Formal framework of recontextualization by means of dependency graphs

Mickaël Kerboeuf, Paola Vallejo, Jean-Philippe Babau

Formal framework of recontextualization by means of dependency graphs

Mickaël Kerboeuf Paola Vallejo Jean-Philippe Babau
University of Brest (France),
Lab-STICC, MOCS Team

{kerboeuf,vallejoco,babau}@univ-brest.fr

1 Model Graph

A model graph is a *graph-based denotation* of a model, *i.e.* a labeled graph composed of *vertices* denoting *instances* and *scalar values*, and *edges* denoting *references* and *attributes*. Figure 1 defines the *name spaces* corresponding to instances, scalar values, attributes and references. These name spaces are *alphabets*, *i.e.* finite non empty sets of symbols.

$\mathcal{I}$: instances $\quad$ $\mathcal{S}$: scalar values $\quad$ $\mathcal{A}$: attribute $\quad$ $\mathcal{R}$: reference

Figure 1: ISAR: name spaces for model graphs

We call *model* and note $\mathfrak{m}$ a triplet composed of a set of instances corresponding to vertices noted V, and two sets of edges noted E_a and E_r (see figure 2). The first set of edges denotes attributes. It relates instances to scalar values through attribute names. The second one denotes references. It relates instances to each other through reference names. We note $\mathfrak{m}.V$, $\mathfrak{m}.E_a$ and $\mathfrak{m}.E_r$ the V, E_a and E_r components of a given model $\mathfrak{m}$.

$$\mathfrak{m} \triangleq \left(V \ , \ E_a \ , \ E_r \right) \qquad \text{where} \begin{cases} V & \subseteq & \mathcal{I} \\ E_a & \subseteq & V \times \mathcal{A} \times \mathcal{S} \\ E_r & \subseteq & V \times \mathcal{R} \times V \end{cases}$$

Figure 2: Model graph

As an illustration, figure 3 shows a model conforming to an Ecore metamodel together with its corresponding graph-based representation. The metadata are willfully out of the scope of the graph-based representation of models.

2 Migration specification

For a given model $\mathfrak{m}$, we call *migration specification* and note $\overrightarrow{\mathfrak{m}}$ a quadruplet composed of $\mathfrak{m}$ and of three sets noted D_i, D_a and D_r. These sets specify the instances, attributes and references of $\mathfrak{m}$ that are intended to be deleted (see figure 4). We note $\overrightarrow{\mathfrak{m}}.D_i$, $\overrightarrow{\mathfrak{m}}.D_a$ and $\overrightarrow{\mathfrak{m}}.D_r$ the D_i, D_a and D_r components of a given migration specification $\overrightarrow{\mathfrak{m}}$.

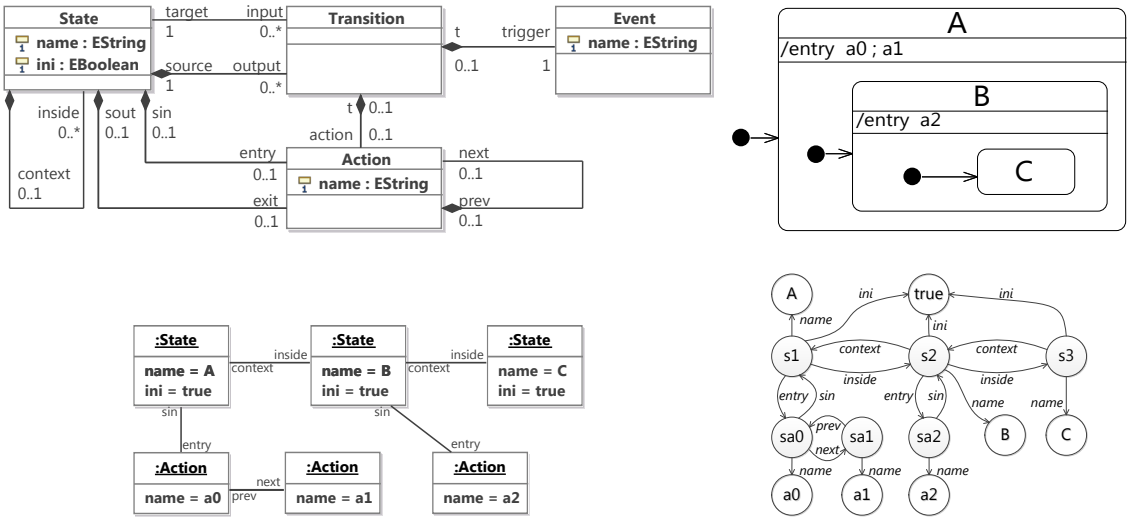


Figure 3: Metamodel, conforming model, concrete syntax and semantics

$$\overrightarrow{\mathfrak{m}} \triangleq \left(\mathfrak{m} \ , \ D_i \ , \ D_a \ , \ D_r \right) \qquad \text{where} \left\{ \begin{array}{lcl} D_i & \subseteq & \mathfrak{m}.V \\ D_a & \subseteq & \mathfrak{m}.V \times \mathcal{A} \\ D_r & \subseteq & \mathfrak{m}.V \times \mathcal{R} \end{array} \right.$$

Figure 4: Migration specification

The migration specification is intended to be used together with the model to produce a *migrated model*. This migration specification is *computed* from a *Modif specification*. A Modif specification makes it possible to state the deletion of a class, an attribute or a reference (corresponding to D_i, D_a and D_r at the model level). See appendix A for more details about the computation of a migration specification from a Modif specification.

3 Migration

We call *migrator* and note M the tool aiming at producing a *migrated model* from a *migration specification*. It is defined in figure 5. It simply commits the *deletion* of instances and of features (attributes and references) on the source model to produce a target model.

$$\begin{array}{rcl} \mathfrak{m}' & = & \mathsf{M}(\overrightarrow{\mathfrak{m}}) \\ \mathfrak{m}'.V & = & \mathfrak{m}.V \setminus \overrightarrow{\mathfrak{m}}.D_i \\ \mathfrak{m}'.E_a & = & \mathfrak{m}.E_a \setminus \{(i,a,s) \in \mathcal{I} \times \mathcal{A} \times \mathcal{S} \mid i \in \overrightarrow{\mathfrak{m}}.D_i \vee (i,a) \in \overrightarrow{\mathfrak{m}}.D_a\} \\ \mathfrak{m}'.E_r & = & \mathfrak{m}.E_r \setminus \{(i,r,i') \in \mathcal{I} \times \mathcal{R} \times \mathcal{I} \mid i \in \overrightarrow{\mathfrak{m}}.D_i \vee i' \in \overrightarrow{\mathfrak{m}}.D_i \vee (i,r) \in \overrightarrow{\mathfrak{m}}.D_r\} \end{array}$$

Figure 5: Computation of a migrated model

As an illustration, we introduce in figure 6 a *variant* of the metamodel of figure 3 where actions are not taken into account.

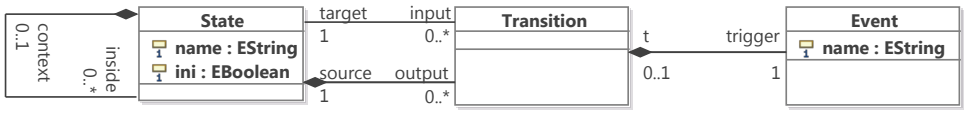


Figure 6: Metamodel of statecharts without actions

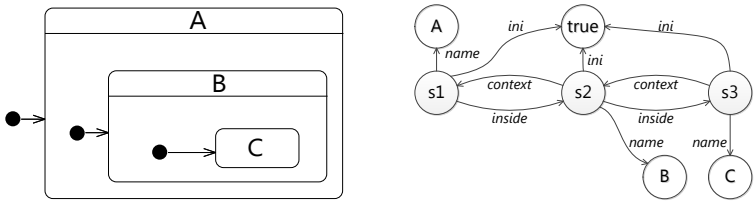
We note $\mathfrak{m}$ the model of figure 3. It is formally and explicitly defined by its components as follows:

$$\begin{aligned}
\mathfrak{m}.V &= \{s_1, s_2, s_3, sa_0, sa_1, sa_2\} \\
\mathfrak{m}.E_a &= \{(s_1, name, A), (s_2, name, B), (s_3, name, C), (s_1, ini, true), (s_2, ini, true), \\
&\quad (s_3, ini, true), (sa_0, name, a_0), (sa_1, name, a_1), (sa_2, name, a_2)\} \\
\mathfrak{m}.E_r &= \{(s_1, inside, s_2), (s_2, inside, s_3), (s_3, contexte, s_2), (s_2, context, s_1), (s_1, entry, sa_0), \\
&\quad (s_2, entry, sa_2), (sa_2, sin, s_2), (sa_0, sin, s_1), (sa_0, next, sa_1), (sa_1, prev, sa_0)\}
\end{aligned}$$

We note $\overrightarrow{\mathfrak{m}}$ the migration specification of $\mathfrak{m}$ to a model conforming to the metamodel of figure 6, *i.e.* a model where *actions* have been *deleted*. This specification is formally and explicitly defined by its components as follows:

$$\overrightarrow{\mathfrak{m}}.D_i = \{sa_0, sa_1, sa_2\} \qquad \overrightarrow{\mathfrak{m}}.D_a = \emptyset \qquad \overrightarrow{\mathfrak{m}}.D_r = \emptyset$$

Once the migrator M is applied to the migration specification $\overrightarrow{\mathfrak{m}}$, we obtain the *migrated model* $\mathfrak{m}'$ depicted and formally defined by figure 7.



$$\begin{aligned}
\mathfrak{m}' &= \mathsf{M}(\overrightarrow{\mathfrak{m}}) \\
\mathfrak{m}'.V &= \{s_1, s_2, s_3\} \\
\mathfrak{m}'.E_a &= \{(s_1, name, A), (s_2, name, B), (s_3, name, C), (s_1, ini, true), (s_2, ini, true), (s_3, ini, true)\} \\
\mathfrak{m}'.E_r &= \{(s_1, inside, s_2), (s_2, inside, s_3), (s_3, contexte, s_2), (s_2, context, s_1)\}
\end{aligned}$$

Figure 7: Concrete syntax and semantics of a migrated model without actions

4 Rewriting tools

In this paper, the focus is put on the reuse of *rewriting tools*. To begin with, this kind of tools has to be specified within our semantic domain.

We call *rewriting tool* and note T the tool aiming at producing an *output model* from an *input model* and conforming to the same metamodel. T can be taken for an *endogenous model transformation*. We note $\mathsf{T}(\mathfrak{m})$ the output model produced by T from an input model $\mathfrak{m}$.

So far, we consider T as a *black box*. As a consequence, the action of T can only be specified by the differences between a given input model and the corresponding output model. Within our semantic domain, the action of T is therefore specified by means of *deleted* and *added* graph

elements for a given model $\mathfrak{m}$. This specification is formally stated in figure 8. As an illustration, we introduce three simple examples of rewriting tools whose scope corresponds to the metamodel of figure 6.

$$
\begin{array}{rcl}
\text{Deleted elements} & : & \left\{ \begin{array}{lcll} \mathsf{T}^i_D(\mathfrak{m}) & = & \mathfrak{m}.V \setminus \mathsf{T}(\mathfrak{m}).V & \text{(deleted instances)} \\ \mathsf{T}^a_D(\mathfrak{m}) & = & \mathfrak{m}.E_a \setminus \mathsf{T}(\mathfrak{m}).E_a & \text{(deleted attributes)} \\ \mathsf{T}^r_D(\mathfrak{m}) & = & \mathfrak{m}.E_r \setminus \mathsf{T}(\mathfrak{m}).E_r & \text{(deleted references)} \end{array} \right. \\
\text{Added elements} & : & \left\{ \begin{array}{lcll} \mathsf{T}^i_A(\mathfrak{m}) & = & \mathsf{T}(\mathfrak{m}).V \setminus \mathfrak{m}.V & \text{(added instances)} \\ \mathsf{T}^a_A(\mathfrak{m}) & = & \mathsf{T}(\mathfrak{m}).E_a \setminus \mathfrak{m}.E_a & \text{(added attributes)} \\ \mathsf{T}^r_A(\mathfrak{m}) & = & \mathsf{T}(\mathfrak{m}).E_r \setminus \mathfrak{m}.E_r & \text{(added references)} \end{array} \right.
\end{array}
$$

Figure 8: Specification of a rewriting tool

4.1 Identity

We call *identity* and note Id the rewriting tool performing *no actions*. It applies to a statechart conforming to the metamodel of figure 6 and it provides it unchanged as a result. This trivial example will be useful later to underline the benefits of our approach.

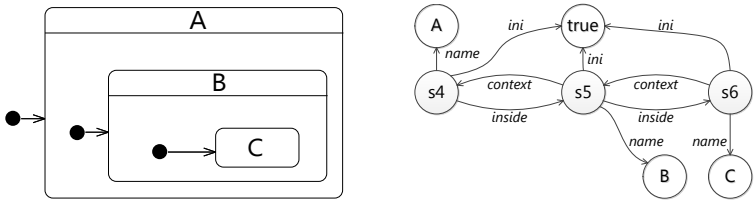
We use $\mathfrak{m}'$ the model depicted by figure 7. By definition, $\mathsf{Id}(\mathfrak{m}') = \mathfrak{m}'$. The action of Id on $\mathfrak{m}'$ is therefore formally stated as follows:

$$
\begin{array}{rcl}
\text{Deleted elements} & : & \left\{ \mathsf{Id}^i_D(\mathfrak{m}') = \mathsf{Id}^a_D(\mathfrak{m}') = \mathsf{Id}^r_D(\mathfrak{m}') = \emptyset \quad \text{(no deleted elements)} \right. \\
\text{Added elements} & : & \left\{ \mathsf{Id}^i_A(\mathfrak{m}') = \mathsf{Id}^a_A(\mathfrak{m}') = \mathsf{Id}^r_A(\mathfrak{m}') = \emptyset \quad \text{(no added elements)} \right.
\end{array}
$$

4.2 Copy

We call *copy* and note Cp the rewriting tool performing a complete copy of an input model. It applies to a statechart conforming to the metamodel of figure 6 and it provides a new model where all the instances are *new*.

We use $\mathfrak{m}'$ the model depicted by figure 7 and we note $\mathfrak{m}'' = \mathsf{Cp}(\mathfrak{m}')$, the result of the copy. The formal definition of $\mathfrak{m}''$ and it's graphical representation is given by figure 9.



$$
\begin{array}{rl}
\mathfrak{m}'' = & \mathsf{Cp}(\mathfrak{m}') \\
\mathfrak{m}''.V = & \{s_4, s_5, s_6\} \\
\mathfrak{m}''.E_a = & \{(s_4, name, A), (s_5, name, B), (s_6, name, C), (s_4, ini, true), (s_5, ini, true), (s_6, ini, true)\} \\
\mathfrak{m}''.E_r = & \{(s_4, inside, s_5), (s_5, inside, s_6), (s_6, contexte, s_5), (s_5, context, s_4)\}
\end{array}
$$

Figure 9: Copy of the model from figure 7

By definition, the action of Cp on $\mathfrak{m}'$ is formally stated as follows:

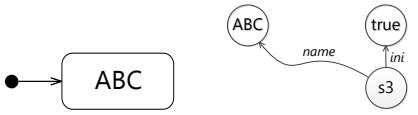
$$\text{Deleted elements} : \begin{cases} \mathsf{Cp}^i_D(\mathfrak{m}') = \mathfrak{m}'.V \\ \mathsf{Cp}^a_D(\mathfrak{m}') = \mathfrak{m}'.E_a \\ \mathsf{Cp}^r_D(\mathfrak{m}') = \mathfrak{m}'.E_r \end{cases}$$

$$\text{Added elements} : \begin{cases} \mathsf{Cp}^i_A(\mathfrak{m}') = \{s_4, s_5, s_6\} \\ \mathsf{Cp}^a_A(\mathfrak{m}') = \{(s_4, name, A), (s_5, name, B), (s_6, name, C), \\ \qquad\qquad (s_4, ini, true), (s_5, ini, true), (s_6, ini, true)\} \\ \mathsf{Cp}^r_A(\mathfrak{m}') = \{(s_4, inside, s_5), (s_5, inside, s_6), (s_6, contexte, s_5), \\ \qquad\qquad (s_5, context, s_4)\} \end{cases}$$

4.3 Flattener

We call *flattener* and note Fl the rewriting tool performing a *flattening* of a *hierarchical* statechart conforming to the metamodel of figure 6. It provides a new model where the *superstates* are deleted and potentially replaced by one new state at least. We suppose we have no more information about the algorithm implemented by the flattener.

We use $\mathfrak{m}'$ the model depicted by figure 7 and we note $\mathfrak{m}'' = \mathsf{Fl}(\mathfrak{m}')$, the result of the flattener. The formal definition of $\mathfrak{m}''$ and it's graphical representation is given by figure 10. We can note that only one instance remains in the result.



$$\begin{aligned} \mathfrak{m}'' &= \mathsf{Fl}(\mathfrak{m}') \\ \mathfrak{m}''.V &= \{s_3\} \\ \mathfrak{m}''.E_a &= \{(s_3, name, ABC), (s_3, ini, true)\} \\ \mathfrak{m}''.E_r &= \emptyset \end{aligned}$$

Figure 10: Flattening of the model from figure 7

By definition, the action of Fl on $\mathfrak{m}'$ is formally stated as follows:

$$\text{Deleted elements} : \begin{cases} \mathsf{Fl}^i_D(\mathfrak{m}') = \{s_1, s_2\} \\ \mathsf{Fl}^a_D(\mathfrak{m}') = \mathfrak{m}'.E_a \setminus \{(s_3, ini, true)\} \\ \mathsf{Fl}^r_D(\mathfrak{m}') = \mathfrak{m}'.E_r \end{cases}$$

$$\text{Added elements} : \begin{cases} \mathsf{Fl}^i_A(\mathfrak{m}') = \emptyset \\ \mathsf{Fl}^a_A(\mathfrak{m}') = \{(s_3, name, ABC)\} \\ \mathsf{Fl}^r_A(\mathfrak{m}') = \emptyset \end{cases}$$

5 Reverse migration

We call *reverse migrator* and note R the tool aiming at undoing the migration operations that have been applied to a given initial model $\mathfrak{m}$. Thus, R applies to a model which is *supposed* to have been migrated and it produces a new model which corresponds as far as possible to the initial model.

If the migration specification $\overrightarrow{\mathfrak{m}}$ associated to the initial model $\mathfrak{m}$ is unknown, then the reverse migration cannot be computed.

If the migration specification is known, and if the migrated model has not been modified, then the reverse migration is already and trivially available:

$$\mathsf{R}(\mathsf{M}(\overrightarrow{\mathfrak{m}})) = \mathfrak{m}$$

We focus now on the more challenging problem of an initial model $\mathfrak{m}$ which has been migrated through a known migration specification $\overrightarrow{\mathfrak{m}}$, and which has been later modified by a rewriting tool T. We note $\mathfrak{m}'$ the model produced by the reverse migrator in this case:

$$\mathsf{R}(\mathsf{T}(\mathsf{M}(\overrightarrow{\mathfrak{m}}))) = \mathfrak{m}' \quad \text{(reversed migrated model)}$$

We aim at recovering as much as possible the initial model *without undoing the action of* T.

The first solution to this problem is a well known and rather obvious approach which consists in considering a reverse migration as a normal case of migration. For a given migration operation, if the corresponding reverse operation is available, then it is applied (*e.g.* renaming of attributes, which is undone by the reverse renaming). If it is not available, then it can leads to a loss of data. This is actually the case on which we focus here, and this is the reason why we only consider *deletion*. As a result, nothing is supposed to be deleted nor added by this reverse migration:

$$\mathsf{R}(\mathsf{T}(\mathsf{M}(\overrightarrow{\mathfrak{m}}))) = \mathsf{M}(\overrightarrow{\mathsf{T}(\mathsf{M}(\overrightarrow{\mathfrak{m}}))}) \qquad \text{where: } \overrightarrow{\mathsf{T}(\mathsf{M}(\overrightarrow{\mathfrak{m}}))} = (\mathsf{T}(\mathsf{M}(\overrightarrow{\mathfrak{m}})), \emptyset, \emptyset, \emptyset)$$

The reverse migration is obtained from this migration specification according to the principles of figure 5. By definition, it leads to *identity*:

$$\mathsf{R}(\mathsf{T}(\mathsf{M}(\overrightarrow{\mathfrak{m}}))) = \mathsf{M}(\overrightarrow{\mathsf{T}(\mathsf{M}(\overrightarrow{\mathfrak{m}}))}) = \mathsf{T}(\mathsf{M}(\overrightarrow{\mathfrak{m}}))$$

To illustrate this first solution, we consider again the three examples of rewriting tools that have been introduced in the previous section, namely *identity* (Id), *copy* (Cp), and *flattener* (Fl). They apply to models conforming to the metamodel of figure 6. We aim at performing a reverse migration of the model of figure 7 after its processing by *identity*, *copy* and *flattener*. The result is a model which is supposed to conform to the metamodel of figure 3.

5.1 Identity

As mentioned in section 4.1, *identity* keeps the model $\mathsf{M}(\overrightarrow{\mathfrak{m}})$ of figure 7 unchanged. Thus, in this case:

$$\mathsf{R}(\mathsf{Id}(\mathsf{M}(\overrightarrow{\mathfrak{m}}))) = \mathsf{Id}(\mathsf{M}(\overrightarrow{\mathfrak{m}})) = \mathsf{M}(\overrightarrow{\mathfrak{m}}) \quad \text{(model of figure 7)}$$

As a result, the reverse migration leads to the model of figure 7 which corresponds to the model of figure 3 *without actions*. Ideally, these actions should have been recovered but actually, they are finally lost.

5.2 Copy

According to the definition of *copy* (cf. section 4.2), $\mathsf{Cp}(\mathsf{M}(\overrightarrow{\mathfrak{m}}))$ leads to the model $\mathfrak{m}''$ of figure 9, and thus:

$$\mathsf{R}(\mathsf{Cp}(\mathsf{M}(\overrightarrow{\mathfrak{m}}))) = \mathsf{Cp}(\mathsf{M}(\overrightarrow{\mathfrak{m}})) = \mathfrak{m}'' \quad \text{(model of figure 9)}$$

The result of the reverse migration is a model which is equivalent to the model of figure 7 except the instances have been *recreated*. As for *identity*, this resulting model corresponds to the model of figure 3 *without actions*. Thus in this case as in the previous one, these actions should have been recovered but actually they are lost.

5.3 Flattener

The result of *flattener* applied to the model of figure 3 is the model noted $\mathfrak{m}''$ in figure 10 (cf. section 4.3), and thus:

$$\mathsf{R}(\mathsf{Fl}(\mathsf{M}(\overrightarrow{\mathfrak{m}}))) = \mathsf{Fl}(\mathsf{M}(\overrightarrow{\mathfrak{m}})) = \mathfrak{m}'' \quad \text{(model of figure 10)}$$

The result of the reverse migration in this case contains a unique state named ABC and without actions. Actually, this state has been modified by the flattener through the initial states named A, B and C. These states had entry actions. These original actions are not taken into account by the reverse migration and then, they cannot be recovered.

6 Recontextualization by keys

So far, the reverse migration does not enable the recovery of elements that are deleted by the initial migration. We call *initial context* of a given model $\mathfrak{m}$ the set of model elements from $\mathfrak{m}$ that are deleted by a migration specification $\overrightarrow{\mathfrak{m}}$. By definition of migration specifications (see figure 4), this initial context corresponds to $\overrightarrow{\mathfrak{m}}.D_i \cup \overrightarrow{\mathfrak{m}}.D_a \cup \overrightarrow{\mathfrak{m}}.D_r$.

We call *recontextualization* the tool aiming at putting back the initial context of a given model $\mathfrak{m}$ in its *reverse migrated version* after the processing of a *rewriting tool*.

In a first approach, we use *instances* as *keys* to reconnect a reverse migrated model to its initial context. We note $\mathsf{C_k}$ this tool:

$$\mathsf{C_k}(\mathsf{R}(\mathsf{T}(\mathsf{M}(\overrightarrow{\mathfrak{m}})))) = \mathfrak{m}' \quad \text{(recontextualized reversed migrated model)}$$

Its action can be informally described as follows:

1. deleted instances are recovered
2. attributes of deleted instances are recovered
3. deleted attributes are recovered as far as their source instances still exist
4. references between two deleted instances are recovered
5. references from deleted instances targeting still existing instances are recovered
6. references from still existing instances targeting deleted instances are recovered
7. deleted references are recovered as far as their source and target instances still exist

More formally, the recontextualization by beys on graphs is specified by figure 11. To illustrate this approach, we consider again the three examples of rewriting tools namely *identity* (Id), *copy* (Cp), and *flattener* (Fl).

$$\begin{array}{rcll}
\mathfrak{m}_\mathsf{r} & = & \mathsf{R}(\mathsf{T}(\mathsf{M}(\overrightarrow{\mathfrak{m}})) & \text{(reversed migrated model)} \\
\mathfrak{m}' & = & \mathsf{C_k}(\mathfrak{m}_\mathsf{r}) & \text{(recontextualized reversed migrated model)} \\
\mathfrak{m}'.V & = & \mathfrak{m}_\mathsf{r}.V \cup \overrightarrow{\mathfrak{m}}.D_i & \\
\mathfrak{m}'.E_a & = & \mathfrak{m}_\mathsf{r}.E_a \cup \{(i,a,s) \in \mathfrak{m}.E_a \mid i \in \overrightarrow{\mathfrak{m}}.D_i \vee ((i,a) \in \overrightarrow{\mathfrak{m}}.D_a \wedge i \in \mathfrak{m}_\mathsf{r}.V)\} & \\
\mathfrak{m}'.E_r & = & \mathfrak{m}_\mathsf{r}.E_r \cup \{(i,r,i') \in \mathfrak{m}.E_r \mid (i,i') \in \mathfrak{m}'.V^2 \wedge (i \in \overrightarrow{\mathfrak{m}}.D_i \vee i' \in \overrightarrow{\mathfrak{m}}.D_i \vee (i,r) \in \overrightarrow{\mathfrak{m}}.D_r)\} &
\end{array}$$

Figure 11: Computation of a recontextualized model by keys

6.1 Identity

As previously mentioned, *identity* keeps the input model unchanged and the reverse migration leads to the model of figure 7 which corresponds to the model of figure 3 *without actions*:

$$\begin{array}{rcll}
\mathfrak{m}_\mathsf{r} & = & \mathsf{R}(\mathsf{Id}(\mathsf{M}(\overrightarrow{\mathfrak{m}}))) & \text{(model of figure 7)} \\
\mathfrak{m}' & = & \mathsf{C_k}(\mathfrak{m}_\mathsf{r}) & \text{(recontextualized reversed migrated model)}
\end{array}$$

By definition of the recontextualization by keys (see figure 11), $\mathfrak{m}'$ is defined as follows:

$$\begin{array}{rcll}
\mathfrak{m}'.V & = & \mathfrak{m}_\mathsf{r}.V & \cup \ \{sa_0, sa_1, sa_2\} \\
\mathfrak{m}'.E_a & = & \mathfrak{m}_\mathsf{r}.E_a & \cup \ \{(sa_0, name, a_0), (sa_1, name, a_1), (sa_2, name, a_2)\} \\
\mathfrak{m}'.E_r & = & \mathfrak{m}_\mathsf{r}.E_r & \cup \ \{(s_1, entry, sa_0), (s_2, entry, sa_2), (sa_2, sin, s_2), (sa_0, sin, s_1), \\
 & & & \quad (sa_0, next, sa_1), (sa_1, prev, sa_0)\}
\end{array}$$

The deleted actions are actually recovered and as expected, the recontextualized reversed migrated model corresponds to the initial model of figure 3:

$$\mathsf{C_k}(\mathsf{R}(\mathsf{Id}(\mathsf{M}(\overrightarrow{\mathfrak{m}})))) = \mathfrak{m}$$

This situation is depicted by figure 12. In this figure, the initial context appear in red and the links between the reversed migrated model and the initial context appear in blue.

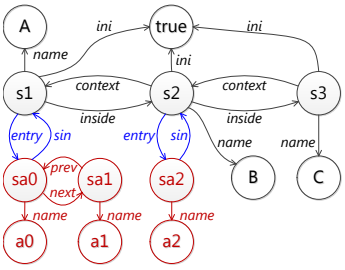


Figure 12: Recontextualization by keys of the model from figure 7

6.2 Copy

As previously mentioned, the *copy* of $\mathsf{M}(\overrightarrow{\mathfrak{m}})$ leads to the model of figure 9 which is equivalent to the model of figure 7 except the instances have been *recreated*:

$$\begin{array}{lcll} \mathfrak{m}_\mathsf{r} & = & \mathsf{R}(\mathsf{Cp}(\mathsf{M}(\overrightarrow{\mathfrak{m}}))) & \text{(model of figure 9)} \\ \mathfrak{m}' & = & \mathsf{C_k}(\mathfrak{m}_\mathsf{r}) & \text{(recontextualized reversed migrated model)} \end{array}$$

As a consequence and by definition of the recontextualization by keys (see figure 11), the initial context can be put back but not reconnected to the reversed migrated model. The resulting model actually contains the recovered instances and the possible links between them (attributes or references), but it does not contains reference links between the recovered instances and the instances newly created by *copy*:

$$\begin{array}{rcll} \mathfrak{m}'.V & = & \mathfrak{m}_\mathsf{r}.V & \cup \quad \{sa_0, sa_1, sa_2\} \\ \mathfrak{m}'.E_a & = & \mathfrak{m}_\mathsf{r}.E_a & \cup \quad \{(sa_0, name, a_0), (sa_1, name, a_1), (sa_2, name, a_2)\} \\ \mathfrak{m}'.E_r & = & \mathfrak{m}_\mathsf{r}.E_r & \cup \quad \{(sa_0, next, sa_1), (sa_1, prev, sa_0)\} \end{array}$$

The deleted actions are actually recovered but they are not connected to states. This situation is depicted by figure 13. In this case, the graph is *not connected*. Each *connected component* of such graphs is the semantics of a sub-model conforming to the initial metamodel (of figure 3 in this example), and which can be valid or not with regard to multiplicity or containment.

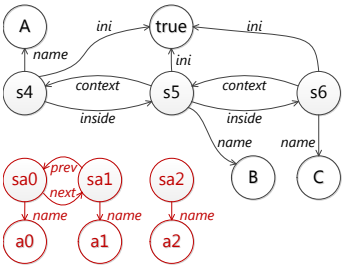


Figure 13: Recontextualization by keys of the model from figure 9

6.3 Flattener

The *flattening* of $\mathsf{M}(\overrightarrow{\mathfrak{m}})$ leads to the model of figure 10 which contains a unique state named ABC and without actions:

$$\begin{array}{lcll} \mathfrak{m}_\mathsf{r} & = & \mathsf{R}(\mathsf{Fl}(\mathsf{M}(\overrightarrow{\mathfrak{m}}))) & \text{(model of figure 10)} \\ \mathfrak{m}' & = & \mathsf{C_k}(\mathfrak{m}_\mathsf{r}) & \text{(recontextualized reversed migrated model)} \end{array}$$

The only instance s_3 was in the initial model. Its name has been changed and by definition of the recontextualization by keys (see figure 11), this change is not undone. The initial instances s_1 and s_2 have been deleted by the flattener and they are obviously not recovered by the recontextualization. Actually, as in the case of *copy*, the resulting model only contains the recovered instances corresponding to actions. However it does not contains reference links between the recovered instances and the remaining instance s_3:

$$\begin{array}{rcll} \mathfrak{m}'.V & = & \mathfrak{m}_\mathsf{r}.V & \cup \quad \{sa_0, sa_1, sa_2\} \\ \mathfrak{m}'.E_a & = & \mathfrak{m}_\mathsf{r}.E_a & \cup \quad \{(sa_0, name, a_0), (sa_1, name, a_1), (sa_2, name, a_2)\} \\ \mathfrak{m}'.E_r & = & \mathfrak{m}_\mathsf{r}.E_r & \cup \quad \{(sa_0, next, sa_1), (sa_1, prev, sa_0)\} \end{array}$$

As in the previous case, it leads to a graph which is *not connected*. This result is depicted by figure 14.

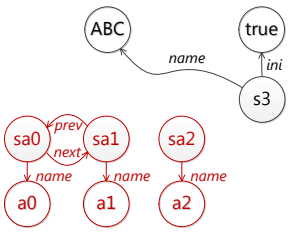


Figure 14: Recontextualization by keys of the model from figure 10

7 Recontextualization by graphs

So far the rewriting tool T is considered as a *black box* and as a consequence, its action can only be specified by the differences between a given input model and the corresponding output model.

In order to improve the recontextualization of a reverse migrated model, T is considered now as a *grey box* where the *computation functions* are not known but where the *computation dependencies* are known. For that purpose, for a given input model $\mathfrak{m}$, we suppose now that the tool T

$$\mathsf{G}(\mathsf{T}, \mathfrak{m}) \triangleq \mathsf{T}(\mathfrak{m}).V \to \mathcal{P}(\mathfrak{m}.V) \qquad \text{(edges from output instances to input instances)}$$

Figure 15: Dependency graph

provides the resulting model $\mathsf{T}(\mathfrak{m})$ together with a *dependency graph* noted $\mathsf{G}(\mathsf{T}, \mathfrak{m})$. For a given instance i of $\mathsf{T}(\mathfrak{m})$, this graph makes it explicit the instances of $\mathfrak{m}$ that have been used to *create* or *update* i. It is formally defined in figure 15 by means of a total function over the output instances $(\mathsf{T}(\mathfrak{m}).V)$ to *parts* of the input instances. Thus, $\mathsf{G}(\mathsf{T}, \mathfrak{m})(i)$ provides the (possibly empty) set of input instances that are used to compute i.

The use of dependency graphs enhances the recontextualization. Indeed, the links between the reverse migrated model and it initial context can be created not only over the remaining instances, but also over new instances through the instances that have been used to compute them.

In this second approach, we note $\mathsf{C_g}$ the tool using *instances* as *vertices* of dependency graphs to reconnect a recontextualized reverse migrated model to its initial context. Its actions complete the actions performed by $\mathsf{C_k}$ (*i.e.* recontextualization by keys). It applies to a model together with a dependency graph:

$\mathsf{C_g}\left(\ \mathsf{C_k}(\mathsf{R}(\mathsf{T}(\mathsf{M}(\overrightarrow{\mathfrak{m}}))))\ ,\ \mathsf{G}(\mathsf{T},\mathsf{M}(\overrightarrow{\mathfrak{m}}))\ \right) = \mathfrak{m}'$ (enhanced recontextualized reversed migrated model)

The actions performed by $\mathsf{C_g}$ can be informally described as follows:

1. deleted attributes of a non-deleted instance A which has been afterwards deleted by the tool are put on each instance B computed from A (according to the dependency graph, B depends on A)
2. references relating a deleted instance X and a non-deleted instance A which has been afterwards deleted by the tool are put between the recovered instance X and each instance B computed from A (according to the dependency graph B depends on A)
3. deleted references relating a non-deleted instance A1 and a non-deleted instance A2 which has been afterwards deleted by the tool are put between the remaining instance A1 and each instance B computed from A2 (according to the dependency graph B depends on A2)
4. deleted references relating the non-deleted instances A1 and A2, afterwards *both* deleted by the tool are put between each instance B1 computed from A1 and each instance B2 computed from A2 (according to the dependency graph B1 depends on A1 and B2 depends on A2)

More formally, the recontextualization by a dependency graph is specified by figure 16. To illustrate this approach, we consider again the three examples of rewriting tools namely *identity* (Id), *copy* (Cp), and *flattener* (Fl).

$$
\begin{array}{rcll}
\mathfrak{m}_\mathsf{k} & = & \mathsf{C_k}(\mathsf{R}(\mathsf{T}(\mathsf{M}(\overrightarrow{\mathfrak{m}})))) & \text{(recontextualized reversed migrated model)} \\
g & = & \mathsf{G}(\mathsf{T},\mathsf{M}(\overrightarrow{\mathfrak{m}})) & \text{(dependency graph)} \\
\mathfrak{m}' & = & \mathsf{C_g}(\mathfrak{m}_\mathsf{k}, g) & \text{(enhanced recontextualized reversed migrated model)} \\
\mathfrak{m}'.V & = & \mathfrak{m}_\mathsf{k}.V & \\
\mathfrak{m}'.E_a & = & \mathfrak{m}_\mathsf{k}.E_a \cup \{(i,a,s) \in \mathsf{T}(\mathsf{M}(\overrightarrow{\mathfrak{m}})).V \times \mathcal{A} \times \mathcal{S} \mid & \\
 & & \quad \exists i' \in g(i) \setminus \mathsf{T}(\mathsf{M}(\overrightarrow{\mathfrak{m}})).V,\ (i',a,s) \in \mathfrak{m}.E_a \wedge (i',a) \in \overrightarrow{\mathfrak{m}}.D_a\} & \\
\mathfrak{m}'.E_r & = & \mathfrak{m}_\mathsf{k}.E_r \cup \{(i_1,r,i_2) \in \mathfrak{m}_\mathsf{k}.V \times \mathcal{R} \times \mathfrak{m}_\mathsf{k}.V \mid & \\
 & & \quad \exists i_1' \in g(i_1) \setminus \mathfrak{m}_\mathsf{k}.V,\ (i_1',r,i_2) \in \mathfrak{m}.E_r \wedge ((i_1',r) \in \overrightarrow{\mathfrak{m}}.D_r \vee i_2 \in \overrightarrow{\mathfrak{m}}.D_i)\} & \\
 & & \vee\ \exists i_2' \in g(i_2) \setminus \mathfrak{m}_\mathsf{k}.V,\ (i_1,r,i_2') \in \mathfrak{m}.E_r \wedge ((i_1,r) \in \overrightarrow{\mathfrak{m}}.D_r \vee i_1 \in \overrightarrow{\mathfrak{m}}.D_i)\} & \\
 & & \vee\ \exists (i_1',i_2') \in (g(i_1) \setminus \mathfrak{m}_\mathsf{k}.V) \times (g(i_2) \setminus \mathfrak{m}_\mathsf{k}.V),\ (i_1',r,i_2') \in \mathfrak{m}.E_r \wedge (i_1',r) \in \overrightarrow{\mathfrak{m}}.D_r\} &
\end{array}
$$

Figure 16: Computation of a recontextualized model by a dependency graph

7.1 Identity

We consider the initial model $\mathfrak{m}$ of figure 3, and the migrated model $\mathsf{M}(\overrightarrow{\mathfrak{m}})$ of figure 7 corresponding to the initial model without actions. The application of *identity* to this migrated model leads to the same model, without any change. The recontextualized model by keys is depicted by figure 12.

We suppose that the dependency graph of *identity* is defined as follows. It simply states that each instance is computed (without any change) from and only from itself:

$$\mathsf{G}(\mathsf{Id}, \mathsf{M}(\overrightarrow{\mathfrak{m}})) \quad : \quad \begin{array}{rcl} \mathsf{Id}(\mathsf{M}(\overrightarrow{\mathfrak{m}})).V & \rightarrow & \mathcal{P}(\mathsf{M}(\overrightarrow{\mathfrak{m}}).V) \\ s_1 & \mapsto & \{s_1\} \\ s_2 & \mapsto & \{s_2\} \\ s_3 & \mapsto & \{s_3\} \end{array}$$

By definition of the recontextualization by dependency graphs (fig. 16), we have:

$$\begin{array}{rcl} \mathfrak{m}' & = & \mathsf{C_g}\ (\ \mathsf{C_k}(\mathsf{R}(\mathsf{Id}(\mathsf{M}(\overrightarrow{\mathfrak{m}})))\ ,\ \mathsf{G}(\mathsf{Id}, \mathsf{M}(\overrightarrow{\mathfrak{m}}))\ ) \\ \mathfrak{m}'.V & = & \mathsf{C_k}(\mathsf{R}(\mathsf{Id}(\mathsf{M}(\overrightarrow{\mathfrak{m}})))).V \\ \mathfrak{m}'.E_a & = & \mathsf{C_k}(\mathsf{R}(\mathsf{Id}(\mathsf{M}(\overrightarrow{\mathfrak{m}})))).E_a \cup \emptyset \\ \mathfrak{m}'.E_r & = & \mathsf{C_k}(\mathsf{R}(\mathsf{Id}(\mathsf{M}(\overrightarrow{\mathfrak{m}})))).E_r \cup \emptyset \end{array}$$

And thus : $\mathfrak{m}' = \mathsf{C_k}(\mathsf{R}(\mathsf{Id}(\mathsf{M}(\overrightarrow{\mathfrak{m}}))))$. In the case of *identity*, the recontextualized model by dependency graph brings nothing compared to the recontextualized model by keys. This situation is depicted by figure 17.

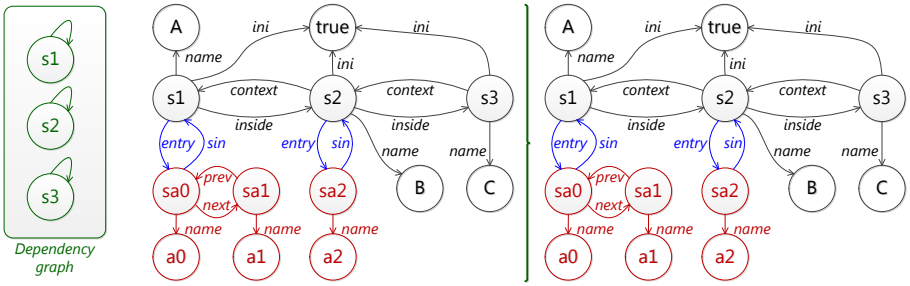


Figure 17: Recontextualization by graph of the model from figure 7

7.2 Copy

We consider again the initial model $\mathfrak{m}$ of figure 3, and the migrated model $\mathsf{M}(\overrightarrow{\mathfrak{m}})$ of figure 7 corresponding to the initial model without actions. The application of *copy* to this migrated model leads to an equivalent model where instances have been *recreated*. In this case, the dependency graph is defined as follows. It simply maps new instances to the corresponding initial instances:

$$\mathsf{G}(\mathsf{Cp}, \mathsf{M}(\overrightarrow{\mathfrak{m}})) \quad : \quad \begin{array}{rcl} \mathsf{Cp}(\mathsf{M}(\overrightarrow{\mathfrak{m}})).V & \rightarrow & \mathcal{P}(\mathsf{M}(\overrightarrow{\mathfrak{m}}).V) \\ s_4 & \mapsto & \{s_1\} \\ s_5 & \mapsto & \{s_2\} \\ s_6 & \mapsto & \{s_3\} \end{array}$$

By definition of the recontextualization by dependency graphs (fig. 16), we have:

$$\begin{array}{rcl} \mathfrak{m}' & = & \mathsf{C_g}\ (\ \mathsf{C_k}(\mathsf{R}(\mathsf{Cp}(\mathsf{M}(\overrightarrow{\mathfrak{m}})))\ ,\ \mathsf{G}(\mathsf{Cp}, \mathsf{M}(\overrightarrow{\mathfrak{m}}))\ ) \\ \mathfrak{m}'.V & = & \mathsf{C_k}(\mathsf{R}(\mathsf{Cp}(\mathsf{M}(\overrightarrow{\mathfrak{m}})))).V \\ \mathfrak{m}'.E_a & = & \mathsf{C_k}(\mathsf{R}(\mathsf{Cp}(\mathsf{M}(\overrightarrow{\mathfrak{m}})))).E_a \cup \emptyset \\ \mathfrak{m}'.E_r & = & \mathsf{C_k}(\mathsf{R}(\mathsf{Cp}(\mathsf{M}(\overrightarrow{\mathfrak{m}})))).E_r \cup \{(s_4, entry, sa_0), (s_5, entry, sa_2), (sa_2, sin, s_5), (sa_0, sin, s_4)\} \end{array}$$

Thanks to this mapping, entry actions can be connected to new instances as far as they are a copy of a deleted instance which was connected to actions before the migration. Finally, an equivalent model of the initial model is actually recovered. This situation is depicted by figure 18.

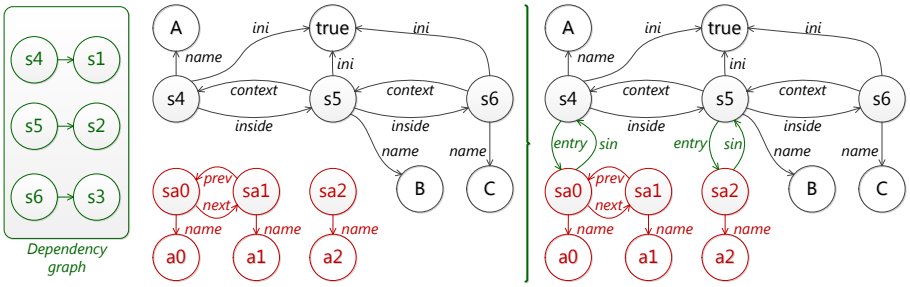


Figure 18: Recontextualization by graph of the model from figure 9

7.3 Flattener

We consider again the initial model $\mathfrak{m}$ of figure 3, and the migrated model $\mathsf{M}(\overrightarrow{\mathfrak{m}})$ of figure 7 corresponding to the initial model without actions. The application of *flatten* to this migrated model leads to the model of figure 10 which contains a unique state named ABC and without actions. As an illustration in this case, we suppose that the dependency graph is defined as follows. The flat state corresponds to a collapsed sequence of nested states. Thus, the remaining state was already a simple state before the flattening and it has been updated by all the super-states to which it belonged:

$$\mathsf{G}(\mathsf{Fl}, \mathsf{M}(\overrightarrow{\mathfrak{m}})) \quad : \quad \begin{array}{rcl} \mathsf{Fl}(\mathsf{M}(\overrightarrow{\mathfrak{m}})).V & \rightarrow & \mathcal{P}(\mathsf{M}(\overrightarrow{\mathfrak{m}}).V) \\ s_3 & \mapsto & \{s_1, s_2, s_3\} \end{array}$$

By definition of the recontextualization by dependency graphs (fig. 16), we have:

$$\begin{array}{rcl} \mathfrak{m}' & = & \mathsf{C_g}\ (\ \mathsf{C_k}(\mathsf{R}(\mathsf{Fl}(\mathsf{M}(\overrightarrow{\mathfrak{m}})))\ ,\ \mathsf{G}(\mathsf{Fl}, \mathsf{M}(\overrightarrow{\mathfrak{m}}))\ ) \\ \mathfrak{m}'.V & = & \mathsf{C_k}(\mathsf{R}(\mathsf{Fl}(\mathsf{M}(\overrightarrow{\mathfrak{m}})))).V \\ \mathfrak{m}'.E_a & = & \mathsf{C_k}(\mathsf{R}(\mathsf{Fl}(\mathsf{M}(\overrightarrow{\mathfrak{m}})))).E_a \cup \emptyset \\ \mathfrak{m}'.E_r & = & \mathsf{C_k}(\mathsf{R}(\mathsf{Fl}(\mathsf{M}(\overrightarrow{\mathfrak{m}})))).E_r \cup \{(s_3, entry, sa_0), (s_3, entry, sa_2), (sa_2, sin, s_3), (sa_0, sin, s_3)\} \end{array}$$

Thanks to this mapping, entry actions can be connected to the remaining instance s_3. However, the resulting graph does not corresponds to a valid model with regard to the initial metamodel (see figure 3). Indeed, the model we obtain contains two entry actions whereas at most one is expected. This situation is depicted by figure 19.

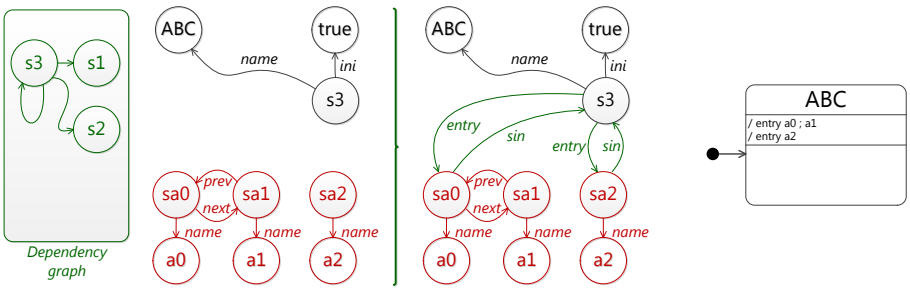


Figure 19: Recontextualization by graph of the model from figure 10

At this point, an update of the recontextualized model is needed in order to make it conform to the initial metamodel. This update requires domain specific knowledge and is therefore out of the scope of automatic recontextualization. In this example, action a2 clearly needs to be added to the first sequence of actions a0 ; a1. By this way, there is only one entry action associated

to the unique state ABC. As a result, we obtain a flattened version of the initial statecharts including entry actions whereas the reused flattener does not take them into account. This last transformation rely on the usual operational semantics of statecharts. It is illustrated by figure 20.

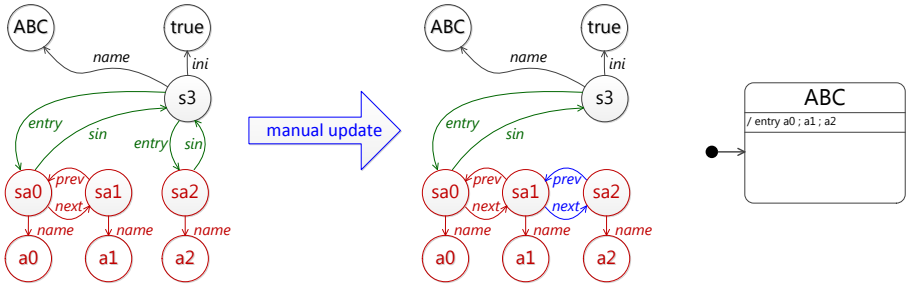


Figure 20: Transformation to a valid model with regard to the initial metamodel

8 Properties

The recontextualization by graphs is intended to extend the recontextualization by keys. The recontextualization by graphs (and therefore especially by keys) must not undo the modifications performed by the rewriting tool. These two properties can be verified within the formal framework of our semantic domain.

8.1 Recontextualization by graphs extends recontextualization by keys

Let $\mathfrak{m}$ be a given model, $\overrightarrow{\mathfrak{m}}$ a migration specification and T a rewriting tool. Let $\mathsf{G}(\mathsf{T}, \mathsf{M}(\overrightarrow{\mathfrak{m}}))$ be a dependency graph over $\mathsf{T}(\mathsf{M}(\overrightarrow{\mathfrak{m}}))$.

An edge of $\mathsf{G}(\mathsf{T}, \mathsf{M}(\overrightarrow{\mathfrak{m}}))$ relating an instance i to itself has no added value. Indeed, this edge states that i is an instance of the result which was already in the input of T, and which value depends (obviously) on itself.

We say that $\mathsf{G}(\mathsf{T}, \mathsf{M}(\overrightarrow{\mathfrak{m}}))$ is *semantically empty* if all its edges have *no added values*. More formally, by definition:

$$\mathsf{G}(\mathsf{T}, \mathsf{M}(\overrightarrow{\mathfrak{m}})) \text{ is } \textit{semantically empty} \quad \triangleq \quad \forall i \in \mathsf{T}(\mathsf{M}(\overrightarrow{\mathfrak{m}})).V,\ \mathsf{G}(\mathsf{T}, \mathsf{M}(\overrightarrow{\mathfrak{m}}))(i) \subseteq \{i\}$$

Property 1 *If* $\mathsf{G}(\mathsf{T}, \mathsf{M}(\overrightarrow{\mathfrak{m}}))$ *is* semantically empty*, then the recontextualization by graphs corresponds to the recontextualization by keys:* $\mathsf{C_g}\left(\mathsf{C_k}(\mathsf{R}(\mathsf{T}(\mathsf{M}(\overrightarrow{\mathfrak{m}})))),\ \mathsf{G}(\mathsf{T}, \mathsf{M}(\overrightarrow{\mathfrak{m}}))\right) = \mathsf{C_k}(\mathsf{R}(\mathsf{T}(\mathsf{M}(\overrightarrow{\mathfrak{m}}))))$

Proof *Let* $\mathfrak{m}_\mathsf{k} = \mathsf{C_k}(\mathsf{R}(\mathsf{T}(\mathsf{M}(\overrightarrow{\mathfrak{m}}))))$ *be the recontextualized reversed migrated model by keys. Let* $g = \mathsf{G}(\mathsf{T}, \mathsf{M}(\overrightarrow{\mathfrak{m}}))$ *be a* semantically empty *dependency graph. Let* $\mathfrak{m}' = \mathsf{C_g}(\mathfrak{m}_\mathsf{k}, g)$ *be the enhanced recontextualized model by* g*. By definition of* $\mathsf{C_g}$ *(fig. 16), we have:*

$$\mathfrak{m}'.V = \mathfrak{m}_\mathsf{k}.V$$

By definition of $\mathsf{C_g}$*, we also have:*

$$\begin{aligned}\mathfrak{m}'.E_a \quad = \quad & \mathfrak{m}_\mathsf{k}.E_a \cup \{(i,a,s) \in \mathsf{T}(\mathsf{M}(\overrightarrow{\mathfrak{m}})).V \times \mathcal{A} \times \mathcal{S} \mid \\ & \exists i' \in g(i) \setminus \mathsf{T}(\mathsf{M}(\overrightarrow{\mathfrak{m}})).V,\ (i',a,s) \in \mathfrak{m}.E_a \wedge (i',a) \in \overrightarrow{\mathfrak{m}}.D_a\}\end{aligned}$$

By definition of a semantically empty graph, $\forall i \in \mathsf{T}(\mathsf{M}(\overrightarrow{\mathfrak{m}})).V,\ g(i) \subseteq \{i\}$. *Thus we have:*

$$\forall i \in \mathsf{T}(\mathsf{M}(\overrightarrow{\mathfrak{m}})).V,\ g(i) \setminus \mathsf{T}(\mathsf{M}(\overrightarrow{\mathfrak{m}})).V = \emptyset$$

And then we have:

$$\begin{array}{rcl} \mathfrak{m}'.E_a & = & \mathfrak{m}_\mathsf{k}.E_a \cup \{(i,a,s) \in \mathsf{T}(\mathsf{M}(\overrightarrow{\mathfrak{m}})).V \times \mathcal{A} \times \mathcal{S} \mid \exists i' \in g(i) \setminus \mathsf{T}(\mathsf{M}(\overrightarrow{\mathfrak{m}})).V, \dots\} \\ & = & \mathfrak{m}_\mathsf{k}.E_a \cup \{(i,a,s) \in \mathsf{T}(\mathsf{M}(\overrightarrow{\mathfrak{m}})).V \times \mathcal{A} \times \mathcal{S} \mid \exists i' \in \emptyset, \dots\} \\ & = & \mathfrak{m}_\mathsf{k}.E_a \cup \emptyset \\ & = & \mathfrak{m}_\mathsf{k}.E_a \end{array}$$

By definition of $\mathsf{C_g}$, *we finally have:*

$$\begin{array}{rcl} \mathfrak{m}'.E_r & = & \mathfrak{m}_\mathsf{k}.E_r \cup \{(i_1,r,i_2) \in \mathfrak{m}_\mathsf{k}.V \times \mathcal{R} \times \mathfrak{m}_\mathsf{k}.V \mid \\ & & \quad \exists i'_1 \in g(i_1) \setminus \mathfrak{m}_\mathsf{k}.V,\ (i'_1,r,i_2) \in \mathfrak{m}.E_r \wedge ((i'_1,r) \in \overrightarrow{\mathfrak{m}}.D_r \vee i_2 \in \overrightarrow{\mathfrak{m}}.D_i)\} \\ & & \vee\ \exists i'_2 \in g(i_2) \setminus \mathfrak{m}_\mathsf{k}.V,\ (i_1,r,i'_2) \in \mathfrak{m}.E_r \wedge ((i_1,r) \in \overrightarrow{\mathfrak{m}}.D_r \vee i_1 \in \overrightarrow{\mathfrak{m}}.D_i)\} \\ & & \vee\ \exists (i'_1,i'_2) \in (g(i_1) \setminus \mathfrak{m}_\mathsf{k}.V) \times (g(i_2) \setminus \mathfrak{m}_\mathsf{k}.V),\ (i'_1,r,i'_2) \in \mathfrak{m}.E_r \wedge (i'_1,r) \in \overrightarrow{\mathfrak{m}}.D_r\} \end{array}$$

For the same reason and by definition of semantically empty graphs, if $i_1 \in \mathfrak{m}_\mathsf{k}.V$, *then* $g(i_1) \subseteq \{i_1\} \subseteq \mathfrak{m}_\mathsf{k}.V$ *and if* $i_2 \in \mathfrak{m}_\mathsf{k}.V$, *then* $g(i_2) \subseteq \{i_2\} \subseteq \mathfrak{m}_\mathsf{k}.V$. *And thus* $g(i_1) \setminus \mathfrak{m}_\mathsf{k}.V = \emptyset$ *and* $g(i_2) \setminus \mathfrak{m}_\mathsf{k}.V = \emptyset$. *Hence:*

$$\begin{array}{rcl} \mathfrak{m}'.E_r & = & \mathfrak{m}_\mathsf{k}.E_r \cup \{(i_1,r,i_2) \in \mathfrak{m}_\mathsf{k}.V \times \mathcal{R} \times \mathfrak{m}_\mathsf{k}.V \mid \exists i'_1 \in \emptyset, \cdots \vee \exists i'_2 \in \emptyset, \cdots \vee \exists (i'_1,i'_2) \in \emptyset^2, \dots\} \\ & = & \mathfrak{m}_\mathsf{k}.E_r \cup \emptyset \\ & = & \mathfrak{m}_\mathsf{k}.E_r \end{array}$$

The three components of $\mathfrak{m}'$ *and* $\mathfrak{m}_\mathsf{k}$ *coincide and thus we actually have* $\mathfrak{m}' = \mathfrak{m}_\mathsf{k}$. □

8.2 Recontextualization does not undo the rewriting

As mentioned in section 4, the action of a rewriting tool T is specified by the differences between a given input model and the corresponding output model. This specification is formally stated in figure 8 by *deleted elements* $\mathsf{T}^i_D(\mathfrak{m}), \mathsf{T}^a_D(\mathfrak{m}), \mathsf{T}^r_D(\mathfrak{m})$ and *added elements* $\mathsf{T}^i_A(\mathfrak{m}), \mathsf{T}^a_A(\mathfrak{m}), \mathsf{T}^r_A(\mathfrak{m})$.

The recontextualization aims at resetting the initial context on the result of T without undoing its specific actions. Thus, the model elements that have been added by the tool must not be deleted by the recontextualization, and the the model elements that have been deleted by the tool must not be recovered by the recontextualization.

This property holds under the following condition which means that the instances that are created by the tool ($\mathsf{T}^i_A(\mathsf{M}(\overrightarrow{\mathfrak{m}}))$) have new names with regard to instances of the initial context.

$$\mathsf{T}^i_A(\mathsf{M}(\overrightarrow{\mathfrak{m}})) \cap \mathfrak{m}.V = \emptyset$$

This condition can be easily satisfied by a renaming of the instances created by the tool (thanks to a kind of *post-processor*).

Property 2 *Let* $\mathfrak{m}$ *be a given model and* T *a rewriting tool. Let* $\mathfrak{m}'$ *be the recontextualized reversed migrated model by graphs:* $\mathfrak{m}' = \mathsf{C_g}\ (\ \mathsf{C_k}(\mathsf{R}(\mathsf{T}(\mathsf{M}(\overrightarrow{\mathfrak{m}})))\ ,\ \mathsf{G}(\mathsf{T}, \mathsf{M}(\overrightarrow{\mathfrak{m}}))\ )$. *Then we have:*

(1) deleted elements are not added:
$\mathsf{T}^i_D(\mathsf{M}(\overrightarrow{\mathfrak{m}})) \cap \mathfrak{m}'.V = \emptyset \quad \wedge \quad \mathsf{T}^a_D(\mathsf{M}(\overrightarrow{\mathfrak{m}})) \cap \mathfrak{m}'.E_a = \emptyset \quad \wedge \quad \mathsf{T}^r_D(\mathsf{M}(\overrightarrow{\mathfrak{m}})) \cap \mathfrak{m}'.E_r = \emptyset$

(2) added elements are not removed:
$\mathsf{T}^i_A(\mathsf{M}(\overrightarrow{\mathfrak{m}})) \subseteq \mathfrak{m}'.V \quad \wedge \quad \mathsf{T}^a_A(\mathsf{M}(\overrightarrow{\mathfrak{m}})) \subseteq \mathfrak{m}'.E_a \quad \wedge \quad \mathsf{T}^r_A(\mathsf{M}(\overrightarrow{\mathfrak{m}})) \subseteq \mathfrak{m}'.E_r$

Proof sketch *The second point is trivial since neither* $\mathsf{C_k}$ *nor* $\mathsf{C_g}$ *remove anything from the model* $\mathsf{T}(\mathsf{M}(\overrightarrow{\mathfrak{m}}))$ *resulting from* T *(cf. figures 11 and 16).*

About the first point, we consider the elements that are added by $\mathsf{C_k}$*, and then by* $\mathsf{C_g}$*. By definition of* $\mathsf{C_k}$ *and* T^i_D*, knowing that* R *is* identity *and that* $\mathsf{M}(\overrightarrow{\mathfrak{m}}).V = \mathfrak{m}.V \setminus \overrightarrow{\mathfrak{m}}.D_i$ *we have:*

$$\begin{array}{rcl} \mathfrak{m}'.V & = & \mathsf{C_k}(\mathsf{R}(\mathsf{T}(\mathsf{M}(\overrightarrow{\mathfrak{m}})))).V \\ & = & \mathsf{R}(\mathsf{T}(\mathsf{M}(\overrightarrow{\mathfrak{m}}))).V \cup \overrightarrow{\mathfrak{m}}.D_i \\ & = & \mathsf{T}(\mathsf{M}(\overrightarrow{\mathfrak{m}})).V \cup \overrightarrow{\mathfrak{m}}.D_i \end{array} \qquad \begin{array}{rcl} \mathsf{T}^i_D(\mathsf{M}(\overrightarrow{\mathfrak{m}})) & = & \mathsf{M}(\overrightarrow{\mathfrak{m}}).V \setminus \mathsf{T}(\mathsf{M}(\overrightarrow{\mathfrak{m}})).V \\ & = & (\mathfrak{m}.V \setminus \overrightarrow{\mathfrak{m}}.D_i) \setminus \mathsf{T}(\mathsf{M}(\overrightarrow{\mathfrak{m}})).V \end{array}$$

An thus we have:

$$\begin{array}{rcl} \mathsf{T}^i_D(\mathsf{M}(\overrightarrow{\mathfrak{m}})) \cap \mathfrak{m}'.V & = & \left((\mathfrak{m}.V \setminus \overrightarrow{\mathfrak{m}}.D_i) \setminus \mathsf{T}(\mathsf{M}(\overrightarrow{\mathfrak{m}})).V \right) \cap \left(\mathsf{T}(\mathsf{M}(\overrightarrow{\mathfrak{m}})).V \cup \overrightarrow{\mathfrak{m}}.D_i \right) \\ & = & \left(\mathfrak{m}.V \setminus (\overrightarrow{\mathfrak{m}}.D_i \cup \mathsf{T}(\mathsf{M}(\overrightarrow{\mathfrak{m}})).V) \right) \cap \left(\mathsf{T}(\mathsf{M}(\overrightarrow{\mathfrak{m}})).V \cup \overrightarrow{\mathfrak{m}}.D_i \right) \\ & = & \emptyset \end{array}$$

The proofs of $\mathsf{T}^a_D(\mathsf{M}(\overrightarrow{\mathfrak{m}})) \cap \mathfrak{m}'.E_a = \emptyset$ *and of* $\mathsf{T}^r_D(\mathsf{M}(\overrightarrow{\mathfrak{m}})) \cap \mathfrak{m}'.E_r = \emptyset$ *follow the same principles. By definition of* $\mathsf{M}(\overrightarrow{\mathfrak{m}}).E_a$ *and of* $\mathsf{M}(\overrightarrow{\mathfrak{m}}).E_a$ *(cf. figure 5),* T *does not takes as input any* edge *(attribute or reference) which is deleted (*$\overrightarrow{\mathfrak{m}}.D_a$ *or* $\overrightarrow{\mathfrak{m}}.D_r$*), or whose source and/or target is deleted (*$\overrightarrow{\mathfrak{m}}.D_i$*).* T *cannot delete such unavailable edges. Now by definition of* $\mathsf{C_k}$ *and* $\mathsf{C_g}$ *(cf. figures 11 and 16) the only added edges are taken from* $\overrightarrow{\mathfrak{m}}.D_a$ *and* $\overrightarrow{\mathfrak{m}}.D_r$*, or they are related to a deleted instance from* $\overrightarrow{\mathfrak{m}}.D_i$*.* □

A Modif

A.1 Modif operators

Among the *metamodel refactoring* operators that are available in Modif, we focus on *delete*. It applies to *classes*, *attributes* and *references*. Its scope and parameters are defined in table 1.

Operator	Scope	Parameters
delete	class	class: c
delete	attribute	class: c; name: a
delete	reference	class: c; name: r

Table 1: Modif operators: scope and parameters

A.2 Modif specification

We note $\mathcal{C}$ the name space for classes. This name space is only relevant at the metamodel level. We call *Modif specification* and note modif a possibly empty sequence of operators noted op (see figure 21).

$$\mathsf{modif} \triangleq \left(\mathsf{op}_1 , \ \ldots \ , \ \mathsf{op}_n \right) \qquad \text{where } \forall i \in \{1, \ldots, n\}, \mathsf{op}_i \in \left\{ \begin{array}{ll} & \{\mathsf{delete}\} \times \mathcal{C} \\ \cup & \{\mathsf{delete}\} \times \mathcal{C} \times \mathcal{A} \\ \cup & \{\mathsf{delete}\} \times \mathcal{C} \times \mathcal{R} \end{array} \right.$$

Figure 21: Modif specification

A Modif specification depends on a given metamodel and is defined by a sequence of Modif operators. It is intended to be used:

- together with the metamodel to produce a *refactored metamodel*
- together with the metamodel and a conforming model to produce a *migration specification*

A.3 Generation of migration specifications

We suppose we have a metamodel noted $\mathfrak{mm}$, a conforming model noted $\mathfrak{m}$ and a valid Modif specification noted modif. The scope of modif is $\mathfrak{mm}$.

We call *migration generator* and note $\mathsf{G_M}$ the tool aiming at producing a valid migration specification from $\mathfrak{mm}$, $\mathfrak{m}$ and modif:

$$\mathsf{G_M}(\mathsf{modif}, \mathfrak{mm}, \mathfrak{m}) = \overrightarrow{\mathfrak{m}}$$

This generator compile the sequence of operator calls in modif to produce the resulting migration specification.