Patterns for Specifying Bidirectional Transformations in UML-RSDS

K. Lano,
S. Yassipour-Tehrani
Dept. of Informatics
King's College London
London, UK
Email: kevin.lano@kcl.ac.uk,
s.yassipour-tehrani@kcl.ac.uk

S. Kolahdouz-Rahimi
Dept of Software Engineering
University of Isfahan
Isfahan, Iran
Email: sh.rahimi@eng.ui.ac.ir

Abstract—**In this paper, we identify model transformation specification and design patterns, which support the property of transformation bidirectionality: the ability of a single specification to be applied either as a source-to-target transformation or as a target-to-source transformation. In contrast to previous work on bidirectional transformations (bx), we identify the important role of transformation *invariants* in the derivation of reverse transformations, and show how patterns and invariants can be used to give a practical means of defining bx in the UML-RSDS transformation language.**

Keywords — Bidirectional transformations; transformation design patterns; UML-RSDS

I. Introduction

Bidirectional transformations (bx) are considered important in a number of transformation scenarios:

- Maintaining consistency between two models which may both change, for example, if a UML class diagram and corresponding synthesised Java code both need to be maintained consistently with each other, in order to implement *round-trip engineering* for model-driven development.
- Where a mapping between two languages may need to be operated in either direction for different purposes, for example, to represent behavioural models as either Petri Nets or as state machines [12].
- Where inter-conversion between two different representations is needed, such as two alternative formats of electronic health record [3].

Design patterns have become an important tool in software engineering, providing a catalogue of 'best practice' solutions to design problems in software [7]. Patterns for model transformations have also been identified [14], but patterns specifically for bx have not been defined.

In this paper, we show how bx patterns can be used to obtain a practical approach for bx using the UML-RSDS language [11].

Section II defines the concept of a bx. Section V describes related work. Section III describes UML-RSDS and transformation specification in UML-RSDS. Section IV gives a catalogue of bx patterns for UML-RSDS, with examples. Section VI gives a conclusion.

II. Criteria for Bidirectionality

Bidirectional transformations are characterised by a binary relation $R : SL \leftrightarrow TL$ between a source language (metamodel) SL and a target language TL. $R(m, n)$ holds for a pair of models m of SL and n of TL when the models consist of data which corresponds under R. It should be possible to automatically derive from the definition of R both forward and reverse transformations

$$R^{\rightarrow} : SL \times TL \rightarrow TL \qquad R^{\leftarrow} : SL \times TL \rightarrow SL$$

which aim to establish R between their first (respectively second) and their result target (respectively source) models, given both existing source and target models.

Stevens [16] has identified two key conditions which bidirectional model transformations should satisfy:

1) Correctness: the forward and reverse transformations derived from a relation R do establish R: $R(m, R^{\rightarrow}(m, n))$ and $R(R^{\leftarrow}(m, n), n)$ for each $m : SL$, $n : TL$.
2) Hippocraticness: if source and target models already satisfy R then the forward and reverse transformations do not modify the models:

$$\begin{array}{lcl} R(m,n) & \Rightarrow & R^{\rightarrow}(m,n) = n \\ R(m,n) & \Rightarrow & R^{\leftarrow}(m,n) = m \end{array}$$

for each $m : SL$, $n : TL$.

The concept of a *lens* is a special case of a bx satisfying these properties [16].

III. BX Specification in UML-RSDS

UML-RSDS is a hybrid model transformation language, with a formal semantics [10] and an established toolset [11]. Model transformations are specified in UML-RSDS as UML use cases, defined declaratively by three main predicates, expressed in a subset of OCL:

1) Assumptions *Asm*, which define when the transformation is applicable.
2) Postconditions *Post*, which define the intended effect of the transformation at its termination. These are an ordered conjunction of OCL constraints (also termed *rules* in the following) and also serve to define a procedural implementation of the transformation.
3) Invariants *Inv*, which define expected invariant properties which should hold during the transformation execution. These may be derived from *Post*, or specified explicitly by the developer.

From a declarative viewpoint, *Post* defines the conditions which should be established by a transformation. From an implementation perspective, the constraints of *Post* also define intended computation steps of the transformation: each computation step is an application of a postcondition constraint to a specific source model element or to a tuple of elements.

For example, an elementary transformation specification τ_{a2b} on the languages S consisting of entity type A and T consisting of entity type B (Figure 1) could be:

$$\begin{array}{l}(Asm):\\ \quad B{\rightarrow}forAll(b \mid b.y \geq 0)\\ (Post):\\ \quad A{\rightarrow}forAll(a \mid B{\rightarrow}exists(b \mid b.y = a.x{\rightarrow}sqr()))\\ (Inv):\\ \quad B{\rightarrow}forAll(b \mid A{\rightarrow}exists(a \mid a.x = b.y{\rightarrow}sqrt()))\end{array}$$

The computation steps α of τ_{a2b} are applications of $B{\rightarrow}exists(b \mid b.y = a.x{\rightarrow}sqr())$ to individual $a : A$. These consist of creation of a new $b : B$ instance and setting its y value to $a.x * a.x$. These steps preserve *Inv*: $Inv \Rightarrow [\alpha]Inv$.

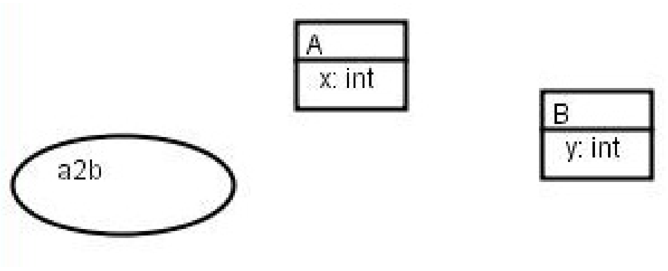

Figure 1. A to B Transformation τ_{a2b}

This example shows a typical situation, where the invariant is a dual to the postcondition, and expresses a form of minimality condition on the target model: that the only elements of this model should be those derived from source elements by the transformation. In terms of the framework of [16], the source-target relation R_τ associated with a UML-RSDS transformation τ is *Post and Inv*. As in the above example, R_τ is not necessarily bijective. The forward direction of τ is normally computed as $stat(Post)$: the UML activity derived from *Post* when interpreted procedurally [10]. However, in order to achieve the correctness and hippocraticness properties, *Inv* must also be considered: before $stat(Post)$ is applied to the source model *m*, the target model *n* must be cleared of elements which fail to satisfy *Inv*.

In the *a*2*b* example, the transformation $\tau^{\times}_{a2b}$ with postcondition constraints:

$$\begin{array}{l}(CleanTarget1):\\ \quad B{\rightarrow}forAll(b \mid not(b.y \geq 0)\ implies\\ \qquad\qquad b{\rightarrow}isDeleted())\\ (CleanTarget2):\\ \quad B{\rightarrow}forAll(b \mid not(A{\rightarrow}exists(a \mid a.x = b.y{\rightarrow}sqrt()))\\ \qquad\qquad implies\ b{\rightarrow}isDeleted())\end{array}$$

is applied before τ_{a2b}, to remove all B elements which fail to be in R_{a2b} with some $a : A$, or which fail to satisfy *Asm*.

This is an example of the Cleanup before Construct pattern (Section IV). Additionally, the $E{\rightarrow}exists(e \mid P)$ quantifier in rule succedents should be procedurally interpreted as “create a new $e : E$ and establish P for e, unless there already exists an $e : E$ satisfying P”. That is, the Unique Instantiation pattern [14] should be used to implement ‘check before enforce’ semantics. The forward transformation $\tau^{\rightarrow}$ is then the sequential composition $\tau^{\times};\ \tau$ of the cleanup transformation and the standard transformation (enhanced by Unique Instantiation).

In the reverse direction, the roles of *Post* and *Inv* are interchanged: elements of the source model which fail to satisfy *Asm*, or to satisfy *Post* with respect to some element of the target model should be deleted:

$$\begin{array}{l}(CleanSource2):\\ \quad A{\rightarrow}forAll(a \mid not(B{\rightarrow}exists(b \mid b.y = a.x{\rightarrow}sqr()))\\ \qquad\qquad implies\ \ a{\rightarrow}isDeleted())\end{array}$$

This cleanup transformation is denoted $\tau^{\sim\times}_{a2b}$. It is followed by an application of the normal inverse transformation $\tau^{\sim}_{a2b}$ which has postcondition constraints *Inv* ordered in the corresponding order to *Post*. Again, Unique Instantiation is used for source model element creation. The overall reverse transformation is denoted by $\tau^{\leftarrow}$ and is defined as $\tau^{\sim\times};\ \tau^{\sim}$.

In the case of separate-models transformations with type 1 postconditions (Constraints whose write frame is disjoint from their read frame), *Inv* can be derived automatically from *Post* by syntactic transformation, the *CleanTarget* and *CleanSource* constraints can also be derived from *Post*, and from *Asm*. This is an example of a higher-order transformation (HOT) and is implemented in the UML-RSDS tools.

In general, in the following UML-RSDS examples, τ is a separate-models transformation with source language S and target language T, and postcondition *Post* as an ordered conjunction of constraints of the form:

$$\begin{array}{l}(Cn):\\ S_i{\rightarrow}forAll(s \mid SCond(s)\ implies\\ \qquad\qquad T_j{\rightarrow}exists(t \mid TCond(t)\ and\ P_{i,j}(s,t)))\end{array}$$

and *Inv* is a conjunction of dual constraints of the form

$$\begin{array}{l}(Cn^{\sim}):\\ T_j{\rightarrow}forAll(t \mid TCond(t)\ implies\\ \qquad\qquad S_i{\rightarrow}exists(s \mid SCond(s)\ and\ P^{\sim}_{i,j}(s,t)))\end{array}$$

where the predicates $P_{i,j}(s,t)$ define the features of t from those of s, and are invertible: an equivalent form $P^{\sim}_{i,j}(s,t)$ should exist, which expresses the features of s in terms of those of t, and such that

$$S_i{\rightarrow}forAll(s \mid T_i{\rightarrow}forAll(t \mid P_{i,j}(s,t) = P^{\sim}_{i,j}(s,t)))$$

under the assumptions *Asm*. Table I shows some examples of inverses $P^{\sim}$ of predicates P. The computation of these inverses are all implemented in the UML-RSDS tools (the *reverse* option for use cases). More cases are given in [11]. The transformation developer can also specify inverses for particular *Cn* by defining a suitable $Cn^{\sim}$ constraint in *Inv*, for example, to express that a predicate $t.z = s.x + s.y$ should be inverted as $s.x = t.z - s.y$.

Each *CleanTarget* constraint based on *Post* then has the form:

$$\begin{array}{l}(Cn^{\times}):\\ T_j{\rightarrow}forAll(t \mid TCond(t)\ and\\ \qquad not(S_i{\rightarrow}exists(s \mid SCond(s)\ and\ P_{i,j}(s,t)))\ implies\\ \qquad\qquad t{\rightarrow}isDeleted())\end{array}$$

Similarly for *CleanSource*.

TABLE I. EXAMPLES OF PREDICATE INVERSES

P(s,t)	$P^{\sim}(s,t)$	Condition
$t.g = s.f$	$s.f = t.g$	Assignable features f, g
$t.g = s.f \rightarrow sqrt()$	$s.f = t.g \rightarrow sqr()$	f, g non-negative attributes
$t.g = K * s.f + L$ Numeric constants K, L, $K \neq 0$	$s.f = (t.g - L)/K$	f, g numeric attributes
$t.rr = s.r \rightarrow including(s.p)$	$s.r = t.rr \rightarrow front()$ *and*	rr, r ordered association ends
$t.rr = s.r \rightarrow append(s.p)$	$s.p = t.rr \rightarrow last()$	p 1-multiplicity end
$t.rr = s.r \rightarrow sort()$	$s.r = t.rr \rightarrow asSet()$	r set-valued, rr ordered
$t.rr = s.r \rightarrow asSequence()$		
$R(s,t)$ *and* $Q(s,t)$	$R^{\sim}(s,t)$ *and* $Q^{\sim}(s,t)$	
$t.rr = TRef[s.r.idS]$ idS primary key of $SRef$, idT primary key of $TRef$	$s.r = SRef[t.rr.idT]$	rr association end with element type $TRef$, r association end with element type $SRef$
$t.g = s.r.idS$ Attribute g	$s.r = SRef[t.g]$	idS primary key of $SRef$, r association end with element type $SRef$
$T_j[s.idS].rr = TRef[s.r.idSRef]$ r has element type $SRef$, rr has element type $TRef$	$S_i[t.idT].r = SRef[t.rr.idTRef]$	$idS, idSRef$ primary keys of S_i, $SRef$ $idT, idTRef$ primary keys of T_j, $TRef$

IV. Patterns for Bx

In this section, we give a patterns catalogue for bx, and give pattern examples in UML-RSDS.

A. *Auxiliary Correspondence Model*

This pattern defines auxiliary entity types and associations which link corresponding source and target elements. These are used to record the mappings performed by a bx, and to propagate modifications from source to related target elements or vice-versa, when one model changes.

Figure 2 shows a typical schematic structure of the pattern.

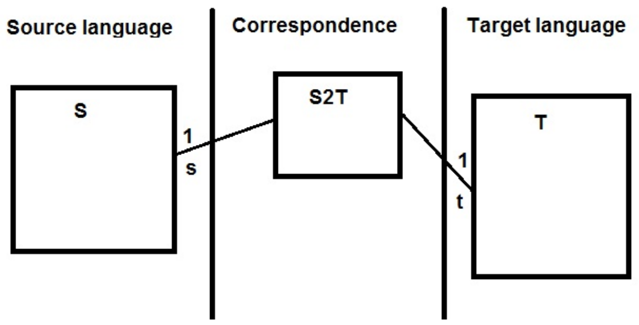

Figure 2. Auxiliary Correspondence Model pattern

Benefits: The pattern is a significant aid in change-propagation between models, and helps to ensure the correctness of a bx. Feature value changes to a source element s can be propagated to changes to its corresponding target element, and vice-versa, via the links. Deletion of an element may imply deletion of its corresponding element.

Disadvantages: The correspondence metamodel must be maintained (by the transformation engineer) together with the source and target languages, and the necessary actions in creating and accessing correspondence elements adds complexity to the transformation and adds to its execution time and memory requirements.

Related Patterns: This pattern is a specialisation of the *Auxiliary Metamodel* pattern of [14].

Examples: This mechanism is a key facility of Triple Graph Grammars (TGG) [1][2], and correspondence traces are maintained explicitly or implicitly by other MT languages such as QVT-R [15].

In UML-RSDS, the pattern is applied by introducing auxiliary attributes into source and target language entity types. These attributes are primary key/identity attributes for the entity types, and are used to record source-target element correspondences. Target element $t : T_j$ is considered to correspond to source element(s) $s_1 : S_1$, ..., $s_n : S_n$ if they all have the same primary key values: $t.idT_j = s_1.idS_1$, etc. The identity attributes are String-valued in this paper. The correspondence between a source entity S_i and a target entity T_j induced by equality of identity attribute values defines a *language mapping* or *interpretation* χ of S_i by T_j in the sense of [13]:

$$S_i \longmapsto T_j$$

with $S_i \rightarrow collect(idS_i) = T_j \rightarrow collect(idT_j)$.

The existence of identity attributes facilitates element lookup by using the *Object Indexing* pattern [14], which defines maps from *String* to each entity type, permitting elements to be retrieved by the value of their identity attribute: $T_j[v]$ denotes the T_j instance t with $t.idT_j = v$ if v is a single String value, or the collection of T_j instances t with $v \rightarrow includes(t.idT_j)$ if v is a collection. The last three cases in Table I show inverse predicates derived using this approach to correspondence models. Note that $T_j[x.idT_j] = x$ for $x : T_j$.

The pattern can be used to define source-target propagation and incremental application of a transformation τ. For postconditions *Cn* of the form

$$\begin{aligned}&S_i{\rightarrow}forAll(s \mid SCond(s)\ implies\\ &\qquad T_j{\rightarrow}exists(t \mid TCond(t)\ and\ P_{i,j}(s,t)))\end{aligned}$$

derived constraints Cn^{Δ} can be defined for the incremental application of *Cn* to model increments (finite collections of creations, deletions and modifications of model elements).

The incremental version τ^{Δ} of a transformation τ is defined to have postconditions formed from the constraints Cn^{Δ} for each postcondition *Cn* of τ, and ordered according to the order of the *Cn* in the *Post* of τ. In a similar way, target-source change propagation can be defined. Change propagation is implemented in UML-RSDS by the *incremental* mode of use case execution.

B. *Cleanup before Construct*

This pattern defines a two-phase approach in both forward and reverse transformations associated with a bx with relation *R*: the forward transformation $R^{\rightarrow}$ first removes all elements from the target model *n* which fail to satisfy *R* for any element of the source *m*, and then constructs elements of *n* to satisfy *R* with respect to *m*. The reverse transformation $R^{\leftarrow}$ operates on *m* in the same manner.

Benefits: The pattern is an effective way to ensure the correctness of separate-models bx.

Disadvantages: There may be efficiency problems because for each target model element, a search through the source model for possibly corresponding source element may be needed. Elements may be deleted in the Cleanup phase only to be reconstructed in the Construct phase. Auxiliary Correspondence Model may be an alternative strategy to avoid this problem, by enforcing that feature values should change in response to a feature value change in a corresponding element, rather than deletion of elements.

Related Patterns: This pattern is a variant of the *Construction and Cleanup* pattern of [14].

Examples: An example is the Composers bx [4]. Implicit deletion in QVT operates in a similar manner to this pattern, but can only modify models (domains) marked as *enforced* [15]. In UML-RSDS, explicit cleanup rules $Cn^{\times}$ can be deduced from the construction rules *Cn*, for mapping transformations, as described in Section III above. If identity attributes are used to define the source-target correspondence, then $Cn^{\times}$ can be simplified to:

$$\begin{aligned}&T_j{\rightarrow}forAll(t \mid TCond(t)\ and\\ &\qquad S_i{\rightarrow}collect(sId){\rightarrow}excludes(t.tId)\ implies\\ &\qquad\qquad t{\rightarrow}isDeleted())\end{aligned}$$

and

$$\begin{aligned}&T_j{\rightarrow}forAll(t \mid TCond(t)\ and\\ &\qquad S_i{\rightarrow}collect(sId){\rightarrow}includes(t.tId)\ and\ s = S_i[t.tId]\\ &\qquad and\ not(SCond(s))\ implies\ t{\rightarrow}isDeleted())\end{aligned}$$

In the case that $TCond(t)$ and $SCond(s)$ hold for corresponding *s*, *t*, but $P_{i,j}(s,t)$ does not hold, *t* should not be deleted, but $P_{i,j}(s,t)$ should be established by updating *t*:

$$\begin{aligned}&S_i{\rightarrow}forAll(s \mid T_j{\rightarrow}collect(tId){\rightarrow}includes(s.sId)\ and\\ &\qquad t = T_j[sId]\ and\ SCond(s)\ and\\ &\qquad TCond(t)\ implies\ P_{i,j}(s,t))\end{aligned}$$

For a transformation τ, the cleanup transformation $\tau^{\times}$ has the above $Cn^{\times}$ constraints as its postconditions, in the same order as the *Cn* occur in the *Post* of τ. Note that $\tau^{\rightarrow}$ is $\tau^{\times};\ \tau$, and τ^{Δ} is $\tau^{\times};\ \tau$ incrementally applied.

C. *Unique Instantiation*

This pattern avoids the creation of unnecessary elements of models and helps to resolve possible choices in reverse mappings. It uses various techniques such as traces and unique keys to identify when elements should be modified and reused instead of being created. In particular, unique keys can be used to simplify checking for existing elements.

Benefits: The pattern helps to ensure the Hippocraticness property of a bx by avoiding changes to a target model if it is already in the transformation relation with the source model. It implements the principle of 'least change' [17].

Disadvantages: The need to test for existence of elements adds to the execution cost. This can be ameliorated by the use of the Object Indexing pattern [14] to provide fast lookup of elements by their primary key value.

Examples: The *key* attributes and check-before-enforce semantics of QVT-R follow this pattern, whereby new elements of source or target models are not created if there are already elements, which satisfy the specified relations of the transformation [16]. The $E{\rightarrow}exists1(e \mid P)$ quantifier in UML-RSDS is used in a similar way. It is procedurally interpreted as "create a new $e : E$ and establish *P* for *e*, unless there already exists an $e : E$ satisfying *P*" [11]. For bx, the quantifier *exists* should also be treated in this way. If a transformation uses identity attributes (to implement Auxiliary Correspondence Model), the quantifier $E{\rightarrow}exists(e \mid e.eId = v\ and\ P)$ can be interpreted as: "if $E[v]$ exists, apply $stat(P)$ to this element, otherwise create a new *E* instance with $eId = v$ and apply $stat(P)$ to it". This ensures Hippocraticness.

D. *Phased Construction for bx*

This pattern defines a bx τ by organising R_{τ} as a union of relations $R_{Si,Tj}$ which relate elements of entities *Si* and *Tj* which are in corresponding levels of the composition hierarchies of the source and target languages. Figure 3 shows the typical schematic structure of the pattern. At each composition level there is a 0..1 to 0..1 relation (or more specialised relation) between the corresponding source and target entity types.

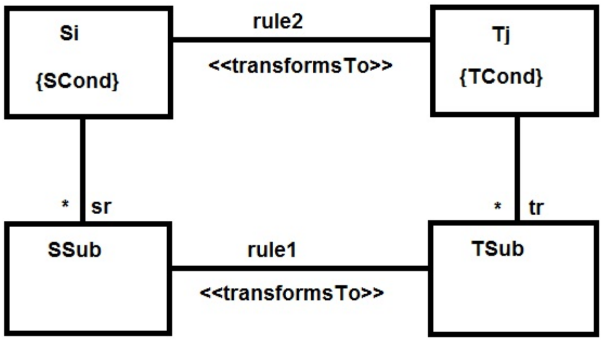

Figure 3. Phased Construction pattern

Benefits: The pattern provides a modular and extensible means to structure a bx.

Examples: The UML to relational database example of [15] is a typical case, where *Package* and *Schema* correspond at the top of the source/target language hierarchies, as do *Class* and *Table* (in the absence of inheritance), and *Column* and *Attribute* at the lowest level.

In UML-RSDS a transformation defined according to this pattern has its *Post* consisting of constraints *Cn* of the form

$$\begin{array}{l} S_i{\rightarrow}forAll(s \mid SCond(s)\ implies \\ \qquad T_j{\rightarrow}exists(t \mid TCond(t)\ and\ P_{i,j}(s,t))) \end{array}$$

where S_i and T_j are at corresponding hierarchy levels, and *Inv* consists of constraints $Cn^{\sim}$ of the form

$$\begin{array}{l} T_j{\rightarrow}forAll(t \mid TCond(t)\ implies \\ \qquad S_i{\rightarrow}exists(s \mid SCond(s)\ and\ P^{\sim}_{i,j}(s,t))) \end{array}$$

No nested quantifiers or deletion expressions $x{\rightarrow}isDeleted()$ are permitted in *SCond*, *TCond* or $P_{i,j}$, and $P_{i,j}$ is restricted to be formed of invertible expressions.

Each rule creates elements t of some target entity type T_j, and may lookup target elements produced by preceding rules to define the values of association end features of t: $t.tr = TSub[s.sr.idSSub]$ for example, where *TSub* is lower than T_j in the target language composition hierarchy (as in Figure 3) and there are identity attributes in the entities to implement a source-target correspondence at each level. Both forward and reverse transformations will conform to the pattern if one direction does. The assignment to $t.tr$ has inverse: $s.sr = SSub[t.tr.idTSub]$.

Two UML-RSDS bx $\tau : S \rightarrow T$, $\sigma : T \rightarrow U$ using this pattern can be sequentially composed to form another bx between S and U: the language T becomes auxiliary in this new transformation. The forward direction of the composed transformation is $\tau^{\rightarrow}$; $\sigma^{\rightarrow}$, the reverse direction is $\sigma^{\leftarrow}$; $\tau^{\leftarrow}$.

E. Entity Merging/Splitting for bx

In this variation of Phased Construction, data from multiple source model elements may be combined into single target model elements, or vice-versa, so that there is a many-one relation from one model to the other. The pattern supports the definition of such bx by including correspondence links between the multiple elements in one model which are related to one element in the other.

Benefits: The additional links enable the transformation to be correctly reversed.

Disadvantages: Additional auxiliary data needs to be added to record the links. The validity of the links between elements needs to be maintained. There may be potential conflict between different rules which update the same element.

Related Patterns: This uses a variant of Auxiliary Correspondence Model, in which the correspondence is between elements in one model, in addition to cross-model correspondences. The attributes used to record intra-model correspondences may not be primary keys.

Examples: An example of Entity Merging is the Collapse/Expand State Diagrams benchmark of [6]. The UML to RDB transformation is also an example in the case that all subclasses of a given root class are mapped to a single table that represents this class. The Pivot/Unpivot transformation of [3] is an example of Entity Splitting.

In the general case of merging/splitting, the inverse of C_n:

$$\begin{array}{l} S_{i1}{\rightarrow}forAll(s1 \mid ... \\ \quad S_{in}{\rightarrow}forAll(sn \mid SCond(s1,...,sn)\ implies \\ \qquad T_{j1}{\rightarrow}exists(t1 \mid ... \\ \qquad\quad T_{jm}{\rightarrow}exists(tm \mid TCond(t1,...,tm)\ and \\ \qquad\qquad\qquad P(s1,...,sn,t1,...,tm))...))\ ...) \end{array}$$

is $Cn^{\sim}$:

$$\begin{array}{l} T_{j1}{\rightarrow}forAll(t1 \mid ... \\ \quad T_{jm}{\rightarrow}forAll(tm \mid TCond(t1,...,tm)\ implies \\ \qquad S_{i1}{\rightarrow}exists(s1 \mid ... \\ \qquad\quad S_{in}{\rightarrow}exists(sn \mid SCond(s1,...,sn)\ and \\ \qquad\qquad\qquad P^{\sim}(s1,...,sn,t1,...,tm))...))...) \end{array}$$

In UML-RSDS, correspondence links between elements in the same model are maintained using additional attributes. All elements corresponding to a single element will have the same value for the auxiliary attribute (or a value derived by a 1-1 function from that value).

F. Map Objects Before Links for bx

If there are self-associations on source entity types, or other circular dependency structures in the source model, then this variation on Phased Construction for bx can be used. This pattern separates the relation between elements in target and source models from the relation between links in the models.

Benefits: The specification is made more modular and extensible. For example, if a new association is added to one language, and a corresponding association to the other language, then a new relation relating the values of these features can be added to the transformation without affecting the existing relations.

Disadvantages: Some features of one entity type are treated in separate relations.

Examples: In UML-RSDS a first phase of such a transformation relates source elements to target elements, then in a second phase source links are related to corresponding target links. The second phase typically has postcondition constraints of the form $S_i{\rightarrow}forAll(s \mid T_j[s.idS].rr = TRef[s.r.idSRef])$ to define target model association ends rr from source model association ends r, looking-up target model elements $T_j[s.idS]$ and $TRef[s.r.idSRef]$ which have already been created in a first phase. Such constraints can be inverted to define source data from target data as: $T_j{\rightarrow}forAll(t \mid S_i[t.idT].r = SRef[t.rr.idTRef])$. The reverse transformation also conforms to the Map Objects Before Links pattern.

An example of this pattern is the tree to graph transformation [9], Figure 4.

A first rule creates a node for each tree:

$$Tree{\rightarrow}forAll(t \mid Node{\rightarrow}exists(n \mid n.label = t.label))$$

A second rule then creates edges for each link between parent and child trees:

$$\begin{array}{l} Tree{\rightarrow}forAll(t \mid \\ \quad Tree{\rightarrow}forAll(p \mid t.parent{\rightarrow}includes(p)\ implies \\ \qquad Edge{\rightarrow}exists(e \mid e.source = Node[t.label]\ and \\ \qquad\qquad\qquad e.target = Node[p.label]))) \end{array}$$

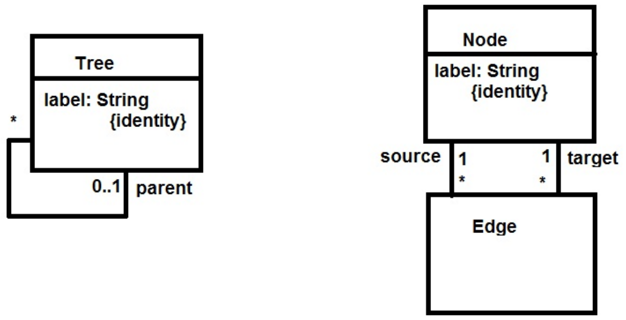

Figure 4. Tree to graph metamodels

The corresponding invariant predicates, defining the reverse transformation, are:

$$Node \rightarrow forAll(n \mid Tree \rightarrow exists(t \mid t.label = n.label))$$

and

$$\begin{array}{l} Edge \rightarrow forAll(e \mid \\ \quad Tree \rightarrow exists(t \mid Tree \rightarrow exists(p \mid \\ \qquad t.parent \rightarrow includes(p) \text{ and} \\ \qquad t.label = e.source.label \text{ and} \\ \qquad p.label = e.target.label))) \end{array}$$

Inv is derived mechanically from *Post* using Table I, and provides an implementable reverse transformation, since $stat(Inv)$ is defined.

V. Related Work

There are a wide range of approaches to bx [8]. Currently the most advanced approaches [2][5] use constraint-based programming techniques to interpret relations $P(s, t)$ between source and target elements as specifications in both forward and reverse directions. These techniques would be a potentially useful extension to the syntactic inverses defined in Table I, however the efficiency of constraint programming will generally be lower than the statically-computed inverses. The approach also requires the use of additional operators extending standard OCL. Further techniques include the inversion of recursively-defined functions [18], which would also be useful to incorporate into the UML-RSDS approach.

In [13] we identify the role of *language interpretations* $\chi : S \rightarrow T$ in specifying transformations. Interpretations are closely related to transformation inversion: at the model level a mapping $Mod(\chi) : Mod(T) \rightarrow Mod(S)$ from the set of models of T to those of S can be defined based on χ: the interpretation of an S language element E in $Mod(\chi)(n)$ for $n : Mod(T)$ is that of $\chi(E)$ in n. Syntactically, the inverse of a transformation τ specified by a language morphism χ can be derived from χ: the value of a feature f of source element s of source entity E is set by $s.f = t.\chi(E :: f)$ in the case of an attribute, and by $s.r = SRef[t.\chi(E :: r).idTRef]$ in the case of a role with element type $SRef$.

Considerable research has been carried out on the theory of bx. One principle which has been formulated for bx is the *principle of least change* [17]. This means that a bx which needs to modify one model in order to re-establish the bx relation R with a changed other model, should make a minimal possible such change to the model. In our approach, *Post* in the forward direction, and *Inv* in the reverse direction, express the necessary minimal conditions for the models to be related by R. The synthesised implementation of these constraints as executable code carries out the minimal changes necessary to establish *Post* and *Inv*, and hence satisfies the principle of least change.

VI. Conclusion

We have defined a declarative approach for bidirectional transformations based on the derivation of forward and reverse transformations from a specification of dual postcondition and invariant relations between source and target models. The approach enables a wide range of bx to be defined, including cases of many-to-one and one-to-many relations between models, in addition to bijections. We have described transformation patterns which may be used to structure bx. The derivation of reverse transformations has been implemented in the UML-RSDS tools [11].

References

[1] A. Anjorin and A. Rensink, "SDF to Sense transformation", TU Darmstadt, Germany, 2014.

[2] A. Anjorin, G. Varro, and A. Schurr, "Complex attribute manipulation in TGGs with constraint-based programming techniques", BX 2012, Electronic Communications of the EASST vol. 49, 2012.

[3] M. Beine, N. Hames, J. Weber, and A. Cleve, "Bidirectional transformations in database evolution: a case study 'at scale'", EDBT/ICDT 2014, CEUR-WS.org, 2014.

[4] J. Cheney, J. McKinna, P. Stevens, and J. Gibbons, "Towards a repository of bx examples", EDBT/ICDT 2014, 2014, pp. 87–91.

[5] A. Cicchetti, D. Di Ruscio, R. Eramo, and A. Pierantonio, "JTL: a bidirectional and change propagating transformation language", SLE 2010, LNCS vol. 6563, 2011, pp. 183–202.

[6] K. Czarnecki, J. Nathan Foster, Z. Hu, R. Lammel, A. Schurr, and J. Terwilliger, "Bidirectional transformations: a cross-discipline perspective", GRACE workshop, ICMT, 2009.

[7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley, 1994.

[8] Z. Hu, A. Schurr, P. Stevens, and J. Terwilliger (eds.), "Report from Dagstuhl Seminar 11031", January 2011, www.dagstuhl.de/11031.

[9] D. S. Kolovos, R. F. Paige, and F. Polack, "The Epsilon Transformation Language", ICMT, 2008, pp. 46–60.

[10] K. Lano and S. Kolahdouz-Rahimi, "Constraint-based specification of model transformations", Journal of Systems and Software, vol. 88, no. 2, February 2013, pp. 412–436.

[11] K. Lano, The UML-RSDS Manual, www.dcs.kcl.ac.uk/staff/kcl/uml2web/umlrsds.pdf, 2015.

[12] K. Lano, S. Kolahdouz-Rahimi, and K. Maroukian, "Solving the Petri-Nets to Statecharts Transformation Case with UML-RSDS", TTC 2013, EPTCS, 2013, pp. 101–105.

[13] K. Lano and S. Kolahdouz-Rahimi, "Towards more abstract specification of model transformations", ICTT 2014.

[14] K. Lano and S. Kolahdouz-Rahimi, "Model-transformation Design Patterns", IEEE Transactions in Software Engineering, vol 40, 2014, pp. 1224–1259.

[15] OMG, MOF 2.0 Query/View/Transformation Specification v1.1, 2011.

[16] P. Stevens, "Bidirectional model transformations in QVT: semantic issues and open questions", SoSyM, vol. 9, no. 1, January 2010, pp. 7–20.

[17] Theory of Least Change, groups.inf.ed.ac.uk/bx/, accessed 3.9.2015.

[18] J. Voigtlander, Z. Hu, K. Matsuda, and M. Wang, "Combining syntactic and semantic bidirectionalization", ICFP '10, ACM Press, 2010, pp. 181–192.