Refactoring information systems: association folding and unfolding

Article *in* ACM SIGSOFT Software Engineering Notes · August 2011

DOI: 10.1145/1988997.1989011 · Source: DBLP

Michael Löwe
Fachhochschule für die Wirtschaft Hannover-FHDW

REFACTORING INFORMATION SYSTEMS
– ASSOCIATION FOLDING AND UNFOLDING –

MICHAEL LÖWE

FHDW Hannover
Freundallee 15, 30173 Hannover, Germany
michael.loewe@fhdw.de

ABSTRACT. Agile software development methods need powerful refactoring tools. For the time being, there is only little support for the refactoring of *complete productive* systems, i. e. data and functional models *together with* possibly large databases and possibly long running processes. In this paper, we present an algebraic framework for model refactorings and induced data migrations. Within the framework theoretical results can be proven which are needed as a solid basis for practical tool support: we investigate sequential composition of refactorings here. The framework in this paper is a simplified version of our previous approaches, provides, however, more modelling power.

1. INTRODUCTION

Agile and test-driven software development needs powerful refactoring support, since changes to the system under development are often. Design patterns[9] are introduced on demand not in advance.[1]

For the time being, the agile method and software refactoring work quite good as long as the system being developed is not in operation. The situation is different for productive real-life systems that possess huge databases. Changing the structure of the software (i. e. the system's model) requires corresponding changes to the data. Here, we do not find satisfactory refactoring support that automatically induces (on-demand) data migrations from model refactorings. To some extent, the necessary theory for a suitable tool support is missing yet.

In [12, 13, 15, 16, 17] we started to develop a formal categorical framework for refactoring-induced data migrations.[2] In this paper, we continue and simplify this work. While all former papers explicitely distinguish between association and inheritance structure, the simplified framework in this paper handles all relations the same way. By this unification, all refactorings that work for inheritance structures become available for arbitrary associations as well. Thus, the introduction of a proxy pattern[9] works the same way as interface extraction, compare section 3.

The paper is organised as follows. Section 2 describes the simplified framework and first theoretical results. Section 3 shows its practical applicability by a small but typical sample refactoring process. Section 4 reestablishes the results for sequential composition of refactorings, that are well-known for the frameworks in [13, 15], in the new context. The conclusions in section 5 discuss issues for future research.

Acknowledgements: The results presented here are the outcome of a master course on the "Theory of Continuous Engineering" held from January to March 2011 at the FHDW Hannover. The participating students were Thomas Bremer, Jonas Büth, Stefan Christian, Sönke Küper, Benjamin Rogge, and Roman Smirnov.

[1]The ideas and mechanisms of software refactoring have already been incorporated in modern IDEs like Eclipse[4].

[2]A similar problem is addressed in [10], where operators on the meta-model level induce model migrations. For applications of graph grammars to the problem of model transformation see [18].

2. The Framework

Definition 1. (Meta-Model) The basis for all constructions in the following is the category of algebras and homomorphisms wrt. the following specification:

```
Meta =
    Sorts:
         O[bject]
    Operations:
         s[ource]: O → O
         t[arget]: O → O
    Axioms:
         (a1) s(s(x)) = s(x)
         (a2) t(t(x)) = t(x)
         (a3) s(x) = x ⟺ t(x) = x
```

In each algebra wrt. this meta-model, we can distinguish *vertices*, i. e. objects n such that $\texttt{source}(n) = n$ and $\texttt{target}(n) = n$, and *edges*, i. e. objects e such that $\texttt{source}(e) \neq e$ and $\texttt{target}(e) \neq e$. The axioms (a1) and (a2) guarantee that there are no edges between edges and the axiom (a3) prevents "half edges". Note that the axioms imply `s(t(x))=t(x)` and `t(s(x))=s(x)`.

Definition 2. (Model and Instance) A *model* is a `Meta`-algebra. An *instance* wrt. a model M is a `Meta`-homomorphism $i : I \to M$ satisfying:

(a4) $i\texttt{(x) = s(}i\texttt{(x))} \implies \texttt{s(x) = x}$
(a5) $i\texttt{(x)} = i\texttt{(y)} \wedge \texttt{s(x) = s(y)} \wedge \texttt{t(x) = t(y)} \implies \texttt{x = y}$

The object-oriented point of view interprets nodes in the model as *classes* and the edges as *associations*. Analogously, nodes in the instance I can be seen as *objects* and edges as *links*. The homomorphism $i : I \to M$ provides the *typing* (or *instance-of*-mapping) for every instance item in I. Note that we do not model inheritance explicitly. The inheritance structure must be modeled by some `is-a` associations in the model and is reflected by corresponding links in the instance.

Every instance being a `Meta`-homomorphism guarantees that nodes in the instance are typed in nodes in the model. Axiom (a4) provides the opposite property, namely that *only* nodes in the instance can be typed in model nodes and instance edges must be typed in model edges. Axiom (a5) forbids multiple edges of the same type between the same source and target node, since these edges cannot be distinguished by the context.

From the categorical point of view, the class of all instances wrt. a fixed model M is the full subcategory of the comma category $\texttt{Meta}{\downarrow}M$ the objects of which satisfy the axioms (a4) and (a5). We call this category $\texttt{Inst}(M)$. As prerequisites for the migration framework that is elaborated below, we need the following properties of this category.

Lemma 3. (Reflection into $\texttt{Inst}(M)$) *For any model M, there is a free construction $\equiv_M$: $\texttt{Meta}{\downarrow}M \to \texttt{Inst}(M)$ wrt. the inclusion functor $\subset$: $\texttt{Inst}(M) \to \texttt{Meta}{\downarrow}M$.*

Proof. We show that $\texttt{Inst}(M)$ is closed under (i) subobjects and (ii) products:

(i) Let $i : I \to M \in \texttt{Inst}(M)$ and $j : J \rightarrowtail I$ be a monomorphism, then $i \circ j : J \to M$ satisfies the axioms:

(a4) Let $i(j(x)) = \texttt{s}(i(j(x)))$. Since i satisfies the axioms, $\texttt{s}(j(x)) = j(x)$. Thus, $j(\texttt{s}(x)) = \texttt{s}(j(x)) = j(x)$ and $s(x) = x$ since j is monic.

(a5) Let $i(j(x)) = i(j(y))$, $\mathtt{s}(x) = \mathtt{s}(y)$, and $\mathtt{t}(x) = \mathtt{t}(y)$. Then $\mathtt{s}(j(x)) = j(\mathtt{s}(x)) = j(\mathtt{s}(y)) = \mathtt{s}(j(y))$ and $\mathtt{t}(j(x)) = j(\mathtt{t}(x)) = j(\mathtt{t}(y)) = \mathtt{t}(j(y))$. Since i satisfies (a5), $j(x) = j(y)$ and $x = y$, since j is monic.

(ii) Let $i : I \to M, j : J \to M \in \mathtt{Inst}(M)$, then the product of i and j is the pullback $(i^* : P \to J, j^* : P \to I)$. Again, we show satisfaction of the axioms[3]:

(a4) Let $i(j^*(x)) = \mathtt{s}(i(j^*(x)))$. Then also $j(i^*(x)) = \mathtt{s}(j(i^*(x)))$. Since i and j satisfy (a4), we conclude $j^*(\mathtt{s}(x)) = \mathtt{s}(j^*(x)) = j^*(x)$ and $i^*(\mathtt{s}(x)) = \mathtt{s}(i^*(x)) = i^*(x)$. Since i^* and j^* are jointly monic, $\mathtt{s}(x) = x$.

(a5) Let $i(j^*(x)) = i(j^*(y))$ [as well as $j(i^*(x)) = j(i^*(y))$], $\mathtt{s}(x) = \mathtt{s}(y)$, and $\mathtt{t}(x) = \mathtt{t}(y)$. Then $\mathtt{s}(j^*(x)) = j^*(\mathtt{s}(x)) = j^*(\mathtt{s}(y)) = \mathtt{s}(j^*(y))$ and $\mathtt{t}(j^*(x)) = j^*(\mathtt{t}(x)) = j^*(\mathtt{t}(y)) = \mathtt{t}(j^*(y))$. Since i satisfies (a5), we obtain $j^*(x) = j^*(y)$. A symmetrical argument shows $i^*(x) = i^*(y)$. Again, i and j being jointly monic provides $x = y$. □

Corollary 4. (Construction of the Reflection into $\mathtt{Inst}(M)$) $\equiv_M (i : I \to M)$ *can be constructed as follows: (i) Let $I_{\equiv^0_{a4}}$ be the quotient of I where $\equiv^0_{a4}$ is the smallest congruence containing $\approx^0_{a4} = \{(x, \mathtt{s}(x)) : i(x) = \mathtt{s}(i(x))\}$ and $\equiv^0_{a4} : I \to I_{\equiv^0_{a4}}$ its natural homomorphism. Since $x \approx^0_{a4} \mathtt{s}(x) \implies i(x) = \mathtt{s}(i(x)) = i(\mathtt{s}(x))$ there is $i' : I_{\equiv^0_{a4}} \to M$ such that $i' \circ \equiv^0_{a4} = i$. (ii) Let $(I_{\equiv^0_{a4}})_{\equiv^0_{a5}}$ be the quotient of $I_{\equiv^0_{a4}}$ where $\equiv^0_{a5}$ is the smallest congruence containing $\approx^0_{a5} = \{(x, y) : i'(x) = i'(y), \mathtt{s}(x) = \mathtt{s}(y), \mathtt{t}(x) = \mathtt{t}(y)\}$ and $\equiv^0_{a5} : I_{\equiv^0_{a4}} \to (I_{\equiv^0_{a4}})_{\equiv^0_{a5}}$ its natural homomorphism. Since $x \approx^0_{a5} y \implies i'(x) = i'(y)$ there is $i'' : (I_{\equiv^0_{a5}})_{\equiv^0_{a5}} \to M$ such that $i'' \circ \equiv^0_{a5} = i'$. Define $\equiv_M (i) = i''$ and the universal morphism $u_i : i \to i''$ by $u_i := \equiv^0_{a5} \circ \equiv^0_{a4}$.*

Proof. (Sketch) Each vertex satisfies axiom (a4). $\equiv^0_{a4} : I \to I_{\equiv^0_{a4}}$ "produces" new vertices only. Therefore $\approx^1_{a4} = \{(x, \mathtt{s}(x)) : i'(x) = \mathtt{s}(i'(x))\}$ contains reflexive pairs only. $\equiv^0_{a5} : I_{\equiv^0_{a4}} \to (I_{\equiv^0_{a4}})_{\equiv^0_{a5}}$ "identifies" edges typed in edges only. Therefore, it does not "produce" any new situation that satisfies the premise of axiom (a4) or (a5). That means that ${\approx''}^0_{a4} = \{(x, \mathtt{s}(x)) : i''(x) = \mathtt{s}(i''(x))\}$ and $\approx^1_{a5} = \{(x, y) : i''(x) = i''(y), \mathtt{s}(x) = \mathtt{s}(y), \mathtt{t}(x) = \mathtt{t}(y)\}$ contain reflexiv pairs only. □

In the following, we need this corollary:

Corollary 5. (Identification of Paths) *If $\equiv_M(i : I \to M) = (i' : I' \to M, u : I \to I')$ and $i(x) = i(y)$ is a vertex in M, then $u(x) = u(y)$, if and only if there is a sequence of items $z_0, x = z_1, \cdots, z_{n-1} = y, z_n$ with $n = 2(m) + 1$, $m \geq 1$ such that $i(z_p) = i(z_q)$ and $u(z_p) = u(z_q)$ for $0 \leq p, q \leq n$ and the sequence constitutes a path, i. e. $(z_{r-1} = \mathtt{s}(z_r) \land \mathtt{t}(z_r) = z_{r+1}) \lor (z_{r-1} = \mathtt{t}(z_r) \land \mathtt{s}(z_r) = z_{r+1})$ for $r = 2s + 1$, $0 \leq s < n/2$.*

Proof. (Sketch) The if-part is trivial. For the only-if-part, we choose the path (1) $z_0 = x, z_1 = x = y, z_2 = y$ in the case that $x = y$, (2) the path $z_0 = \mathtt{t}(x), z_1 = x, z_2 = \mathtt{s}(x) = y, z_3 = y, z_4 = y$ if $x = \mathtt{s}(x)$ due to axiom (a4), (3) the path $z_0 = \mathtt{s}(x), z_1 = x, z_2 = \mathtt{t}(x) = y, z_3 = y, z_4 = y$ if $x \equiv \mathtt{t}(x)$ due to $x \equiv \mathtt{s}(x)$[4], (4) the paths in inverse order for symmetry, and (5) the composition of paths for the transitive closure of $\equiv$. □

Lemma 6. (Pullback Functor) *Let $i : I \to M$ be in $\mathtt{Inst}(M)$, $l : M \leftarrow N$ be any $\mathtt{Meta}$-homomorphism, and $(i^* : J \to N, l^* : J \to I)$ be a pullback of the pair (i, l), then $i^* \in \mathtt{Inst}(N)$.*

[3]The extension from finite to infinite products is straightforward.

[4]Note that the operational closure of $\equiv$ provides $\mathtt{t}(x) \equiv \mathtt{t}(\mathtt{s}(x))$ if $x \equiv \mathtt{s}(x)$. Axioms (a1) - (a3) imply $\mathtt{t}(\mathtt{s}(x)) \equiv \mathtt{s}(x)$. Putting everything together we obtain $\mathtt{t}(x) \equiv \mathtt{t}(\mathtt{s}(x)) \equiv \mathtt{s}(x) \equiv x$.

Proof. Since all participating algebras satisfy axioms (a1) - (a3), the pullback object J also satisfies them, since it is constructed as a subobject of the product of I and N. It remains to show that i^* satisfies the axioms (a4) and (a5):

(a4) Let $i^*(x) = \mathtt{s}(i^*(x))$. Then (i) $i^*(x) = \mathtt{s}(i^*(x)) = i^*(\mathtt{s}(x))$, $l(i^*(x)) = l(i^*(\mathtt{s}(x)))$, and (by $l \circ i^* = i \circ l^*$) $i(l^*(x)) = i(l^*(\mathtt{s}(x))) = \mathtt{s}(i(l^*(x)))$. Since i satisfies (a4), (ii) $l^*(x) = \mathtt{s}(l^*(x)) = l^*(\mathtt{s}(x))$. The morphisms i^* and l^* being jointly monic together with (i) and (ii) provides $x = \mathtt{s}(x)$.

(a5) Let (i) $i^*(x) = i^*(y)$, $\mathtt{s}(x) = \mathtt{s}(y)$, and $\mathtt{t}(x) = \mathtt{t}(y)$. Then $l(i^*(x)) = l(i^*(y))$ and (by $l \circ i^* = i \circ l^*$) $i(l^*(x)) = i(l^*(y))$. We also obtain $\mathtt{s}(l^*(x)) = l^*(\mathtt{s}(x)) = l^*(\mathtt{s}(y)) = \mathtt{s}(l^*(y))$ and $\mathtt{t}(l^*(x)) = l^*(\mathtt{t}(x)) = l^*(\mathtt{t}(y)) = \mathtt{t}(l^*(y))$. Since i satisfies (a5), (ii) $l^*(x) = l^*(y)$. The properties (i) and (ii) provide $x = y$, since l^* and i^* are jointly monic. □

Corollary 7. (Free Construction wrt. the Pullback Functor) *If $r : M \to N$ is any* `Meta`*-homomorphism, there is a free construction* $\mathrm{F}(r) : \mathtt{Inst}(M) \to \mathtt{Inst}(N)$ *with respect to the pullback functor* $\mathrm{PB}(r) : \mathtt{Inst}(N) \to \mathtt{Inst}(M)$.

Proof. The free construction $\mathrm{F}(r)$ is obtained by the composition $\equiv_N \circ\, \mathrm{C}(r) : \mathtt{Inst}(M) \to \mathtt{Inst}(N)$ of (i) the left-adjoint $\mathrm{C}(r) : \mathtt{Meta} \downarrow M \to \mathtt{Meta} \downarrow N$ to the pullback functor $\mathrm{PB}(r) : \mathtt{Meta} \downarrow N \to \mathtt{Meta} \downarrow M$, which maps $i : I \to M$ to $r \circ i : I \to N$, and (ii) the reflection $\equiv_N : \mathtt{Meta}{\downarrow}N \to \mathtt{Inst}(N)$. □

Definition 8. (Refactoring) A *model refactoring* from a model M_o to a model M_n is given by a span of homomorphisms $M_o \overset{l}{\leftarrow} M^* \overset{r}{\rightarrow} M_n$.

Note that neither the left-hand side l nor the right-hand side r is required to be monic or epic. The left-hand side of a refactoring is able to express (i) deletion of model items[5], i. e. the objects not in the image of l, and (ii) unfolding or copying of objects, i. e. objects having more than one preimage under l. The right-hand side is able to express (iii) addition of model items, i. e. the objects not in the image of r, and (iv) folding of objects, i. e. objects having more than one preimage under r.

Definition 9. (Induced Migration) Given an instance $i_o : I_o \to M_o$, a refactoring $M_o \overset{l}{\leftarrow} M^* \overset{r}{\rightarrow} M_n$ induces the *migration* of i_o to $i_n = \mathrm{F}(r) \circ \mathrm{PB}(l)(i_o)$, where $\mathrm{PB}(l) : \mathtt{Inst}(M_o) \to \mathtt{Inst}(M^*)$ is the pullback functor induced by the left-hand side l of the refactoring and $\mathrm{F}(r) : \mathtt{Inst}(M^*) \to \mathtt{Inst}(M_n)$ is the free construction induced by the right-hand side. The whole situation is depicted in figure 1. We also write $i_n = (l, r)(i_o)$ for the application of a refactoring to a suitable instance.

$$\begin{tikzcd}
M_o & M^* \arrow[l, "l"'] \arrow[r, "r"] & M_n \\
I_o \arrow[u, "i_o"] & I^* \arrow[l, "l'"'] \arrow[r, "r' = \equiv_{M_n}"] \arrow[u, "{i^* = \mathrm{PB}(l)(i_o)}"'] & I_n \arrow[u, "{i_n = \mathrm{F}(r)(i^*)}"']
\end{tikzcd}$$

FIGURE 1. Induced Migration

[5]Most practical refactorings do not use deletion, since deletion of types requires deletion of all objects typed in these types.

Note that the effect of an induced migration does not depend on the concrete choice of the intermediate model M^* in the refactoring: Refactorings $M_o \xleftarrow{l_1} M_1^* \xrightarrow{r_1} M_n$ and $M_o \xleftarrow{l_2} M_2^* \xrightarrow{r_2} M_n$ are *equivalent* if there is an isomorphism $\cong\colon M_1^* \to M_2^*$ with $l_2 \circ \cong = l_1$ and $r_2 \circ \cong = r_1$. If $I_o \xleftarrow{l_1'} I_1^* \xrightarrow{i_1^*} M_1^*$ and $I_o \xleftarrow{l_2'} I_2^* \xrightarrow{i_2^*} M_2^*$ are the pullbacks of (i_o, l_1) and (i_o, l_2) respectively, we obtain an isomorphism $\cong^*\colon I_1^* \to I_2^*$ with $l_2' \circ \cong^* = l_1'$ and $i_2^* \circ \cong^* = \cong \circ i_1^*$. And if $I_1^* \xrightarrow{r_1'} I_n^1 \xrightarrow{i_n^1} M_n$ and $I_2^* \xrightarrow{r_2'} I_n^2 \xrightarrow{i_n^2} M_n$ are the free constructions on the right-hand side for (i_1^*, r_1) and (i_2^*, r_2) respectively, we obtain an isomorphism $\cong_n\colon I_n^1 \to I_n^2$ such that $\cong_n \circ r_1' = r_2' \circ \cong^*$ and $i_n^2 \circ \cong_n = i_n^1$. Thus, the result of a migration is unique up to isomorphism independent of the choice of the intermediate model in the applied refactoring.

That these notions of refactoring and migration are reasonable from a practical as well as from a theoretical point of view is shown in the next two sections.

3. Examples

In this section, we demonstrate how practical refactorings, like *interface extraction* or *introduction of a proxy*, can be modelled in the proposed framework. We also demonstrate how the induced migrations work on the instance level.

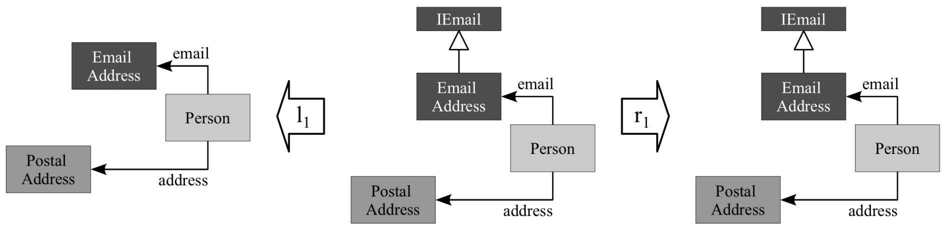

Figure 2. Interface Extraction (1)

Example 10. (Interface Extraction) A typical refactoring is the extraction of an interface. Figure 2 depicts an example of such a refactoring in our framework: The left-hand side consists of three classes[6], namely “Person”, “Postal Address”, and “Email Address”, and two associations, i. e. “email” and “address”. The right-hand side of the refactoring coincides with the intermediate model and possesses an additional class “IEmail” and an additional (unnamed) inheritance association[7]. The mapping of the homomorphisms (l_1, r_1) is indicated by various shades of grey for classes. The mapping of the associations is – in most cases – uniquely determined by the homomorphism condition. In figure 2, r_1 is the identity and l_1 maps “IEmail”, “Email Address”, and the inheritance relation in the intermediate model to “Email Address” in the left-hand side. The rest is mapped one to one.

The refactoring in figure 2 is an example for a non-injective left-hand side. Its effect on the instance level is the unfolding of structure out of a single object. A migration that is induced by “Interface Extraction (1)” is depicted in figure 3.

Note that we represent the inheritance structure on the instance level. That means that inheritance structures objects the same way as it structures classes. Due to

[6]In the examples, we call the vertices and the edges in the model classes and associations resp. and use an UML2-like[14] notation.

[7]Note that our model does not distinguish between associations and inheritance relations. The different notation in the examples is for illustrative purposes only.

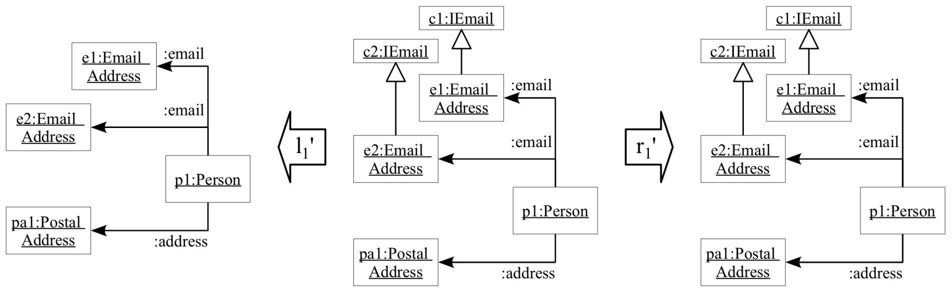

FIGURE 3. Induced Migration of "Interface Extraction (1)"

this similar structure of objects and classes, the typing (or instance-of relation) – in figure 3 indicated by the UML2-like notation for objects – becomes a simple homomorphism in our framework.

Figure 4 depicts another refactoring that extracts an interface, in this case from "Postal Address".

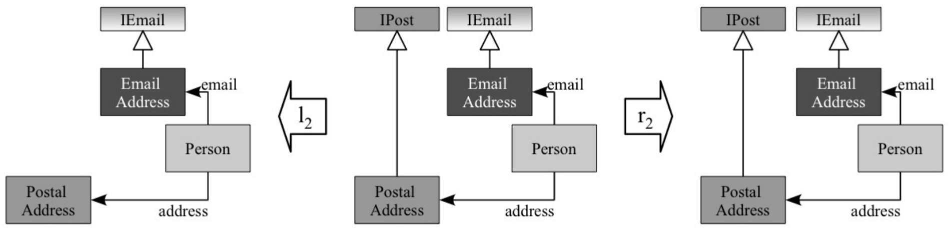

FIGURE 4. Interface Extraction (2)

Example 11. (Type Abstraction) The interface abstractions we presented above prepare a type abstraction, namely the abstraction of "IEmail" and "IPost" to an abstract class "Channel". The appropriate refactoring in our framework is depicted in figure 5. Here, the left-hand side l_3 is the identity and the right-hand side r_3 is not injective. It maps "IPost" and "IEmail" to "Channel"[8]. The migrations induced by this refactoring do not change the instance structure, they only change the typing of the objects, i. e. objects of type "IEmail" and "IPost" possess the same type "Channel" after migration.

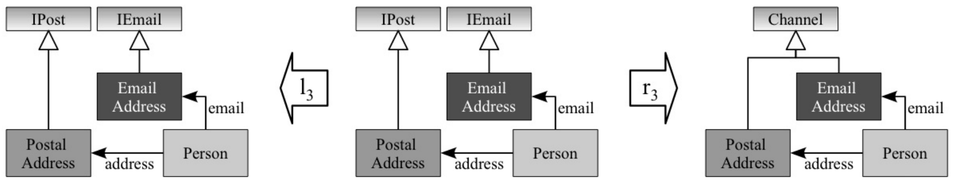

FIGURE 5. Type Abstraction (3)

[8]Note that the names for classes and associations in the examples are not relevant to the refactoring homomorphisms. They are just used as an easy reference to items in the example descriptions.

The model in figure 5 allows to abstract the two associations "email" and "address" into one. This goal is achieved by three steps described in the next example.

Example 12. (Pulling up Associations) The refactoring in figure 6 demonstrates how associations can be pulled up to a more abstract type within the proposed framework.

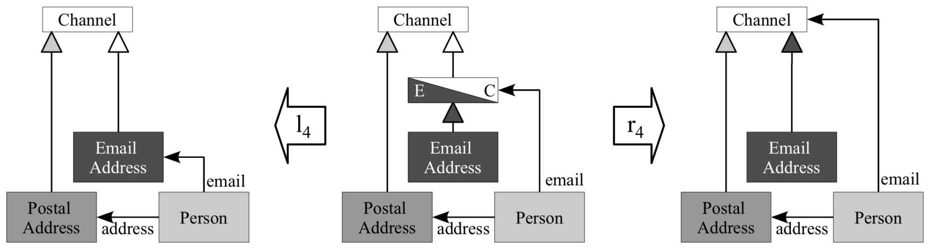

FIGURE 6. Pulling up Associations (4)

Note that neither l_4 nor r_4 is injective in this example. The left-hand side l_4 maps the 'helper class' "E\C", the class "Email Address", and the inheritance between them to the class "Email Address" in the refactoring's left-hand side. The right-hand side r_4 maps the 'helper class' "E\C", the class "Channel", and the inheritance between them to the class "Channel" in the refactoring's right-hand side. (The mapping of the inheritance relations is indicated by the different shades of grey in the arrow heads.)

Continuing the refactoring process in our example, we can also pull up the "address" association by the same mechanism as in figure 6. Using "Association Abstraction" (without example), a refactoring comparable to the one in figure 5 for associations, we can obtain the model which is the left-hand side in the refactoring of figure 7.

The application of all presented sample refactorings to any suitable instance does not delete or add any item, it just unfolds or folds structure out of resp. into objects.

Example 13. (Proxy Insertion) Figure 7 depicts a more complex refactoring, namely the insertion of a proxy pattern for the "Channels". This example shows that the framework is not restricted to refactorings concerning inheritance structures only. The left-hand homomorphism l_5 unfolds the class "Channel" into 4 classes ("Channel", "IChannel", "CProxy", and "ICProxy"), two inheritance relations (white arrow heads), and one association ("obj"). The right-hand side r_5 abstracts the two types "IChannel" and "ICProxy" to the final interface "IChannel".

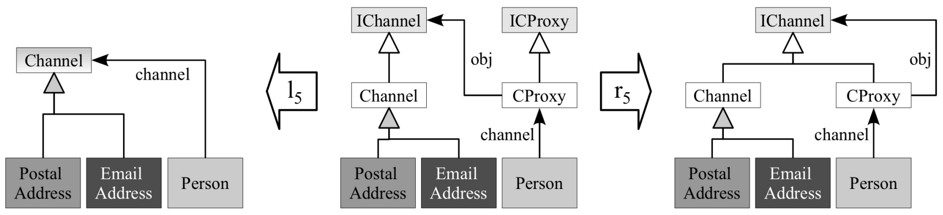

FIGURE 7. Proxy Insertion (5)

These examples demonstrate that the presented framework is able to model many practical refactorings on the model level and provides suitable induced migrations on the instance level. The examples showed that a practical refactoring process uses a sequence of restructuring steps in order to achieve a final goal: the (poor) model in the left-hand side of figure 2 has been restructured by several steps to the model in the right-hand side of figure 7[9]. Thus, the question arises if refactorings can be composed such that the induced migrations compose as well. This question is investigated in the next section.

4. Composition of Refactorings and Migrations

The composition of refactorings is provided by the standard categorical composition of spans (composition of categorical multi-relations):

Definition 14. (Composition of Refactorings) Refactorings $M_o^1 \xleftarrow{l_1} M_1^* \xrightarrow{r_1} M_n^1$ and $M_o^2 \xleftarrow{l_2} M_2^* \xrightarrow{r_2} M_n^2$ can be composed if $M_n^1 = M_o^2$. In this case, the composition $(l_2, r_2) \circ (l_1, r_1)$ is given by the refactoring $M_o^1 \xleftarrow{l_1 \circ l_2^*} M_{12}^* \xrightarrow{r_2 \circ r_1^*} M_n^2$ where $M_1^* \xleftarrow{l_2^*} M_{12}^* \xrightarrow{r_1^*} M_2^*$ is the pullback of $M_1^* \xrightarrow{r_1} M_n^1 = M_o^2 \xleftarrow{l_2} M_2^*$.
Note that the composition of two refactorings is given up to isomorphism only due to the pullback construction involved. But the variation of the pullback object always leads to equivalent refactorings.

Theorem 15. *(Composition of Migrations) If two refactorings $(l_1, r_1) : M_o^1 \to M_n^1$ and $(l_2, r_2) : M_o^2 \to M_n^2$ can be composed, then for any suitable instance $i : I \to M_o^1$, we obtain $[(l_2, r_2) \circ (l_1, r_1)](i) = (l_2, r_2)[(l_1, r_1)(i)]$, i. e. the application of the composed refactoring results in the same instance (up to isomorphism) as the sequential application of the individual refactorings.*

$$\begin{tikzcd}
 & & & M_{12}^* \arrow[ddll, "l_2^*"'] \arrow[drr, "r_1^*"] & & & & \\
 & & & & & M_2^* \arrow[rr, "r_2"] \arrow[dddlll, "l_2"'] & & M_n^2 \\
M_o & M_1^* \arrow[l, "l_1"'] \arrow[ddr, "r_1"] & & I_{12}^* \arrow[uu, "i_{12}^*"] \arrow[dddll, "l_2^{*\prime}"'] \arrow[drrr, "r_1^{*\prime}"] \arrow[ddrr, "u"] & & & & \\
 & & & & & & I'' \arrow[ul, "i''"'] \arrow[dl, "f"] \arrow[r, "r_2''"] & I_n^2 \arrow[uu, "i^2_n"'] \\
 & & M_n^1 & & & I_2^* \arrow[uuu, "i_2^*"] \arrow[ddlll, "l_2'"'] & & \\
I_o \arrow[uuu, "i_o"] & I_1^* \arrow[l, "l_1'"] \arrow[uuu, "i_1^*"] \arrow[dr, "r_1'"'] & & & & & & \\
 & & I_n^1 \arrow[uu, "i'_n = i^2_o"] & & & & &
\end{tikzcd}$$

Figure 8. Composition of Migrations

Proof. Consider the diagram in figure 8. It depicts the migration with (l_1, r_1), parts of the migration with (l_2, r_2), and the migration with $(l_2, r_2) \circ (l_1, r_1) = (l_1 \circ l_2^*, r_2 \circ r_1^*)$: (i_1^*, l_1') is a pullback and the left-hand side of the first migration, (i_n^1, r_1') is the right-hand side of the first migration, and (i_2^*, l_2') is a pullback and the

[9]Note that the induced migrations do not drop any objects!

left-hand side of the second migration. Note that r_1' is surjective by construction, compare corollary 4. The pair $(l_1' \circ l_2^{*\prime}, i_{12}^*)$ is a pullback and depicts the left-hand side of the migration with the composed refactoring. Since pullbacks can be decomposed, $(l_2^{*\prime}, i_{12}^*)$ is pullback of (i_1^*, l_2^*). Since pullbacks compose, $(r_1^* \circ i_{12}^*, l_2^{*\prime})$ is pullback of $(r_1 \circ i_1^*, l_2)$. The right-hand side of the composed migration is constructed in two steps, namely $i'' = \mathrm{F}(r_1^*)(i_{12}^*)$ and $i_n^2 = \mathrm{F}(r_2)(i'')$. Since free constructions compose, $(i_n^2, r_2'' \circ r_1^{*\prime})$ is the right-hand side of the migration with the composed refactoring.

Since I_2^* is pullback, we obtain a homomorphism u such that $i_2^* \circ u = r_1^* \circ i_{12}^*$ and $(u, l_2^{*\prime})$ is pullback of (r_1', l_2'). Note that u is surjective since r_1' is and pullbacks in `Meta` preserve surjectivity.

Since $i'' = \mathrm{F}(r_1^*)(i_{12}^*)$, we get (e1) $i'' \circ r_1^{*\prime} = r_1^* \circ i_{12}^*$. If $(\mathrm{PB}(r_1^*)(i''), x)$ is the pullback of i'' along r_1^*, there is the universal morphism m for the free construction with (e2) $\mathrm{PB}(r_1^*)(i'') \circ m = i_{12}^*$ and (e3) $x \circ m = r_1^{*\prime}$. Since i_2^* is pullback of i_o^2, it satisfies the axioms (a4) and (a5), hence it is in $\mathrm{Inst}(M_2^*)$. Let $(\mathrm{PB}(r_1^*)(i_2^*), y)$ be its pullback along r_1^*, then there is the pullback morphism n with (e4) $y \circ n = u$ and (e5) $\mathrm{PB}(r_1^*)(i_2^*) \circ n = i_{12}^*$. Since i'' is free, there is f with (e6) $i_2^* \circ f = i''$ such that (e7) $\mathrm{PB}(r_1^*)(f) \circ m = n$. Since $\mathrm{PB}(r_1^*)(f)$ is the pullback image of f, we obtain additionally (e8) $f \circ x = y \circ \mathrm{PB}(r_1^*)(f)$. Combining all these results, we get $f \circ r_1^{*\prime} \overset{e3}{=} f \circ x \circ m \overset{e8}{=} y \circ \mathrm{PB}(r_1^*)(f) \circ m \overset{e7}{=} y \circ n \overset{e4}{=} u$. Therefore the whole diagram in figure 8 commutes.

In the rest of the proof, we show that $(i'', l_2' \circ f)$ is a pullback of (l_2, i_o^2). In this case, $(i'', l_2' \circ f)$ is the left-hand side of the second migration and (i_n^2, r_2'') its right-hand side. Hence the sequential application of the individual refactorings results in the same instance as the application of the composed refactoring.

Figure 9 shows the corresponding situation. (We introduced shorter names for some of the participating homomorphisms.) We know that the outer diagram is a pullback, i. e. $(g, i'' \circ h)$ is pullback of $(l_2, p \circ k)$, that $p \in \mathrm{Inst}(M_n^1)$, and that $i'' \in \mathrm{Inst}(M_2^*)$. We show the two sufficient conditions for pullbacks in `Meta`:

$$\begin{tikzcd}
M_n^1 & & M_2^* \arrow[ll, "l_2"'] \\
I_n^1 \arrow[u, "i^2_o := p"] & I_2^* \arrow[l, "l_2' := q"'] \arrow[ur, "i_2^*"] & I'' \arrow[l, "f"] \arrow[u, "i''"'] \\
I_1^* \arrow[u, "r_1' := k"] & & I_{12}^* \arrow[ll, "l_2^{*\prime} := g"'] \arrow[ul, "u"] \arrow[u, "r_1^{*\prime} := h"']
\end{tikzcd}$$

Figure 9. Pullback by Free Construction

(a) For each pair (x, y) with $l_2(x) = p(y)$, there is z with $i''(z) = x$ and $q \circ f(z) = y$. Let $l_2(x) = p(y)$ be given. Since k is surjective, there is w with $k(w) = y$. Since the outer diagram is a pullback, there is v with $g(v) = w$ and $i'' \circ h(v) = x$. Set $z = h(v)$. We get $i''(z) = i''(h(v)) = x$ and $q \circ f(z) = q \circ f \circ h(v)) = k \circ g(v) = k(w) = y$.

(b) If $i''(x) = i''(y)$ and $q \circ f(x) = q \circ f(y)$, then $x = y$, i. e. the homomorphisms i'' and $q \circ f$ are jointly monic. In the following, let x' and y' be fixed with $h(x') = x$ and $h(y') = y$. (x',y' exist since ${r_1^*}'$ is surjective.) We distinguish three cases:

(b1) i'' maps x and y to a vertex: Then $p \circ k$ maps $g(x')$ ans $g(y')$ to the vertex $l_2(i''(x) = i''(y))$. If $g(x') = g(y')$, the pullback property of the outer diagram in figure 9 provides $x' = y'$ and $x = y$. If $g(x') \neq g(y')$, we obtain $k(g(x')) = q(f(h(x'))) = q(f(x)) = q(f(y)) = q(f(h(y'))) = k(g(y'))$ and, by corollary 5, a path $z'_0, g(x') = z'_1, \cdots, z'_{n-1} = g(y'), z'_n$, such that $k(z'_i) = k(z'_j)$ for all $0 \leq i, j \leq n$. Since the outer diagram in figure 9 is a pullback, there is a path $z_0, x' = z_1, \cdots, z_{n-1} = y', z_n$ and for all $0 \leq i \leq n$ $g(z_i) = z'_i$ as well as $i''(h(z_i)) = i''(x) = i''(y)$. Since $i'' \in \mathrm{Inst}(M_2^*)$, $h(z_j) = h(z_i)$ for all $0 \leq i, j \leq n$. Thus, $x = h(x') = h(z_1) = h(z_{n-1}) = h(y') = y$.

(b2) i'' maps x and y to an edge, that is mapped to an edge by l_2: The interesting case is again $g(x') \neq g(y')$. Since k identifies $g(x')$ and $g(y')$, these two edges must violate axiom (a5). Hence $\mathtt{s}(g(x')) = \mathtt{s}(g(y'))$ and $\mathtt{t}(g(x')) = \mathtt{t}(g(y'))$. Since g is homomorphism, we conclude $g(\mathtt{s}(x')) = g(\mathtt{s}(y'))$ and $g(\mathtt{t}(x')) = g(\mathtt{t}(y'))$. Since i'' and h are homomoprhisms, $i'' \circ h(\mathtt{s}(x')) = i'' \circ h(\mathtt{s}(y'))$ and $i'' \circ h(\mathtt{t}(x')) = i'' \circ h(\mathtt{t}(y'))$. Thus, $\mathtt{s}(x') = \mathtt{s}(y')$ and $\mathtt{t}(x') = \mathtt{t}(y')$ as well as $\mathtt{s}(x) = \mathtt{s}(y)$ and $\mathtt{t}(x) = \mathtt{t}(y)$. Since $i'' \in \mathrm{Inst}(M_2^*)$, it must satisfy axiom (a5), which means $x = y$.

(b3) i'' maps x and y to an edge, that is mapped to a vertex by l_2: In this case $q \circ f(x) = q \circ f(y)$ is a vertex, because $p \in \mathrm{Inst}(M_n^1)$. Thus, $g(x')$ and $g(y')$ are mapped to the same vertex by k, which, in case of $g(x') \neq g(y')$, must be an identification due to axiom (a4). It follows that $\mathtt{s}(g(x'))$, $\mathtt{t}(g(x'))$, $\mathtt{s}(g(y'))$, and $\mathtt{t}(g(y'))$ are also mapped to the same vertex by k. Repeating the argument of the first case, it follows that $\mathtt{s}(x')$ and $\mathtt{s}(y')$ are identified by h as well as $\mathtt{t}(x')$ and $\mathtt{t}(y')$. Since $i'' \in \mathrm{Inst}(M_2^*)$, it satisfies (a5), which provides $x = h(x') = h(y') = y$. □

5. Conclusions

The framework introduced here is at least as powerful as the former frameworks we presented for example in [13, 15]: It can handle more refactorings in practical applications and has the same potential for theoretical results. But is is simpler: (1) the theoretical background is just Horn-formulae over equality and (2) the meta-model (definition 1) does not introduce any special structure in order to handle inheritance. Therefore, it is more general as well: many refactorings are not restricted to inheritance and can be applied to arbitrary associations. The introduction of a proxy pattern in example 13 demonstrates this fact.

There are two aspects for future research:

(1) Additional modeling features of the object-oriented analysis and design language UML2[14], like multiplicity constraints, shall be integrated into the meta-model. It shall be investigated if theorem 15 still holds in the extended meta-model.

(2) An algebraic graph transformation approach shall be designed that produces model refactorings as rule applications. All three existing approaches, namely double-pushout rewriting[6], single-pushout grammars[11], and the sesqui-pushout approach[5] are not sufficient since they cannot handle arbitrary foldings and unfoldings.

References

[1] Ambler, Scott W.: Agile Database Techniques. John Wiley & Sons, 2003.

[2] Ambler, Scott W.: Refactoring Databases: Evolutionary Database Design. Addison-Wesley, 2006.

[3] Ambler, Scott W.: Test-Driven Development of Relational Databases. IEEE Software, 24:37–43, 2007.

[4] D'Anjou, Jim et al. The Java Developer's Guide to Eclipse (2nd Edition). Addison-Wesley, 2005.

[5] Corradini, Andrea, Tobias Heindel, Frank Hermann, and Barbara König: Sesqui-pushout rewriting. In Proceedings ICGT2006, pages 30–45, Springer LNCS, 2006.

[6] Ehrig, H., K. Ehrig, U. Prange und G. Taentzer: Fundamentals of Algebraic Graph Transformation. Springer, 2006.

[7] Fowler, Martin: Refactoring - Improving the Design of Existing Code. Addison-Wesley, 1999.

[8] Fowler, Martin: Patterns of Enterprise Application Architecture. Addison-Wesley, 2002.

[9] Gamma, Erich, Richard Helm, Ralph Johnson und John Vlissides: Design Patterns. Addison-Wesley Professional, 1995.

[10] Herrmannsdoerfer, Markus, S. D. Vermolen, G. Wachsmuth: An Extensive Catalog of Operators for the Coupled Evolution of Metamodels and Models. In Proceeedings International Conference on Software Language Engineering (SLE 2010), LNCS 6563, 2010.

[11] Löwe, Michael: Algebraic approach to single-pushout graph transformation. Theor. Comput. Sci., 109(1&2):181–224, 1993.

[12] Löwe, Michael, Harald König, Christoph Schulz and Michael Peters: Refactoring Information Systems – A Formal Framework. In: Proceedings WMSCI 2006, Volume 1, pages 75–80, 2006.

[13] Löwe, Michael, Harald König, Christoph Schulz and Michael Peters: Refactoring Information Systems – Handling partial composition. In: Electronic Communications of the EASST, Volume 3, 2006.

[14] Rumbuagh, James, Ivar Jacobson, Grady Booch: The Unified Modeling Language Reference Manual (2nd Edition). Addison-Wesley, 2004.

[15] Schulz, Christoph: Transformation objektorientierter Systeme basierend auf algebraischen Graph-Transformationen, Dissertation, Technische Universität Berlin, 2010.

[16] Schulz, Christoph, M. Löwe, H. König: Categorical framework for the transformation of object-oriented systems: Operations and methods, Electronic Communications of the EASST, Vol. 26, 2010.

[17] Schulz, Christoph, M. Löwe, H. König: Categorical framework for the transformation of object-oriented systems: Models and data, Journal of Symbolic Computation, Vol. 46, Nr. 3, S. 316 – 337, 2011.

[18] Taentzer, Gabriele et al.: Model Transformation by Graph Transformation: A Comparative Study. In Proceedings Workshop Model Transformation in Practice, Montego Bay, Jamaica, October 2005.