Dagstuhl Seminar on Bidirectional Transformations (BX)

Zhenjiang Hu
National Institute of Informatics, Japan
hu@nii.ac.jp

Andy Schนrr
Technische Universitat Darmstadt
andy.schuerr@es.tu-darmstadt.de

Perdita Stevens
University of Edinburgh
perdita@inf.ed.ac.uk

James F. Terwilliger
Microsoft Corporation
james.terwilliger@microsoft.com

1. OVERVIEW

The Dagstuhl BX seminar, held January 16–21, 2011, brought together researchers from 13 countries across disciplines that study bidirectional transformations. It was a follow-up of the GRACE International Meeting on Bidirectional Transformations held in December 2008 near Tokyo, Japan [5]. This consisted of short introductions from each of the participants on their background and work, followed by presentations and demonstrations on representative technologies from each field, and some open discussion time. A major benefit of the GRACE meeting was the opportunity for the disciplines to get some initial exposure to each other.

The Dagstuhl seminar intended to go a step further and begin to identify commonalities between the disciplines and start to set a cross-disciplinary research agenda. The first part of the seminar consisted of tutorials from each of the four represented disciplines. The second part consisted of cross-disciplinary working groups dedicated to investigating specific examples of commonality between solutions or identifying requirements, terminology, or scenarios that may reach across fields. There were also sessions in which participants gave position statements on their own work.

Participants at both the Dagstuhl and GRACE seminars came from four disciplines: (1) Programming Languages, (2) Graph Transformations, (3) Software Engineering, and (4) Databases. At Dagstuhl, each of the first three disciplines made up about 2/7 of the participants, while databases took the remaining 1/7 out of about 45 participants. Representation from the database field was, nevertheless, an improvement over the turnout from the GRACE meeting.

2. TUTORIALS

The seminar opened with in-depth tutorials by representatives from each of the four disciplines on the various solutions to bidirectional transformation problems offered by that discipline. In this section, we present an overview of the material presented at those sessions.[1]

[1]For a complete list of all presentations and working groups held at the seminar, consult the Dagstuhl site at http://bit.ly/dagstuhl-bx.

Programming Languages (PL)

Recently, many linguistic approaches have been proposed for describing bidirectional transformations in the programming languages community [6, 3]. An easy but non-ideal approach is to write a pair of transformation functions, one in each direction. Although this approach can use existing languages and works for simple transformations, scalability and maintenance are difficult. Other approaches specify both transformations with a single description. This tutorial surveyed recent work and presented promising approaches in detail.

First, Nate Foster gave an overview of design choices in languages for bidirectional transformations and introduced a common vocabulary for comparing different approaches. Important design choices include state-based versus operation-based updates, the set of constraints on the forward and backward transformations to ensure that they work well together, whether to allow freedom to choose a backward transformation for a forward transformation, and whether the backward transformation should handle every pair of data sources.

Second, Robert Glück described *reversible computing and reversible languages* [18]. Reversible computing is the study of computing models that exhibit both forward and backward determinism, while reversible languages are used to describe injective functions that can be effectively inverted. Reversible languages include explicit postcondition assertions, the ability to “un-call” a reversible procedure, and the possibility of clean and garbage-free computation of injective functions.

Janis Voigtländer described a *complement-based approach* to bidirectionalization [17] that automatically constructs a backward function from a forward function based on derivation of a constant-complement. Three methods are introduced: algorithmic generation based on the syntactic representation of the forward function, a semantic approach from a polymorphic forward function using parametricity and free theorems, and an integration of the previous two.

Next, Benjamin Pierce described *lens combinators* [3], the use of type systems to establish well-behavedness, and the issues surrounding the handling of ordered data. The fundamental concepts of bidirectional programming were explored in the simplest imaginable setting, where data are strings, types are regular expressions, and computation is finite state transduction. Their design emphasizes both robustness and ease of use, guaranteeing totality as well as strong well-behavedness conditions formulated as round-tripping laws.

Finally, Zhenjiang Hu introduced *trace-based bidirectionalization*, which is used for *bidirectionalizing graph transformations* [10]. This talk explained the basic idea of trace-based bidirectionalization, and showed how to compute traces effectively and how to propagate the view updates to the original database safely.

Databases (DB)

The *query-defined view* is a widely studied way to express a relationship between two models. Using queries to define model relationships has several benefits, including well-understood formal properties of query languages, the availability of robust and mature implementations, and the convenience of using the same language to specify both queries and mappings.

Despite copious research (e.g., [2]), practical support for updatable views in commercial systems falls well short of what research offers. Usability is among the reasons for this. Many developers prefer to specify the reverse mapping to a view manually as triggers. Also, it is difficult to provide constructive feedback as to why a given query is not reversible and how to fix the problem.

The database tutorial, presented by Jean-Luc Hainaut, Anthony Clève, and James Terwilliger, focused on alternatives for constructing updatable or bidirectional mappings. For instance, an *Object-Relational Mapping* (ORM) constructs an updatable object-oriented view of relational data. ORM tools are limited to transformations that are updatable (e.g., horizontal and vertical partitioning) and useful for bridging the object-relational impedance mismatch (e.g., inheritance mapping strategies like Table-per-Hierarchy) [12].

Data Exchange mappings are expressed using queries in a subset of first-order predicate calculus [1]. Research on inverting such mappings has expanded to include definitions of invertibility beyond those covered by classical updatable views. A mapping $\mathcal{M}$ may have an inverse $\mathcal{M}^{-1}$ where $\mathcal{M}^{-1} \circ \mathcal{M}$ is not the identity, but $\mathcal{M} \circ \mathcal{M}^{-1} \circ \mathcal{M} \equiv \mathcal{M}$. At least three such notions of inverse have now been studied, and each has relative advantages and disadvantages.

One characteristic that query-defined views, ORMs, and data exchange mappings all have in common is a top-down, holistic specification. The alternative approach to specifying a bidirectional mapping declaratively is to construct a mapping out of incremental components. Each component has proven bidirectional properties (not unlike a lens [4]), as well as other properties that make the component well-suited to the particular domain. For instance, in the *channels* framework, each component is capable of transforming schema evolution primitives and constraint declarations as well as queries and updates against its view schema [16].

A prime example of a component-based solution is *DB-MAIN*, an approach for managing multiple model artifacts even across levels of abstraction [8]. It can establish a relationship between a conceptual model, its corresponding logical model, and that model's corresponding physical model. Components in the *DB-MAIN* model describe how to translate constructs in one level of abstraction into another, and can thus handle most of the operations typically handled by ORMs.

Graph Transformations (GT)

Graph grammars were invented in the 1970s as a generalization of Chomsky Grammars. Graph transformation tools are often used for the formal specification of in-place model transformations or unidirectional transformations between different modeling and graph languages. *Triple Graph Grammars* (TGGs) have been developed for declarative and rule-based description of bidirectional transformations between related graph languages [15]. Formally, a TGG describes a language of graph triples with the first components being elements from the source language, the second components being instances of traceability relationships between source and target language elements, and the third components being elements from the target language.

The first part of the GT tutorial, given by Andy Schürr, introduced the basic ideas of, and motivations for, the TGG formalism. It allows for the high-level description of functional and non-functional relationships between pairs of graphs. A family of graph transformations is derived from such a TGG specification that supports batch transformation and incremental change propagation scenarios in both directions as well as checking the consistency of given pairs of graphs. Traceability relationships between elements of related pairs of graphs are created and updated as a side effect [11]. The first part of the tutorial thereby prepared the ground for the second part presented by Frank Herrmann. This part sketched the formal background of TGGs based on category theory and related techniques for the verification of important properties of TGGs and derived graph translators. It also introduced a number of analysis techniques for the formal verification of desirable properties of TGGs including:

- *Correctness* which guarantees that graph tuples resulting from generated graph translators are instances of the relevant TGG language

- *Completeness* which guarantees that the generated graph translators can always translate updates of schema-compliant source graphs into updates of schema-compliant target graphs (and vice-versa)

Related techniques are also used for detection and resolution of conflicts between different translation rules and for improving the efficiency of generated engineering tool integrators [9].

Software Engineering (SE)

Model transformations are a key ingredient of model-driven development[13], a paradigm in which most design decisions are embodied in (graphical) models rather than in (textual) code. Several different models may be used in conjunction, each in a notation chosen to suit a particular task. Models, and code, may be partly or completely generated from other models using transformations. In the simplest case, a transformation may be just a kind of compilation: for example, code may be generated from a model by a transformation, and if the model changes, the code can be generated afresh.

As Krzysztof Czarnecki explained, the question of whether an individual model is valid is already quite complex; a model must meet both local structural requirements and constraints which may be global. One of the challenges for a model transformation language, even in the relatively simple unidirectional case, is that the transformation must not modify a model in such a way as to make it cease to conform to its metamodel.

The central challenge for this Dagstuhl meeting, however, was bidirectionality. There will typically be human input into both models, and each model will embody information that is not representable in the other. That is, this typical situation is *symmetric*. The job of the transformation is now to *maintain consistency*, bidirectionally. An important subproblem is to be able to check consistency – bearing in mind that the consistency relation may be non-bijective – and for this reason bijective model transformation formalisms tend to be relational in the sense that the consistency relation is more clearly apparent from the notation than the procedures for restoring consistency.

Relations in such a formalism (e.g., QVT-R [14]) are superficially similar to rules in graph transformations. There is a notion of matching a pattern in a model, that is, identifying a part of a model which is relevant to a particular relation; each match provokes some check of, or modification to, the other model. Beyond this opinions and formalisms differ widely. There may or may not be a well-defined corresponding part of the other model, which may or may not be recorded in some kind of linking structure. A linking structure may be formally defined as part of the definition of the transformation language, may be left implicit, or may have to be constructed by a human with heuristic tool support.

A related issue is the use, or not, of a record of the changes made to a model. Such a record can ease consistency maintenance at some pragmatic cost. As one point in the space of design possibilities, Stephan Hildebrandt introduced the language MoTE [7], which uses a combination of explicit link information and change notification to manage consistency.

3. WORKING GROUPS

What follows is a sampling of the seven working groups that met, and some key conclusions from them.

Taxonomy and Scenarios

Two groups focused on documenting commonality across disciplines by examining common terminology and scenarios. Despite working towards similar and sometimes identical research goals, the words used to describe concepts in those disciplines were vastly different. For instance, the word “model” in one discipline corresponds to “meta-model” in another discipline, and to “instance” in yet another discipline. Also, for a given scenario — say, object-relational mappings — different disciplines had different pivot points of research; GT focuses on managing model relationships via grammar rules expressed at the meta-model level, while DB focuses on managing instances via formal properties expressed at the model level. The documentation of these groups will hopefully serve as fodder in future workshops on establishing a bidirectional transformation benchmark.

Mathematical Foundations of Lenses

The group on mathematical foundations of lenses served as a tutorial to most of the group on the connections between lenses and mathematical formalism. Michael Johnson demonstrated how lenses can be represented mathematically either as monads or as co-monads. Using these mathematical formalisms, one can prove why (and when) certain properties of lenses are important. One can also leverage monads to prove or discover properties of lenses that might not be immediately apparent.

One unfinished line of thought surrounded a potential link between lenses and data exchange. Database literature formalizes a data exchange mapping using predicate calculus, but practitioners outside the database field sometimes formalize similar mappings as a pair of adjacent lenses with opposing polarity. A mathematical link between lenses and data exchange may be intrinsically interesting, and might yield new results when considered within monadic formalism as well.

Reversible Programming and Graph Transformations

This group started with a broader discussion and classification of concepts for specifying and implementing a bidirectional programming language. Janus was, among other things, used as a running example for this purpose. The focus later turned towards a comparison of the pros and cons of BX programming languages like Janus on one hand and TGGs on the other hand. The conclusion was drawn that languages like Janus are well-suited for handling projections and arithmetic operation, but have problems handling complex data structures. TGGs are exactly the opposite, so the combination of these two lines of research seems promising.

4. FUTURE WORK

We went to Dagstuhl knowing that longer-term ideas like a common research agenda or benchmark would take more than a single week. The participants decided on several follow-up actions to keep work progressing:

- A follow-up meeting in the same style as GRACE and Dagstuhl to continue collaborating on a cross-disciplinary research agenda
- Workshops at conferences associated with each discipline to work toward specific, targeted goals (a first one has already been scheduled associated with GTTSE 2011, and will focus on developing a benchmark[2]; a second follow-up event has just been accepted as a satellite workshop for ETAPS)
- Tutorials and other education-minded events at conferences to continue bringing awareness of bidirectional solutions from other disciplines, as well as awareness of the general BX effort
- Smaller-scale research cooperations that combine techniques from different fields like merging concepts from bidirectional programming languages and triple graph grammars as envisaged in one of the seminar's working groups.

In particular, a goal of the upcoming seminars and workshops is to increase database community participation. The bidirectional transformation problem has origins deep in the database community [2], but has grown so that solutions are being driven from many directions in different fields across computer science. The plan is to hold some of the tutorials or workshops at database venues to help solicit ideas and opportunities for collaboration; details will be made available once they are scheduled.

5. REFERENCES

[1] M. Arenas, P. Barceló, L. Libkin, F. Murlak. Relational and XML Data Exchange. *Synthesis Lectures on Data Management.*

[2] F. Bancilhon, N. Spyratos. Update Semantics of Relational Views. *ACM Transactions on Database Systems*, December 1981, 6(4).

[3] A. Bohannon, J. N. Foster, B. C. Pierce, A. Pilkiewicz, A. Schmitt. Boomerang: resourceful lenses for string data. *POPL 2008.*

[4] A. Bohannon, B. C. Pierce, J. A. Vaughan. Relational lenses: a language for updatable views. *PODS 2006.*

[5] K. Czarnecki, J. N. Foster, Z. Hu, R. Lämmel, Andy Schürr, J. F. Terwilliger. Bidirectional Transformations: A Cross-Discipline Perspective. *ICMT 2009.*

[6] J. N. Foster, M. B. Greenwald, J. T. Moore, B. C. Pierce, A. Schmitt. Combinators for bi-directional tree transformations: a linguistic approach to the view update problem. *POPL 2005.*

[7] H. Giese, S. Neumann, S. Hildebrandt. Model Synchronization at Work: Keeping SysML and AUTOSAR Models Consistent. *Graph Transformations and Model Driven Enginering — Essays Dedicated to Manfred Nagl on the Occasion of his 65th Birthday*, LNCS 5765.

[8] J.-L. Hainaut. The Transformational Approach to Database Engineering. *GTTSE 2006.*

[9] F. Hermann, H. Ehrig, F. Orejas, U. Golas. Formal Analysis of Functional Behaviour of Model Transformations Based on Triple Graph Grammars. *Proc. Int. Conf. in Graph Transformation ICGT 2010.*

[10] S. Hidaka, Z. Hu, K. Inaba, H. Kato, K. Matsuda, K. Nakano. Bidirectionalizing graph transformation. *ICFP 2010.*

[11] F. Klar, M. Lauder, A. Königs, A. Schürr. Extended Triple Graph Grammars with Efficient and Compatible Graph Translators. *Graph Transformations and Model Driven Enginering — Essays Dedicated to Manfred Nagl on the Occasion of his 65th Birthday*, LNCS 5765.

[12] S. Melnik, A. Adya, P. A. Bernstein. Compiling mappings to bridge applications and databases. *ACM Trans. Database Syst.*, 33(4).

[13] Object Management Group. MDA Guide V1.0.1. *omg/03-06-01*

[14] Object Management Group. Queries, Views and Transformations. *formal/2011-01-01*

[15] Specification of Graph Translators with Triple Graph Grammars. *Proc. Int. Workshop on Graph-Theoretic Concepts in Computer Science (WG 1994).*

[16] J. F. Terwilliger, L. M. L. Delcambre, D. Maier, J. Steinhauer, S. Britell. Updatable and Evolvable Transforms for Virtual Databases. *PVLDB 3(1).*

[17] J. Voigtländer, Z. Hu, K. Matsuda, M. Wang. Combining syntactic and semantic bidirectionalization. *ICFP 2010.*

[18] T. Yokoyama, H.B. Axelsen, R. Glück. Reversible flowchart languages and the structured reversible program theorem. *ICALP 2008.*

[2]http://www.di.univaq.it/CSXW2011/