Bidirectional Transformations: A Cross-Discipline Perspective

GRACE meeting notes, state of the art, and outlook

Krzysztof Czarnecki[1],
J. Nathan Foster[2],
Zhenjiang Hu[3],
Ralf Lämmel[4],
Andy Schürr[5], and
James F. Terwilliger[6]

[1] University of Waterloo, Canada
[2] University of Pennsylvania, USA
[3] National Institute of Informatics, Japan
[4] Universität Koblenz-Landau, Germany
[5] Technische Universität Darmstadt, Germany
[6] Microsoft Research, USA

Abstract. The GRACE International Meeting on Bidirectional Transformations was held in December 2008 near Tokyo, Japan. The meeting brought together researchers and practitioners from a variety of subdisciplines of computer science to share research efforts and help create a new community. In this report, we survey the state of the art and summarize the technical presentations delivered at the meeting. We also describe some insights gathered from our discussions and introduce a new effort to establish a benchmark for bidirectional transformations.

1 Introduction

Bidirectional transformations (*bx*) are a mechanism for maintaining the consistency of two (or more) related sources of information. Researchers from many different areas including software engineering, programming languages, databases, and document engineering are actively investigating the use of *bx* to solve a diverse set of problems, for example:

- *Model-Driven Software Development*: to compute and synchronize views of software models [5, 93, 97, 113].
- *Graphical User Interfaces*: to maintain the consistency of a GUI and the underlying application model in the model-view-controller paradigm [79].
- *Visualization With Direct Manipulation*: to visualize abstract data and animate algorithms [100].

- *Relational Databases*: to construct updatable views [10, 16, 27, 59].
- *Data Transformation, Integration, and Exchange*: to map data across paradigms, merge it from multiple sources, and exchange it between sources [44, 47, 49, 57, 68, 82, 86].
- *Data Synchronizers*: to bridge the gap between replicas in different formats [17, 41, 58].
- *Macro Systems*: to give feedback to the programmer (e.g., from a type checker or a debugger) in terms of the original program elements prior to macro expansion [23, 62].
- *Domain-Specific Languages (DSLs)*: to translate between run-time values of the object language (the DSL) and the corresponding values of the host language in embedded interpreters [13, 88].
- *Structure Editors*: to provide convenient interfaces for editing complicated data sources [54, 55, 75].
- *Serializers:* to mediate between external data (binary or sequential data representations on the wire or the file system) and structured objects in memory [32, 39].

Although researchers are actively working on bidirectional transformations in several communities, so far there has been very little cross-discipline interaction and cooperation. The purpose of the GRACE International Meeting on Bidirectional Transformations (GRACE-BX), held in December 2008 near Tokyo, was to bring together international elites, promising young researchers, and leading practitioners to share problems, discuss solutions, and open a dialogue towards understanding the common underpinnings of *bx* in all these areas.

This report summarizes the main results of the GRACE-BX meeting and records the technical presentations that were delivered there. Section 2 surveys the essential ideas and important papers in key disciplines of *bx* and describes the specific work presented at the meeting. Section 3 identifies some of the common themes and cross-links that emerged during the meeting. Section 4 describes the beginning of an effort to develop a benchmark suite of canonical bidirectional transformations for comparing the capabilities and characteristics of different *bx* approaches, technologies, scenarios and implementations thereof. Section 5 concludes this report.

All of the material presented at the meeting is available at the GRACE-BX website.[7]

2 Bidirectional Transformation Subcommunities

Bidirectional transformations are being used in a variety of disciplines including programming languages, database management systems, model-driven engineering, and graph query and transformation systems. The precise details of how *bx*

[7] `http://grace.gsdlab.org/`

work in each area vary greatly between disciplines, but there is general agreement that a *bx* between two sources of information A and B (e.g., a database source and view, two different software models or graph structures, or the input and output of a program) comprises a pair of unidirectional transformations: one from A to B and another from B back to A.[8] In many cases, the flow of data from A to B dominates the flow of data from B to A—i.e., A acts as a master for B. In these cases, A is called the input (source, master) and B is called the output (target, slave) of the *bx*. Consequently, the transformation from A to B is often called the forward transformation and the transformation from B to A is called the backward or reverse transformation.

In current practice, *bx* are usually implemented by programming (or specifying) compatible forward and backward transformations in a *unidirectional* transformation language. Recently, however, an exciting new approach has started to be used in which *bx* are implemented in *bidirectional* transformation languages—formalisms where every program (or specification) describes a forward and a backward transformation simultaneously. A major advantage of this approach is that the compatibility of the transformations can be guaranteed by construction.

In the rest of this section, we give a brief overview of uses of *bx* in each community and describe the presentations that were delivered at the GRACE-BX meeting. We start with the area of bidirectional and reversible programming languages, and continue with databases and data management, model-driven engineering, graph query and graph transformation systems. We conclude this section by describing presentations on novel applications of *bx* including coupled grammar transformations and a system for computing updatable views of configuration files.

2.1 Programming Languages

Recently, a number of researchers have investigated programming languages where programs can be run both forwards and backwards. Broadly speaking, these languages can be classified according to two features: the *semantic laws* obeyed by programs and the *mechanisms* by which programs are made bidirectional.

At the level of semantics, the key distinction is between bijectiveness and bidirectionality. In *bijective languages*, the forward transformation denoted by every program is an injective function, and the backward transformation is the corresponding inverse. By contrast, in *bidirectional languages*, the forward transformation can be an arbitrary function. Since the forward transformation may discard information in general, the backward transformation typically takes two arguments: an updated output as well as the original input. It weaves these two together, yielding a new input where the information contained in the output has been propagated and the discarded information from the input has been restored. Bidirectional programs are typically required to obey "round-tripping"

[8] Generalizations to more than two sources of information have also been studied, but are beyond the scope of this report; see [61] for an example.

laws, which ensure that information is maintained by the transformation in each direction.

Several mechanisms for equipping programs with a bidirectional semantics have been investigated. In *reversible* models of computation, each step of computation is invertible [116]. Although these models can only realize injective functions, it has been shown that this does not represent a serious restriction—arbitrary functions can be made injective by adding the information discarded by the forward transformation, often called a complement, to its output. This approach is often used in *bidirectionalization* of unidirectional transformations. In general, there are many ways to represent the discarded information [12, 55, 77], and each results in a different backward transformation. Recently, semantic approaches to bidirectionalization have also been explored [108].

Other languages do not require explicit reversibility at each step. For example, programs in the languages biXid and XSugar, consist of pairs of intertwined grammars [17, 58], and the transformations are obtained by parsing according to the rules in one grammar and pretty printing according to the rules in the other. Another approach, used in many bidirectional languages, is to have primitives that denote two transformations and combining forms that preserve bidirectionality [15, 16, 42, 43, 84].

Presentations at the meeting

Holger Bock Axelsen gave two presentations on reversible models of computation. In his first talk, he described a general class of reversible abstract machine architectures and instruction sets. The individual instructions and control logic are both reversible and are based on a notion of reversible updates [9]. His second presentation described a result in computability theory: structured programming using a reversible flowchart language is as expressive as unstructured programming [117]. The result uses the concept of *r-Turing completeness*: a model of computation is r-Turing complete if it is capable of simulating Turing machines *cleanly*—i.e., without generating any extraneous output data. Unlike classical approaches [12], reversible flowcharts are r-Turing complete.

Kazutaka Matsuda described recent work on bidirectionalization—i.e., the task of deriving suitably-related backward transformations from the program describing the forward transformations. He described a method that, given a program written in ordinary λ-calculus notation, produces a corresponding well-behaved backward transformation [76]. The method derives "complement" functions and also uses an injectivity analysis to produce a backward transformation capable of handling a broad class of updates. Using a tupling transformation, the technique can also be cleanly extended to situations where the forward transformation duplicates parts of the input [77].

Janis Voigtländer presented a different approach to bidirectionalization that works purely semantically [108]. He described a higher-order function that takes a (polymorphic) forward transformation as an argument and constructs an appropriate backward function as a result. No special language for describing the

forward transformation is needed, and the function that calculates the backward transformation does not inspect the program used to describe the forward function. The work is inspired by relational parametricity [89] and uses free theorems [111] to prove several round-tripping laws.

Meng Wang described an application of bidirectionalization for building algebraic views [112]. In this context, a *view* is an abstraction of the actual implementation of an abstract data type that provides an interface that is convenient for programmers to use and robust to changes [110]. Implementers of views are typically required to come up with pairs of conversion functions that are each other's inverses, a condition that is difficult for programmers to check and maintain. Using a bidirectionalization transformation, these properties can be obtained automatically.

Nate Foster presented work on lenses [15, 16, 42] that addresses the problem of handling inessential data in bidirectional transformations [43]. The foundation of most bidirectional languages is rooted in "round-tripping" laws which require that information be preserved in both directions. In practice, however, these laws are too strong: realistic bidirectional transformations do not obey them "on the nose," but only modulo equivalence relations that capture the essential parts of the data being manipulated. He described a general framework of *quotient lenses* and gave a syntax and a type system for building and reasoning about well-behaved quotient lenses.

Yingfei Xiong described Beanbag, a language for describing synchronization policies for software models [114]. Whenever one part of a model is modified, the Beanbag system computes the updates that need to be applied to the other parts of the model (as well as to related models) to reach a consistent state. Beanbag provides declarative syntax for describing inter-relations between models and intra-relations within a single model.

2.2 Databases and Data Management

In database literature, the fundamental unit of data movement or transformation is the *query*, which leverages the structure of potentially large quantities of data to provide declarative syntax, clean semantics, and efficient execution. The specification of a transformation may occur at some higher level such as schema matching, and is then compiled or converted into queries.

Until recently, most database research into *bx* studied whether an existing transformation specified in a query language, e.g., SQL, Datalog, or XQuery, can be "reversed" in some meaningful way, i.e., create a new transformation from the target of the original transformation to its source, rather than starting with bidirectionality by design, e.g., [27, 37]. A database transformation between models $\mathcal{M}$ and $\mathcal{M}'$ can be bidirectional in two ways: *operationally* (by transforming both read and write operations on $\mathcal{M}$ into equivalent operations on $\mathcal{M}'$) or *by instance* (by determining to what degree instances of $\mathcal{M}$ can remain unaltered when being transformed into an instance of $\mathcal{M}'$ and back again).

Instance-at-a-time transformation is also known as *data exchange* in database literature [95, 44]. The goal in data exchange is to transform instances of source

model $\mathcal{S}$ into instances of target model $\mathcal{T}$ in such a way that a set of constraints $\Sigma_{\mathcal{ST}}$ that associate instances of $\mathcal{S}$ and $\mathcal{T}$ are satisfied. The constraints may be specified in whatever query or logical language suits the models, such as tuple-generating dependencies of first-order logic [38] or second-order logic [37] for relations or tree-based pattern matching for XML [7]. For a mapping $(\mathcal{S}, \mathcal{T}, \Sigma_{\mathcal{ST}})$ to be bidirectional, the goal is to construct another mapping $(\mathcal{T}, \mathcal{S}, \Sigma_{\mathcal{TS}})$ such that composing the two is the identity mapping [37]. Such inverses may not always exist, or may only be partial [8].

A concrete case of instance-at-a-time transformation is the *cross-metamodel mapping*, where the source and target models of a transformation are in different metamodels. In database research, the most common cases involve translating between three metamodels in particular: objects (O), XML (X), and relations (R) [68]. The O-R case has well-established commercially-available tools and mature research [20, 60, 81, 86]. The X-O and X-R cases are substantially more difficult, given the difference in the expressive power of the metamodels [69]. There are several approaches to translating XML document instances into objects and vice versa. The most common cases involve either constructing XML-like objects that support an XML-like interface [50, 80] or compiling an XML schema into a canonical class representation that supports translation of instances in either direction [65, 66, 73, 115]. A more recent research effort allows declarative mappings to be specified between classes and XML schemas [82, 103].

The operational case is often called the *view update problem*: given a target model $\mathcal{T}$ specified as a set of views over a source model $\mathcal{S}$ by a set of queries $\mathcal{Q}$, determine if it is possible to intercept updates to $\mathcal{T}$ and instead update $\mathcal{S}$ such that re-running queries $\mathcal{Q}$ on $\mathcal{S}$ regenerates the updated instance of $\mathcal{T}$ exactly. The update to $\mathcal{S}$ must be unambiguous, i.e., there can be exactly one way to do it [27]. Research into the view update problem generally focuses on identifying semantic or syntactic constraints on $\mathcal{Q}$ that can determine if a view defined using $\mathcal{Q}$ is updatable, and if the exact method of update translation can be determined strictly by examining syntax.

One area of research around updatable views pertains to what invariants should be respected with respect to view updates. For instance, the *constant complement* property suggests that not only should base data be unambiguously updatable, but that data not referenced by the view should remain constant [10].

A recent development in bidirectional database transformations is the study of smaller algebraic transformations with known properties. For instance, the foundation of both relational lenses [16] and Both-as-View [78] is a collection of atomic query transformations that are known to be bidirectional, from which more complex transformations can be built.

In the proximity of *bx*, there is a data management scenario for co-evolution of database schemas and database instances [48]. This is important when schema-level transformations (e.g., schema refactorings) need to be coupled with instance-level transformations so that existing instances can be adapted for use with a new schema. Likewise, transformations of XML schemas and XML-document transformations may be coupled [67]. Also, coupling is not limited to schema and

instance transformations. In addition, co-transformation of schema-dependent programs may be an issue [22, 107]. These are all instances of the notion of coupled transformations [14, 64, 106, 107].

Presentations at the meeting

Jácome Cunha covered HaExcel, which is a tool that correlates spreadsheets and databases, and prevents update anomalies in spreadsheet tables [24]. HaExcel uses data mining techniques to discover functional dependencies in spreadsheet data. These functional dependencies can be exploited to derive a normalized relational database schema. Finally, HaExcel applies data calculation laws to the derived schema in order to reconstruct a sequence of refinement steps that connects the relational database schema back to the tabular spreadsheet.

James Terwilliger presented two bidirectional frameworks; the first was Microsoft Entity Framework (EF), which serves as an object-relational mapping tool and is publicly available as part of the .Net framework [20]. In EF, the developer specifies a declarative mapping between an extended entity-relationship model and a relational database. This specification is then compiled into a pair of views (one view for query transformation, one for update transformation) that translates model operations into equivalent database operations [81].

Finally, James presented Guava, a system that treats the user interface of a business application as an updatable view of that system's database. Guava encapsulates middleware operations like data pivoting, unpivoting, merging and partitioning, tuple augmentation with environment data, and role-based security into algebraic operators. Each operator translates queries, updates, and schema modifications on its input into equivalent expressions on its output [101, 102].

2.3 Model-Driven Engineering

Model-driven engineering (MDE) promotes the use of formal or structured specifications, which are referred to as *models*, with the goal of automating the derivation of implementations from such specifications. The approach involves many kinds of related models, such as requirements, design, and test specifications, and developers have to maintain complex relationships among the models and code. Examples of such relationships are *refinement* of design models to code and the *conformance* of models to their respective metamodels. Model transformations are mechanisms for establishing—and re-establishing, in the presence of change—the relationships among models and code [26]. Bidirectional model transformations are of particular interest if the related artifacts can be edited independently [5, 98]. Such edits are necessary if different stakeholders require viewing information in different specialized notations, possibly at different abstraction levels, or some of the related artifacts contain independent pieces of information that cannot be derived from other artifacts or both.

The Object Management Group (OMG) acknowledged the importance of *bx* to MDE by including a *bx* language in their Query View Transformation (QVT)

standard. Interestingly, Perdita Stevens analyzed the language from the viewpoint of the round-tripping laws (Section 2.1), pointing out several weaknesses. Other languages and systems to bidirectional model transformations have been proposed; see [30, 45, 51, 113, 114] for some more recent discussions and contributions.

The model-transformation scenario for co-evolution of metamodels and models, where metamodel adaptations must be coupled with corresponding model transformations [109], is closely related to *bx*. Such co-evolution is an instance of the notion of coupled transformations [64, 106].

Presentations at the meeting

Davide Di Ruscio gave a presentation on co-evolution of metamodels and models. This work is based on the axiom that metamodels must be considered one of the basic concepts of MDE and, accordingly, they are expected to evolve during their life cycle. As a consequence, models conforming to changed metamodels have to be updated to preserve their well-formedness. The presented work deals with the coupled transformation of metamodels and models by using higher-order model transformations which take a difference model for the metamodel level as input and produce a model transformation able to co-evolve the involved models as output [21].

Antonio Vallecillo gave a presentation on correspondences in viewpoint modeling for complex software systems. Viewpoint modeling is a technique for specifying software systems in terms of a set of independent viewpoints and correspondences between them. Correspondences specify the relationships between the elements in different views, together with the constraints that guarantee the consistency among these elements. Correspondences are hard to specify due to a lack of adequate notations, mechanisms, and tools. Also, specifications become unmanageable when the number of elements in a system is large. The presentation described efforts that are focused on the development of a generic framework and a set of tools to represent correspondences [104], which are able to manage and maintain viewpoint synchronization in evolution scenarios, as reported in [36]. The approach is based on modeling correspondences both intensionally and extensionally and the use of model transformation techniques to connect these two specifications.

Andrzej Wasowski gave a short progress report on his efforts to develop a flexible editing model for feature diagrams, including reversible editing steps and editing of broken (inconsistent) models. The presented topics included semantics for the editing process and algorithms for validation and guidance during modeling, under inconsistency. The aim of this work is to (ultimately) build a 'very intelligent' and highly flexible editor for feature models.

Bernd Fischer gave a short presentation of his work in progress on model-based code generation. The transformations used to generate code are fundamentally unidirectional because the generated code contains significantly more details than the model from which it was generated. This makes it hard to modify the generated code without causing model and code to get out of sync. In

round-trip engineering, the direction of code generation it to be complemented by an inverse transformation (which is known to be a hard problem). In contrast, the presentation proposed to employ aspect-oriented techniques to achieve the desired code modifications by controlled modifications of the transformation. The core insight is that the concepts from the generator's domain model can be used to systematically derive the required join points.

Krzysztof Czarnecki first presented ongoing work (with Michal Antkiewicz) on an infrastructure for mapping domain-specific languages (DSLs) to code [2, 4, 6, 71]. The infrastructure supports extracting domain-specific models from code as code views and bidirectional update propagation with conflict resolution. The extraction and update propagation rely on declarative mapping definitions, which relate elements of the DSL to structural and behavioral code patterns. The mapping definitions can be executed bidirectionally thanks to predefined code queries and code update transformations approximating the semantics of the mappings. The second presentation outlined an ongoing effort (with Zinovy Diskin and Michal Antkiewicz) to recast the design space of heterogeneous synchronization [5] in terms of synchronizers with category-theoretic underpinnings.

Zinovy Diskin gave a presentation on the algebraic foundations of model management (work in progress). Model management scenarios are often described by informal diagrams: nodes denote models and arrows are model transformations (of different types). The goal of the work is to reveal *diagrammatic* algebraic foundations underlying such diagrams and thus make the notation precise without translating it into formula-based formalisms. Some general ideas can be found in [29], and a recent promising application is described in [31].

2.4 Bidirectional Graph Transformation

The graph transformation research community, with its roots in the 1970's, can be considered today to be a special subarea of the model-driven engineering community. If we replace the terms "model", "metamodel", and "model transformation" by the related terms "graph", "graph type/schema", and "graph transformation", then it becomes obvious that graph transformation languages and tools are essentially model transformation languages and tools with a precisely defined semantics. Roughly speaking, one can identify three different families of graph transformations that either use category theory, non-standard logics, or set theory as their foundation. For a survey of related activities, we refer the reader to the LNCS Proceedings of the International Conference on Graph Transformation ICGT, as well as the set of so-called graph grammar handbooks [33, 34].

One can distinguish at least two different sorts of bidirectional graph transformation approaches: (1) reversible graph transformation languages, which rewrite a given input graph step by step into a new output graph, and (2) truly bidirectional graph transformation languages, which manipulate pairs of graphs linked together by means of so-called *correspondence links*. All graph transformation languages that support the so-called "double pushout" category-theoretic graph transformation approach can be classified as reversible transformation languages. Conditions on rule application guarantee that all rewriting steps can be undone

by simply applying the involved rewrite rule with exchanged left- and right-hand side [35]. *Triple Graph Grammars* (TGGs) [92, 94] as the descendants of pair grammars [87] are a special brand of coupled grammars. They belong to the class of bidirectional transformation languages [25]. Pairs of uni-directional forward and backward transformations can be derived automatically from a given TGG that defines a language of related pairs of graphs.

Presentations at the meeting

Andy Schürr gave a tutorial-style introduction to TGGs. TGGs are a bidirectional model transformation formalism, where a single specification generates a language of related graph tuples (pairs of models) together with an intermediate correspondence graph (traceability link database). A single TGG specification is used as input for a compiler that generates corresponding consistency checking, traceability link creating, and forward/backward model transformation implementations. The TGG tutorial reviewed the history of TGGs and sketched their formal definition relying on the theory of the algebraic/category-theoretic branch of graph grammars. Finally, the meta-modeling tool MOFLON was presented. MOFLON's implementation of TGGs adopts the visual notation of QVT Relational, the OMG standard bidirectional model transformation language.

Soichiro Hidaka, Hiroyuki Kato, Shin-Cheng Mu, and Keisuke Nakano gave presentations on different aspects of a functional approach to bidirectional graph transformation. This work aims at the development of an algebraic framework for bidirectional model transformation by integrating the state-of-the-art technologies on bidirectional tree transformations and the algebraic graph querying language UnQL+ [52, 53], which is an extension of the known UnQL [18]. The theoretical foundation of the work is related to the family of category-theoretic graph transformations called *algebraic graph transformations*. The resulting bidirectional graph transformation approach comes with a powerful automatic bidirectionalization method for the automatic derivation of a backward graph transformation from a given forward graph transformation. For this purpose, a bidirectional semantics for an existing graph algebra based on structural recursion called UnCAL is used, which has been well studied in the database community. Hence, this work belongs to the class of reversible as well as to the class of truly bidirectional graph transformation languages. Moreover, the algebraic framework supports the systematic development of efficient large-scale bidirectional model transformations in a compositional manner.

2.5 Further applications

GRACE-BX covered a number of presentations that we feel are best collected in a list of "further applications". It goes without saying that these applications regularly interact with issues of programming languages for *bx*, data management, or model-driven engineering.

Keisuke Nakano described the Vu-X approach to website construction that is based on bidirectional transformations [85] (as opposed to unidirectional transformations that simply translate data from a database into web content). Hence, users can directly modify a generated website, and the modification is automatically reflected in the database—without the need to update the database directly. The Vu-X system is also implemented as a web server so that users can edit it in WYSIWYG style within their web browsers.

Hui Song talked about runtime management of systems at the level of an intuitive, high-level architecture model [56, 96]. Management agents use the architecture model to monitor and control a running system. A key component for architecture-based runtime management is the synchronizer that propagates changes between the architecture model and the system state, and maintains the correspondence between them. The presented approach supports automated generation of such synchronizers based on bidirectional transformations.

David Lutterkort presented a configuration API for Linux systems called Augeas [75]. Augeas parses configuration files in their native formats and transforms them into a tree. Configuration changes are made by manipulating this tree and saving it back into native configuration files. The string-to-tree transformation is specified by lenses so that some details can be left out from the tree level, but they are still preserved when writing back changes. For instance, Augeas makes an effort to preserve comments and formatting in the textual configuration files.

Kathleen Fisher presented recent work on PADS, a system for processing ad hoc data sources (such as log files) [39]. The PADS compiler takes declarative descriptions of data formats as input and generates a variety of software artifacts including a parser, an in-memory representation for the data, and a pretty printer, among others. The presentation described a mechanism for generating a suite of useful data processing tools, including a semi-structured query engine, several format converters, a statistical analyzer, and data visualization routines directly from ad hoc data, without human intervention [40]. The key technical contribution is a multi-phase algorithm that automatically infers the structure of an ad hoc data source and produces a format specification in the PADS data description language. Programmers wishing to implement custom data analysis tools can use such descriptions to generate printing and parsing libraries for the data.

Ralf Lämmel talked about grammar transformations [63, 70]—specifically on coupled grammar transformations and their applications in XML-data binding and concrete/abstract syntax mapping. Grammar transformations are expressed in terms (of sequences) of primitive combinators, which can be applied both to grammars and instances (such as parse trees or documents). In the simpler, better understood cases, these grammar transformations are information-preserving, and an inverse is defined for each possible combinator. In more general cases such as mapping a rich concrete syntax to a more abstract syntax, bidirectionality is more difficult to achieve, subject to future work.

Zhenjiang Hu gave a presentation on the use of automatic function inversion as a means to obtain divide-and-conquer parallel programs from sequential programs [83]. This approach allows programmers to use the often more intuitive, sequential encoding style, while, under certain conditions, efficient parallel programs in the form of list homomorphisms can be derived automatically. These parallel programs would be more difficult to develop by programmers in the first place. The heavy lifting of the approach for the extraction of a list homomorphism is the automatic derivation of weak right inverses from sequential programs. Experimental results show the practical efficiency of the automatic parallelization algorithm and demonstrate that the generated parallel programs achieve good speedups.

3 Synthesis: key concepts and properties of BX

The meeting included two discussion plenary sessions: one on the terminology and key concepts used across the represented communities and another on properties of *bx*. We briefly summarize the main discussion points in this section.

Lack of common and well-established terminology The participants generally agreed that each represented community has developed its own terminology and that there is little sharing of terms across disciplines. Moreover, central terms such as "transformation" and "view" are overloaded. For example, "transformation" is sometimes used to mean "transformation specification", "transformation implementation", or "transformation execution". Likewise, although "view" has a precise and well-established meaning in databases, it is used in a different way in the programming languages community [110], and its meaning in MDE is not clear.

Transformation vs. synchronization A hotly-debated topic was the distinction between transformation and synchronization. All of the participants shared a common understanding of transformations as executable operations on structured artifacts (data, models, programs) that establish well-defined relationships between the inputs and outputs, but the definition of synchronization was less clear. After discussion, a key difference was identified: synchronization (re-)establishes relationships among (partial) *replicas*, i.e., semantically overlapping artifacts that exist in parallel. Thus, although synchronization can be viewed as a kind of transformation, such as transforming code to make it consistent with an updated design, not all transformations do synchronization. For example, the reversible edits presented by Wasowski (see Section 2.3) are *bx* but not synchronizers since they relate two consecutive *revisions* of a single artifact being edited. It was also noted that, in general, transformations may take place in spaces with multiple dimensions including revisions, replicas, languages, and features [11, 31]. Exploring *bx* in these multidimensional spaces was proposed as an important area for future work.

State-based vs. operation-based Many of the projects presented at the meeting focused on using *bx* to propagate changes made to the source or target of the transformation. These update translators fall into two distinct categories. In *state-based* approaches, the update translator operates on the source and target structures themselves. For example, to translate an update made to the target back to the source, the translator takes as an argument the post-update state of the target (as well as the original state of the source) and calculates an appropriately-modified source [15, 16, 41, 42]. By contrast, in *operation-based* approaches, updates are expressed in a transformation language, and the update translator propagates the updates themselves through the transformation. In either case, update translation produces instances that are consistent with respect to the *bx* [81, 102].

Properties of bx Several different properties of *bx* were discussed at the meeting. The properties of the relation that captures when the source and target are consistent—e.g., whether the relation is an injective source-to-target function or a total target-to-source function or both, or even if the relation is a function in either direction at all—impact *bx* in fundamental ways. Most systems stipulate that the transformations a *bx* comprises must obey the *definitional properties* of *correctness* and *hippocraticness* [97] (these properties roughly correspond to PutGet and GetPut laws in the lens framework [42]; for details of this correspondence, see elswhere [30, 99]), with *undoability* [97] (corresponding to PutPut for lenses [42]) as an optional property. The *reversibility* of a transformation was discussed as an example of an *operational property*: not only must the inverse exist, it must also be computable efficiently. Lastly, properties such as *incrementality* [46], *minimality of changes* [19, 44], and *preservation of recent changes* were classified as *quality properties*. Although these properties have intuitive appeal, they are not well understood formally and are an important area for further study.

Constructing bx Several different techniques for constructing *bx* were discussed at the meeting. There are three main approaches. In the first approach, the user programs the forward transformation and the backward transformation is obtained (almost) for free, because the forward transformation either was built from smaller primitives that are bidirectional (e.g., [78, 101, 102]) or can be made bidirectional (e.g., [76, 108]). The second approach, used in lenses, is similar but subtly different: the programmer specifies the backward transformation and obtains the forward transformation for free (e.g., [16, 42]). In the third approach, the user defines a *mapping*—i.e., the specification of the relation that captures when the source and target are consistent—and the forward and backward transformations are derived from the mapping automatically, possibly with additional manual refinement (e.g., [6, 81, 93]). A related topic is the composition of bidirectional translators, e.g., lenses and synchronizers, into larger systems ([30, 31, 41]).

4 Towards a BX benchmark

The GRACE-BX meeting clarified the need to have effective criteria for comparing different *bx* approaches and assessing progress in the field. Benchmarks are widely used in academia and industry as a framework for comparing competing approaches and technologies. Insightful examples of benchmark suites exist for scenarios with cross-discipline application, including graph transformation [105], query and transformation of XML [91], generic programming (query and transformation of tree-shaped data) [90], and schema mappings [1].

Accordingly, the meeting included a discussion session on a potential benchmark suite for bidirectional transformations, codenamed BXBenchmark. The following goals of BXBenchmark were identified:

- Provide a *platform* for comparing expressiveness, usability, and efficiency of different *bx* approaches and technologies.
- Catalog proven *bx scenarios* and interesting variations thereof (e.g., the ubiquitous classes-to-tables mapping and variations thereof).
- Collect *implementations* of *bx* scenarios.
- Clarify the relevance of the scenarios across different approaches and technologies and thereby *reveal commonalities and differences*.

Organization of the benchmark suite

To help picturing the envisaged benchmark suite BXBenchmark, we sketch its possible organization. At the top level, the suite is organized by a collection of *scenarios*. Each scenario comprises components as described below. (We use the ubiquitous classes-to-tables mapping as a running example.)

Name For instance: "Classes to Tables".

Rationale For instance: "Cover a baseline scenario of model-driven development with just a few modeling concepts for classes in the sense of the OO paradigm and tables in the sense of the relational paradigm".

Metadata Based on an appropriate *taxonomy* of bidirectional transformations, the scenarios are to be semantically annotated. To provide a simple example, scenarios would be annotated as being graph- vs. string- vs. tree-based. (The scenario of the running example is graph-based.) We discuss some elements of an emerging taxonomy shortly.

Sample data Each scenario will be accompanied by a collection of types (or classes, schemas, regular expressions, or other relevant *metamodel* information) that model sample data for the scenario, as well as actual conformant *sample data*. In addition, generators and mappers for metamodels and sample data may provide access to sample data across platform, format, and programming paradigm while meeting constraints on size or others.

Configurations The term "configuration" is used here as a proxy for comprehensively describing the execution of a bidirectional transformation including sample data (inputs and results), a description of updates and auxiliary parameters such as size measures when data is generated.

Specialization Scenarios may be related by specialization. For instance, we may think of "Classes to Tables" as an abstract scenario with fundamentally different O/R mapping options as concrete specializations.

Any scenario can now be implemented by any number of implementations using particular technologies. For instance, one technology may address the scenario by describing both directions of the *bx* scenario separately, whereas another technology may leverage a designated transformation language for *bx*. We also need some way of documenting the assumed difficulty, feasibility, completeness, or infeasibility of a specific implementation (option). To this end, the role of metadata, as announced above, will be extended.

An emerging taxonomy for metadata

We expect this taxonomy to cover properties of *bx* that are useful in documenting (by means of metadata) commonalities and differences between the different scenarios, the different implementations thereof, the different *bx* approaches and technologies. A few candidate properties are illustrated:

Injectivity Given a scenario, is the basic forward transformation injective? If it is not injective, what other properties are possibly exploited to obtain a useful backward transformation?

Kinds of changes Given a scenario or a *bx* approach, what kinds of changes are needed or supported? For instance, one can distinguish updates vs. insertion vs. deletion. When focusing on updates, one can distinguish structured vs. primitive updates.

Synchronization orientation Given an implementation of a scenario or a technology, is synchronization involved? If so, how can we further classify and qualify the form of synchronization at hand? (For instance, do we face synchronization based on a constraint mechanism?)

Semantics preservation Given a scenario that involves a sort of (programming) language as the domain of the sources related by *bx*, does the *bx* promise or require semantics preservation? (See the scenario "For-loop Desugaring" presented shortly.) Other programming language-related notions may be similarly leveraged to contribute properties for taxonomy, e.g., type preservation and dependency on flow analysis.

Some candidate scenarios for BXBenchmark

"Roman/Arabic Number Conversion" This is a trivial example which can be used to provide a basic demonstration of any transformation technology. The conversions are bijective and can be reasonably represented by a pair of unidirectional transformations.

"Add Index to Address Book" (inspired by [55]) Given a simple collection of addresses of persons (say represented as an XML tree), an index for the names of the persons is added by the forward transformation. The redundancy created by the extra name index triggers update challenges.

"Classes to Tables" This scenario (described earlier) is a baseline scenario of model-driven development. Various technologies have readily addressed some variant of it. It appears to be promising to try organizing all these variants. Conceptually, it is an (O/R) mapping example which hence involves two paradigms: the OO paradigm and the relational paradigm. Object models and relational schemas may both be represented as graphs (with edges for associations or key constraints). In fact, whereas a tree-based approach suffices for the previous scenario, a graph-based approach may be more appropriate for the present scenario. (We view "Classes to Tables" as a representative of the larger class of relatively direct metamodel transformations: WSDL to/from EMF, BPMN to/from BPEL, EMF to/from XSD, UML sequence to/from communication diagrams, etc.)

"Collapse/Expand State Diagrams" Starting from a hierarchical state diagram (involving some nesting), a flat view is to be provided, and any modifications on the flat view should be reflected eventually in the hierarchical view. The basic flattening transformation is non-injective, and hence, the view complement must be taken into account when mapping back the flattened and possibly updated state diagram to a nested one.

"For-loop Desugaring" Given a tiny (imperative) programming language, provide a syntax desugaring transformation, e.g., the translation of for-loops to while loops. The desugared syntax is further subjected to a refactoring transformation. The bidirectional transformation challenge is to be able to revert desugaring even past refactoring. Just like the previous scenario, desugaring is non-injective. The specific contribution of the scenario lies in its well-understood interaction with programming language types and semantics. Ideally, one would want to establish, by construction, that desugaring and its reversion are type- and semantics-preserving. (A related but distinctive scenario, inspired by [28, 74], is the preservation of whitespace and comments by transformations that abstract from whitespace and comments.)

"Round-trip Engineering for Java Applet Models and Code" (inspired by [2, 3]) In the forward direction, JApplet framework use is expressed at the modeling level in terms of a feature model, subject to a code-generating interpretation of the model. In the backward direction, framework use is extracted from code by queries and presented as feature models. Updates may be performed both on code and feature model. This scenario stands out with its involvement of two rather distant abstraction levels: imperative OO code vs. more grammatical and declarative feature models. This scenario is also relatively challenging in that it requires flow analysis.

"Textual and Graphical Program Editor" Given (a subset of) a Java/C#-like language, provide a capability for simultaneous editing in textual and graphical mode. It is assumed that the graphical mode provides a limited view on programs. For instance, it may be limited to types and relationships. This scenario involves "Text to Model" and "Model to Text" components, and thereby involves specific challenges such as layout preservation (of both textual and graphical layout) and comment preservation (where we assume that comments only appear in the textual representation). Further, this sce-

nario requires the ability to incrementally (locally) change the graphical view in reply to local changes to the textual view, and vice versa.

"Reversible FFT" (inspired by [72, 116]) A Fast Fourier Transform is an efficient algorithm to compute the Discrete Fourier Transform and its inverse. Compared to the above scenarios, this scenario specifically involves a computationally expensive problem.

Status and future prospects

An open source project has been created to host future efforts on BXBenchmark.[9] The project is actively seeking contributors. Designated workshops or bird-of-a-feather sessions and hackathons may be scheduled to make progress on the suite. Work on the benchmark suite is likely to feature prominently at any follow-up meetings.

5 Conclusion

Bidirectional transformation is a field of interest that spans many sub-disciplines of computer science. The GRACE-BX meeting served as a useful checkpoint to match capabilities and research efforts, and to examine differences in approaches and assumptions. The references cited in this report constitute the beginnings of a comprehensive bibliography of *bx*-related work across the sub-disciplines. The new benchmark suite will continue the collaboration effort by allowing researchers with different research aims to contribute solutions to common problems. In addition to the emerging benchmark, we will continue to foster collaboration between communities by applying for a Dagstuhl seminar. We may also hold workshops at conferences as part of the effort to establish the benchmark.

Acknowledgments We would like to thank all the participants of the GRACE-BX meeting for their contributions at the meeting and their input during the preparation of this report: Holger Bock Axelsen, Jean Bezivin, Jácome Cunha, Davide Di Ruscio, Zinovy Diskin, Bernd Fischer, Kathleen Fisher, Soichiro Hidaka, Robert Glück, Hiroyuki Kato, David Lutterkort, Kazutaka Matsuda, Shin-Cheng Mu, Alfonso Pierantonio, Keisuke Nakano, Ali Razavi, Hui Song, Antonio Vallecillo, Janis Voigtläender, Yingfei Xiong, Meng Wang, and Andrzej Wasowski.

References

1. B. Alexe, W.-C. Tan, and Y. Velegrakis. STBenchmark: towards a benchmark for mapping systems. *Proceedings of the VLDB conference*, 1(1):230–244, 2008. See also `http://www.stbenchmark.org/`.
2. M. Antkiewicz. *Framework-Specific Modeling Languages*. PhD thesis, University of Waterloo, Electrical and Computer Engineering, 2008.

[9] `https://sourceforge.net/projects/bxbenchmark/`

3. M. Antkiewicz, T. T. Bartolomei, and K. Czarnecki. Automatic extraction of framework-specific models from framework-based application code. In *ASE '07: Proceedings of the twenty-second IEEE/ACM international conference on Automated Software Engineering*, pages 214–223. ACM, 2007.
4. M. Antkiewicz and K. Czarnecki. Framework-Specific Modeling Languages with Round-Trip Engineering. In *Model Driven Engineering Languages and Systems, 9th International Conference, MoDELS 2006, Proceedings*, volume 4199 of *LNCS*, pages 692–706. Springer, 2006.
5. M. Antkiewicz and K. Czarnecki. Design Space of Heterogeneous Synchronization. In *Generative and Transformational Techniques in Software Engineering II, International Summer School, GTTSE 2007, Revised Papers*, volume 5235 of *LNCS*, pages 3–46. Springer, 2008.
6. M. Antkiewicz, K. Czarnecki, and M. Stephan. Engineering of Framework-Specific Modeling Languages. *IEEE Transactions on Software Engineering*, 2009. To appear.
7. M. Arenas and L. Libkin. XML data exchange: Consistency and query answering. *Journal of the ACM*, 55(2), 2008.
8. M. Arenas, J. Pérez, and C. Riveros. The recovery of a schema mapping: bringing exchanged data back. In *PODS '08: Proceedings of the twenty-seventh ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 13–22. ACM, 2008.
9. H. B. Axelsen, R. Glück, and T. Yokoyama. Reversible Machine Code and Its Abstract Processor Architecture. In *Computer Science – Theory and Applications, Proceedings*, volume 4649 of *LNCS*, pages 56–69. Springer, 2007.
10. F. Bancilhon and N. Spyratos. Update semantics of relational views. *ACM Transactions on Database Systems (TODS)*, 6(4):557–575, 1981.
11. D. S. Batory, M. Azanza, and J. Saraiva. The objects and arrows of computational design. In *Model Driven Engineering Languages and Systems, 9th International Conference, MoDELS 2008, Proceedings*, volume 5301 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2008.
12. C. H. Bennet. Logical Reversibility of Computation. *IBM Journal of Research and Development*, 17(6):525–532, 1973.
13. N. Benton. Embedded interpreters. *Journal of Functional Programming*, 15(4):503–542, 2005.
14. P. Berdaguer, A. Cunha, H. Pacheco, and J. Visser. Coupled Schema Transformation and Data Conversion for XML and SQL. In *Practical Aspects of Declarative Languages, 9th International Symposium, PADL 2007, Proceedings*, volume 4354 of *LNCS*, pages 290–304. Springer, 2007.
15. A. Bohannon, J. N. Foster, B. C. Pierce, A. Pilkiewicz, and A. Schmitt. Boomerang: Resourceful lenses for string data. In *Proceedings of ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL 2008)*, pages 407–419, Jan. 2008.
16. A. Bohannon, B. Pierce, and J. Vaughan. Relational lenses: a language for updatable views. In *PODS '06: Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 338–347. ACM, 2006.
17. C. Brabrand, A. Møller, and M. I. Schwartzbach. Dual Syntax for XML Languages. *Information Systems*, 33(4–5):385–406, 2008. Short version in DBPL '05.

18. P. Buneman, M. F. Fernandez, and D. Suciu. UnQL: a query language and algebra for semistructured data based on structural recursion. *VLDB Journal: Very Large Data Bases*, 9(1):76–110, 2000.
19. P. Buneman, S. Khanna, and W.-C. Tan. On propagation of deletions and annotations through views. In *PODS '02: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 150–158. ACM, 2002.
20. P. Castro, S. Melnik, and A. 0Adya. ADO.NET entity framework: raising the level of abstraction in data programming. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 1070–1072. ACM, 2007.
21. A. Cicchetti, D. Di Ruscio, R. Eramo, and A. Pierantonio. Automating Co-evolution in Model-Driven Engineering. In *12th International IEEE Enterprise Distributed Object Computing Conference, ECOC 2008, Proceedings*, pages 222–231. IEEE Computer Society, 2008.
22. A. Cleve and J.-L. Hainaut. Co-transformations in Database Applications Evolution. In *Generative and Transformational Techniques in Software Engineering, International Summer School, GTTSE 2005, Revised Papers*, volume 4143 of *LNCS*, pages 409–421. Springer, 2006.
23. R. Culpepper and M. Felleisen. Debugging macros. In *GPCE '07: Proceedings of the 6th international conference on Generative programming and component engineering*, pages 135–144. ACM, 2007.
24. J. Cunha, J. Saraiva, and J. Visser. From spreadsheets to relational databases and back. In *PEPM '09: Proceedings of the 2009 ACM SIGPLAN workshop on Partial evaluation and program manipulation*, pages 179–188. ACM, 2008.
25. K. Czarnecki and S. Helsen. Classification Of Model Transformation Approaches. In *2nd OOPSLA Workshop on Generative Techniques in the context of Model Driven Architecture*, 2003. Available at `http://www.softmetaware.com/oopsla2003/czarnecki.pdf`.
26. K. Czarnecki and S. Helsen. Feature-based survey of model transformation approaches. *IBM Systems Journal*, 45(3):621–646, 2006.
27. U. Dayal and P. A. Bernstein. On the Correct Translation of Update Operations on Relational Views. *ACM Transactions on Database Systems (TODS)*, 7(3):381–416, Sept. 1982.
28. M. L. V. de Vanter. Preserving the Documentary Structure of Source Code in Language-Based Transformation Tools. In *1st IEEE International Workshop on Source Code Analysis and Manipulation (SCAM 2001), Proceedings*, pages 133–143. IEEE Computer Society, 2001.
29. Z. Diskin. Mathematics of generic specifications for model management. In Rivero, Doorn, and Ferraggine, editors, *Encyclopedia of Database Technologies and Applications*, pages 351–366. Idea Group, 2005.
30. Z. Diskin. Algebraic Models for Bidirectional Model Synchronization. In *Model Driven Engineering Languages and Systems, 11th International Conference, MoDELS 2008, Proceedings*, volume 5301 of *LNCS*, pages 21–36. Springer, 2008.
31. Z. Diskin, K. Czarnecki, and M. Antkiewicz. Model-versioning-in-the-large: algebraic foundations and the tile notation. In *Comparison and versioning of software models. 3rd Int. Workshop affiliated with ICSE 2009*, 2009. To appear in IEEE Digital Library.
32. D. T. Eger. Bit Level Types, 2005. Unpublished manuscript. Available from `http://www.yak.net/random/blt/blt-drafts/03/blt.pdf`.

33. Ehrig, Engels, Kreowski, and Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 1. World Scientific Publishing, 1997.
34. Ehrig, Engels, Kreowski, and Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 2. World Scientific Publishing, 1999.
35. H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2006.
36. R. Eramo, A. Pierantonio, J. Romero, and A. Vallecillo. Change Management in Multi-Viewpoint Systems using ASP. In *Proceedings of 5th International Workshop on ODP for Enterprise Computing (WODPEC 2008)*, Sept. 2008.
37. R. Fagin. Inverting schema mappings. *ACM Transactions on Database Systems (TODS)*, 32(4), 2007.
38. R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: semantics and query answering. *Theoretical Computer Science*, 336(1):89–124, 2005.
39. K. Fisher and R. Gruber. PADS: a domain-specific language for processing ad hoc data. In *PLDI '05: Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation*, pages 295–304. ACM, 2005.
40. K. Fisher, D. Walker, K. Zhu, and P. White. From dirt to shovels: fully automatic tool generation from ad hoc data. In *POPL '08: Proceedings of the 35th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 421–434. ACM, 2008.
41. J. N. Foster, M. B. Greenwald, C. Kirkegaard, B. C. Pierce, and A. Schmitt. Exploiting Schemas in Data Synchronization. *Journal of Computer and System Sciences*, 73(4), June 2007. Short version in DBPL '05.
42. J. N. Foster, M. B. Greenwald, J. T. Moore, B. C. Pierce, and A. Schmitt. Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 29(3), 2007.
43. J. N. Foster, A. Pilkiewcz, and B. C. Pierce. Quotient lenses. In *ICFP '08: Proceeding of the 13th ACM SIGPLAN international conference on Functional programming*, pages 383–396. ACM, 2008.
44. A. Fuxman, P. G. Kolaitis, R. J. Miller, and W. C. Tan. Peer data exchange. *ACM Transactions on Database Systems (TODS)*, 31(4):1454–1498, 2006.
45. H. Giese and R. Wagner. From model transformation to incremental bidirectional model synchronization. *Software and Systems Modeling*, 8(1):21–43, 2009.
46. T. J. Green, Z. G. Ives, and V. Tannen. Reconcilable differences. In *ICDT '09: Proceedings of the 12th International Conference on Database Theory, Proceedings*, pages 212–224. ACM, 2009.
47. T. J. Green, G. Karvounarakis, Z. G. Ives, and V. Tannen. Update Exchange with Mappings and Provenance. In *Proceedings of the 33rd International Conference on Very Large Data Bases, VLDB 2007*, pages 675–686. ACM, 2007.
48. J.-L. Hainaut, C. Tonneau, M. Joris, and M. Chandelon. Schema Transformation Techniques for Database Reverse Engineering. In *Entity-Relationship Approach - ER'93, 12th International Conference on the Entity-Relationship Approach, Proceedings*, volume 823 of *LNCS*, pages 364–375. Springer, 1994.
49. A. Y. Halevy, Z. G. Ives, D. Suciu, and I. Tatarinov. Schema Mediation in Peer Data Management Systems. In *Proceedings of the 19th International Conference on Data Engineering, ICDE 2003*, pages 505–516. IEEE Computer Society, 2003.

50. M. Harren, M. Raghavachari, O. Shmueli, M. Burke, R. Bordawekar, I. Pechtchanski, and V. Sarkar. XJ: facilitating XML processing in Java. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 278–287. ACM, 2005.
51. T. Hettel, M. Lawley, and K. Raymond. Model Synchronisation: Definitions for Round-Trip Engineering. In *Theory and Practice of Model Transformations, First International Conference, ICMT 2008, Proceedings*, volume 5063 of *LNCS*, pages 31–45. Springer, 2008.
52. S. Hidaka, Z. Hu, H. Kato, and K. Nakano. A Compositional Approach to Bidirectional Model Transformation. In *New Ideas and Emerging Results Track of 31st International Conference on Software Engineering (ICSE 2009, NIER Track)*, May 2009. To appear.
53. S. Hidaka, Z. Hu, H. Kato, and K. Nakano. Towards Compositional Approach to Model Transformation for Software Development. In *24th Annual ACM Symposium on Applied Computing*, Mar 2009.
54. Z. Hu, S.-C. Mu, and M. Takeichi. 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*, pages 178–189. ACM, 2004. See [55] for a journal version.
55. Z. Hu, S.-C. Mu, and M. Takeichi. A programmable editor for developing structured documents based on bidirectional transformations. *Higher-Order and Symbolic Computation*, 21(1-2):89–118, 2008. See [54] for a short version.
56. G. Huang, H. Mei, and F. Yang. Runtime recovery and manipulation of software architecture of component-based systems. *Automated Software Engineering*, 13(2):257–281, 2006.
57. G. Karvounarakis and Z. G. Ives. Bidirectional Mappings for Data and Update Exchange. In *11th International Workshop on the Web and Databases, WebDB 2008, Proceedings*, 2008. Available at `http://webdb2008.como.polimi.it/images/stories/WebDB2008/paper35.pdf`.
58. S. Kawanaka and H. Hosoya. biXid: a bidirectional transformation language for XML. In *ICFP '06: Proceedings of the eleventh ACM SIGPLAN international conference on Functional programming, Proceedings*, pages 201–214. ACM, 2006.
59. A. M. Keller. The Role of Semantics in Translating View Updates. *Computer*, 19(1):63–73, 1986.
60. A. M. Keller, R. Jensen, and S. Agarwal. Persistence software: bridging object-oriented programming and relational databases. In *SIGMOD '93: Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, pages 523–528. ACM, 1993.
61. A. Königs and A. Schürr. MDI - a Rule-Based Multi-Document and Tool Integration Approach. *Journal of Software and System Modeling*, 5(4):349–368, December 2006. Special Section on Model-based Tool Integration.
62. S. Krishnamurthi, Y.-D. Erlich, and M. Felleisen. Expressing Structural Properties as Language Constructs. In *Programming Languages and Systems, 8th European Symposium on Programming, ESOP'99, Proceedings*, volume 1576 of *LNCS*, pages 258–272. Springer, 1999.
63. R. Lämmel. Grammar Adaptation. In *FME 2001: Formal Methods for Increasing Software Productivity, International Symposium of Formal Methods Europe, Proceedings*, volume 2021 of *LNCS*, pages 550–570. Springer, 2001.

64. R. Lämmel. Coupled Software Transformations (Extended Abstract). In *First International Workshop on Software Evolution Transformations*, Nov. 2004. 5 pages; available online at `http://homepages.cwi.nl/~ralf/CoupledSoftwareTransformations/`.
65. R. Lämmel. LINQ to XSD. In *Proceedings, PLAN-X 2007, Programming Language Technologies for XML, An ACM SIGPLAN Workshop collocated with POPL 2007*, pages 95–96, 2007. `http://www.plan-x-2007.org/plan-x-2007.pdf`.
66. R. Lämmel. Style normalization for canonical X-to-O mappings. In *PEPM '07: Proceedings of the 2007 ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation*, pages 31–40. ACM, 2007.
67. R. Lämmel and W. Lohmann. Format Evolution. In *Proceedings of 7th International Conference on Reverse Engineering for Information Systems (RETIS 2001)*, volume 155 of *books@ocg.at*, pages 113–134. OCG, 2001.
68. R. Lämmel and E. Meijer. Mappings Make Data Processing Go 'Round. In *Generative and Transformational Techniques in Software Engineering, International Summer School, GTTSE 2005, Revised Papers*, volume 4143 of *LNCS*, pages 169–218. Springer, 2006.
69. R. Lämmel and E. Meijer. Revealing the X/O impedance mismatch (Changing lead into gold). In *Spring School on Datatype-Generic Programming, Lecture Notes*, volume 4719 of *LNCS*, pages 285–367. Springer, 2007.
70. R. Lämmel and V. Zaytsev. An Introduction to Grammar Convergence. In *Integrated Formal Methods, 7th International Conference, IFM 2009, Proceedings*, volume 5423 of *LNCS*, pages 246–260. Springer, 2009.
71. H. Lee, M. Antkiewicz, and K. Czarnecki. Towards a Generic Infrastructure for Framework-specific Integrated Development Environment Extensions. In *2nd International Workshop on Domain-Specific Program Development (DSPD), in association with GPCE 2008*, 2008. `http://www.labri.fr/perso/reveille/DSPD/2008`.
72. J. Li. Low noise reversible MDCT (RMDCT) and its application in progressive-to-lossless embedded audio coding. *IEEE Transactions on Signal Processing*, 53(5):1870–1880, 2005.
73. Liquid XML. http://www.liquid-technologies.com/.
74. W. Lohmann and G. Riedewald. Towards Automatical Migration of Transformation Rules after Grammar Extension. In *7th European Conference on Software Maintenance and Reengineering (CSMR 2003), Proceedings*, pages 30–39. IEEE Computer Society, 2003.
75. D. Lutterkort. Augeas–A Configuration API. In *Proceedings of the Linux Symposium, Ottawa, ON*, pages 47–56, July 2008.
76. K. Matsuda, Z. Hu, K. Nakano, M. Hamana, and M. Takeichi. Bidirectionalization transformation based on automatic derivation of view complement functions. In *ICFP '07: Proceedings of the 12th ACM SIGPLAN international conference on Functional programming*, pages 47–58. ACM, 2007.
77. K. Matsuda, Z. Hu, K. Nakano, M. Hamana, and M. Takeichi. Bidirectionalizing Programs with Duplication through Complementary Function Derivation. *JSSST Journal: Computer Software*, 26(2), 5 2009. In Japanese. To appear.
78. P. McBrien and A. Poulovassilis. Data Integration by Bi-Directional Schema Transformation Rules. In *19th International Conference on Data Engineering, ICDE 2003, Proceedings*, pages 227–238. IEEE Computer Society, 2003.
79. L. Meertens. Designing Constraint Maintainers for User Interaction. Manuscript, available at `http://www.kestrel.edu/home/people/meertens`, June 1998.

80. E. Meijer, B. Beckman, and G. Bierman. LINQ: reconciling object, relations and XML in the .NET framework. In *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 706–706. ACM, 2006.
81. S. Melnik, A. Adya, and P. Bernstein. Compiling mappings to bridge applications and databases. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 461–472. ACM, 2007.
82. R. Miller, M. Hernández, L. Haas, L. Yan, C. Ho, R. Fagin, and L. Popa. The Clio Project: Managing Heterogeneity. *ACM SIGMOD Record*, 30(1):78–83, 2001.
83. K. Morita, A. Morihata, K. Matsuzaki, Z. Hu, and M. Takeichi. Automatic inversion generates divide-and-conquer parallel programs. In *PLDI '07: Proceedings of the 2007 ACM SIGPLAN conference on Programming language design and implementation*, pages 146–155. ACM, 2007.
84. S.-C. Mu, Z. Hu, and M. Takeichi. An Algebraic Approach to Bi-directional Updating. In *Programming Languages and Systems: Second Asian Symposium, APLAS 2004, Proceedings*, volume 3302 of *LNCS*, pages 2–20. Springer, 2004.
85. K. Nakano, Z. Hu, and M. Takeichi. Consistent Web Site Updating based on Bidirectional Transformation. In *10th IEEE International Symposium on Web Site Evolution (WSE 2008)*, Oct. 2008.
86. J. Oliveira. Transforming Data By Calculation. In *Generative and Transformational Techniques in Software Engineering II, International Summer School, GTTSE 2007, Revised Papers*, volume 5235 of *LNCS*, pages 134–195. Springer, 2008.
87. T. W. Pratt. Pair Grammars, Graph Languages and String-to-Graph Translations. In *Journal of Computer and System Sciences*, volume 5, pages 560–595. Academic Press, 1971.
88. N. Ramsey. Embedding an interpreted language using higher-order functions and types. In *IVME '03: Proceedings of the 2003 workshop on Interpreters, virtual machines and emulators*, pages 6–14. ACM, 2003.
89. J. Reynolds. Types, abstraction and parametric polymorphism. In *Information Processing 1983, Proceedings*, pages 513–523. Elsevier, 1983.
90. A. Rodriguez, J. Jeuring, P. Jansson, A. Gerdes, O. Kiselyov, and B. Oliveira. Comparing libraries for generic programming in Haskell. In *Haskell '08: Proceedings of the first ACM SIGPLAN symposium on Haskell*, pages 111–122. ACM, 2008.
91. A. Schmidt, F. Waas, M. Kersten, D. Florescu, I. Manolescu, M. Carey, and R. Busse. The XML Benchmark Project. Technical report, CWI, Amsterdam, The Netherlands, Apr. 2001. Technical Report INS-R0103. Available at `http://www.xml-benchmark.org/`.
92. A. Schürr. Specification of Graph Translators with Triple Graph Grammars. In Mayr and Schmidt, editors, *Proceeding of WG'94 Workshop on Graph-Theoretic Concepts in Computer Science*, pages 151–163, 1994.
93. A. Schürr. Specification of Graph Translators with Triple Graph Grammars. In *International Workshop Graph-Theoretic Concepts in Computer Science*, volume 903 of *LNCS*. Springer, 1995.
94. A. Schürr and F. Klar. 15 Years of Triple Graph Grammars - Research Challenges, New Contributions, Open Problems. In *4th International Conference on Graph Transformation, Proceedings*, volume 5214 of *LNCS*, pages 411–425. Springer, 2008.

95. N. C. Shu, B. C. Housel, R. W. Taylor, S. P. Ghosh, and V. Y. Lum. EXPRESS: A Data EXtraction, Processing, amd REStructuring System. *ACM Transactions on Database Systems (TODS)*, 2(2):134–174, 1977.
96. H. Song, Y. Xiong, Z. Hu, G. Huang, and H. Mei. A Model-Driven Framework for Constructing Runtime Architecture Infrastructures. Technical report, National Institute of Informatics, Japan, Dec. 2008. Technical Report GRACE-TR-2008-05. Available at `http://grace-center.jp/en/rsc_tr.html`.
97. P. Stevens. Bidirectional Model Transformations in QVT: Semantic Issues and Open Questions. In *International Conference on Model Driven Engineering Languages and Systems (MoDELS 2007), Proceedings*, volume 4735 of *LNCS*, pages 1–15. Springer, 2007.
98. P. Stevens. A Landscape of Bidirectional Model Transformations. In *Generative and Transformational Techniques in Software Engineering II, International Summer School, GTTSE 2007, Revised Papers*, volume 5235 of *LNCS*, pages 408–424. Springer, 2008.
99. P. Stevens. Towards an Algebraic Theory of Bidirectional Transformations. In *Graph Transformations, 4th International Conference, ICGT 2008, Proceedings*, volume 5214 of *LNCS*, pages 1–17. Springer, 2008.
100. S. Takahashi, S. Matsuoka, K. Miyashita, H. Hosobe, and T. Kamada. A Constraint-Based Approach for Visualization and Animation. *Constraints*, 3(1):61–86, 1998.
101. J. Terwilliger, L. Delcambre, and J. Logan. Querying through a user interface. *Data & Knowledge Engineering*, 63(3):774–794, 2007.
102. J. F. Terwilliger. *Graphical User Interfaces as Updatable Views*. PhD thesis, Portland State University, 2009.
103. J. F. Terwilliger, S. Melnik, and P. A. Bernstein. Language-integrated querying of XML data in SQL server. *PVLDB*, 1(2):1396–1399, 2008.
104. A. Vallecillo. A Journey through the Secret Life of Models. In *Model Engineering of Complex Systems (MECS)*, number 08331 in Dagstuhl Seminar Proceedings, 2008. `http://drops.dagstuhl.de/opus/volltexte/2008/1601`.
105. G. Varró, A. Schürr, and D. Varro. Benchmarking for Graph Transformation. In *VLHCC '05: Proceedings of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing*, pages 79–88. IEEE Computer Society, 2005.
106. S. Vermolen and E. Visser. Heterogeneous Coupled Evolution of Software Languages. In *Model Driven Engineering Languages and Systems, 11th International Conference, MoDELS 2008, Proceedings*, volume 5301 of *LNCS*, pages 630–644. Springer, 2008.
107. J. Visser. Coupled Transformation of Schemas, Documents, Queries, and Constraints. *ENTCS*, 200(3):3–23, 2008.
108. J. Voigtländer. Bidirectionalization for free! (Pearl). In *POPL '09: Proceedings of the 36th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 165–176. ACM, 2009.
109. G. Wachsmuth. Metamodel Adaptation and Model Co-adaptation. In *ECOOP 2007 - Object-Oriented Programming, 21st European Conference, Proceedings*, volume 4609 of *LNCS*, pages 600–624. Springer, 2007.
110. P. Wadler. Views: a way for pattern matching to cohabit with data abstraction. In *POPL '87: Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 307–313. ACM, 1987.
111. P. Wadler. Theorems for free! In *FPCA '89: Proceedings of the fourth international conference on Functional programming languages and computer architecture*, pages 347–359. ACM, 1989.

112. M. Wang and J. Gibbons. Translucent Abstraction: Safe Views through Bidirectional Transformation, 2008. Available from `http://www.comlab.ox.ac.uk/files/711/Bidi.pdf`.
113. Y. Xiong, D. Liu, Z. Hu, H. Zhao, M. Takeichi, and H. Mei. Towards automatic model synchronization from model transformations. In *ASE '07: Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, pages 164–173. ACM, 2007.
114. Y. Xiong, H. Zhao, Z. Hu, M. Takeichi, H. Song, and H. Mei. Beanbag: Operation-based Synchronization with Intra-relations. Technical Report GRACE-TR-2008-04, Center for Global Research in Advanced Software Science and Engineering, National Institute of Informat iontics, Japan, dec 2008. http://grace-center.jp/downloads/GRACE-TR-2008-04.pdf.
115. XML Beans. http://xmlbeans.apache.org/.
116. T. Yokoyama, H. B. Axelsen, and R. Glück. Principles of a Reversible Programming Language. In *CF '08: Conference on Computing Frontiers, Proceedings*, pages 43–54. ACM, 2008.
117. T. Yokoyama, H. B. Axelsen, and R. Glück. Reversible Flowchart Languages and the Structured Reversible Program Theorem. In *Automata, Languages and Programming, Proceedings, Part II*, volume 5126 of *LNCS*, pages 258–270. Springer, 2008.