Feature-Based Classification of Bidirectional Transformation Approaches

Soichiro Hidaka, Massimo Tisi, Jordi Cabot, Zhenjiang Hu

Software and Systems Modeling manuscript No.
(will be inserted by the editor)

Feature-Based Classification of Bidirectional Transformation Approaches

Soichiro Hidaka · Massimo Tisi · Jordi Cabot · Zhenjiang Hu

Received: date / Revised version: date

Abstract Bidirectional model transformation is a key technology in Model-Driven Engineering (MDE), when two models that can change over time have to be kept constantly consistent with each other. While several model transformation tools include at least a partial support to bidirectionality, it is not clear how these bidirectional capabilities relate to each other and to similar classical problems in computer science, from the view-update problem in databases to bidirectional graph transformations. This paper tries to clarify and visualize the space of design choices for bidirectional transformations from an MDE point of view, in the form of a feature model. The selected list of existing approaches are characterized by mapping them to the feature model. Then the feature model is used to highlight some unexplored research lines in bidirectional transformations.

Keywords Bidirectional model transformation – Feature model

M. Tisi and J. Cabot

AtlanMod, INRIA & École des Mines de Nantes, LINA, France

{massimo.tisi, jordi.cabot}@inria.fr

S. Hidaka and Z. Hu

National Institute of Informatics & SOKENDAI (The Graduate University for Advanced Studies), Japan

{hidaka, hu}@nii.ac.jp

1 Introduction

Information in software systems is encapsulated and structured in several artifacts that evolve over time. Artifacts are generally not independent from each other, being connected by syntactic or semantic relationships that can be a stable part of the system or evolve together with the artifacts. When the relationships hold we say that the whole system is *consistent*. When the

relationship is such that an artifact can be generated from another, a program can be implemented, executing this generation. This kind of programs are generally called *transformations*. In the simplest case transformations are unidirectional, specifying how to derive a target artifact given an up-to-date source artifact, and ignoring the case of manual modifications of the target. However, in most practical cases, artifacts connected by a consistency relationship can be created or updated independently and an automatic system is needed to guarantee the preservation of this consistency.

In this setting, a *bidirectional transformation* is a system to enforce the consistency between two artifacts. We introduce the notion of bidirectional transformation and related concepts following [20, 55]. We consider bidirectional transformations between a set of *source* artifacts S and a set of *target* artifacts T. A (unidirectional) transformation $get : S \to T$ from S to T creates a target artifact from a given source artifact. Then the two artifacts $s \in S$ and $t \in T$ are *consistent*[1] if and only if $t = get\, s$. A bidirectional transformation system also performs the reflection of updates on the target artifacts to the source artifacts. In this case, the backward transformation $put : S \times T \to S$ usually takes the original source $s \in S$ in addition to the updated target t' to obtain the updated target $s' = put(s, t')$.

Alternatively, a bidirectional transformation may be symmetrically specified by a relation $R \subset S \times T$. In this case, the two artifacts $s \in S$ and $t \in T$ are consistent if and only if $(s, t) \in R$. Forward transformation and backward transformation will be denoted by $\overrightarrow{R} : S \times T \to T$ and $\overleftarrow{R} : S \times T \to S$, respectively[2] .

Bidirectional transformation has various applications including synchronization of replicated data in different formats [20], presentation-oriented structured document development [30], interactive user interface design [46], coupled software transformation [42], and view updating mechanisms which have been intensively studied in the database community [3, 13].

In this paper, we propose a feature model to compare different bidirectional model transformation approaches and we apply it to examples of existing approaches. A feature model [37] is a hierarchical tree commonly used to organize and visualize the features

[1] Here we consider consistency among artifacts. We also mention consistency between forward and backward *transformations* in Section 3.4.7.

[2] A more general scheme may be considered – synchronization with respect to transformation f achieved by the function $\mathsf{sync}_f : S \times S \times T \to S \times T$ which takes the original source s, updated source and updated target (original target is equal to $f(s)$ so is not in the signature) and returns a pair of updated source and target. All our discussion can be generalized to this scheme, so we will discuss it explicitely only when needed, i.e. in Section 3.4.7.

of a generic domain and specify their variability. Our objectives for building such a feature model are 1) to propose a classification for existing bidirectional transformation systems, so that we can clarify the design space and reason about current approaches and 2) to highlight some unexplored research lines. We apply our feature model to a set of existing transformation systems that is not meant to be exhaustive, but representative of a wide range of approaches. The list includes all the bidirectional transformation systems we know about that 1) natively transform models and 2) provide a public implementation we can analyze. We also extend the classification to examples of graph-, tree-, and text-based systems that have been, or have the potential to be, adapted to handle models.

The rest of the paper is organized as follows: Section 2 introduces a simple case of bidirectional transformation that will be used to exemplify the concepts of the paper. Section 3 applies domain analysis to bidirectional transformation approaches and proposes a feature model. Section 4 analyzes the existing approaches we have chosen to classify based on our feature model. Section 5 is devoted to discussing the proposed classification, identifying unexplored features and proposing further research activities. Section 6 compares our categorization with related efforts in the literature; Section 7 draws the conclusions. In this paper, we have tried our best to make the explanation accessible to a wider audience than the bidirectional transformation community, and thus we omitted when possible formal notations and definitions.

2 Running Case

To exemplify the illustration of the feature model in the next section, we introduce a simple transformation that we will use as a running case throughout the paper. In the example a system manages the anagraphical data of a set of individuals, by grouping them in families. Fig. 1 shows the structure of this representation, in the form of an Ecore metamodel: elements of type Family are characterized by a last name and contain elements of type Member, for which the system stores first name and gender. While Ecore is a common format in the MDE community, other environments describe artifact structure by different representations, e.g., as an XML schema or a set of grammar rules. Each one of these formats is associated with a corresponding type of artifact, e.g., instances of an Ecore metamodel are expressed as instance models (e.g. object diagrams), instances of an XML schema are XML files and instances of a grammar specification are generic textual artifacts. In Listing 1 we show an example of Family artifact in XML format.

Listing 1 An instance of Families.

```
1 <Families>
```

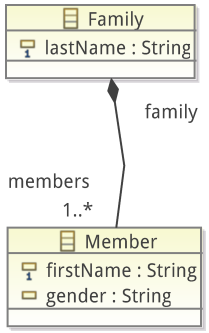


Fig. 1 Families metamodel

```
  <Family lastName="March">
    <members firstName="Jim" gender="male"/>
    <members firstName="Cindy" gender="female"/>
    <members firstName="Brandon" gender="male"/>
    <members firstName="Brenda" gender="female"/>
  </Family>
  <Family lastName="Sailor">
    <members firstName="Peter" gender="male"/>
    <members firstName="Jackie" gender="female"/>
    <members firstName="David" gender="male"/>
    <members firstName="Dylan" gender="male"/>
    <members firstName="Kelly" gender="female"/>
  </Family>
</Families>
```

Let's assume that the system creates a view of this data structure as a flat list of Persons, distinguishing males and females, according to the metamodel in Fig. 2. Listing 2 shows the instance of the Persons metamodel consistent with Listing 1.

Listing 2 An instance of Persons.

```
<Persons>
  <Male firstName="Jim" lastName="March"/>
  <Male firstName="Brandon" lastName="March"/>
  <Male firstName="Peter" lastName="Sailor"/>
  <Male firstName="David" lastName="Sailor"/>
  <Male firstName="Dylan" lastName="Sailor"/>
```

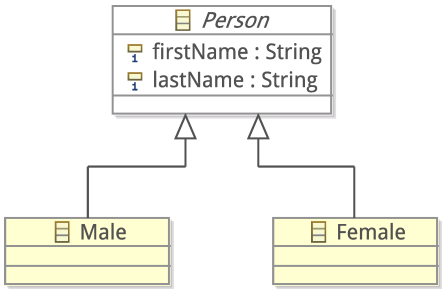


Fig. 2 Persons metamodel

```
  <Female firstName="Brenda" lastName="March"/>
  <Female firstName="Cindy" lastName="March"/>
  <Female firstName="Jackie" lastName="Sailor"/>
  <Female firstName="Kelly" lastName="Sailor"/>
</Persons>
```

In Listing 3 and Fig. 3 we show two possible representations of this transformation, respectively in the model transformation language ATL and as a Triple Graph Grammar. The ATL code is composed by two rules, each one describing which element to generate in the target model (*to* section) when an element with specific properties is traversed in the source model (*from* section). The TGG transformation in Fig. 3 uses a short notation to describe the consistent couples source-target as triple grammar rules: left- and right-hand side of a rule are depicted in one triple graph and the elements to be created have the label ++. The first rule initially creates a Family together with one male Member in the source model, the corresponding Male in the target model and the explicit correspondence structure. The second rule requires an existing Family and creates a new male

member. The third rule extends two corresponding males by their first names. The TGG contains similar rules (not depicted) for female family members.

Listing 3 ATL SimpleFamilies2Persons transformation.

```
1  rule Member2Male {
2    from
3      s : Families!Member (s.gender = 'male')
4    to
5      t : Persons!Male (
6        firstName <- s.firstName,
7        lastName <- s.family.lastName
8      )
9  }
10 rule Member2Female {
11   from
12     s : Families!Member (s.gender = 'female')
13   to
14     t : Persons!Female (
15       firstName <- s.firstName,
16       lastName <- s.family.lastName
17     )
18 }
```

With these transformations the system developer represents the consistency relation between Families artifacts and Persons artifacts. Depending on the situation the user may want to exploit this relation in different ways. For instance she may expect the system to restore consistency whenever a first name is changed in the instance of the Persons metamodel. Note that in this scenario handling families with the same last name may not be trivial. In other cases she may need to concurrently update the Families and Persons models and have the system handle possibly conflicting changes. In the next section we will provide a categorization of the expectations that are fulfilled by the main approaches in bidirectional transformations.

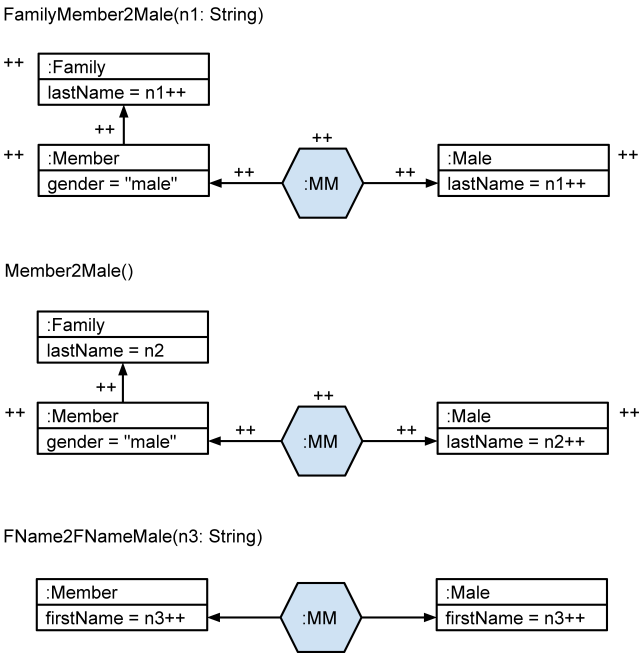


Fig. 3 SimpleFamilies2Persons transformation in TGG (three analogous rules for *Female* are omitted)

3 Features for Bidirectional Transformation Approaches

In this section we apply domain analysis to bidirectional transformation approaches in a similar way to [12]. Domain analysis aims at identifying and modeling the commonalities and variabilities of the elements of a particular domain. Feature diagrams are a popular method to visualize these variabilities [37] by organizing the features of a generic domain element in hierarchical trees. A *feature configuration* is a set of features owned by

a member of the domain and it is permitted by a feature model if and only if it does not violate constraints imposed by the model. Main constraints are specified as relationships between a parent feature and its child features (or subfeatures), by the notation illustrated in Table 1. Given a feature configuration that contains a certain feature F, the configuration 1) has to contain any mandatory child feature of F, 2) may contain any optional child feature of F, 3) may contain one or more child features in an *OR group* of F, 4) must contain one of the child features in an *Alternative group* of F.

Table 1 Feature diagram notation

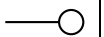
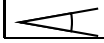
	Mandatory feature
	Optional feature
	Or group
	Alternative group

Figure 4 shows the toplevel feature diagram. We distinguish four major areas of variation:

Technical Space. Bidirectional approaches have a strong dependency on the form of the artifacts they transform. In this feature, approaches are characterized based on the artifact representation they refer to.

Correspondence. Every bidirectional tool provides a means of definition of a correspondence relation between the source and target artifact sets. In this variation point, approaches are characterized based on the kind of relation they allow users to define. Note that we only deal with correspondence between two artifacts. Multi-directional transformation, to keep more than two artifacts consistent, is beyond the scope of this paper and we plan to address it in future work.

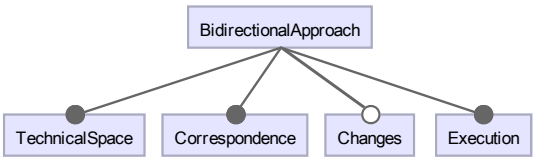


Fig. 4 Top-level feature diagram

Changes. Several approaches assume the existence of an initial state in which the artifacts are supposed to be consistent to each other, and allow the user to define a set of changes to one of the models, providing a certain level of bidirectional synchronization. This optional macro-feature analyzes which kind of updates are allowed by the tool.

Execution. Once artifacts, correspondence, and updates are defined, the execution semantics of the approach reifies the core bidirectional strategy. This variation point structures the possible choices in the tool execution semantics.

3.1 Technical Space

This feature classifies bidirectional transformation tools based on the concept of technical space described in [10]. For the classification we consider the way the tool accesses the artifact, i.e. which assumptions the tool makes on the artifact structure.

As shown in Fig. 5, we distinguish tools that assume their artifacts to be:

- *Text*ual artifacts, optionally conforming to grammar rules, e.g. in BNF (Backus-Naur Form);
- well-formed *XML* files, optionally conforming to a schema;
- *MDE* meta-modeled artifacts, typed attributed graphs conforming to a MOF-like metamodel;
- generic *Graph* artifacts with an optional graph schema, including untyped graphs such as edge-labeled graphs.

Note that transformation tools can in principle be applied also to other technical spaces by first translating the artifacts to the tool native format. However, most bidirectional approaches implement algorithms that are strongly dependent on their native technical space. Hence, in this feature we only refer to the native artifact representation.

Note also that in some cases artifacts in a technical space can be directly transformed by tools in other technical spaces, but with a loss in expressivity. For instance a tool in the *Text* technical space is able to transform any textual artifact. Hence it can also transform XML files, but without exploiting the XML structure or a provided XML schema. Similarly, MDE models may be saved on disk in an XML format (or another textual format). A tool in the XML technical space may transform the XML serialization of the model, but it would not be able to exploit the full semantics of the Ecore-based structure, since concepts like opposite references or multiple inheritance do not have a representation in XML schema. Similar arguments stand for graph artifacts. The metamodels in Fig. 1 and 2 are examples of artifacts in the MDE technical space, while Listings 1 and 2 belong to the XML technical space.

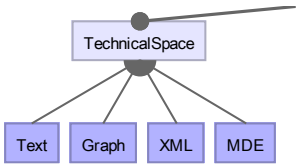


Fig. 5 Technical Space

Since the view-update [3,13] problem is well-studied in the database community and the capabilities are supported in many relational database management systems, another technical space *Relation* could be added to our model. However no transformation tool we know, in the context of MDE, operates directly on relations.

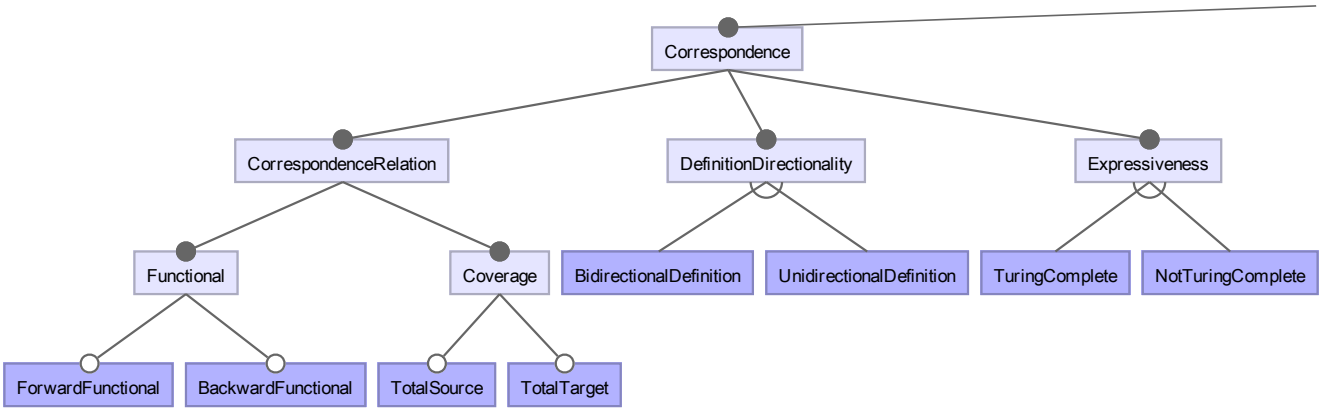


Fig. 6 Correspondence

3.2 Correspondence

All the approaches we analyzed rely on the definition, with different degrees of explicitness, of a correspondence relation R between two sets of artifacts. This relation connects artifacts that are considered to be consistent with each other in the considered environment, and it is sometimes referred to as consistency relationship. Functional relation of function $get : S \rightarrow T$ is defined by the relation $R_{get} \subset S \times T$ such that $(s, t) \in R_{get}$ if and only if $get(s) = t$. Listing 3 and Fig. 3 exemplify two alternative ways to describe a correspondence relation. Figure 6 models the design choices in the definition mechanism of the correspondence relationship. Note that the *Correspondence* feature refers to correspondence between artifacts as a whole, and not to the fine-grained correspondence between components inside artifacts, e.g. between model elements in the MDE technical space. The latter type of correspondence is analyzed in the *Traces* feature (Section 3.4.6).

The topology of the relation is the first feature we consider to characterize the notion of bidirectionality underlying the approaches under study. Relationships can be *Functional* in one of the two directions (or both) when they associate one artifact to at most one image. With the features *Forward Functional* and *Backward Functional* we refer to tools that allow users to define only functional relationships. Such tools generally have a simpler and more intuitive bidirectional synchronization process, since ambiguous correspondences are avoided by construction. This comes at the cost of a limited expressivity in the supported transformations. Note that an approach without these functional correspondence may still exhibit functional *behavior*. We revisit this aspect as a separate feature in Section 3.4.7. The conjunction of Forward Functional and Backward Functional implies bijectivity. Bijective correspon-

dences disallow information-losing transformations, with a strong reduction of the expressive power of the approach. The ATL transformation in Listing 3 defines a forward-functional correspondence: given one Families model a single Persons model is generated by the transformation. The ATL language gives no guarantee of backward-functionality: different Families models could be associated to the same Persons model, e.g., in case of families with the same last name. The same stands for the graph transformation in Fig. 3. Note that the same transformations would be bijective if we constrained families to have a unique last name (or if we added a family identifier to the source and target metamodels).

The correspondence relation can totally cover one or both of the artifact sets under transformation, meaning that it can connect every possible artifact with a correspondent in the other side. Requiring total coverage (total in the sense of total function) in one of the two sides (*Total Source* or *Total Target*) is a strong limitation of the transformation expressive power, but gives the guarantee that any model produced by the user will be a correct input for the bidirectional transformation. Our example totally covers the source and target domains (every model conforming to the Families metamodel can be associated to a model conforming to the Persons metamodel and vice-versa).

An important categorization of the tools is related to the directionality of the consistency definition (*Definition Directionality*). Some tools are based on unidirectional transformations and they use the transformation definition as the user-provided correspondence relation (*Unidirectional Definition*). In these cases the backward transformation is not explicitly provided, and can be derived or just simulated by the bidirectional engine. This is the case of the ATL specification of Listing 3. Other tools rely on a natively bidirectional definition, executable or not in both directions (*Bidirectional Definition*). These cases, that include the TGG specification of Fig. 3, are usually less expressive than the unidirectional ones, but they avoid the costly bidirectionalization process — Bidirectionalization is the process to provide a unidirectional transformation system with a mechanism to perform backward transformation. Bidirectionalization usually requires an enhancement of the forward transformation semantics (for example, by generating support information described in Section 3.4.6) to facilitate the backward transformation, as well as the definition of an algorithm to compute the backward transformation results, while respecting the forward transformation semantics. On the contrary, when a bidirectional definition is provided, it already includes semantics in both directions, that can be executed by direct interpretation of the definition. —

Finally the definition of correspondence will make use of a language whose expressive power can vary. In this feature model we only express whether the expressiveness of the correspondence language reaches *Turing-complete*ness or not (*Not Turing-Complete*). Note that some approaches may prefer sub-Turing-complete expressiveness to achieve other desirable properties such as guaranteed termination of the transformation in both directions. In our examples the ATL language is Turing-complete. Conversely, information-preserving TGG [40], a restriction of the TGG language to information-preserving transformation, is an example of approach that sacrifices Turing-completeness.

3.3 Changes

Several approaches allow the user to make modifications to one of two consistent artifacts and provide the propagation of these changes to the other artifact. These tools can come with artifact editors that allow the definition of these changes and check their correctness, before activating the synchronization engine. We divide this feature (Fig. 7) into two subfeatures, respectively related to the mechanism for defining updates and to the kind of supported updates.

The feature *Change Definition* is divided in two mandatory features: *Change Representation* distinguishes systems that represent changes as sequences of states (*State Based* systems) or sequences of operations (*Operation Based* systems). In the following, when talking of the running case, we will refer to two specific existing tools as bidirectional systems: 1) a state-based synchronization system for TGG[3], that analyzes two subsequent version of a Persons model to compute how to correspondently update the Families model [40]; 2) an operation-based system for ATL, that uses a record of the atomic operations on one model to calculate the updates on the other [60].

The feature *Change Input* analyzes the insertion method for changes in the bidirectional approach. Systems are called *Live* when they allow users to insert updates only during the system execution, so that they can monitor the user interaction and record the exact sequence of actions. Other systems are *Offline*, meaning that the updated artifact (or the sequence of operations to update it) can be edited with external tools and provided to the bidirectional engine asynchronously. Both the systems in our running case are offline, and they do not require to monitor the user-interaction.

The other main feature related to *Changes* is *Change Support*, characterizing the set of changes that

[3] TGG also provides theoretical foundations for operation-based execution. The feasibility of this approach is shown in [24].

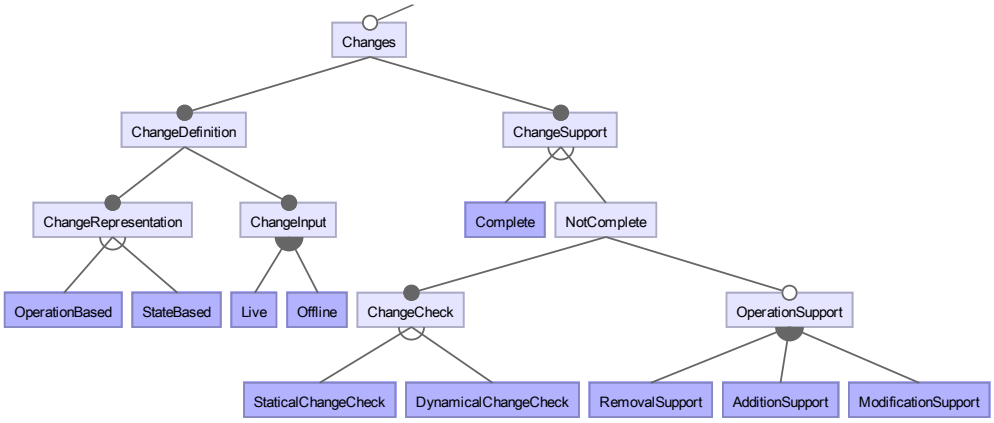


Fig. 7 Changes

the tool allows on the synchronized artifacts. Some tools provide *Complete* change support, meaning that the user can freely modify both artifacts without the risk to run into unreflectable states. In other words, any change made by the user can be reflected[4] to the other artifacts. In cases when the support is *Not Complete*, every update is checked by the tool to detect invalid updates. A simple example of invalid update is the modification of a constant value created by the transformation. Such update could not be allowed since any value other than the constant would be out of the range of the transformation. The (information-preserving) bidirectional engine for TGG in [40] supports all possible changes in the correspondent artifacts, while the ATL engine in [60] does not support addition of elements to the target model of the original transformation. To a system like Boomerang [5], we assign a *Complete* change support. For example, in Boomerang, codomain (or range) of forward transformations — or equivalently, domain of backward transformations (*putback*) — is clearly defined, and these functions are total. Therefore, any change of the target artifacts is allowed within the domain of the putback function. For the transformation that consists of a constant value mentioned above, the domain of the forward transformation is the singleton set $\{c\}$ of the constant c. Therefore, the only considered artifact after update is the constant c, so the change support is still complete (within the domain $\{c\}$).

We distinguish two ways of checking the validity of a change. Systems in which the check function over the artifact is generated statically, by analyzing the transformation code (*Statical Change Check*) and systems

[4] The kind of change on the other artifacts that reflect the change made by the user is not necessarily the same as the kind of the change made by the user.

in which the check is performed dynamically, by explicitly running the back-propagation and detecting runtime exceptions (*Dynamical Change Check*). It may be possible to further refine the Dynamic Change Check feature. For example, specifying whether the check is done with respect to entire transformation specifications or only the fragments of the specifications, during the transformation or after the transformation is finished. While the TGG system in [40] does not need to perform update validity checks, because of its complete change support, the ATL engine in [60] is only able to detect certain errors by executing the transformation, i.e. it implements dynamical checks.

Finally bidirectional systems show separate problems when dealing with different kinds of update operations (i.e. removal, addition, modification). For this reason several systems completely disallow some problematic operation types. The feature model presents an explicit feature indicating the support on the specific operation type (*Removal Support*, *Addition Support*, *Modification Support*). Note that in systems like VDL [45] distinction between addition, removal and modification is somehow meaningless (being the system based on constructors like *cons* and *nil* in Lisp). In VDL pure deletion is not supported, but can be *represented* by replacement.

3.4 Execution

The execution of the bidirectional engine is subject to several variability points, as shown in Fig. 8. We describe its first-level subfeatures (*Semantics, Execution Automation, Application, Approach, Backward Transformation, Support Information, Well-behavedness*) in the following sections.

3.4.1 Semantics.

This feature (Fig. 9) distinguishes systems that can bidirectionally *Check* consistency between two models from systems that can proceed with enforcing the consistency if it is missing (*Enforcement*). Depending on the approach, consistency may be enforced by modifying one or both of the corresponding artifacts.

Note that enforcement can sometimes be performed without the tool actually checking for pre-existing consistency, e.g. in cases when one of the artifacts is always regenerated from the other.

3.4.2 Execution Automation.

In some cases a single artifact can potentially have multiple, equally consistent counterparts on the other side. Conceptually such situations require a choice, to be performed by the bidirectional engine or directly by the user. The feature *Execution Automation* in Fig. 9 expresses the different kinds of user interaction. We dis-

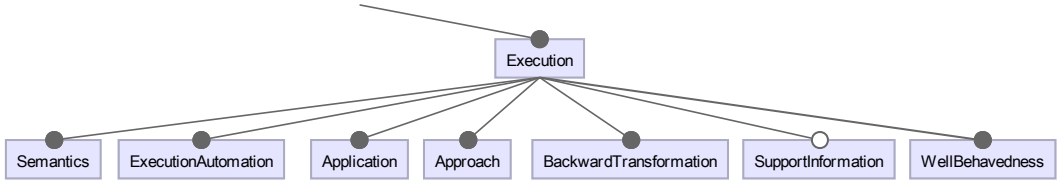


Fig. 8 Execution, fist-level feature diagram

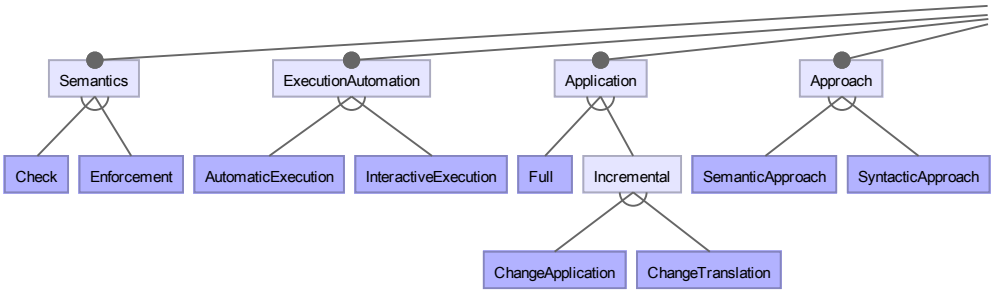


Fig. 9 Semantics, ExecutionAutomation, Application, Approach

tinguish a completely *Automatic Execution* where all the decisions, deterministic or not, are taken by the system and an *Interactive Execution*, which requires a certain kind of user intervention. In this last case the amount of interaction can vary from minimal, limited to disambiguating between equivalent options, to the extreme case in which the tool just checks consistency while leaving to the user the task of restoring it. In-between these extreme cases, the checking capability could guide the user to restore consistency: the user may be required to provide incremental input to narrow down the set of alternatives, and the system may return meaningful feedback by checking the (partial) consistency of the current solution.

In our running case, the synchronization problem shows some ambiguity. For instance, if two Families have the same name and we add a Person with this last name, the system will not be able to autonomously identify the correspondent family. To solve this ambiguity, a system may proceed interactively, e.g. first asking the user if the Person belongs to an existing family (otherwise a new Family gets created) and in affirmative case it could offer the possibility to select the correct family.

3.4.3 Application.

Some bidirectional transformation systems enforce consistency between artifacts without considering their history. We refer to this kind of systems by the feature *Full Application*. On the contrary other systems are able to synchronize artifacts only if they are provided with an initial consistent state for those artifacts. We refer to this second class as *Incremental Application* (Fig. 9).

Note that we do not consider a system incremental unless it explicitly deals with update operations, even if the old source is utilized to resolve ambiguity in non-bijective transformations, as it normally happens by the *put* function. The use of the ATL engine SyncATL [60] in our running case is an example of incremental application. The user could start from a consistent state, made by the two models in Listings 1 and 2, update the name of a Person in the target model and backpropagate the change to the source artifact. Only the updated name of the appropriate family member would be recomputed. Conversely an approach with *Full Application* would recompute the whole source artifact without considering the previous consistent state.

In cases of incremental consistency application, most bidirectional approaches just propagate the changes from one model to the other, arriving at an updated version of the two models. Some systems can instead perform an explicit translation of the change sequence, from one side to the other. This translation approach, typical in operation-based systems, requires at least an explicit formalization of the update language for the two artifacts. We refer to the first feature as *Change Application* and to the second as *Change Translation*. The ATL engine in [60] directly applies the changes to the other side, without an explicit translation. Conversely, a *Change Translation* would explicitly encode the target update operation (e.g., Person[id=2].firstName:"Peter"→"Pete") and translate this representation in an update on the source model (e.g., Member[id=6].firstName:"Peter"→"Pete").

3.4.4 Approach.

Most tools in our list make use of the transformation code to drive the bidirectional synchronization. We denote these systems with the feature *Syntactic Approach*, meaning that they have access to the transformation syntax.

A few tools consider the transformation as a black-box, and try to derive the synchronization actions from different sources, for instance feeding the transformation with sequences of test data. Literature refers to these tools by the name *Semantic Approach* [57] (Fig. 9). Semantics here correspond to the behavior of the transformation in the sense that if two different transformations result in the same targets for all sources, then the two transformations are considered semantically equivalent. For the transformation in the running case, a semantic approach would for instance execute the transformation of a "dummy" collection of Family, each with unique last name and first name according to the position of the Family she belongs to (`March` will be 1 and `Sailor` will be 2), and the position inside the Family (`Jim` will be 1/1, `Cindy`

1/2, ... `Peter` 2/1 `Jackie` 2/2 and so on.). Then the system would observe the correspondence between the unique names, and build the correspondence table between the names, so that the table can define a positional correspondence between source and targets. Then the system would use the positional information to figure out how to propagate updates. Note that such a system only uses the external behavior of the transformation (and possibly general semantic information assumed in the transformations) but never analyses the transformation code.

3.4.5 Backward Transformation.

Fig. 10 shows the feature diagram for the two subfeatures of *Backward Transformation* and *Support Information*. In the *Backward Transformation* feature we express whether or not the system contains an explicit definition of the backward transformation (*Explicit Backward* or *Implicit Backward*).

The explicit definition can be provided by the user, and in this case it coincides with (part of) the consistency definition (see feature *Definition Directionality*). Otherwise the explicit backward transformation can be automatically derived by the tool (literature talks in these cases of *Inversion*). While the ATL engine in [60] does not build an explicit inverse transformation, an inversion approach for ATL is available.

3.4.6 Support Information.

Figure 10 shows the feature *Support Information* and its hierarchical structure.

Bidirectional transformation approaches make use of two different kinds of support information, which we represent with the features *Complement* and *Traces*. When a bidirectional engine directly executes both directions of the transformation, the kind of support information available and the way how the information is stored is up to the bidirectional transformation approach. On the other hand, in some cases the bidirectional transformation tool is built on top of a preexisting unidirectional transformation tool. In these situations the bidirectional tool relies on the support information that is made available by the unidirectional part. For instance the ATL engine in [60] is built on top of the standard ATL unidirectional engine and inherits its tracing mechanism.

The first kind of support information is created when there is some information in one of the artifacts, which is not contained in the other. The missing information can be still transferred to the other side and it is often called *Complement* [3,45]. The ATL transformation of the running case loses the information of the family affiliations of persons with identical last names. The system may then store this information as

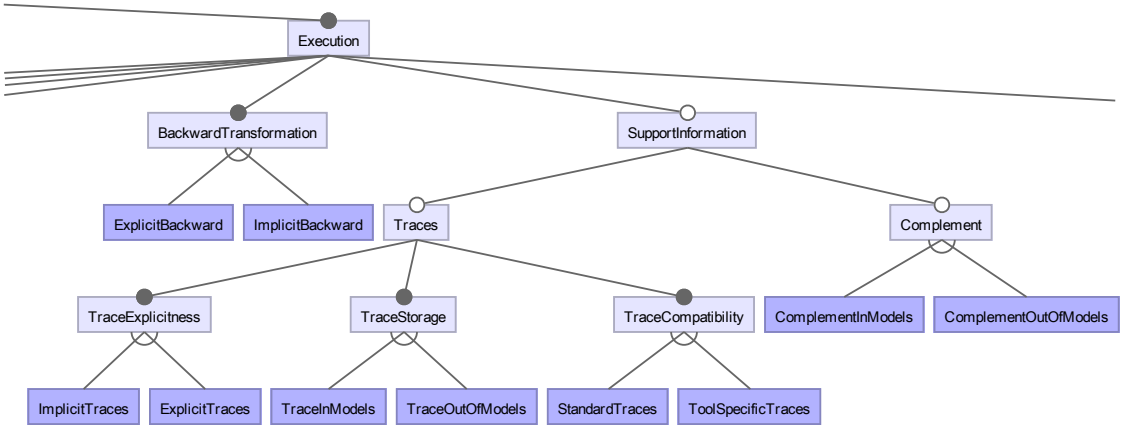


Fig. 10 Backward Transformation, Support Information

a complement, and use this complement information during backward transformation to determine, if one of the person is deleted, which person from which family can be deleted accordingly.

We distinguish the cases in which this information is stored in models (*Complement In Models*) and out of models (*Complement Out Of Models*). The first case requires to use annotations, comments, or to extend the structure of the artifact to contain the additional information. The second case explicitly separates complement information in a newly generated model. In both cases complement information usually requires the user to edit artifacts with a tool (possibly ad-hoc) that can correctly manage the complements to make them consistent with the artifact.

Also in transformations without information loss, the approach may store during the forward transformation some additional data that may help the bidirectionalization. The most notable case is *Traces*, persistent representations of the fine-grained correspondence between elements in the left and the right hand sides (they are called *Correspondences* in some approaches like TGG [53]). Some tools, like the TGG engine in [40], store *Explicit Traces* in the form of inter-artifact references and others have compact implicit representations for them (*Implicit Traces*). For instance, making tracing information explicit significantly helped formalizing the implementation of GRoundTram [26]. Traces can be stored in models (*Trace In Models*), by extending the left and the right artifacts or out of models (*Trace Out of Models*), as a third artifact produced by the transformation engine. For instance in Boomerang trace information is stored by building a separate dictionary as a correspondence between key and associated data. Finally, traces are usually stored in a tool specific format (*Tool Specific Traces*), requiring the user to perform artifact editing and transformation within the same environment. Otherwise traces could be available

in a standard format (*Standard Traces*) which could be read by several tools. This would enable the user to choose different tools for editing and transformation, with the additional cost of having another artifact to manage. *Traces Out of Models* and in *Standard Format* are usually provided with enhanced interoperability in mind.

It is worth noting that complements and traces may not be independent. Recall that complements are information lost during transformation. Formally, given a function $f : S \to T$, function $f^c : S \to C$ is a complement function if $(f, f^c) : S \to T \times C$ is injective [3]. Note that under this requirement, f^c may be arbitrarily rich[5] (in the extreme case f^c may be the identity function, returning all the information in the source) and may even generate traces. In that case, the complement subsumes traces. If the only information lost during transformation is the traces, then complement and traces coincide. On the contrary, if a transformation discards information other than traces (e.g., if in the running example only male family members were transformed), then the traces cannot hold the discarded information. Therefore, traces cannot subsume complements in general.

[5] Apart from the updatability of the target discussed in [45].

It is also worth noting that the support information is optional in our feature model. For example, Focal (original version of lens [20]) does not rely on either complement or traces. However, since the backward transformation *put* has full access to the original source, it can at least deal with non-bijective transformations without complements. While from a purely semantical point of view, any bidirectional transformation approach theoretically uses some information to conduct backward transformation, we discuss in this feature only support information computed and stored in some form by the tool. Additionally, the feature Support Information may be selected even without selecting any of its subfeatures. This is meant to leave open the possibility to categorize future approaches with different structures of support information.

Traces can be considered as one of the technical solutions to the model alignment problem, and Diskin et al. [17] treat them as horizontal delta in their delta-based framework. This is also an important viewpoint for unified treatment with another delta – vertical delta that corresponds to the representation of Changes in our feature model. We will revisit the delta-based framework to see how it helps in reasoning about relationships between state-based and operation-based approaches in Section 3.4.7.

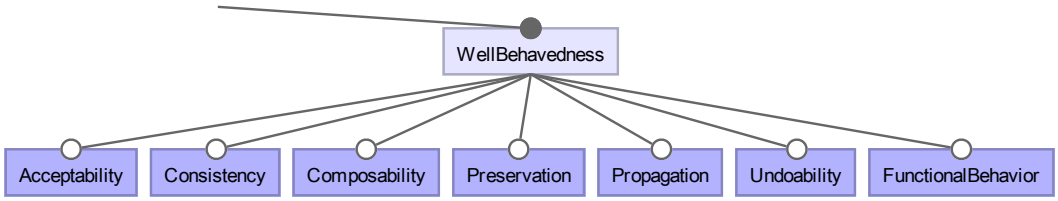


Fig. 11 Well-behavedness properties

3.4.7 Well-behavedness properties.

Fig. 11 shows the feature *Well-behavedness (round-trip) properties*. These properties capture the formal guarantees that the system provides during bidirectional transformation. In general, the more expressive power the transformation has, the less guarantees on well-behavedness the system can provide. In the extreme case, Turing-complete transformation languages can express complex forward transformations, making it difficult to define a backward transformation that, when combined with the forward one, will form a well-behaved system.

The features discussed in this section include the ones that are specific to operation-based synchronizers that is not only a bidirectional transformation approach (by having explicit notion of forward transformation function f), but also a synchronization approach, by taking updated source and target as their inputs [60]. So, when these properties are discussed, we further generalize the bidirectionalization scheme we already introduced in Section 1 to the synchronization with respect to forward transformation $f : S \to T$ achieved by the function $\mathsf{sync}_f : S \times S \times T \to S \times T$ which takes the original source $s \in S$, updated source in S and updated target in T (original target is equal to $f(s)$ so is not in the signature) and returns a pair of source and target in which updates are reflected. In this scheme, update operations are denoted by functions $\psi \in \Psi$ on the source and the target, denoted by $\psi_s : S \to S$ when it is applied to $s \in S$, and $\psi_t : T \to T$ when they are applied to $t \in T$ that, when applied to original artifacts, generates updated artifacts. sync_f can be related with *get* and *put* by $get(s) = f(s)$ and $put(s,t) = \pi_1(\mathsf{sync}_f(s,s,t))$ where $\pi_1(s,t) \stackrel{\text{def}}{=} s$.

The first two properties are well-recognized in the bidirectional transformation community.

– *Acceptability* says that no modification on the target artifact leads to no modification on the source artifact. Some literature [20] refer to this property as GetPut because it is characterized by $s = put(s, get\ s)$. In our running case, suppose the Family model in Listing 1 has been transformed to a Person model in Listing 2. If the Person model without any modification is fed to backward

transformation, then the result should be identical to the Family model in Listing 1.

This feature is also called *Stability* in operation-based approaches like SyncATL [60] in the sense that if no update operation is applied to both source and target, then both source and target remain unchanged after transformation.

– *Consistency*, or PutGet [20], says that another forward transformation yields the same target artifact as the previous target artifact that was fed to the backward transformation. It is characterized using *put* and *get* by $get(put(s,t)) = t$. In our running case, suppose the Family model in Listing 1 has been transformed to a Person model in Listing 2, and then `Bill March` is added to the Person model. Then, after backward transformation immediately followed by the forward transformation, `Bill March` should still stay in the Person model while the other persons stay unchanged.

Language X [29,30] supports a slightly relaxed version of Consistency, due to the fact that its transformation language has a built-in support for duplication (so that during backward transformations, modification to only one of the copies is propagated to the source) and it handles dependencies inside the target artifacts. Since the next forward transformation would update the other copy, Consistency will be violated.

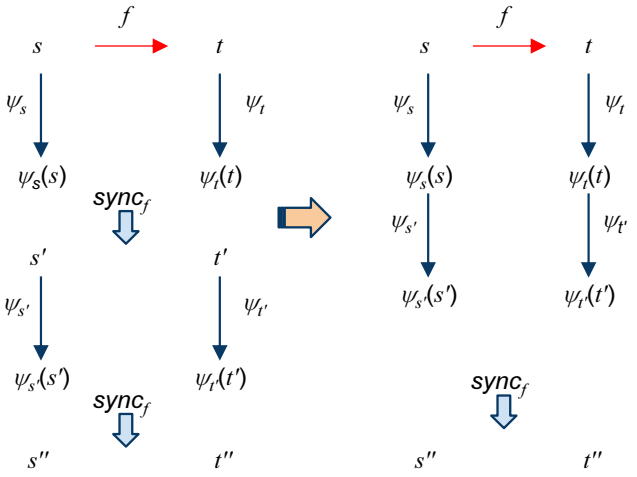


Fig. 13 Composability Feature

The next two properties are specific to operation-based synchronizers that accepts updates on both source and target [60].

– *Preservation* says that when modifications are applied to the source and/or target artifacts, then after transformation (synchronization) the effect of these modifications is preserved in the artifacts. Modification ψ, as function that takes artifacts and returns the same type of artifacts, is considered preserved if it becomes idempotent after synchronization, i.e., $\psi(x) = x$. An example of modification operations is to set a component of an artifact to a specified value. The function being idempotent suggests that the value is already set to the value.

– *Propagation* says that when modifications are applied to the artifacts, and propagated by transfor-

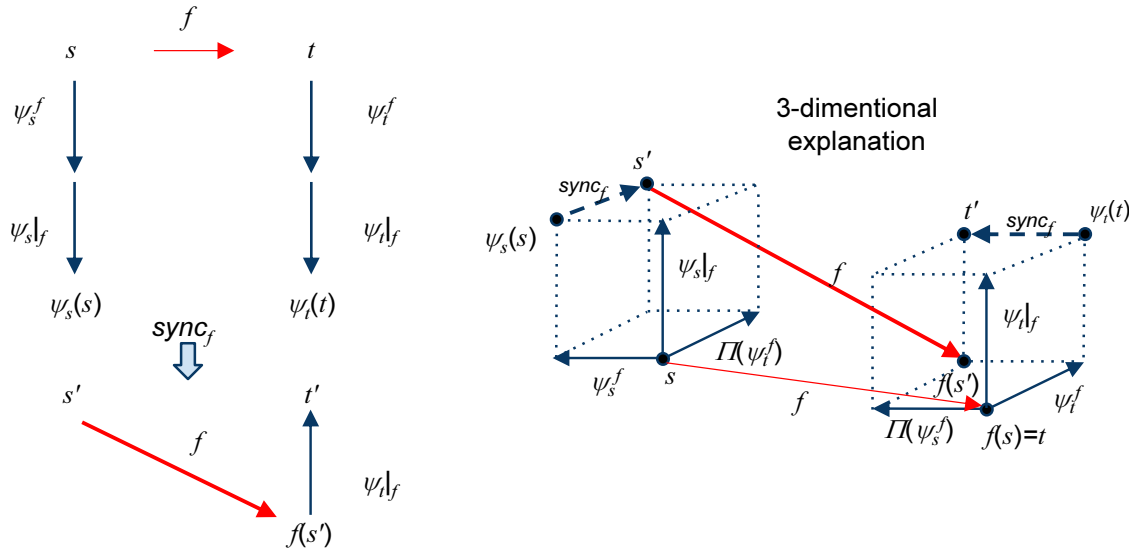


Fig. 12 Propagation Feature

mation, then when another forward transformation is applied on the source, the new target as the result of the transformation remains within the previous state of the target, modulo the modification on the target that is not propagable to the source in terms of the definition of transformation. This means that modifications on the target are correctly propagated to the source.

For a concrete example, suppose we want to synchronize a UML model s with Java code t using transformation f from UML model to Java code. Then an example of "the modification on the target that is not propagable to the source" (denoted $\psi_t|_f$ in the following explanation) will be a modification on the comments in Java code that f does not produce. Example of the modification on the source that is not propagable to the target (denoted $\psi_s|_f$) will be a modification on the element of UML that f projects away. So, after updates on both sides followed by synchronization that produces the pair of UML model and Java code s' and t', if s' is transformed by f, it is not equal to t' because updates on the comments on Java are lost, but if the comments are edited again by $\psi_t|_f$, then the resultant Java code will be equal to t' if Propagation property is satisfied.

Propagation is specific to approaches like SyncATL [60] which is not only a bidirectional transformation approach (by having explicit notion of forward transformation function f), but also a synchronization approach, as introduced in the beginning of this subsection. Updates ψ are always assumed to be decomposable into sequences of distinct atomic updates and the order of applying

these updates is insignificant. Further, updates are classified into 1) a sequence of updates that are reflectable through f to artifacts on the other side and 2) updates that are not reflectable. These sequences are denoted by ψ^f, respectively $\psi|_f$. The Propagation property with respect to synchronization $\mathsf{sync}_f : S \times S \times T \to S \times T$ is characterized as the following: Given $f : S \to T, s \in S, \psi_s, \psi_t \in \Psi$. If $\mathsf{sync}_f(s, \psi_s(s), \psi_t(f(s))) = (s', t')$, then we have $\psi_t|_f(f(s')) = t'$.

The rationale behind is [60]:

Target to source propagation. If an update on the target that is propagable to the source is not propagated to the source, which means, s' does not include that updates, then, $\psi_t|_f(f(s')) = t'$ does not hold (because t' includes that modification while $f(s')$ does not).

Source to target propagation. If an update on the source that is propagable to the target is not propagated to the target, which means, t' does not include that updates, then, $\psi_t|_f(f(s')) = t'$ does not hold because $f(s')$ includes that modification while t' does not.

Figure 12 (left) explains the feature. We start with the original source s and the original target t that is created by applying f to s. Then we apply modifications ψ_s on the source which is decomposed into ψ_s^f that is reflectable to the target, and $\psi_s|_f$ that is not. Update ψ_t that is decomposed similarly into ψ_t^f and $\psi_t|_f$ is applied to target. Then synchronization $\mathsf{sync}_f(s, \psi_s(s), \psi_t(f(s)))$ produces the updated pair of artifacts (s', t'). Then, if f is applied to s', then it is not equal to t', but if $\psi_t|_f$ is further applied, then it is equal to t'. The right part of Figure 12 explains the same thing in a three-dimensional manner to clarify each component of updates, where on the source side, updates from s to s' can be decomposed into three displacements: ψ_s^f, $\psi_s|_f$ and $\Pi(\psi_t^f)$, where $\psi_s = \psi_s^f \circ \psi_s|_f$ and $\Pi(\psi_t^f)$ is the update on the source that is generated by propagating (denoted by Π) update ψ_t^f. Similar decomposition can be done on the updates on the target. On the source side, before synchronization, only ψ_s has been applied, so the source remains on the plane that also includes vectors ψ_s^f and $\psi_s|_f$. After synchronization, the point is displaced along with the vector $\Pi(\psi_t^f)$ (which already assumes the Preservation property). Same displacement occurs on the target side. Since s' includes the update ψ_t^f that has been propagated by Π via synchronization, $f(s')$ is on the plane that ψ_t^f points to, and since ψ_s^f is reflectable, $f(s')$ is also on the plane that $\Pi(\psi_s^f)$ points to, but since $\psi_t|_f$ is not included in s' because it is not reflectable to the source, $f(s')$ is on

the plane that includes the origin of vector $\psi_t|_f$. So applying $\psi_t|_f$ to $f(s')$ will send $f(s')$ to t'.

– *Composability*, or PutPut [20], says that if two successive backward transformations with respect to successive modification operations on the target lead to source s'', then the combined modifications on the target without intermediate backward transformation also leads to source s''. This implies that the backward transformation has only to be performed once after multiple sequence of modification operations, not every time after each modification operation.

  Figure 13 also explains the Composability feature. If two successive synchronizations on a pair of source s and target t with respect to (ψ_s, ψ_t) and (ψ'_s, ψ'_t) arrive at state (s'', t''), then the same sequence of modification without intermediate synchronization will arrive at the same state (s'', t''). This implies that the synchronization has only to be performed only once after multiple sequence of modifications, not every time after each pair of modifications.

  A counterexample [20] is made by: a forward transformation from an artifact and its version number, to extract the artifact part only; a backward transformation that increments the version number if the target has been updated. Then two consecutive backward transformations lead to double increments in the version number, whereas only one backward transformation leads to one increment only, thus violating composability.

The last two features characterize both state- and operation-based systems:

– *Undoability* says that the original source can be restored by using the original target. In other words, the propagated updates on the source can be undone by propagating "reverse" updates that rollback the target to the original state. It is characterized using *get* and *put* by $put(put(s,t), get\, s) = s$. For example, consider the situation right after the initial backward transformation in the scenario described in the *Consistency* feature. In the scenario, update on the target model to add `Bill March` had been propagated to add `Bill` in the family `March`. Then, if `Bill March` is removed from the target and backward transformation is executed, the source model returns to the Family model in Listing 1.

– *Functional Behavior* guarantees that the result of transformation is always uniquely determined. In the running case, *Functional Behavior* guarantees that for any given Family model, the corresponding Persons model is always uniquely determined, and vice versa. Note that this feature covers both deterministic systems and systems in which some nondeterminism is allowed but due to additional con-

fluence properties the final result is always uniquely determined. Also note that systems that don't have a Functional CorrespondenceRelation (Section 3.2) can either exhibit a functional or non-functional behavior.

The above-mentioned fact that some of these properties are applicable only to operation-based systems is formalized by the feature diagram along with the following constraints:

```
Preservation  implies  OperationBased;
Propagation   implies  OperationBased;
```

It is worth noting that a delta-based framework [17] that formalizes operation-based approaches is shown to subsume state-based approaches as a special case, by giving delta (= updates) as pair of states before and after modifications (discrete delta). Therefore, any state-based approach can be tested for operation-based properties, by considering the approach as a special delta-based approach that only accepts discrete delta. However, in an asymmetric state-based approach that does not satisfy PutGet, it is difficult to determine the Preservation property in this way. GRoundTram [26] is among such approaches and claims WPutGet, a relaxed variant of PutGet. WPutGet (discussed later), $put(s, t') = put(s, get(put(s, t')))$ suggests that the update from original target $t = get(s)$ to t' is "preserved" in the view $t'' = get(put(s, t'))$, as the effect of delta (t, t') and (t, t'') is the same for the source. Without clear notion of operations that define the update inclusion relation between (t, t') and (t, t''), the Preservation property is difficult to discuss. As for the Propagation property, we also consider it only concerns operation-based approaches, because a modification should be decomposed into propagable and non-propagable modifications, while in state-based systems, a modification is just a monolithic difference between two states.

Although we have illustrated the main well-behavedness properties for bidirectional model transformation, every new system could add its own well-behavedness property. For instance "Update-preservation", introduced in [30], is a relaxed variant of *Consistency* that allows view side-effects, meaning that the result of another forward transformation t'' is different from the target right after modification t', but requires the updated portion to remain identical. For example, consider a variation of the running case where the Family metamodel has a constraint that at least two persons exist in each Family. Then suppose after forward transformation starting from source model s in Listing 1 resulting in the target model t in Listing 2, `Bill Clinton` is added (to make target state t'). Further, suppose the backward transformation algorithm creates a dummy member if a new Family

is created with only one member (to make source state s'). Then, another forward transformation would create `XXX Clinton` as the default person that belongs to the new Family `Clinton` (to make target state t''). This situation satisfies Update-preservation since, although t'' is different from t', added `Bill Clinton` as well as other persons originally in s remain in t''. To formalize this property we need order ($\preceq$) between models to represent how "updated" the model is [30]. In the above example, we observe that $t' \preceq t''$.

"WPutGet"(Weak PutGet) is another weaker variant of *Consistency*("PutGet"), satisfied by GRoundTram, that allows view side-effects but guarantees that another backward transformation will result in the source previously produced by t'. It is characterized using *get* and *put* by $put(s, get(put(s, t'))) = put(s, t')$. In the above scenario, if the backward transformation with t'' results in s' again, then the system satisfies "WPutGet". WPutGet corresponds to bflnv in Diskin et al. [18]. We can put WPutGet into Diskin at al.'s context, by taking user's modification sending the original target $t = get(s)$ produced by transformation from original source s, to updated target t' as b, and the hypothetical modification sending the original target t to the one produced by backward transformation $put(s, t')$ on user updated target t' relative to the original source s immediately followed by another forward transformation, i.e, $get(put(s, t'))$ as b'. Suppose $get(s) = t \Leftrightarrow (s, t) \in r$. WPutGet says $put(s, t') = put(s, get(put(s, t')))$, meaning that t' and $get(put(s, t'))$ have the same effect on the source. It corresponds to update equivalence between b and b', i.e., $b \sim_r b'$, thus bflnv is implied. Diskin et al. [18] also considers the law of weak undoability.

A relaxed notion of composability by taking monotonicity into account (deletion followed by deletion, or insertion by insertion, rather than deletion followed by insertion) that holds in most systems was described by Johnson and Rosebrugh [35]. An accurate definition of delta lenses [18] with monotonic composability can be found in [15].

We plan to keep the feature model updated by considering for inclusion new well-behavedness properties based on their community acceptance.

4 Considered Approaches

This section summarizes the existing approaches we are classifying by the means of our feature model. We especially gave preference to approaches that have a concrete implementation publicly available at the date of

June 2012[6]. The approaches we review include (in alphabetical order):

- **AMW2ATL** [14] is an approach based on bidirectional specification of the correspondence between two models by means of a third model called *weaving*. From this model, a couple of forward and backward ATL transformations is generated. The code is maintained at `http://www.eclipse.org/gmt/amw/`.
- **ATL-Inversion** is a higher-order transformation to generate an ATL backward transformation, given the forward one. While only a very basic prototype is available, able to invert only trivial transformations, we include the approach our list as an instance of higher-order inversion. The prototype can be found at `http://www.emn.fr/z-info/atlanmod/index.php/ATL_Inversion`.
- **GRoundTram** [26] is a graph roundtrip transformation system for models. It is based on bidirectional interpretation of graph query language UnQL [8]. Implementation is available at `http://www.biglab.org/`.
- Bijective **BOTL** [6,7,44] is a restriction to the bijective case of the bidirectional object-oriented transformation language by Frank Marschall et al. Implementation is available at `http://sourceforge.net/projects/botl/`.
- **Boomerang** [5] is an implementation of lenses [20], a state-based, linguistic approach to build complex bidirectional transformations by combining small ones. The approach provides comprehensive discussion about well-behavedness. Implementation is available at `http://www.seas.upenn.edu/~harmony/`
- **Information-Preserving TGG** [40] is based on Triple Graph Grammars [53]. In this paper, we only consider relatively well-studied bijective cases [19] [7] [8]. Once bijectivity is imposed, semantic properties become rather well-behaved. Implementation of TGG is available at `http://www.moflon.org`.
- **Vdl** [45] is based on the constant-complement approach: information discarded during the transformation is saved as a *complement* and restored dur-

[6] We did not include commercial tools such as Microsoft BizTalk Server [48] and tools from ERP frameworks [33] for transforming structured data. Although these tools may belong to technical spaces other than the ones we have considered, our classification approach may also be applied.

[7] In pure TGG, there must be a bijection between source and target patterns [25]. There are literature on an implementation of a variant of TGG that explicitly address non-bijective usages [34], but formal bidirectional properties were not the scope of the paper.

[8] It *is* possible to use TGG for specifying bidirectional transformations with non-bijective consistency relations [54].

ing backward transformation. An implementation is available at `http://www-kb.is.s.u-tokyo.ac.jp/~kztk/b18n2/`.

- **QVT-Relations (QVT-R)** is the part of the threefold OMG's standard Query View Transformation [50] that allows for the specification of bidirectional relations among models and provides their enforcement. Several implementations are available, with slight differences in semantics, as shown in literature [22]. In our analysis we refer to implementation provided by Medini QVT at `http://projects.ikv.de/qvt`.
- **SyncATL [60]** is based on extending the ATL virtual machine to synchronize source and target artifacts that are related by an ATL transformation. In [60] operation-based well behavedness is extensively discussed. Implementation is available at `http://sei.pku.edu.cn/~xiongyf04/modelSynchronization.html`.
- **TCS** [36] is a model-to-text and text-to-model transformation system, that generates from a bidirectional specification in the TCS language an ANTLR grammar, and performs parsing/pretty-printing into/from EMF models. The tool is available at `http://www.eclipse.org/gmt/tcs/`.
- **Voigtländer**'s approach [57] is based on semantic bidirectionalization, an approach where the syntax of the program is not necessary to conduct backward transformation. Building a table collecting extensional behavior (treating the program as black box) is sufficient to backward transformation. Implementation is available from `http://www-ps.iai.uni-bonn.de/cgi-bin/bff.cgi`
- **X** [29,30] supports synchronization between structured documents and their view for editing. Implementation is available at `http://takeichi.ipl-lab.org/~scm/sw/Inv.tar.gz`).
- **Deterministic Incremental TGG** [21] proposes an incremental execution algorithm for model synchronization based on Triple Graph Grammars [53]. Formal properties of the tool, including conditions on guaranteeing deterministic (functional) behavior, has been formulated in [24]. An implementation is provided as a plug-in of the Fujaba toolsuite (version 4) and available at `http://www.fujaba.de/projects/triple-graph-grammars.html`.

Our list of approaches includes both languages that have been designed for bidirectionality, and languages that do not naturally support bidirectional transformation and have been adapted to it. Literature presents also further alternative methods to implement bidirectional computation, that are not included in this paper because, while a transformation framework could be based on them, currently no such attempt

has been made. For instance, reversible computation frameworks [61] like Janus[9] build computation by reversible primitives like `x := x + 1` (its inverse is `x := x - 1`).

We exclude from our analysis the approaches that, instead of directly transforming artifacts, translate the transformation definition into a constraint solving problem and then let a solver compute the resulting artifacts. In the MDE community some effort has been done in representing a model transformation by a set of constraints, forming a so called *transformation model* [9]. In this setup the inverse transformation could be theoretically performed by solving a constraint satisfaction problem. Transformation frameworks based on this method, have been recently proposed [43, 23, 51, 41, 1].

5 Discussion

Table 2 presents in tabular format the set of feature equations derived from manually applying the feature model to the tools under study[10]. The table can be manually or automatically analyzed to study the current design space of bidirectional transformation. Table 3 summarizes the semantics of the features.

[9] Implementation is available at `http://topps.diku.dk/pirc/janus/intro.html`.

[10] Feature equations are validated against the presented feature model using the tool FeatureIDE [38].

Features, or combinations of features, that have no known implementation can be considered unexplored design space. Clearly not every arbitrary combination of features can be selected. The feature model itself expresses exclusiveness of sibling features by an alternative group. Features incompatible with each other are expressed by constraints as exemplified in the previous section. For example, some choices in other parts of the feature model may prevent to achieve some of the well-behavedness properties. In our feature model we formalize some of these constraints but we do not mean to be exhaustive.

5.1 Unexplored Features and Future Research Proposal

For some of the features, Table 2 does not include any implementing tool. These features are well-known in other transformation domains (e.g., databases) but are apparently under-studied in model transformation. The unexplored features are:

Interactive. All systems in our list perform bidirectional transformation in a completely automatic way. In some of the systems, artifact editors may provide limited information about reaching an incorrect state (e.g., the text editor in TCS performs static checks and highlight non-parsable

texts). However none of the approaches implements an interactive, human-in-the-loop, transformation system[11]. In such a system users could be asked for additional information when the transformation engine can't associate the target state with a unique source state. Database community has important previous work in this respect, like [39].

Change translation. Several systems are capable of back-propagating to the source model the sequence of updates made to the target model. However this back-propagation is never performed as an explicit translation of the target-updates to a sequence of source-update operations. A translational approach to back-propagation would generate a source-update artifact, that could be for instance useful when transformation is performed over the network, with attention to bandwidth usage. This direction towards translational approach is apparently emerging, and theoretical work like [28] and [18] would facilitate this direction.

[11] Becker et al. [4] reported a TGG implementation with a specific focus on user interactions, but we could not find implementation that is publicly available currently.

Table 2 Characterization of systems using proposed features

Terminal Feature	ATL-AMW	ATL-Inversion	BiG	Bijective BOTL	Boomerang	InfoPreservingTGG	Vdl	QVTR	SyncATL	TCS	Voigtlander 2009	X	Incr. TGG
Text					x					x			
Graph			x										
XML							x				x	x	
MDE	x	x		x		x		x	x	x			x
ForwardFunctional		x	x	x	x	x	x		x	x	x	x	
BackwardFunctional		x		x		x							
TotalSource		x	x	x	x	x	x	x	x	x	x	x	
TotalTarget		x		x	x	x		x	x				
BidirectionalDefinition	x			x		x		x					x
UnidirectionalDefinition		x	x		x		x		x	x	x	x	
TuringComplete								x	x		x	x	
NotTuringComplete	x	x	x	x	x	x	x			x	x		x
OperationBased									x			x	
StateBased	x	x	x	x	x	x	x	x		x	x		x
Live										x		x	
Offline	x	x	x	x	x	x	x	x	x	x	x		x
Complete		x			x	x		x					
StaticalChangeCheck							x						
DynamicalChangeCheck	x		x	x					x	x	x		x
AdditionSupport			x	x			x				x	x	x
RemovalSupport			x	x			x		x		x	x	x
ModificationSupport			x	x			x		x		x	x	x
Check								x					
Enforcement	x	x	x	x	x	x	x	x	x	x	x	x	x
AutomaticExecution	x	x	x	x	x	x	x	x	x	x	x	x	x
InteractiveExecution													
Full	x	x		x		x				x			
ChangeApplication			x		x		x	x	x		x	x	x
ChangeTranslation													
SemanticApproach											x		
SyntacticApproach	x	x	x	x	x	x	x	x	x	x		x	x
ExplicitBackward	x	x					x				x		
ImplicitBackward			x	x	x			x	x	x		x	x
ImplicitTraces					x		x		x			x	
ExplicitTraces	x		x			x		x			x		x
TraceInModels			x				x		x			x	
TraceOutOfModels	x				x	x		x			x		x
StandardTraces	x												
ToolSpecificTraces			x		x	x	x	x	x		x	x	x
ComplementInModels													
ComplementOutOfModels			x				x				x		
Acceptability	x	x	x	x	x	x	x		x	x	x		x
Consistency	x			x	x	x	x				x		x
Composability	x	x		x		x	x		x	x	x		
Preservation									x				
Propagation									x				
Undoability	x	x			x		x			x	x	x	
FunctionalBehavior	x	x	x	x	x	x	x	x	x	x	x	x	x

Table 3 Semantics of features

Terminal Feature	L1 feature	L2 feature	L3 feature	L4 feature	Semantics
Text	TechnicalSp.				The tool natively supports textual artifacts
Graph	TechnicalSp.				The tool natively supports graph artifacts
XML	TechnicalSp.				The tool natively supports XML artifacts
MDE	TechnicalSp.				The tool natively supports MDE artifacts
ForwardFunctional	Corresp.	CorrRelation	Functional		The tool guarantees or requires consistency relations that are functional in the forward direction
BackwardFunctional	Corresp.	CorrRelation	Functional		The tool guarantees or requires consistency relations that are functional in the backward direction
TotalSource	Corresp.	CorrRelation	Coverage		The tool guarantees or requires consistency relations that cover the full source domain
TotalTarget	Corresp.	CorrRelation	Coverage		The tool guarantees or requires consistency relations that cover the full target domain
BidirectionalDefinition	Corresp.	DefDir.			The transformation definition language is bidirectional
UnidirectionalDefinition	Corresp.	DefDir.			The transformation definition language is unidirectional
TuringComplete	Corresp.	Express.			The transformation definition language is turing-complete
NotTuringComplete	Corresp.	Express.			The transformation definition language is not turing-complete
OperationBased	Changes	ChangeDef.	ChangeRep.		Changes are represented as sequences of operations
StateBased	Changes	ChangeDef.	ChangeRep.		Changes are represented as sequences of states
Live	Changes	ChangeDef.	ChangeInput		Changes can be recorded by a specific running editor
Offline	Changes	ChangeDef.	ChangeInput		Changes can be provided as an input artifact
Complete	Changes	ChangeSupp.	Complete		The tool is able to transform back any change on the src or tgt artifact within the consistency relation
StaticalChangeCheck	Changes	ChangeSupp.	ChangeChk	ChangeChk	The tool does not allow all changes and detects invalid changes without running the trans.
DynamicalChangeCheck	Changes	ChangeSupp.	ChangeChk	ChangeChk	The tool does not allow all changes and detects invalid changes by running the trans.
AdditionSupport	Changes	ChangeSupp.	OperSupport	OperSupport	The tool only allows some change operations and addition of new info. to the artifacts is supported
RemovalSupport	Changes	ChangeSupp.	OperSupport	OperSupport	The tool only allows some change operations and removal of info. from the artifacts is supported
ModificationSupport	Changes	ChangeSupp.	OperSupport	OperSupport	The tool only allows some change operations and modification of info. from the artifacts is supported
Check	Execution	Semantics			The tool can run in a check-only mode to check for consistency of source and target
Enforcement	Execution	Semantics			The tool can enforce consistency between source and target
AutomaticExecution	Execution	ExecAuto.			The tool execution can run in a completely automatic mode
InteractiveExecution	Execution	ExecAuto.			The tool execution can ask users for interaction during transformation
Full	Execution	Application			The tool always construct a new source/target artifact without taking its history into account
ChangeApplication	Execution	Application	Incremental		The tool propagates changes to the other side
ChangeTranslation	Execution	Application	Incremental		The tool explicitely translates changes on one side to changes on the other
SemanticApproach	Execution	Approach			The tool uses the transformation as a black-box
SyntacticApproach	Execution	Approach			The tool has access to the transformation code
ExplicitBackward	Execution	BwdTrans.			The tool generates an explicit unidirectional definition of the backward transformation
ImplicitBackward	Execution	BwdTrans.			The tool does not generate an explicit unidirectional definition of the backward transformation
ImplicitTraces	Execution	SpportInfo.	Traces	TraceExpl.	The tool makes use of trace information for bidirectional transformation but no explicit inter-artifact reference is stored
ExplicitTraces	Execution	SpportInfo.	Traces	TraceExpl.	An explicit mechanism for inter-artifact reference is used for traces
TraceInModels	Execution	SpportInfo.	Traces	TraceStorage	Traces are stored in the target (and/or source) artifact and delivered with it
TraceOutOfModels	Execution	SpportInfo.	Traces	TraceStorage	Traces are kept separate from artifacts
StandardTraces	Execution	SpportInfo.	Traces	TraceCompat.	Traces are stored in a well-documented format for external reuse
ToolSpecificTraces	Execution	SpportInfo.	Traces	TraceCompat.	Traces are only meant to be used internally
ComplementInModels	Execution	SpportInfo.	Complement		All complement information is stored in the target (and/or source) artifact and delivered with it
ComplementOutOfModels	Execution	SpportInfo.	Complement		Complement information is kept separate from artifacts
Acceptability	Execution	WellBeh.			No modification on the target leads to no modification on the source For synchronizer, if no update operation is applied to both src and tgt. then both of them remains unchanged after trans.
Consistency	Execution	WellBeh.			Another fwd trans. yields the same target as the previous tgt fed to the backward trans.
Composability	Execution	WellBeh.			Successive bwd trans. for successive modifications are equivalent to bwd trans. for combined modification
Preservation	Execution	WellBeh.			The effects of modifications applied for both source and target are preserved after transformation
Propagation	Execution	WellBeh.			Another fwd trans. sends src to tgt that differ only in modification non-propagable to source
Undoability	Execution	WellBeh.			Original source can be restored by using the original target
FunctionalBehavior	Execution	WellBeh.			Tool execution is deterministic

Complement in Models. While several approaches store complement information from the source model, to deal with information-losing forward transformations, every tool handles this complement as a separate artifact. In some cases it would be useful instead to embed the complement information in the target model, to be sure that this information will be always delivered with it. This way third parties would be always able to reconstruct the source model from the target. An efficient way of integrating complement information in the target model has yet to be studied.

Non-Functional Behavior. All the approaches in our list exhibits functional behavior. In bidirectional transformations with non functional behavior it might be difficult to reason about properties like well-behavedness. However it might be useful to explore this direction to deal with systems that inherently include non-functional behavior. Related to the Interactive feature discussed above, users may be required to help the system in choosing from multiple candidate artifacts the system presents as a result of such non-functional behavior.

Other features have been implemented only in combination with a set of other features. These features seem to be particularly suitable for experimentation in different setups. For instance the Statical Change Check feature is implemented only by VDL, that is a state-based system. Hence, the problem of having a statical check of the validity of target updates in operation-based systems remains open. As another example, having standard trace formats according to feature Standard Traces is particularly important for system interoperability. For example, [59] can incorporate an existing state-based bidirectional transformation system as a black box and make it incremental. This kind of integration would be substantially facilitated by standard formats, but currently only AMW-ATL seems to make an effort on trace standardization. Similar features that could inspire further research include: Check, Semantic Approach, Composability, Preservation, Propagation.

A typical way to address unexplored combinations is to introduce approaches that have been successfully used in systems for simpler technical spaces (e.g., trees) to systems for more complex technical spaces (e.g. graphs). Following this principle we are currently in the effort of fulfilling some of the needs highlighted by our analysis, by developing a novel model transformation tool based on ATL. Some of the notable features of our proposal will be Interactive, Statical Change Check, Live, Standard Traces, thus covering some of the unexplored feature combinations.

6 Related Work

Czarnecki and Helsen in [12] present a feature model for model transformation approaches, where they mention directionality, but do not focus on it. The feature model we introduce in this paper can be seen as complementary to [12] in the sense that: 1) the two feature models do not overlap on any common features and 2) all our features refer implicitly to a bidirectional engine, hence they depend on the *multidirectional* feature of the model in [12].

Similarly Mens and Van Gorp in [47] analyze the whole field of model transformations and Taentzer et al. [56] extend the comparison to graph transformations. In both cases the systems under study are not necessarily bidirectional.

Rose et al. in [52] propose a feature model on model-to-text (M2T) transformation approaches based on domain analysis, to facilitate user's selection of existing languages and promote discussions among M2T language developers. The paper and ours generally share the same goal but the paper focuses on M2T transformation domain but bidirectionality itself is not the focus. We focus on bidirectionality in MDE.

Specific surveys about bidirectionality are provided by Stevens in [55] and by Antkiewicz and Czarnecki in [2]. An overview over bidirectional problems is given in [11], which summarizes the discussion in the GRACE workshop on bidirectional model transformations, and later in [31,32], which summarizes the discussion in the Dagstuhl seminar on bidirectional transformations (BX). Recently, a broad classification of bidirectional transformations in purely semantic terms appeared in [16]. With respect to these works, our paper provides a structured view as a feature model and a complete characterization of the design space.

7 Conclusion and Future Work

The feature model presented in this paper highlights the space of possible choices in the design of a bidirectional model transformation approach. We have illustrated the feature model by discussing some of the main existing tools. Finally, we have pointed out some areas where further work is needed. We hope our feature model will be a valuable contribution to the bidirectional transformation community by clarifying the design space, facilitating reasoning about current approaches, and inspiring future research.

Some of the current terminal features (features at the leaves) could be further refined into sub-features. For instance *Enforcement* Semantics could be refined in terms of how the conflicts between changes are handled, or in terms of rule-application strategies, e.g. whether backtracking is used, in case of rule-based

languages. Several more well-behavedness properties are under evaluation for inclusion.

We plan to keep the list of tools updated, including some approaches that are now in embryonic state. GRoundTram has only recently been provided with a statical check of update validity [49] but no correspondent implementation has been made public yet. Boomerang does not yet include some extensions that have been presented in recent work: Symmetric Lenses [27] treat forward and backward transformations symmetrically and Edit Lenses [28] explicitly deal with edit operations. We will consider also recent combinations of syntactic and semantic approaches, like [58]. These new lines of research suggest that different approaches can be integrated in a solid theoretical basis.

As part of our research agenda, we also plan to involve as much as possible the BX community to our categorization effort. We provide a public website[12] to illustrate the current version of the feature model and to gather suggestions from the community. We also publish on the website the list of analyzed tools, and their feature set, to obtain validation and additional inputs from the experts of the tools.

[12] `http://www.emn.fr/z-info/atlanmod/index.php/Bidirectional_Transformations`

Finally it would be useful to provide *performance* information for the different approaches. For instance, incremental approaches expect better performances over non-incremental ones, since an incremental tool does not have to recompute the whole artifacts every time, provided that updated portions are relatively small compared to the whole artifacts so that the cost of bookkeeping the updates remains moderate. Complexity analysis could help in providing quantitative estimations for the performance benefits.

Acknowledgements We thank the anonymous reviewers for their detailed and helpful feedback on our paper, and Frédéric Jouault, Keisuke Nakano, Kazutaka Matsuda, Hiroyuki Kato, Kazuhiro Inaba for their valuable comments and suggestions on earlier versions of this research. We would also like to thank Fabian Büttner for comments on earlier version of this paper. We also would like to thank participants of BIRS workshop on Bi-directional transformations (BX) - Theory and Applications Across Disciplines (13w5115), especially Alcino Cunha and the other members of the working group on Benchmarking Bidirectional Transformations, for inspiring discussions. This research has been supported by the EU FP7 MONDO project and the MOU grant from National Institute of Informatics.

References

1. Anjorin, A., Varró, G., Schürr, A.: Complex attribute manipulation in tggs with constraint-based programming techniques. ECEASST **49** (2012). First International Workshop on Bidirectional Transformations (BX 2012)

2. Antkiewicz, M., Czarnecki, K.: Design space of heterogeneous synchronization. Generative and Transformational Techniques in Software Engineering II **pages**, 3–46 (2007). URL `http://www.springerlink.com/index/j166jvx06r50k776.pdf`
3. Bancilhon, F., Spyratos, N.: Update semantics of relational views. ACM Trans. Database Syst. **6**(4), 557–575 (1981)
4. Becker, S.M., Herold, S., Lohmann, S., Westfechtel, B.: A graph-based algorithm for consistency maintenance in incremental and interactive integration tools. Software and System Modeling **6**(3), 287–315 (2007)
5. Bohannon, A., Foster, J.N., Pierce, B.C., Pilkiewicz, A., Schmitt, A.: Boomerang: resourceful lenses for string data. In: POPL '08: ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages, pp. 407–419 (2008)
6. Braun, P., Marschall, F.: BOTL : The bidirectional object oriented transformation language. Tech. Rep. TUM-I0307, Technische Universität München (2003)
7. Braun, P., Marschall, F.: Transforming Object Oriented Models with BOTL. Electronic Notes in Theoretical Computer Science **72**(3), 103 – 117 (2003). DOI DOI: 10.1016/S1571-0661(04)80615-7. GT-VMT'2002, Graph Transformation and Visual Modeling Techniques (First International Conference on Graph Transformation)
8. Buneman, P., Fernandez, M.F., Suciu, D.: UnQL: a query language and algebra for semistructured data based on structural recursion. VLDB J. **9**(1), 76–110 (2000)
9. Büttner, F., Cabot, J., Gogolla, M.: On validation of ATL transformation rules by transformation models. Proceedings of the 8th International Workshop on Model-Driven Engineering, Verification and Validation - MoDeVVa p. 1 (2011). DOI 10.1145/2095654.2095666. URL `http://dl.acm.org/citation.cfm?doid=2095654.2095666`
10. Bzivin, J., Kurtev, I.: Model-based technology integration with the technical space concept. In: In: Proceedings of the Metainformatics Symposium, Springer-Verlag. Springer-Verlag (2005)
11. Czarnecki, K., Foster, J.N., Hu, Z., Lämmel, R., Schürr, A., Terwilliger, J.F.: Bidirectional Transformations: A Cross-Discipline Perspective. In: Theory and practice of model transformations: second international conference, ICMT 2009, Zürich, Switzerland, June 29-30, 2009: proceedings, vol. 5563, pp. 260–283. Springer-Verlag New York Inc (2009). DOI 10.1007/978-3-642-02408-5
12. Czarnecki, K., Helsen, S.: Feature-based survey of model transformation approaches. IBM Systems Journal **45**(3), 621–645 (2006). DOI 10.1147/sj.453.0621
13. Dayal, U., Bernstein, P.A.: On the correct translation of update operations on relational views. ACM Trans. Database Syst. **7**(3), 381–416 (1982)
14. Del Fabro, M., Bézivin, J., Jouault, F., Breton, E., Gueltas, G.: AMW: a generic model weaver. In: Proc. of the 1ères Journées sur l'Ingénierie Dirigée par les Modèles (2005). URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.108.1294`
15. Diskin, Z., Maibaum, T.S.E.: Category theory and model-driven engineering: From formal semantics to design patterns and beyond. In: U. Golas, T. Soboll (eds.) ACCAT, *EPTCS*, vol. 93, pp. 1–21 (2012)
16. Diskin, Z., Wider, A., Gholizadeh, H., Czarnecki, K.: Towards a rational taxonomy for increasingly symmetric model synchronization. In: D. Di Ruscio, D. Varró (eds.) Theory and Practice of Model Transformations, *Lecture Notes in Computer Science*, vol. 8568, pp. 57–73. Springer International Publishing (2014). DOI

10.1007/978-3-319-08789-4_5. URL http://dx.doi.org/10.1007/978-3-319-08789-4_5

17. Diskin, Z., Xiong, Y., Czarnecki, K.: From state- to delta-based bidirectional model transformations: the asymmetric case. Journal of Object Technology **10**, 6: 1–25 (2011)
18. Diskin, Z., Xiong, Y., Czarnecki, K., Ehrig, H., Hermann, F., Orejas, F.: From state- to delta-based bidirectional model transformations: The symmetric case. In: J. Whittle, T. Clark, T. Kühne (eds.) Model Driven Engineering Languages and Systems, *Lecture Notes in Computer Science*, vol. 6981, pp. 304–318. Springer Berlin Heidelberg (2011). DOI 10.1007/978-3-642-24485-8_22. URL http://dx.doi.org/10.1007/978-3-642-24485-8_22
19. Ehrig, H., Ehrig, K., Ermel, C., Hermann, F., Taentzer, G.: Information preserving bidirectional model transformations. Fundamental Approaches to Software Engineering pp. 72–86 (2007). URL http://www.springerlink.com/index/D3298714G2112360.pdf
20. Foster, J.N., Greenwald, M.B., Moore, J.T., Pierce, B.C., Schmitt, A.: Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem. ACM Trans. Program. Lang. Syst. **29** (2007). DOI 10.1145/1232420.1232424. URL http://doi.acm.org/10.1145/1232420.1232424
21. Giese, H., Wagner, R.: From model transformation to incremental bidirectional model synchronization. Software & Systems Modeling **8**(1), 21–43 (2008). DOI 10.1007/s10270-008-0089-9. URL http://www.springerlink.com/index/10.1007/s10270-008-0089-9
22. Greenyer, J., Kindler, E.: Comparing relational model transformation technologies: implementing Query/View/Transformation with Triple Graph Grammars. Software & Systems Modeling **9**(1), 21–46 (2010)
23. Guerra, E., Lara, J., Orejas, F.: Pattern-based model-to-model transformation: Handling attribute conditions. In: Proceedings of the 2nd International Conference on Theory and Practice of Model Transformations, ICMT '09, pp. 83–99. Springer-Verlag, Berlin, Heidelberg (2009). DOI 10.1007/978-3-642-02408-5_7. URL http://dx.doi.org/10.1007/978-3-642-02408-5_7
24. Hermann, F., Ehrig, H., Orejas, F., Czarnecki, K., Diskin, Z., Xiong, Y.: Correctness of Model Synchronization Based on Triple Graph Grammars. In: Proceedings of the 14th international conference on Model driven engineering languages and systems, MODELS'11, pp. 668–682. Springer-Verlag (2011). DOI 10.1007/978-3-642-24485-8_49. URL http://www.springerlink.com/content/t178m5644w6588t7/
25. Hettel, T., Lawley, M., Raymond, K.: Model synchronisation: Definitions for round-trip engineering. In: Proceedings of the 1st international conference on Theory and Practice of Model Transformations, ICMT '08, pp. 31–45. Springer-Verlag, Berlin, Heidelberg (2008)
26. Hidaka, S., Hu, Z., Inaba, K., Kato, H., Matsuda, K., Nakano, K.: Bidirectionalizing graph transformations. In: Proceedings of the 15th ACM SIGPLAN international conference on Functional programming, pp. 205–216. ACM (2010). URL http://portal.acm.org/citation.cfm?id=1863573
27. Hofmann, M., Pierce, B., Wagner, D.: Symmetric lenses. In: Proceedings of the 38th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages, POPL '11, pp. 371–384. ACM, New York, NY, USA (2011). DOI 10.1145/1926385.1926428. URL http://doi.acm.org/10.1145/1926385.1926428
28. Hofmann, M., Pierce, B., Wagner, D.: Edit lenses. In: Proceedings of the 39th annual ACM SIGPLAN-

SIGACT symposium on Principles of programming languages, POPL '12, pp. 495–508. ACM, New York, NY, USA (2012). DOI 10.1145/2103656.2103715. URL http://doi.acm.org/10.1145/2103656.2103715

29. Hu, Z., Mu, S.C., Takeichi, M.: A programmable editor for developing structured documents based on bidirectional transformations. In: PEPM '04: Proceedings of the 2004 ACM SIGPLAN Symposium on Partial Evaluation and Semantics-based Program Manipulation, pp. 178–189. ACM Press, New York, NY, USA (2004)
30. Hu, Z., Mu, S.C., Takeichi, M.: A programmable editor for developing structured documents based on bidirectional transformations. Higher-Order and Symbolic Computation **21**(1-2), 89–118 (2008)
31. Hu, Z., Schürr, A., Stevens, P., Terwilliger, J.F.: Bidirectional transformation "bx" (dagstuhl seminar 11031). Dagstuhl Reports **1**(1), 42–67 (2011)
32. Hu, Z., Schürr, A., Stevens, P., Terwilliger, J.F.: Dagstuhl seminar on bidirectional transformations (bx). SIGMOD Record **40**(1), 35–39 (2011)
33. Jacobs, F.R., Jr., F.T.W.: Enterprise resource planning (ERP) – A brief history. Journal of Operations Management **25**(2), 357 – 363 (2007)
34. Jakob, J., Königs, A., Schürr, A.: Non-materialized Model View Specification with Triple Graph Grammars. In: Proceedings of the Third international conference on Graph Transformations, ICGT'06, pp. 321–335. Springer-Verlag, Berlin, Heidelberg (2006). URL http://dx.doi.org/10.1007/11841883_23
35. Johnson, M., Rosebrugh, R.D.: Lens put-put laws: monotonic and mixed. ECEASST **49** (2012). First International Workshop on Bidirectional Transformations (BX 2012)
36. Jouault, F., Bézivin, J., Kurtev, I.: TCS:: a DSL for the specification of textual concrete syntaxes in model engineering. In: Proceedings of the 5th international conference on Generative programming and component engineering, p. 254. ACM, New York, New York, USA (2006). DOI 10.1145/1173706.1173744. URL http://portal.acm.org/citation.cfm?id=1173706.1173744
37. Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, A.: Feature-Oriented Domain Analysis (FODA) Feasibility Study. Tech. rep., DTIC Document (1990)
38. Kästner, C., Thüm, T., Saake, G., Feigenspan, J., Leich, T., Wielgorz, F., Apel, S.: FeatureIDE: Tool Framework for Feature-Oriented Software Development. In: Proceedings of the 31th International Conference on Software Engineering (ICSE), pp. 611–614. IEEE Computer Society (2009). Formal Demonstration paper
39. Keller, A.M.: Choosing a view update translator by dialog at view definition time. In: Proceedings of the 12th International Conference on Very Large Data Bases, VLDB '86, pp. 467–474. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1986). URL http://www.sigmod.org/publications/dblp/db/conf/vldb/Keller86.html
40. Königs, A.: Model Transformation with Triple Graph Grammars. In: Model Transformations in Practice Satellite Workshop of MODELS (2005). URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.113.2894&rep=rep1&type=pdf
41. Lambers, L., Hildebrandt, S., Giese, H., Orejas, F.: Attribute Handling for Bidirectional Model Transformations: The Triple Graph Grammar Case. ECEASST **49** (2012). URL http://journal.ub.tu-berlin.de/eceasst/article/view/706. First International Workshop on Bidirectional Transformations (BX 2012)

42. Lämmel, R.: Coupled Software Transformations (Extended Abstract). In: First International Workshop on Software Evolution Transformations (SET 2004), pp. 31–35 (2004). URL http://post.queensu.ca/~zouy/files/set-2004.pdf#page=38

43. Macedo, N., Cunha, A.: Implementing QVT-R Bidirectional Model Transformations Using Alloy. In: Proceedings of the 16th International Conference on Fundamental Approaches to Software Engineering, FASE'13, pp. 297–311. Springer-Verlag, Berlin, Heidelberg (2013). DOI 10.1007/978-3-642-37057-1_22. URL http://dx.doi.org/10.1007/978-3-642-37057-1_22

44. Marschall, F., Braun, P.: Model transformations for the mda with BOTL. In: A. Rensink (ed.) Model Driven Architecture: Foundations and Applications, *CTIT Technical Report*, vol. TR-CTIT-03-27, pp. 25–36. University of Twente (2003)

45. Matsuda, K., Hu, Z., Nakano, K., Hamana, M., Takeichi, M.: Bidirectionalization transformation based on automatic derivation of view complement functions. In: ICFP 2007, pp. 47–58 (2007)

46. Meertens, L.: Designing constraint maintainers for user interaction (1998). http://www.kestrel.edu/home/people/meertens/

47. Mens, T., Van Gorp, P.: A taxonomy of model transformation. Electronic Notes in Theoretical Computer Science **152**, 125–142 (2006). URL http://linkinghub.elsevier.com/retrieve/pii/S1571066106001435

48. Microsoft: Microsoft BizTalk Server. http://www.microsoft.com/biztalk/

49. Nakano, K., Hidaka, S., Hu, Z., Inaba, K., Kato, H.: Simulation-based graph schema for view updatability checking of graph queries. Tech. Rep. GRACE-TR11-01, GRACE Center, National Institute of Informatics (2011)

50. Object Management Group: Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification. Version 1.0 (2008). URL http://www.omg.org/spec/QVT/1.0/

51. Petter, A., Behring, A., Mühlhäuser, M.: Solving constraints in model transformations. In: R. Paige (ed.) Theory and Practice of Model Transformations, *Lecture Notes in Computer Science*, vol. 5563, pp. 132–147. Springer Berlin Heidelberg (2009). DOI 10.1007/978-3-642-02408-5_10. URL http://dx.doi.org/10.1007/978-3-642-02408-5_10

52. Rose, L.M., Matragkas, N., Kolovos, D.S., Paige, R.F.: A feature model for model-to-text transformation languages. In: Proceedings of the 4th International Workshop on Modeling in Software Engineering (MiSE), pp. 57–63 (2012)

53. Schürr, A.: Specification of Graph Translators with Triple Graph Grammars. In: E.W. Mayr, G. Schmidt, G. Tinhofer (eds.) Graph-Theoretic Concepts in Computer Science, 20th International Workshop, WG '94, Herrsching, Germany, *Lecture Notes in Computer Science*, vol. 903, pp. 151–163. Springer (1995)

54. Stevens, P.: A Landscape of Bidirectional Model Transformations. In: Generative and Transformational Techniques in Software Engineering II: International Summer School, GTTSE 2007, *Lecture Notes in Computer Science*, vol. 5235, pp. 408–424. Springer (2008). URL http://www.springerlink.com/index/d27135233t815525.pdf

55. Stevens, P.: Bidirectional model transformations in QVT: semantic issues and open questions. Software and System Modeling **9**(1), 7–20 (2010)

56. Taentzer, G., Ehrig, K., Guerra, E., de Lara, J., Lengyel, L., Levendovszky, T., Prange, U., Varró, D., Varró-Gyapay, S.: Model transformation by graph transformation: A comparative study. In:

Proc. Workshop Model Transformation in Practice, Montego Bay, Jamaica. Citeseer (2005). URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.97.2827&rep=rep1&type=pdf

57. Voigtländer, J.: Bidirectionalization for free! (pearl). In: POPL '09: ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages, pp. 165–176. ACM, New York, NY, USA (2009)
58. Voigtländer, J., Hu, Z., Matsuda, K., Wang, M.: Combining syntactic and semantic bidirectionalization. In: ACM SIGPLAN International Conference on Functional Programming, pp. 181–192. ACM (2010)
59. Wang, M., Gibbons, J., Wu, N.: Incremental updates for efficient bidirectional transformations. In: Proceedings of the 16th ACM SIGPLAN international conference on Functional programming, ICFP '11, pp. 392–403. ACM, New York, NY, USA (2011). DOI 10.1145/2034773.2034825. URL http://doi.acm.org/10.1145/2034773.2034825
60. Xiong, Y., Liu, D., Hu, Z., Zhao, H., Takeichi, M., Mei, H.: Towards automatic model synchronization from model transformations. Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering - ASE '07 p. 164 (2007). DOI 10.1145/1321631.1321657. URL http://portal.acm.org/citation.cfm?doid=1321631.1321657
61. Yokoyama, T., Axelsen, H.B., Glück, R.: Principles of a reversible programming language. In: Proceedings of the 5th conference on Computing frontiers, CF '08, pp. 43–54. ACM, New York, NY, USA (2008). DOI http://doi.acm.org/10.1145/1366230.1366239. URL http://doi.acm.org/10.1145/1366230.1366239