BX Theory and Applications Across Disciplines: Challenge Problems

Assessing Bidirectional Model Transformation Tools (Alcino Cunha, Minho University)

From the end-user perspective it is currently very difficult to understand the expressiveness of (and differences between) existing bidirectional model transformation tools. This makes it difficult to pick a tool for the task in mind, and helps hinder the adoption of BX by general software engineering practitioners. Even for researchers in the field, it is sometimes difficult to compare approaches to BX due to the lack of a common vocabulary among different BX communities (one of the challenges this BIRS workshop intends to address). To alleviate this problem, we are currently putting together an example-driven comparative study of existing bidirectional model transformation tools. The idea is to define a set of very simple (but illustrative) examples of bidirectional transformations, together with a set of unit tests for each example, and then (try to) implement them in different tools and evaluate the results. The current focus is on expressivity: trying to assess the class of examples each tool is suitable to implement, and which bidirectional laws it guarantees. Of course, this study will only be truly useful if it is a community effort, and I would like to challenge current tool developers to join it (implementing the examples in their own tools, and suggesting other examples and unit tests).

A Transformation Zoo for BX (Erhan Leblebici, Technical University of Darmstadt)

Incremental model synchronization is a challenging task in a BX context. Depending on the synchronization scenario, a particular aspect (e.g., automization, decision making, information preservation, language capabilities, efficiency) might be more interesting/relevant than others. A set of exemplary synchronization scenarios is necessary to determine the capabilities of a particular approach. Experience over the years has shown that well-known BX-examples within particular groups, which share a common theoretical foundation (e.g, Triple Graph Grammars), are quite useful for researchers and tool developers to understand challenges. Establishing a transformation zoo beyond specific approaches would be a huge step towards a systematic comparison and understanding of different incremental BX approaches.

Computability of Combinator-Based BX (Yingfei Xiong, Peking University)

Many BX languages are built upon a set of combinators, and many new languages are being designed based on new sets of combinators. It is natural to ask to what extent this approach could reach: what computational power could a combinator-based language possibly reach? For example, could a combinator-based BX language express all well-defined lenses that could be described by a Turing machine? It does not seem so because the totality requirements require guaranteed halts, but a formal proof is needed. Furthermore, is there a weaker form of computation model that a combinator-based BX could possible achieve? If so, what would it be?

What Can BX Make from Sense? (Arend Rensink, Twente University)

As a small case study I have been working on an M2M transformation which should be bidirectional - but I know of no way to specify it so that it can be both executed and proved correct. The target language is richer than the source language, so the backwards transformation is left but not right inverse to the forward transformation. I can provide Ecore metamodels for the source and target formats, and three example models in both formats; in the presentation, I will define the problem more precisely.

Challenging BX as a Software Engineer! What is a problem? How much BX do we need? (Ekkart Kindler, Technical University of Denmark)

We often claim that certain relations between some artifacts (models) in the software engineering process should be formalized and that these transformations need to be bi-directional. In many situations, however, a complete formalisation is very expensive, or conceptually not even possible to define a priori. And bi-directionality of transformations might solve some problems of software engineering, but is actually not strictly necessary -- some similar but weaker properties would do. This presentation should challenge us, to identify in a clear way which tasks of software engineering really need bi-directional transformations and justify the costs of formalizing it, or whether "cheaper" forms of transformations or even informal relations would do.

Here are some examples to look at
- "MDA model chain": CIM, PIM, PSM
- code generation & integration with manual changes
- model/code migration
- traceability

Model/Implementation Synchronisation from a Programming Language Design Perspective (Meng Wang, Chalmers University)

In an ideal situation, any modification to the software is done through the model, and the implementation is simply regenerated to reflect the change. However, for a variety of reasons one often needs to reverse engineer the above process to produce a revised high-level model from an updated implementation. This problem is rather central to software evolution, but is less explored in the context of Domain-Specific Languages (DSLs), which are frequently developed as source-to-source translations, where the program written in the DSL is the high-level model and the code generated (in a different language) is the implementation. Without the ability to synchronise the domain-level program with a changed implementation, one will be forced to abandon the DSL approach altogether once the generated code is modified. I will start the discussion by showing examples of DSLs, and hope to spark exchanges among different communities who are facing similar problems.