ISSN 2186-7437

NII Shonan Meeting Report

No. 2016-13

Bidirectional Transformations

Anthony Anjorin
Zinovy Diskin
Meng Wang
Yingfei Xiong

September 26-29, 2016

National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-Ku, Tokyo, Japan

Bidirectional Transformations

Organizers of the meeting:
Anthony Anjorin (University of Paderborn)
Zinovy Diskin (McMaster University and the University of Waterloo)
Meng Wang (University of Kent)
Yingfei Xiong (Peking University)

September 26-29, 2016

Bidirectional transformations (BX) represent a common pattern of computing: transforming data from one format to another, and requiring a transformation in the opposite direction that is in some sense an inverse. The most well-known instance is the view-update problem from relational database design: a "view" represents a kind of virtual database table, computed on-the-fly from concrete source tables rather than being represented explicitly, and the challenge arises when mapping an update of the view back to a "corresponding" update on the source tables. In a similar way, the problem is central to model transformations and plays a crucial role in software evolution: having transformed a high-level model into a lower-level implementation, one often needs to restore consistency between the two parts that may evolve separately.

Giving this widespread applicability, research on BX naturally spans multiple disciplines: (1) Programming Languages (PL), (2) Graph Transformations (GT), (3) Software Engineering (SE), and (4) Databases (DB).

1. In PL research, the goal is to design languages that are suitable for programming BX (i.e., two opposing transformations are encoded in a single definition rather than with two separate definitions). Notable techniques include lenses – collection of combinators, which can be read in two ways, and bidirectionalization – program transformation that constructs bidirectional programs from unidirectional ones.

2. In GT research, a rule-based approach is taken by specifying consistency as a direction-agnostic graph grammar, i.e., a set of high-level graph transformation rules that generate the language of consistent pairs of graphs. Incremental forward and backward synchronizers with desirable properties are then automatically derived by operationalizing this grammar.

3. In SE research, the goal is to support different software engineering activities with BX. In software development, people usually create different kinds of artefacts, and it has been a long-standing problem to synchronize these artefacts and keep them consistent. By applying BX to these problems, SE researchers have contributed to different aspects of BX research, such as synchronization of object-oriented models, delta-based BX, and exploring the design space of BX.

4. In DB research, BX concerns the classical view-update problem of relational databases, the dual problem of incremental view maintenance, and modern manifestations of synchronization problems such as data exchange and provenance. XML transformations is another active research area of BX, effectively allowing queries to be made bidirectional.

The Shonan meeting proposed in 2016 will build on the momentum and results generated at previous meetings of similar nature (Dagstuhl 2011 and Banff 2013) to further develop cross-disciplinary research agenda and integration effort.

List of Participants

- Faris Abou-Saleh, University of Oxford
- Anthony Anjorin, University of Paderborn
- Dominique Blouin, University of Potsdam
- Zinovy Diskin, McMaster University and the University of Waterloo
- Jeremy Gibbons, University of Oxford
- Soichiro Hidaka, Hosei University
- Frédéric Jouault, ESEO
- Michael Johnson, Macquarie University
- Ekkart Kindler, Technical University of Denmark
- Hsiang-Shang Ko, NII
- Yngve Lamo, Bergen University College
- Ralf Lämmel, University of Koblenz-Landau
- Erhan Leblebici, Technische Universität Darmstadt
- Kazutaka Matsuda, Tohoku University
- James McKinna, University of Edinburgh
- Keisuke Nakano, University of Electro-Communications, Japan
- Andy Schürr, Technische Universität Darmstadt
- Tarmo Uustalu, Tallin University of Technology
- Meng Wang, University of Kent
- Jens Weber, University of Victoria
- Bernhard Westfechtel, University of Bayreuth
- Yingfei Xiong, Peking University
- Vadim Zaytsev, Raincode
- Haiyan Zhao, Peking University
- Hao Zhong, Shanghai Jiao Tong University
- Albert Zündorf, University of Kassel

Meeting Schedule

- Oct. 25th, Sunday Evening
  - Welcome Reception
- Oct. 26th, Monday Morning
  - Opening
  - Self Introduction
- Oct. 26th, Monday Afternoon
  - Overview of BX Approaches
  - Short Research Talks
  - Panel Discussion: State- vs. Delta-Based BX: Pros and Cons
- Oct. 26th, Monday Evening
  - Tool Demos – MOTE and BiGUL
- Oct. 27th, Tuesday Morning
  - Short Research Talks
- Oct. 27th, Tuesday Afternoon
  - Working Groups
  - Panel Discussion: How to Expand BX Beyond the Boundaries of the BX Community.
- Oct. 27th, Tuesday Evening
  - Tool Demos – eMoflon, CX and Hobit
- Oct. 28th, Wednesday Morning
  - Working Groups
- Oct. 28th, Wednesday Afternoon
  - Excursion
- Oct. 28th, Wednesday Evening
  - Banquet
- Oct. 29th, Thursday Morning
  - Working Groups
  - Wrapping up

Overview of Talks

Introduction to BX

Jeremy Gibbons, University of Oxford

I presented an opening lecture introducing representative BX scenarios and approaches. The scenarios included data conversions, for example between a file format and a GUI presentation; the view-update problem in databases; model transformation in model-driven engineering; and the simple "Composers" example from the BX example repository. The approaches included the basic relational model of BX by Meertens and Stevens; varieties of lens from Foster, Pierce, Hofmann etc; generalizations to ordered updates, delta-based BX, and categorical approaches; and Schürr's triple graph grammars. We discussed formal models and their properties; in particular, we focused on forms of property variously called "very well-behavedness", "history ignorance", and "the Put-Put law".

Cyber Security and other New Applications of BX

Michael Johnson, Macquarie University

We already have examples of applications of BX being used for cyber security reasons, and to satisfy privacy and other legislative requirements. Bidirectional transformations also have implications for reverse engineering and other fundamental cyber security problems. BX can be used both to "open up" systems (by giving interconversions between different views of them) and conversely to construct them as safely sealed modules which interact only in a controlled way through bidirectional transformations that can be written and managed with partitioned knowledge (supporting military style "need-to-know" while minimizing the "need" for any one individual or organization). Is it not time for us to jointly start discussing these and other novel applications?

The talk outlined three current approaches to constructing system interoperations via symmetric bidirectional transformations: (1) Symmetric lenses of various kinds (defined up to an equivalence relation), (2) Spans of asymmetric lenses of corresponding kinds (modulo an equivalence relation), and (3) Co-spans of asymmetric lenses or weaker structures (see [Joh07], which already identified some of the implications for confidentiality). Approach (1) involves writing interoperations code (the bidirectional transformation) that understands and operates upon both systems (or at least their APIs) with serious implications for cyber security risks. Approach (2) can be seen as lifting this process (and its negative implications) to the cloud. Approach (3) in contrast permits each system owner to have solely its own staff write code that operates on its own API and there is a clear narrow-band communication, at the head of the cospan, between the programs written by each owner. (This raises new mathematical questions including when such cospans exist.)

The talk then went on to describe a number of less developed ideas including disciplined bug fixing (as suggested by Yingfei Xiong), information leak management and detection, narrow bandwidth obfuscated communication, refactoring, and the internal use of BX for communication between objects in programs to

make them more robust. There are also implications for real-time and reactive systems (in which we develop bidirectional transformations although we can only program one system – the other is a physical system in the real-world).

Reality Check: Does BX Really Help? Model-Code Round Trip as a Case Study

Bernhard Westfechtel, University of Bayreuth

Bidirectional model transformation languages promise to support software engineers in specifying bidirectional transformations concisely at a high level of abstraction. In a bidirectional transformation language, software engineers need to provide only a single specification which may be executed in both directions, ensuring consistent behavior in forward and backward transformations. Furthermore, several languages and tools exist which provide incremental behavior for free, without any need to have it specified explicitly. This talk examines a well-known scenario in model-driven software engineering: the model-code round trip, where the model is defined by a set of UML class diagrams and code is expressed in an object-oriented programming language (e.g., Java). In a model-code round trip, changes may be applied at both ends, implying the need to propagate them to the respective opposite end. Since there are no general frameworks available supporting bidirectional model-to-text transformations, we solve the problem by representing Java code as models and providing bidirectional model-to-model transformations between UML and Java models. We consider three alternatives: an implementation with the help of the TGG Interpreter, based on triple graph grammars, a transformation definition in QVT Relational, executed with the tool Medini QVT, and a hand-crafted triple graph transformation system encoded in Xtend, an object-oriented programming language. Surprisingly, the declarative solutions (TGG and QVT-R based) turn out to be considerably larger than the hand-crafted solution in Xtend, even though both transformation directions and incremental behavior have to be programmed explicitly. In particular, this case study calls for further improvements to the declarative approaches to reduce redundancy among transformation rules.

Profunctor Optics

Jeremy Gibbons, University of Oxford

I summarized the work undertaken by my former student Matthew Pickering for his undergraduate project, which itself was an investigation into the design of lens-like libraries for compositional data access by Edward Kmett, Shachaf Ben-Kiki, Eric Mertens, and Twan van Laarhoven.

A (very well-behaved, asymmetric) *lens* onto a component of type A within a larger structure of type S can be represented as a pair of functions, $get : S \to A$ and $put : S \times A \to S$, satisfying three laws (GetPut, PutGet, PutPut). For example, with $S = A \times B$, the get function projects the left component from a pair, and the put function updates the left component. Dually, a *prism* provides access to data A in a sum type $S = A + B$, via functions $match : S \to A + S$ and $build : A \to S$, again subject to certain laws. Specializing both of these,

an *iso* provides access to a view A of an isomorphic source S, via functions $from : S \to A$ and $to : A \to S$. And generalizing both, a *traversal* provides access to some number of elements of type A in a container of type $S = T(A)$, via functions $contents : T(A) \to A^n$ and $refill : T(A) \times B^n \to T(B)$. (To be precise, the contents and refill functions should be combined, and the common size n existentially quantified.)

However, those four data access mechanisms all have different structures, and they do not compose easily: it is not straightforward to express the composite data accessor "the left component B of the right variant $B \times C$ in the type $A+B\times C$", and similarly for other heterogeneous combinations. The key insight for resolving this incompatibility is to observe that they all have equivalent representations as higher-order functions of type

$$\forall P.\ Profunctor\ P \Rightarrow \forall\ S\ T\ A\ B.\ P\ A\ B \to P\ S\ T$$

fully polymorphic in the type parameters S, T, A, B, but with various specializations of the profunctor parameter P. Here, one can think of the type $P\ A\ B$ for profunctor P as representing "a process that consumes As and produces Bs", so having a functorial action

$$dimap :: (A' \to A) \to (B \to B') \to P\ A\ B \to P\ A'\ B'$$

In particular, the function arrow $(\to)$ is a profunctor. Isos are precisely equivalent to this type; lenses add the constraint that the profunctor should be *strong*, i.e. coherent with product; prisms that it should be a *choice* profunctor, i.e. coherent with sum; and traversals that it should be both. But because all four kinds of data access are now simply functions, of the same structure, they now compose in a completely straightforward way.

Synchronizing Feature Models and Use Cases

Haiyan Zhao, Peking University

Models are widely used in software development to manage complexity and communicate information to various stakeholders. To manage the diverse models used across the whole life cycle for business processes, system requirements, architecture, design, and tests, we have proposed a feature model centric framework with the idea of adopting feature models as core artefacts to organize and manage reusable requirements, constructing traceability relationships between feature models and other models/artefacts in the process. Based on the feature models and the traces between feature models and other artefacts, requirements reuse can be achieved by configuring the feature models first, and deriving the other corresponding artefacts automatically from the feature model configuration.

One of the most challenging aspects of our proposed framework is how to maintain the consistency among the heterogeneous models considering the continuous evolution of artefacts. Bidirectional transformation techniques provide a good opportunity for implementing this kind of maintenance. As a first attempt, this talk investigates how to achieve synchronization between use cases and feature models by programming in BiGUL, a putback-based bidirectional programming language.

Inter-Model Consistency Checking using Triple Graph Grammars and Linear Optimization Techniques

Erhan Leblebici, Technische Universität Darmstadt

Triple Graph Grammars (TGGs) are a rule-based BX language used to describe the consistency of two models together with correspondence links. While TGGs are promising not only for forward/backward transformations but also for consistency checking between two models, the substantial search space involved in determining the "optimal" set of rule applications in a consistency check has arguably prevented mature tool support so far. In this talk, I present the idea of combining TGGs with linear optimization techniques to circumvent this problem. Formalizing decisions between single rule applications of a consistency check as integer inequalities, consistency checking with TGGs can be formulated as a linear optimization problem. Experimental evaluation results with respective tool support show that the approach is applicable to consistency checking between real-world models.

Taking updates seriously

Tarmo Uustalu, Institute of Cybernetics at TUT, Estonia

I will show how "taking updates seriously" takes us from state-based lenses to update lenses and further; we will witness a hierarchy of types of lens that arises in a systematic way. Lenses of each type are characterized either as coalgebras of certain types of co-monads or morphisms between certain types of co-monads. In each case a lens is simulation between two transition systems for suitable notions of transition system and simulation.

Summaries of Tool Demos

MoTE

Dominique Blouin, University of Potsdam

In this demo, I presented the MoTE bidirectional model transformation tool. MoTE takes as input Triple Graph Grammars which are compiled to story diagrams and used as operational specifications for batch and incremental bidirectional transformations as well as model consistency checking. Story diagrams are interpreted at runtime to provide efficient execution of the transformation taking into account runtime characteristics (e.g. cardinalities) of the model elements to be transformed and serving as a guide to derive efficient pattern matching plans. Bidirectional batch transformations were demonstrated for a simple model transformation example and synchronization was illustrated for a more complex language. This consists of synchronizing two models for the AADL language that are expressed using two different metamodels exhibiting important structural differences. This demo showed the difficulty of model synchronization for the complex and rich AADL language and pointed out the need for better expressivity of the TGG language in order to tackle the challenge of model synchronization for industrial tools. A particular case of erroneous model

merge was also illustrated for the EMF Compare tool when only the states of the updated and previous models are considered, while knowledge of the actual change (delta) that has been performed on the model would have provided sufficient information for a correct merge of the models.

BiGUL (Bidirectional Generic Update Language)

Hsiang-Shang Ko, NII

This demo presented a minimalist bidirectional programming language BiGUL (short for Bidirectional Generic Update Language), which is a small collection of asymmetric lens combinators designed for writing put programs. Put transformations are comparable with stateful computation: Implicitly, there are a source state and a view state; using the BiGUL constructs, the programmer gradually changes and matches the shapes of both states, and transfers all information in the view to the source. It was shown in the demo that this way of programming works nicely for specifying list alignment strategies (for which Barbosa et al. had to invent highly specialised "matching lenses" to express). Even though BiGUL is asymmetric, for symmetric problems the programmer can define a co-span by identifying the common information shared by the two sources and writing two BiGUL programs putting the common information into the two sources. Based on this idea, a BiGUL solution to the "cat pictures problem" formulated by Hofmann in the Oxford BX Summer School was briefly presented. The solution was fairly short, achieved the five goals set by Hofmann, and could be extended to exhibit more sophisticated behaviour.

eMoflon

Erhan Leblebici, Technische Universität Darmstadt

In this tool demo, different use cases of Triple Graph Grammars (TGGs) with eMoflon were demonstrated including bidirectional model transformation, model synchronization, and consistency checking based on a small but non-trivial example. Furthermore, the new textual syntax for TGGs as well as different visualization components for models, TGG rules, and other TGG-related data structures were shown. These components are implemented with TGGs as well and make the new eMoflon a completely Eclipse-based and open source tool in contrast to its predecessors.

Logic-Based, Executable Megamodels of Coupled Transformations

Ralf Lämmel, University of Koblenz-Landau

Coupled transformations (CX) are concerned with keeping collections of software artefacts consistent in response to changes of individual artefacts. We capture patterns (scenarios) of CX as megamodels; we enable the instantiation of these patterns for the purpose of testing concrete CX implementations. The approach is implemented in the Prolog-based software language repository YAS. There is a higher-level logic-based megamodeling language LAL which is used

for representing CX patterns including consistency relationships. There is a lower-level logic-based megamodeling language Ueber which is used for representing test cases for CX. The LAL models are translated into Ueber models automatically, subject to configuration for pattern instantiation. The tool demo exercised all relevant aspects of YAS; in particular, illustrative CX examples as well as the underlying megamodels.

HOBiT

Kazutaka Matsuda, Tohoku University

In this tool demonstration, we presented our on-going implementation of the bidirectional programming language HOBiT, which mixes unidirectional programming and bidirectional programming. This not only enables us to compose bidirectional transformations via unidirectional higher-order functions but also gives us more control over robustness. The latter advantage is especially useful for implementing bidirectional versions of certain program transformations that involve complex multiple traversals of long-living but intermediate data structures. An example of such program transformations is alpha renaming; it uses a mapping between old and new variables, which is traversed and updated frequently during the transformation but does not appear in the final result. The idea underlying HOBiT is the Yoneda lemma, which, for instance, gives us a bijective mapping between (well-behaved) lenses and a certain class of ordinary functions [MW15]. Thanks to this underlying theory, we can guarantee well-behavedness of bidirectional transformations defined in HOBiT. We demonstrate the power of HOBiT by using a simplified version of alpha renaming transformation.

Overview of Panels

Panel 1: State- vs. Delta-Based BX: Pros and Cons.

Any sync tool necessarily uses deltas. These deltas can be represented and managed in very different ways, but they should be there (unless the tool invokes some magic). Hence, the state-based setting is a rather abstract math model of the delta-based one. Formally, the former can be seen as a specialization of the latter, in which deltas are pairs of states with a trivial composition (a,b);(b,c)=(a,c). This is a very special specialization, which does not make too much sense for some, but is considered reasonable and useful by others.

Either way, the state-based framework is a math model of inherently delta-based tooling. The goal of the panel was to discuss the pros and cons of this model, and come to a reasonable conclusion on where and when this model should be used (if at all). Each of the panelists was asked to prepare a short 3-4 minute statement with a couple of slides (if needed), and share it with the other panelists and the audience. The first round of presentations was followed by a discussion between the panelists, followed by a discussion with the audience.

Panelists: Ekkart Kindler, Frédéric Jouault, James McKinna, Josh Ko, Soichiro Hidaka, Vadim Zaytsev

Moderator: Zinovy Diskin

Summary.

The panel agreed that deltas are inherent to any BX tool and are somehow detected and managed by the tool. But what part of this internal work is to be accessible, controlled by the user, and governed by well-behavedness laws is a different story.

The discussion additionally revealed the need for a common understanding of basic terminology used in BX. Even the term "'delta" appears to be confusing, especially when compared with other related but different terms such as "diffs", "updates", "edits", "rule applications", and "changes".

Different advantages and disadvantages of abstracting from "internal details of delta handling" were also discussed including:

- Providing, or better, ensuring that explicit deltas are available and accessible can have a positive effect on scalability.
- Stating expectations based on perfect deltas can be misleading and unreasonable, as long as this is not yet the case in practical scenarios. Currently, most tools and technology we have to interface with are state-based, and expecting to get access to explicit deltas is not yet realistic. Doing this can lead to tools that should work well theoretically, but don't in practice.

Panel 2: Yes, we can, but maybe we shouldn't: How to expand BX beyond the boundaries of the BX community.

Synchronization of several artefacts is a practical problem of extreme importance for software engineering and databases. However, not only practitioners, but also researchers developing methodologies and experimenting with tools, manage this problem in ad-hoc ways ignoring the basic ideas and terminology (not to mention methods and results) developed by the BX community. Its not quite clear why this is the case. Maybe the models BX employs are inadequate, and/or the math is too complex, and/or the advertisement is too subtle. Or perhaps its just normal, as the adoption of a new framework takes time.

How should the BX community react to this? One position, arguably wise, is to leave the situation as is, hoping that at some later time BX methods will be recognized and acknowledged by the wider community. Another position is to consider the status quo as something wrong, and try to fix it. Then the question is how: Should we develop a few pilot applications, achieving a breakthrough in some easily marketable domains such as security or healthcare? Or should we develop a series of tutorial papers and tutorials for major conferences? Publish a textbook? Or/and ?

The goal of the panel is to discuss the problem in any relevant context not necessarily limited to the ideas above. Each of the panelists (listed below) was asked to prepare a short 3-4 minute statement with a couple of slides if needed, and share it with the other panelists and the audience. The first round of presentations was followed by a discussion between the panelists, followed by a discussion with the audience.

Panelists: Andy Schürr, Jens Weber, Jeremy Gibbons, Mike Johnson, Ralf Lämmel, Yingfei Xiong

Moderator: Zinovy Diskin

Summary.

The BX Expansion Problem (let's refer to it this way) consists of the following components:

- Enlightening (Teaching and Advertising — "Let them know about BX").
- Developing the BX framework beyond the boundaries of lenses as we understand them now (either state- or delta-based) towards better/wider applicability to practical synchronization scenarios.
- Finding new applications for BX (e.g., cyber-security, healthcare).
- Improving our tools.

In each of these directions, we should be careful not to promise too much! It would be more effective and practical to proceed with a series of small useful steps, which Michael Johnson nicely coined as *hammering* (of course, making bold steps is not prohibited).

A special issue addressed by the panelists was the name "bidirectional transformation", which only captures a part of the problems the BX community deals with, and thus sometimes seems to be hindering rather than promoting. A good name encompassing the entire range of BX problems would be *consistency management* (suggested by Ralf Lämmel). But after an extensive discussion, it was decided to stick with the current name as it is already well branded, and finding a name that "fits all purposes" is hopeless anyway.

A special highlight of the panel was a BX emblem proposed by Jens Weber: a claw hammer that manages nails in a BX way :).

Summaries of Working Groups

Adequate Synchronization of Models after Performing Concurrent Changes

Vadim Zaytsev, Ekkart Kindler

Goal.

When a set of related models which are consistent with respect to some BX are changed concurrently by different people, the set of models needs to be made consistent again after these change. We refer to this as model synchronization.

This is a natural application domain for bidirectional transformations (BX) and is a practical issue in engineering. Still, it turns out that most of the synchronizations in theory as well as in tool implementations are done for two models only. Moreover in the setting with two models, making models consistent after changes is achieved by updating one model only and leaving the other unchanged. A more symmetric setting where synchronization would be achieved by possibly updating all involved models is very often excluded by the general mathematical settings, which would require one model to be the master (which is not changed) and the other the slave (which is changed for making models consistent again).

This working group was supposed to raise the awareness for this blind spot of BX research, and define a more general framework, which does not rule out more symmetric synchronization scenarios, and scenarios in which more than two models are involved.

Summary.

The working group was very successful in raising awareness. There seemed to be general agreement that there is a gap between some important engineering scenarios and what BX theory and tools can actually support.

Even though there exists some theory for symmetric synchronizations – we briefly discussed Triple Graph Grammars and the work of Xiong [Xio09] – there are still many limitations and the actual tool support falls behind the theory. Many other theoretical frameworks such as lenses and co-spans exclude the symmetric setting up-front.

In the working group, we discussed some scenarios, where symmetric synchronization would be needed, e.g., MDA [Coo14] and extended TGGs [KS05]. Moreover, we discussed different mathematical formulations of the problem that would not a priori exclude symmetric and multi-model synchronizations. This would help encourage tool developer and theoreticians alike to fill in this gap.

Some participants of the working group agreed to work on a report that works out some details of the general mathematical framework (and a clarification of terminology) which includes symmetric and multi-model synchronizations.

BenchmarX Reloaded: Concrete Steps Towards an Executable Benchmark for BX Tools

Anthony Anjorin, Ralf Lämmel, Bernhard Westfechtel

Goal.

A BX benchmark (a "BenchmarX") is a standardized problem or test suite that serves as a basis for evaluation and comparison of BX languages and tools. In December 2013 at the Banff BX Seminar, requirements for and the basic structure of a BX benchmark (called a benchmarx) were discussed at the seminar and then proposed in form of a joint publication [ACG+14]. This paper was meant to inspire concrete benchmarx proposals, but up until today there is still no satisfactory benchmarx for even a single example.

Summary.

In this working group, we discussed concrete steps towards an executable benchmark for BX tools, based on the previous conceptual work on benchmarx. A benchmark is executable if a technical infrastructure is provided including test data as well as support for automated tests such that solution providers may plug in their transformations and run them on the provided infrastructure. A major challenge concerning the design and implementation of an executable benchmark is posed by the heterogeneity of BX languages and tools, which

were not only developed against diverging requirements, but also live in different technological spaces. Heterogeneity impedes comparison of solutions both at the conceptual and at the implementation level. To cope with heterogeneity at the conceptual level, an example is needed which may be implemented in a wide variety of BX approaches. As a candidate example, a variant of the well-known families to persons transformation was proposed. The example may be handled in BX tools based on trees, graphs, or models as underlying data structures. It includes several challenges such as loss of information, restructuring, and renaming, such that it may be used as a discriminator with respect to the capabilities of BX approaches. For some candidate example such as the families to persons transformation, a test suite has to be developed which should be structured carefully into groups of test cases such that compatibility levels may be defined. The compatibility levels and respective test cases should be defined according to the capabilities of BX approaches, e.g., batch vs. incremental, deterministic vs. non-deterministic, or state- vs. delta-based. Furthermore, the test suite should be open such that new test cases can be added easily. The designers of the test suite have to face the oracle challenge: for each test case, the expected behavior has to be defined. To this end, general principles such as e.g. least change may be applied. However, the expected behavior may not always be formally deducable. In such cases, the expected behavior should be defined as a result of a community effort, i.e., the community should agree on a plausible result for the respective test case. Altogether, the working group succeeded in developing initial proposals towards an executable benchmark for BX languages and tools, based on previous work on benchmarx. This work will be continued and elaborated, resulting in an actual executable benchmark along the lines determined through the discussions in the working group.

Dependently Typed Edit Lenses on Containers

Jeremy Gibbons, Faris Abou-Saleh, James McKinna

Goal.

Hofmann, Pierce and Wagner introduced an edit language for container types in their formulation for edit lenses. Recently we introduced a dependently-typed formulation of delta-based lenses. We have a skeleton proposal for the corresponding edit language for containers wrt such BX, but many details remain to be settled. We welcome discussion on appropriate language design for compositional editing on rich data structures; similarly for consistency relations/generalized lens complements in the sense of our recent work at BX2016-@ETAPS.

Summary.

We began by reviewing the definitions of containers, and a simple class of transformations between them, known as container morphisms. We expressed these definitions in the language of dependent types, which did not seem to lead to confusion. This suggests that the overheads of using such language is hopefully not too problematic for the audience.

We identified a potential source of confusion about the terminology "containers", which actually refers to type constructors ("lists"), as opposed to the types they construct ("lists of integers") or even values of those types (a particular list of integers). It may sometimes be convenient to abuse terminology and refer to all these as "containers" or "instances of containers", but the discussion highlighted the need for care.

Container morphisms serve as a simple candidate for forward-transformations ("gets") from instances of input containers into output containers; they provide horizontal "correspondences" which trace positions in the output back to their source in the input.

We then asked how these morphisms might be bidirectionalized i.e. how to define an appropriate backwards-transformation ("put"), telling us how updates to the output can be back-propagated into updates on the input. First we have to identify a suitable class of such updates, or "vertical deltas". Then we have to specify how (and when) these deltas are back-propagated.

We largely proceeded with the simplest possibility, essentially "vanilla" container morphisms. (We did not attempt to describe a structured approach to "shape editing" in the vertical deltas — so that the user simply overwrites the old shape — but we identified this as an important point to explore.)

For this notion of vertical delta, we managed to sketch how bidirectionalization would work. The first stage is to identify every possible "new" input shape, that would give rise to the new output shape. For each candidate shape, by definition the forward-transformation tells us how to trace the output positions back into what the input should have been essentially, it fills the new input shape for us. However, the resulting traces may not be consistent; for instance, two output-positions, containing different values, might nonetheless be traced back to the same input position. Moreover, some positions in the new input may not even be referenced by the output meaning that we are free to fill those positions with anything we like. Thus, there may be multiple candidate inputs, or none. However, each candidate allows us to "fill in" a vertical delta from the original input into this new input by following the trace-links around the diagram, all the way back to the original input. This gives a tantalizing sketch of how to complete the back-propagation of the change to the input.

Our working group attempted to tackle an open-ended question with several axes of variation, and naturally we could not explore all of them. (For instance, we could not touch on the question of how to describe sub-shapes and locality.) However we found our way towards a simple BX scenario, by considering simple instances of these possibilities (on each axis) — and we made good progress in sketching how bidirectionalization might work in this scenario.

Design of Bidirectional Programming Languages

Kazutaka Matsuda

Goal.

Since the original lens framework came out, a lot of studies on bidirectional transformations have been done; these studies include finer control on update (edit/delta) translation and customization on backward behaviors. But how can

we "utilize" these results? More specifically, how can programming languages provide us with access to such results? For example, Matsuda and Wang [MW15] recently proposed a higher-order and applicative (non-point-free) programming framework for classical lenses; can we have similar stories for other models of bidirectional transformations? For example, how can a programming language tell users what kind of update is translatable and how it is translated? Can we use a type system or static analysis for this? Another question is: how can we evaluate such bidirectional programming languages? In this working group, we discussed how we can address these questions, and summarized pros and cons of programming language features, and when they are incompatible. Benchmarking examples/problems were also in our focus.

Summary.

In the working group, we discussed three topics:

- Bx-Turing completeness
- A technique to unify some of the bidirectional programming techniques.
- Next goals of bidirectional-programming-language research

It is convenient to have a way to measure the expressive power of a programming language independent from programming paradigms it provides, and without using examples. The well-known notion of Turing completeness is one of such measures. Being Turing complete, a language can express any computation. It must also be convenient to have a similar measure for bidirectional programming languages, like the reversible Turing machine and r-Turing completeness for reversible computation. However, there is a difficulty to have such measures for bidirectional transformations: we expect different bidirectional properties for different purposes. Thus, we might need different measures for different bidirectional properties. For example, for well-behaved asymmetric lenses, a measure would be the ability to define all computable "good" puts [FHP15].

We also confirmed the idea underlying [MW15] and showed how lenses and bidirectionalization approaches are unified by the idea. The idea itself is not limited to asymmetric get-based lenses and would also be beneficial for put-based or symmetric lenses.

There are three major sorts of approaches for bidirectional transformations:

- Constraint-based approaches
- Rule-based approaches
- Approaches based on functional programming techniques

Each has been researched individually and many interesting results have come out from each area. Few studies, however, have been done towards bridging the different approaches, even though such bridging would enable us to enjoy the combined advantages of different approaches. In this discussion, we mainly focused on how we can bridge the second and third sorts.

We found that there is some similarity to some program transformation framework, in which we control where and when we apply individual program

transformations by using a strategy language. This similarity suggests that a direction towards this bridging would be to make rule-based approaches able to invoke bidirectional transformations obtained by functional programming approaches. To make this possible, functional programming approaches, or lenses and bidirectionalization, should be able to define bidirectional transformations that are powerful enough to be used in rule-based approaches.

Regarding bridging of the first and third sorts, we might be able to use rich type systems (such as dependent and refinement type systems) to verify the conformance of defined bidirectional transformations by using such type systems. Moreover, it might be possible to create data conforming to a certain BX by using program synthesis techniques.

Do Lenses Really Address the View-Update Problem?

Zinovy Diskin

Goal.

In a well-known series of empirical studies [WHR+13, HWRK11], Jon Whittle et al. have shown that a major bottleneck for MDE adoption in industry is miscommunication between tool builders and tool users. The latter often misunderstand what a model management tool can actually do, because the former do not have a precise language for specification of scenarios to be addressed by the tool, nor for how these scenarios are actually processed by the tool. Model synchronization can be especially sensitive to this sort of problems, which makes a proper classification of synchronization scenarios an important issue for BX.

It was shown in [DGWC16] that an accurate classification of BX scenarios requires going beyond the classical lens framework: the well-known informational (a)symmetry should be paired with another important BX parameter called organizational symmetry. Roughly, one side of a lens organizationally dominates the other side, if propagating updates from the former to the latter is less restricted (if at all) than the inverse propagation. Pairing two symmetries allows us to formally distinguish several important scenarios: view maintenance vs. view update vs. code generation vs. model compilation vs. reverse engineering, and so on. The total number of different synchronization types formally distinguishable by different combination of two symmetries is surprisingly large and is equal to 24.

The goal of the working group was to discuss these results and their possible applications to both BX theory and BX tooling. Specifically, a technically challenging topic is how to enrich the taxonomic space with concurrent updates (so far, the taxonomy has been built without considering concurrent updates). As for BX tooling, the builder-user miscommunication problem seems to require discussing both the cultural (cf. [WHR+13]) and the technical aspects of BX. Perhaps, we need to talk about a wider-than-technical-view of BX tools.

Summary.

A better alignment between BX tool vendors and MDE users is necessary. To accomplish this, the BX community should (a) go beyond lenses in their cur-

rent form and address new scenarios, and (b) develop a framework, in which synchronization scenarios managed by a BX tool can be accurately specified.

With respect to (a), the group emphasized the need for the following scenarios: (a1) concurrent updates (with and without conflicts), (a2) considering "backward" updates as a possible sync solution, (a3) unidirectional update propagation is more interesting than is usually considered.

With respect to (b), the lens framework is to be augmented with the organizational symmetry dimension (roughly, tagging updates as propagatable or not). Specifically, we need a language for specifying sets of updates to distinguish propagatable and non-propagatable ones. We may refer to this as an *access control language*.

Non-deterministic Bidirectional Transformation

Frédéric Jouault

Goal.

A bidirectional transformation (BX) is generally linked to a consistency relation between several model spaces. For simplicity reasons, we will only consider the binary case here, and call the two related model spaces A and B.

In general, a given model from A may be consistent with several models from B, and vice versa. However, most current BX approaches typically perform the selection of a single model when computing an element of B (resp. A) from an element of A (resp. B), or when updating an element of B (resp. A) to synchronize it in reaction to changes in its corresponding element from A (resp. B). The choice of which model should be selected is typically influenced by additional properties of the BX, for instance:

- A **hippocratic** BX starting from two consistent models will not select a different model from B (resp. A) if no change occurred in the corresponding model from A (resp. B).
- **Least change** will result in a model that is closest to the previous one. As a particular case, no change should be performed on a model from one space after a change in the other space if it is not necessary (i.e., if the two models are still consistent after the change).

In this working group, we attempt to go beyond these limitations along two main aspects:

- **Generalizing model selection**. For instance, we discussed the fact that model selection may actually be seen as an optimization problem. An objective function (e.g., on the models, and/or on the changesets) would then be used to specify which model to select. This objective function may either operate solely on internal information in the models, or also consider external information. The evolution itself may be constrained (not just the models). Traditional BX properties such as hippocracticness or least change could be encoded in such an objective function.

- **Avoiding model selection**. For instance, models may be represented in a parametric way (or with feature models), so that we can consider a whole set of consistent models at once. This is related to the notion of incompleteness in databases. A simple case would be, for instance, to leave some model element properties in an unknown state (i.e., their value could be any value allowed by their type), or to allow them to have any value from a more specific set. It should also be possible to modify such parametric models to change the set of concrete models they represent, and it should be possible to synchronize such changes. Monads and functional logic programming were also mentioned as possible means to deal with nondeterminism, even in functional settings.

A combination of both aspects could be considered, for instance by avoiding selection for some time before ultimately selecting a specific model.

A tentative naming proposal is to call these extended *BX non-deterministic*, or *uncertain*.

Summary.

Several possible representations of the consistency relations have been discussed:

- **Constraints** between the model spaces. Such a representation is basically relational.
- **Coupled grammars** specifying operationally how models from each space can be jointly constructed. The corresponding rules may be incomplete (notion of island grammars). Such a representation is also called rule-based.

A few open questions have been identified:

- What representation(s) of model-spaces would be appropriate, notably in the case where we want to avoid selection?
- Which laws (e.g., related to round-tripping) should such BX follow?
- How should we diagrammatically represent such BX?

Applications and examples of such non-deterministic BX still need to be explored. Software engineering may provide some applications, especially related to forward and reverse engineering. Another aspect that we explicitly decided not to consider in the working group is the possibility that metamodels (or domains, or model spaces) may evolve during the lifetime of the BX.

Can we Put Put-Put to Bed Now?

Michael Johnson

Goal.

In a recent review of types of lenses we note that all but one were explicitly required by their authors to satisfy an appropriately phrased Put-Put law (although sometimes the authors themselves were surprised to see that that was what they had required). (The one exception was "well-behaved set-based lenses".) This is not an attempt to proselytise, but a genuine discussion as there are real empirical questions to be considered. It is an attempt to step back from pre-formed views and see if we can come to common understandings as well as drawing on each others' empirical experiences. It would be good for our community to clarify this topic.

Summary.

The working group began by identifying a need to clarify terminology, avoid prejudice, and analyze carefully, and ultimately it concluded that yes, Put-Put need not be a matter for controversy. Along the way, myths and misconceptions were identified including (1) That Put-Put is bad (in fact it's a very useful property), (2) That Put-Put is always there if you look at it the right way (while that's sometimes true, there are real cases, including the parable of Jeremy's red shoes, which violate Put-Put, (3) several misconceptions related to Deltas (that led to some of us wanting to clarify the terminology), (4) misconceptions about compositions (including mistaken assumptions that they are codiscrete, or free), and finally the misremembered "theorem" that Put-Put implies constant complement (and this is probably the source of myth number (1) above).

After clarifying terminology including basic terms like delta, update and propagation, along with the need to be clear about atomicity (what size of update does it make sense to propagate), we spent some time discussing incrementality but ultimately passed that off to another working group.

Finally, we reviewed Put-Put as implying a *coherent* update strategy, we noted how Put-Put therefore supports incremental analysis and we laid out the definition of Put-Put as propagation that respects composition, leading to the observation that Put-Put is important for Least-Surprise. In this light, a careful analysis of the parable of Jeremy's red shoes leads to useful understanding of why Put-Put fails, proposals for how it might be restored, and a warning that our role is to inform and satisfy a client, not to dictate whether or not their updates need to be coherent.

References

[ACG+14] Anthony Anjorin, Alcino Cunha, Holger Giese, Frank Hermann, Arend Rensink, and Andy Schürr. BenchmarX. In S. Hidaka and J. Terwilliger, editors, *Bidirectional Transformations (BX 2014) - Workshops of the EDBT/ICDT 2014 Joint Conference*, volume 1133 of *CEUR Workshop Proceedings*, pages 82–86, Aachen, Germany, March 2014. CEUR-WS.org.

[Coo14] Steve Cook. Domain-specific modeling and model driven architecture. *MDA Journal*, pages 2–10, Jan 2014.

[DGWC16] Zinovy Diskin, Hamid Gholizadeh, Arif Wider, and Krzysztof Czarnecki. A three-dimensional taxonomy for bidirectional model synchronization. *J. Syst. Softw.*, 111(C):298–322, January 2016.

[FHP15] Sebastian Fischer, ZhenJiang Hu, and Hugo Pacheco. The essence of bidirectional programming. *Science China Information Sciences*, 58(5):1–21, 2015.

[HWRK11] John Hutchinson, Jon Whittle, Mark Rouncefield, and Steinar Kristoffersen. Empirical assessment of mde in industry. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 471–480, New York, NY, USA, 2011. ACM.

[Joh07] Michael Johnson. Enterprise interoperability: New challenges and approaches. In *Enterprise Software with Half-Duplex Interoperations*, pages 521–530. Springer-Verlag, 2007.

[KS05] Alexander Königs and Andy Schürr. Multi-domain integration with mof and extended triple graph grammars. Dagstuhl Seminar Proceedings 04101 `http://drops.dagstuhl.de/opus/volltexte/2005/22`, 2005.

[MW15] Kazutaka Matsuda and Meng Wang. Applicative bidirectional programming with lenses. In *Proceedings of the 20th ACM SIGPLAN International Conference on Functional Programming*, ICFP 2015, pages 62–74, New York, NY, USA, 2015. ACM.

[WHR+13] Jon Whittle, John Hutchinson, Mark Rouncefield, Håkan Burden, and Rogardt Heldal. *Industrial Adoption of Model-Driven Engineering: Are the Tools Really the Problem?*, pages 1–17. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[Xio09] Yingfei Xiong. *A Language-based Approach to Model Synchronization in Software Engineering*. PhD thesis, The University of Tokyo, 2009. `http://sei.pku.edu.cn/~xiongyf04/papers/PhDThesis.pdf`.