS1),
       not node'(NHSM,S1,state).
22 :- node(HSM,S1,state), edge(HSM,OWN!,owningCompositeState,S1,CS1),
       node(HSM,CS1,compositeState), node'(NHSM,S1,state).
23 :- node(NHSM,S1,state), not trace_node(HSM,S1,compositeState),
       not node'(HSM,S1,state).
24 :- node(NHSM,S1,state), trace_node(HSM,S1,compositeState),
       node'(HSM,S1,state).
25 relation ("r2", HSM, compositeState).
26 relation ("r2", NHSM, state).
27 :- node(HSM,S1,compositeState), not node'(NHSM,S1,state).
28 :- node(NHSM,S1,state), trace_node(HSM,S1,compositeState),
       not node'(HSM,S1,compositeState).
29 [...]
```

Relations and constraints of the HSM2NHSM transformation are encoded in the bottom part of the listing. In particular, the terms in lines `1-2` define the relation called `"r1"` between the metaclass `State` belonging to the `HSM` metamodel and the metaclass `State` belonging to the `NHSM` metamodel. An ASP constraint expresses an invalid condition: for example, constraints in line `3-6` impose that each time a state occurs in the HSM model, the correspondent one in the NHSM model is generated only if the source element is not a sub-state, vice versa each state in the NHSM model is mapped in the HSM model. In fact, if each atom in its body is true then the correspondent solution candidate is eliminated. In similar way, the relation between the metaclasses `Composite state` and `State` is encoded in line `7-8`. Constraints in line `9-10` impose that each time a composite state occurs in the HSM model a correspondent state in the NHSM model is generated, and vice versa. Missing sub-states in a NHSM model can be generated again in the HSM model by means of trace information (see line `5-6` and `10`). Trace elements are automatically generated each time a model element is discarded by the mapping and need to be stored in order to be regenerated during the backward transformation.

The transformation process logically consists of the following steps: *1)* given the input (meta)models, the execution engine induces all the possible solution candidates according to the specified relations; *2)* the set of candidates is refined by means of constraints. According to the example described in Section 1, the HSM2NHSM transformation is able to generate the flat model from the hierarchical model (see Figure 1.b) and to propagate the changes occurred in the generated flat model by generating a set of hierarchical model (see Figure 1.a).

As said above, the execution engine may generate a large number of alternatives when only the information encoded in the transformation is used. The constraints play a key role in the transformation process. In particular, transformations can be mapped into logical rules which are constrained by context information which consistently *narrow* the solution space to only those models which are relevant. The answer sets may be refined in subsequent steps, as following: *i)* the answer set is filtered according to the constraints induced by the source metamodel, *ii)* the answer set is further reduced by considering the constraints induced by the tracing information and *iii)* additional user-defined constraints can be added to browse the space of solutions. For instance, the

designer may decide to reduce the alternatives obtained in the previous example by means of the following constraint

```
1 :- node(NHSM,S1,state), node(NHSM,T1,transition), edge(NHSM,TR1,association,source
     ,S1,T1), not trace_edge(HSM,TR1,association,source,SX,TY), node'(HSM,S1,state
     ), node'(HSM,T1,transition), not edge'(HSM,TR1,association,source,S1,T1).
```

In particular, it impose that each new transition occurring in the NHSM model is mapped in a correspondent transition between the parent states neglecting the nested states. The use of constraints during the generation of a solution may reduce back-tracking because it allows for early detection of dead-ends and permits to reduce the space of solutions in sub-sequent steps. Interestingly, a wide range of application scenarios not limited to round-tripping, consistency checking, and synchronization, are enabled depending on the used constraints.

4 Conclusions

Non-bijective bidirectionality is commonplace in many practical situations where two or more models are loosely coupled according to some correspondence definition. Complex processes, such as round-tripping or consistency checking, require both pragmatic qualities and theoretical soundness as pointed in [2] where the semantic idiosyncrasies of a prominent language as QVT-R are discussed.

In our opinion, the study of non-bijective transformations cannot neglect non-determinism and its management. In fact, many approaches deal with non-determinism in a programmatic style or by implementing forms of back-tracking which makes its maintenance and refinement very difficult. In other words, we advocate the adoption of the constraint-based approach presented above for its logical soundness, its pragmatic quality, and finally for its inherent capability of dealing with the multiplicity of the solutions.

References

1. Tratt, L.: Model transformations and tool integration. Jour. on Software and Systems Modeling (SoSyM) **4**(2) (May 2005) 112–122
2. Stevens, P.: Bidirectional model transformations in QVT: semantic issues and open questions. Software and Systems Modeling **8** (2009)
3. Czarnecki, K., Foster, J.N., Hu, Z., Lämmel, R., Schürr, A., Terwilliger, J.F.: Bidirectional Transformations: A Cross-Discipline Perspective—GRACE meeting notes, state of the art, and outlook. In: ICMT2009 - International Conference on Model Transformation, Proceedings. Volume 5563 of LNCS., Springer (2009) 260–283
4. Cicchetti, A., Di Ruscio, D., Eramo, R., Pierantonio, A.: JTL: a bidirectional and change propagating transformation language. In: 3rd International Conference on Software Language Engineering (SLE). (October 2010) to appear.
5. Gelfond, M., Lifschitz, V.: The Stable Model Semantics for Logic Programming. In Kowalski, R.A., Bowen, K., eds.: Proceedings of the Fifth Int. Conf. on Logic Programming, Cambridge, Massachusetts, The MIT Press (1988) 1070–1080
6. Budinsky, F., Steinberg, D., Merks, E., Ellersick, R., Grose, T.: Eclipse Modeling Framework. Addison Wesley (2003)
7. Bézivin, J., Jouault, F., Rosenthal, P., Valduriez, P.: Modeling in the Large and Modeling in the Small. In: Model Driven Architecture, European MDA Workshops: Foundations and Applications. Volume 3599 of LNCS., Springer (2004) 33–46
8. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV System for Knowledge Representation and Reasoning (2004)
9. Czarnecki, K., Helsen, S.: Feature-based Survey of Model Transformation Approaches. IBM Systems J. **45**(3) (June 2006)