Bidirectional Synchronization of Multiple Views of Software Models

Miguel Garcia

Institute for Software Systems (STS)
Hamburg University of Technology (TUHH), 21073 Hamburg
http://www.sts.tu-harburg.de/~mi.garcia

Abstract: Current best-practices for defining Domain-Specific Modeling Languages call for metamodeling techniques, which do not take into account the future use of such languages in multiview design environments. Tool implementers have tried a variety of ad-hoc techniques to maintain views in-synch, with modest results. To improve this state of affairs, a declarative approach is elaborated to automate multiview synchronization, building upon existing metamodeling techniques and recent advances in the field of function inversion for bidirectionalization. An application of these ideas to EMOF and a discussion of the resulting Declarative MVC software architecture are also provided. A significant benefit of the approach is the resulting comprehensive solution to a recurrent problem in the software modeling field.

1 Introduction

Most modeling languages provide different visual notations to highlight different aspects of the System Under Development (SUD). Most notably, UML2 defines a total of thirteen diagram types, grouped into three categories (structure, behavior, and interaction). In general, the same situation arises for Domain-Specific Modeling Languages (DSMLs). There is thus no escape from using several notations when modeling non-trivial software systems, a fact that vendors of modeling tools acknowledge by providing multiview capabilities. At some point in the development process the issue of *inter-view consistency* [Egy06] requires automation due to the complexity of the SUD. For example, determining consistency between a sequence diagram and the statechart for a single traffic-light may be done manually. However, tool support is required for models of realistic complexity (railroad crossings, reservation systems, consumer electronis, etc.)

The definition of a modeling language that introduces views is thus expected to provide an algorithm to determine whether a set of views is consistent. Using metamodeling terminology, the check for consistency is formulated as follows: (a) for each diagram type a metamodel has been defined, whose instances constitute the views manipulated by the modeler, including geometric information; (b) each such metamodel defines its intra-view Well-Formedness Rules (WFRs); and (c) additional WFRs ensure consistency encompassing several views. Given that WFRs are boolean-valued predicates over an object population, a yes/no answer can be provided about the consistency of the *integrated model*, i.e.,

the set of all views prepared by the modeler. Unless inter-view consistency is addressed at the level of the language definition itself, disagreement will otherwise ensue. For example, the workshop series *Consistency Problems in UML-based Software Development* was devoted to overcoming such disagreement for UML 1.x.

As useful as they are, yes/no answers about consistency contribute only partially to productivity. In a multiview setting, additional use cases demand automation (*multiview synchronization*, *model refactoring* [MB05, CW07], and *model completion* [SBP07]). In this paper, we address the multiview synchronization problem (defined below), leveraging on the lessons learnt from the related problem of inter-view consistency: we rely on a formal technique and address this concern at the language definition level.

Keeping multiple views in-synch requires propagating changes in two directions: (a) change requests validated against the WFRs of the integrated model are to be reflected on views; and (b) user-initiated *view updates* are to be processed in the opposite direction. The algorithm for realizing (a) is fixed once a *view definition* is available: given that the integrated model includes geometric information, updating views amounts to evaluating a function again. The situation is not so simple for (b), where partial information is available. For example, a particular view definition may select only those items at odd-numbered positions in a list. Inserting into the view then raises the question as to where to add an item in the underlying list (which is part of the integrated model). Such kind of decision problems are not solved by the current best-practices around tool implementation: Model-View-Controller architecture (MVC), runtime evaluation of WFRs expressed in OCL, transparent undo/redo. Rather, the particular realization of (b) is left to the criteria of tool vendors, thus opening the door to non-standard implementations. Our contribution in this paper improves on this state of affairs, not by building a tool with multiview synchronization capabilities (which is a task for industry) but by disclosing the inner workings of such solution (which industry refrains from doing).

1.1 Benefits of the proposed approach

The lack of tool compatibility (and sometimes correctness) around multiview synchronization stems from the fact that the specific policy governing synchronization is encoded manually in the Controller module of MVC (by each tool vendor, usually in an imperative language). In contrast, a declarative formulation, available as part of the language definition itself, allows both generating such implementation as well as statically analyzing the bidirectional transformations at design time. We call this approach DMVC, for Declarative MVC. The resulting productivity gain is particularly relevant for DSMLs, as the cost of developing tooling for them has to be amortized over a much smaller number of projects than for their general purpose counterparts. The DMVC approach is in line with recent advances in the definition of visual notations, where geometric constraints are used at runtime to automate the maintenance of diagram layouts, as discussed in detail by Grundy [LHG07] and exemplified in an Eclipse-based modeling tool generator[1].

[1] Marama meta-tools, `https://wiki.auckland.ac.nz/display/csidst/Marama+Tatau`

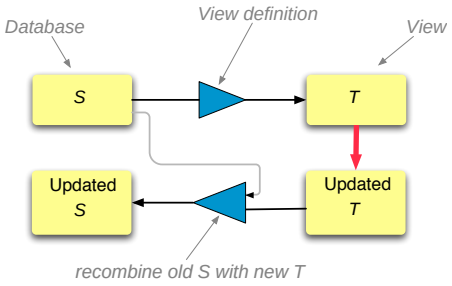

Figure 1: A bidirectional transformation, consisting of forward and backward functions [Pie06]

Our proposed architecture for DMVC builds upon a bidirectional transformation engine fulfilling formal guarantees. For example, given a view definition written by the DSML author, the engine can automatically derive its corresponding backward transformation. Importantly, the backward transformations [Ste07] thus obtained can cope with many-to-one mappings (i.e., non-injective functions, where different inputs are mapped to the same output, as for example in $f(x, y) = x + y$). This is achieved with *stateful transformations*, which track the information needed to complement that lost by the mapping (in the example, keeping a copy of either x or y allows handling user updates to $x + y$). Additionally, it is common practice for a backward transformation to take as input, besides the updated view, the original source. The intuition behind this scheme is depicted in Figure 1.

Statefulness and recombination distinguish our problem space from plain function inversion, which is enough in the particular case where each of (source, view) can be fully reconstructed from the other, i.e., whenever there is a one-to-one correspondence (a bijection) between source and target domains. As argued by Stevents [Ste07], such situation constitutes the exception rather than the rule in multiview modeling languages. For example, *dual syntaxes* [BMS07] do exhibit this property: a human-oriented syntax is defined for pretty-printing Abstract Syntax Trees (ASTs), while an accompanying XML-based syntax is defined for tool interchange. An MVC editor displaying a dedicated view for each representation needs no further information than the contents of an updated view in order to refresh the other, as no information is elided in the alternative syntaxes. In terms of our adopted approach, bijections are handled the same as the non-injective case (the latter being the "interesting" one from the point of view of multiview synchronization).

In summary, bidirectional transformations increase the productivity of the tooling process for DSMLs, and the quality of multiview environments. The structure of this paper is as follows. Competing methodologies around view update are reviewed in Sec. 2, followed in Sec. 3 by the application of one of them to the multiview synchronization problem, comprising the definition of EMOF-level operators for view specification (Sec. 3.1) that are well-behaved from a bidirectionalization point of view (Sec. 3.2). Given that any realistic multiview modeling tool will rely on available MVC frameworks, these practical aspects are discussed in Sec. 4. By placing the reported techniques in perspective, Sec. 5 reviews our contributions and concludes. Knowledge is assumed about language metamodeling, a background in functional programming is helpful but not required.

2 Candidate approaches distilled

All the approaches reviewed in this section share the common goal of enabling bidirectional transformations between pairs of data structures, with differences spanning the preferred representation (e.g., unordered trees vs. semi-structured data) and the available transformation operators (which may or may not allow user updates to alter the transformation as a side-effect). Besides highlighting the innovative aspects of each technique, their comparison allows introducing terminology to better characterize the multiview synchronization problem. However, the reader interested in the *resulting* Declarative MVC architecture may focus on Sec. 2.1 and skip the remaining subsections, moving directly to Sec. 3.

For ease of reference, candidate approaches are loosely grouped into (a) general purpose techniques (program inversion, data synchronization, and virtual view update); and (b) techniques aiming at supporting model transformations (QVT-Relations and graph-grammar based). This classification has more to do with the current level of adoption in the model-driven community than with any inherent capability of each approach.

2.1 Program inversion, Data synchronization, and Virtual view update

Program inversion. We discuss this technique first as it constitutes the basis for Declarative MVC. *Program inversion* [MHN+07] in the context of functional programming refers to determining, given a function $f(x_1, \ldots x_n)$, its inverse, so as to obtain the arguments given a result. For our problem at hand, an insight consists in choosing the building blocks for expressing view definitions such that they fulfill three *bidirectional properties*:

a) *Stability*: unmodified views are transformed back into the same source that gave origin to them (i.e., backward transformations introduce no spurious information);

b) *Restorability*: all updates on a source (that affect a view) can be canceled by updates on the view (i.e., the user has means to restore the integrated model to a previous state by just acting on the view)

c) *Composability*: the backward transformation is oblivious to the order in which updates took place (what counts is the end state).

Moreover, any composition of building blocks fulfilling these properties defines again a well-behaved bidirectional transformation. A Haskell implementation of an algorithm for the above is available[2]. We do not reproduce here the theory behind this algorithm, which is proficiently covered by Matsuda et. al. in their ICFP 2007 paper [MHN+07].

Program inversion fulfilling the above properties has been applied to particular cases: Liu [LHT+07b] presents a Java library for the bidirectional transformation of XML documents

[2]Generation of backward transformation programs based on derivation of view complement functions, `http://www.ipl.t.u-tokyo.ac.jp/~kztk/bidirectionalization/`

(the transformation operators constituting the BiXJ language). A subset of XQuery is translated into BiXJ in [LHT07a], thus allowing using a mainstream language for view definition, again with a prototype realization available[3]. Along the same lines, Xiong [XLH+07] translates a subset of ATL (Atlas Transformation Language), thus achieving bidirectionality[4].

Data synchronization. Algorithms developed to synchronize intermittently connected data sources (such as file systems or address books, between mobile and stationary devices) can also be applied to keep complex software artifacts in-synch. An exponent of this approach is the Harmony project [FGM+07] whose engine[5] implements Focal, a language with building blocks that allow writing only functions that always behave as *lenses*, i.e., bidirectional transformations. Focal is a low-level language operating on tree-shaped data structures (specifically, edge-labeled unordered trees). Standard encodings for mainstream data structures (lists, XML) are available, as well as libraries of higher-level lenses defined in terms of primitive ones. The design of Focal reflects its theoretical underpinnings in the field of type systems for programming languages, as static assurances can be obtained about the detailed type of inputs and outputs, to avoid runtime checks. In contrast, implementations such as BiXJ resort to returning a default value (e.g., unchanged input) or throw an exception whenever a function argument lies outside the function's domain.

The capabilities of EMOF-based modeling infrastructures (in particular undo/redo and evaluation of OCL invariants) grant a large degree of tolerance to inconsistent input, a feature that proves extremely valuable during the initial exploratory phases of DSML language engineering (which comprises the definition of transformations for each view). Moreover, experience shows that modelers frequently perform a series of editing operations which temporarily result in WFRs being broken. We aim at preserving this flexibility, to avoid usability problems similar to those that plagued syntax-directed text editors. In summary, we strike a balance between static assurances and ease of use by relying on runtime checks to capture side conditions not enforceable at design-time. If needed, static assurances beyond those amenable to static type checking can still be obtained by offline *model checking*, as shown in the case studies of [ABK07] (for transformations expressed as LHS $\rightarrow$ RHS production rules) and [GM07] (for imperative transformations).

Update of virtual views in databases. The view update problem has been studied in the context of databases, where the mechanism to define views is taken as given (relational algebra or calculus) and the kinds of view updates that may be propagated back without loss of information are determined. Recent work focuses on updating virtual and materialized XML views (in the latter case, incrementally). Most results have been incorporated into the program inversion and data synchronization techniques [MHN+07, FGM+07].

Given that the EMOF data model is richer than its relational counterpart (because of object IDs, ordered collections, reference handshaking), techniques targeting relational databases are overly restrictive when adapted to EMOF, limiting the operators available for view definition. The incremental maintenance of materialized views in OO databases is studied by Ali [AFP03].

[3] BiXJ and Bi-CQ, `http://www.ipl.t.u-tokyo.ac.jp/~liu/`

[4] Bi-ATL, `http://www.ipl.t.u-tokyo.ac.jp/~xiong/modelSynchronization.html`.

[5] Harmony Project, `http://www.seas.upenn.edu/~harmony`.

2.2 QVT-Relations and Graph-grammars

QVT-Relations. QVT-Relations was designed to encode input-output relationships by means of pattern-matching guarded by preconditions. At any given point in time, all but one of the models participating in a transformation are considered as non-updatable, thus constraining the solution space to a well-defined set of (updates, instantiations, deletions) on the *target model*.

Erche et. al. [EWH07] point out that metamodel-based language specs do not specify the connection between concrete and abstract syntax and propose QVT-Relations to bridge that gap. Given that such transformations are bidirectional, their architecture aims at solving the same problem space as Declarative MVC. There is no detailed discussion in [EWH07] on whether every QVT-Relations transformation is well-behaved in terms of the conditions defined by Matsuda et. al. [MHN$^+$07] (Sec. 2.1). Irrespective of the particular transformation mechanism adopted, Duboisset [DPKS05] recognizes that not all geometric constraints relevant for concrete visual syntaxes can be expressed in EMOF + OCL metamodels, offering as an example topological constraints in spatial databases. It is clear that QVT-Relations can support roundtripping over one-to-one mappings, however a discussion of its capabilities to back-propagate updates on non-injective views is missing in the literature. Our approach around geometric constraints is covered in Sec. 4.

Triple Graph Grammars (TGGs). TGGs [GGL05] build upon directed typed graphs and graph morphisms. Informally, a TGG transformation rule consists of three graphs (left, interface, and right) and two morphisms (from the interface graph to each of left, right) which together describe the correspondence between embeddings of these graphs in source and target. In other words, such rule also states the inter-consistency conditions between source and target, besides specifying a transformation. Figure 2 depicts an example, the compilation of `if-then-else` into lower-level constructs (conditional jumps). Before a TGG transformation can be applied, its *positive* and *negative application conditions* are evaluated. These conditions demand a required context (certain nodes or edges must exist) or forbid a context (certain nodes or edges must be absent) connected in a certain topology.

An extension of TGG transformations to accomodate N-way relations is offered in [KS06]. For our purposes, this capability is not necessary as our architecture revolves around a single integrated model (i.e., to synchronize N different view types N bidirectional transformations are defined, as depicted in Figure 3). Graphical IDE support is available[6], and modularization has been proposed to cope with large-scale transformations. Similar to other rewriting techniques, the control flow aspect of a complex transformation (when to apply which rules) suggests breaking up large transformations into several more focused ones, to be applied sequentially.

As with the data synchronization approach, an encoding of EMOF models is necessary (in this case, into directed typed graphs), as well as expressing transformations in terms of graph morphisms guarded by application conditions. In our setting, some features of

[6] Some TGG-based tools: (a) MOFLON, `http://www.moflon.org/`; (b) MoTE/MoRTEn (as FUJABA plugins), `http://wwwcs.uni-paderborn.de/cs/fujaba/projects/tgg/`; (c) AToM3, `http://atom3.cs.mcgill.ca/`

the program inversion approach (Sec. 2.1) prove beneficial over TGGs: (a) OCL expressions in view definitions can be used directly by Matsuda's bidirectionalization algorithm [MHN$^+$07], and (b) no explicit rules need be declared to delete view elements not supported anymore by source elements. The runtime overhead of encoding EMOF models into graphs can be reduced with the *Adapter* design pattern, at the cost of an indirection level (as with any approach, these design decisions would need to be revisited if the modeling infrastructure natively managed models in the format of the transformation engine).

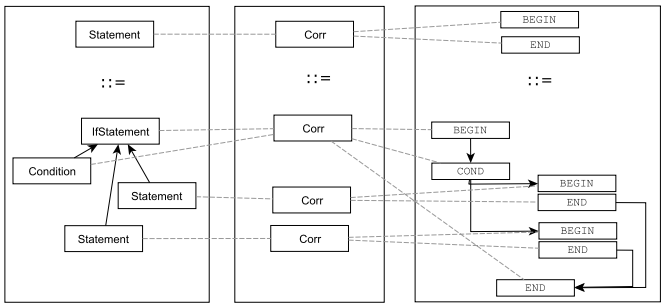

Figure 2: Sample TGG-based translation (`if-then-else` into conditional jumps, [Lei06])

3 Integration in an EMOF-based modeling infrastructure

After settling on the bidirectional program inversion technique [MHN$^+$07], the interfacing of its functional inversion algorithm with current metamodeling infrastructure has to be addressed. A canonical approach consists in encoding EMOF models into inductive data types to automatically apply the inversion algorithm to each view definition, expressed as an affine function in treeless form [MHN$^+$07]. Alternatively, a *fixed catalog* of bidirectional operators can be defined for EMOF models, fulfilling the three stated bidirectional properties. Both alternatives are explored, in Sec. 3.2 and Sec. 3.1 resp. Briefly, the advantage of the canonical approach is the open-ended set of base operators that can be defined, while the existing EMOF-based ones can only be recombined. On the other hand, adopting the EMOF-based operators avoids the detour to the inductive-data-types representation. Besides the performance gain, usability is also improved, as modelers are accustomed to conceptualizing transformations in terms of EMOF-level constructs. In any case, the approaches are not mutually exclusive, and any of them can be adopted to define views (injective or not) as part of the Declarative MVC (DMVC) architecture (Sec. 4).

As with Bidirectional-XQuery [LHT07a] and Bidirectional-ATL [XLH$^+$07] a possibility consists in finding a subset of QVT-Relations amenable to encoding with bidirectional operators. Given that DSML authors already master the concepts required to understand the building blocks of bidirectionalization, directly using them results in making available their full expressive power. In a next step, subsets of OCL and QVT-Relations, which are already EMOF-aware, can be recast in terms of the operators in the next two subsections.

3.1 Operators for two-way transformations in Ecore: TwEcore

Each operator consists of a *forward* and a *backward* function. Borrowing notation from [HLM+06], $[\![X]\!]_F(s)$ stands for the application of the (possibly composite) operator X to the source s in the forward direction, resulting in a view t. The backward function, $[\![X]\!]_B(s, t')$ takes as argument the *unmodified* source s, the *updated* view t', and returns a pair (s', X') consisting of an updated source s' as well as a possibly updated operator X', to be used in further invocations. This statefulness is exemplified by X = `twRenameProp(old, new)`, to rename the property `p` named `old`, where s denotes an EMOF class. In this case $[\![X]\!]_F(s)$ is a clone of s save for renaming the cloned property `p` from `old` to `new`. In turn, $[\![X]\!]_B(s, t') = (s', X')$ where t' may have user updates, including renaming of property `p` itself. The backward function returns in s' such changes save for any renaming of `p`, whose name is restored to `old`. An updated property name `new'` provided by the user on the view t' is recorded instead in the state of X' = `twRenameProp(old, new')`. Therefore, a successive application of X will involve again the latest name entered by the user.

The basic example of a composite transformation is function composition, represented by X = `twSeq(`X_0 `...`X_n`)`, where the simpler transformations $X_0 \ldots X_n$ are applied so that s'_i is the updated source for t_i $(0 \leq i \leq n-1)$. The definition for this generic operation is reproduced from [HLM+06]:

$$
\begin{array}{rcl}
[\![X]\!]_F(s) & = & t \\
[\![X]\!]_B(s, t') & = & (s', \ \texttt{twSeq}\ [X'_0 \ldots X'_n]) \\
\texttt{where}\ t_0 & = & [\![X_0]\!]_F(s) \\
\ldots & & \\
t & = & [\![X_n]\!]_F(t_{n-1}) \\
(s'_{n-1}, X'_n) & = & [\![X_n]\!]_B(t_{n-1}, t') \\
\ldots & & \\
(s', X'_0) & = & [\![X_0]\!]_B(s, s'_0)
\end{array}
$$

In addition to the operator definitions, an EBNF-based concrete syntax is necessary to facilitate the discussion and exchange of view definitions (an area for future work in TwEcore). In terms of implementation, such syntax proves useful as it enables the interpretation of ad-hoc, or dynamically generated, view-definition scripts. In fact, this use case was foreseen by the authors of [MHN+07] and is supported in a Haskell-based bidirectional XML editor where users can update not only sources and views, but also transformations connecting them.

3.2 Encoding of EMOF models using inductive data types

This subsection explores the implications (for multiview synchronization) of implementing an EMOF infrastructure using functional programming (FP) instead of Java. This

exercise is not as far-fetched as it might seem at first sight because: (a) several bidirectionalization approaches are naturally expressed with FP; (b) functional programs are more amenable to static analysis than their OO counterparts; and (c) most of the proposed new language features for post-Java languages originate in FP[7]. The Declarative MVC architecture does not impose a functional realization, with this subsection serving as outlook for readers sharing an interest in functional programming.

Porting an EMOF infrastructure to the functional paradigm comprises devising encodings for (a) EMOF data structures, and (b) algorithms for views and transformations in EMOF. Regarding (a), given that EMOF models are typed, labelled graphs, the encoding proposed by Erwig is applicable [Erw01]. Regarding (b), the algorithms to port fall into two categories: (b.1) those already formulated in terms of OO concepts (e.g., written in QVT-Relations, ATL[8], or Java); and (b.2) those written as affine functions in treeless form, as expected by the algorithm for well-behaved bidirectionalization of Matsuda et. al. [MHN+07]. For (b.1) an encoding style is required that at least preserves the type-checking capabilities of the OO representation. The OOHaskell approach (Kiselyov and Lämmel [KL05]) fulfills these requirements by exploiting the type checking and type inference mechanisms of Haskell. As a result, Haskell-based processing following an OO style never results in a runtime errors like "method not found" that the OO version would have detected at compile-time.

While the pragmatic approach of TwEcore (and BiXJ, Bi-XQuery, and Bi-ATL) accelerates the construction of proofs of concept for DMVC tools, the same benefits could be easily achieved in a modeling infrastructure based on functional programming.

4 Diagram-based views and Geometric Constraint Solvers

Existing MVC frameworks for modeling infrastructures (e.g., EMF.Edit) support out-of-the-box a particular case of synchronization between view and model, namely bijections, where updates originating in a view are applied as-is to the single correspoding item in the associated model. The Declarative MVC architecture leverages this *data binding* capability (for EMF[9], GUI widgets[10], and the Eclipse Graphical Modeling Framework[11]). Interactions of this kind do not have to cope with non-injectiveness (as in $f(x, y) = x + y$). For comparison, they are shown bracketed with (A) in Figure 3, while those requiring bidirectionalization are marked with (B).

In case an update to the integrated model requires adding figures to a diagram view, default values have to be provided for the figure's position, size, layer, color, etc. While these values cannot be computed by the bidirectional transformation engine, they can still be managed declaratively with the help of a *geometric constraint solver* (e.g., [MC02]) which assumes the role of a *local Controller* in one of the MVC subsystems depicted in Figure 3

[7] Scala programming language, `http://www.scala-lang.org/`
[8] ATL, ATLAS Transformation Language, `http://www.eclipse.org/m2m/atl/`
[9] EMF Data Binding, `https://bugs.eclipse.org/bugs/show_bug.cgi?id=75625`
[10] JFace Data Binding, `http://wiki.eclipse.org/index.php/JFace_Data_Binding`
[11] Eclipse GMF, `http://www.eclipse.org/gmf/`

(i.e., it processes a subset of the view-level change requests, forwarding the non-filtered ones to the main Controller). Constraint solvers are responsible for enforcing geometric invariants, such as: (a) ensuring area inclusion between substates and their parent state in a statechart diagram, (b) ensuring non-overlap of the 2D regions for different figures.

The constraints on the layout of figures mandated by a *visual syntax* do not usually comprise the heuristics (such as crossings minimization) that distinguish a diagram with a comfortable layout from another which is hard to decipher. After computing a layout that fulfills those cognitive quality measures, small user edits should not cause a full re-arrangement, as becoming familiar with a new layout places a cognitive load on the user. This dynamic aspect is not normally considered in graph layout algorithms [Dub06]. Moreover, capturing all relevant *visual aesthetics* of a given visual notation is nowhere near straightforward, as their relative weight on diagram understanding may be discovered only with empirical studies [Dub06, p. 5]:

> A followup study reveals a visual aesthetic not previously considered by the graph drawing community. This new aesthetic, continuity, is the measure of the angle formed by the incoming and outgoing edges of a vertex. For the task of finding a shortest path between two vertices, continuity can become even more important than edge crossings.

The example reveals that well known visual aesthetics for graph layout overlook constraints that can be specified declaratively. Geometric constraint solvers are widely used in CAD tools (feature-based parametric modeling, [HJA05]) and are starting to be adopted by graphical frameworks for metamodeling [LHG07].

5 Conclusions

The complexity around keeping views in-synch in multiview authoring environments requires a comprehensive solution. As shown in this paper, one such solution relies on metamodeling, well-behaved bidirectional transformations, and geometric constraint solvers. Current EMOF infrastructures have paved the way for functional extensions, such as bidirectionalization (which requires moderate integration effort) and geometric constraint solving (which is only now starting to be considered part of metamodeling [LHG07]).

Existing tools for general-purpose modeling have been developed following a traditional (non-declarative) MVC architecture, and are not expected to migrate overnight to a new paradigm. Instead, the primary candidates to benefit from Declarative MVC are Domain-Specific Modeling Languages (DSMLs). More generally, we argue that applying to DSMLs the same (metamodel-based) definition techniques as for UML 1.x will impair their adoption, as such techniques overlook the connection between concrete and abstract syntax, do not handle multiview synchronization, and lack precise semantics for backpropagating updates from non-injective views. The techniques brought together in this paper address those weaknesses identified in previous efforts around the definition and tooling of DSMLs.

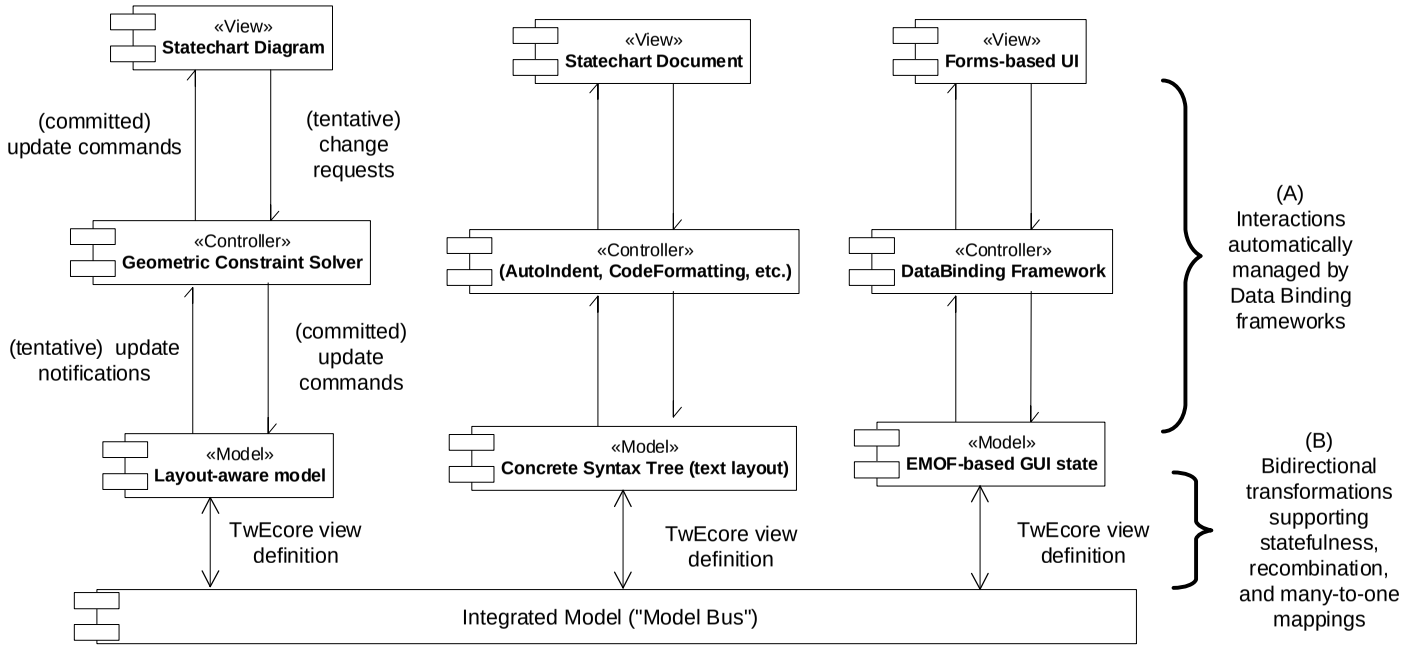

Figure 3: Software Architecture for a Multiview Design Environment supporting Bidirectionality, instantiated for a statechart editor supporting three kinds of views: diagram, textual syntax [Pro05], and forms-based

References

[ABK07] Anastasakis, K, Bordbarand, B, and Küster, J. M. Analysis of Model Transformations via Alloy. In Faivre, B. B. A, Ghosh, S, and Pretschner, A, editors, *4th MoDeVVa workshop Model-Driven Engineering, Verification and Validation*, pages 47–56, In conjunction with MoDELS07, Nashville, TN, USA, 2007.

[AFP03] Ali, M. A, Fernandes, A. A. A, and Paton, N. W. MOVIE: an incremental maintenance system for materialized object views. *Data Knowl. Eng.*, 47(2):131–166, 2003.

[BMS07] Brabrand, C, Møller, A, and Schwartzbach, M. I. Dual Syntax for XML Languages. *Information Systems*, 2007. Earlier version in Proc. 10th International Workshop on Database Programming Languages, DBPL '05, Springer LNCS vol. 3774. pp. 27-41.

[CW07] Correa, A and Werner, C. Refactoring Object Constraint Language Specifications. *Software and Systems Modeling*, 6:113–138, 2007.

[DPKS05] Duboisset, M, Pinet, F, Kang, M.-A, and Schneider, M. Precise Modeling and Verification of Topological Integrity Constraints in Spatial Databases: From an Expressive Power Study to Code Generation Principles. In Delcambre, L. M. L, Kop, C, Mayr, H. C, Mylopoulos, J, and Pastor, O, editors, *ER*, volume 3716 of *LNCS*, pages 465–482. Springer, 2005. `http://www.isima.fr/~kang/pinet/ER_paper.pdf`.

[Dub06] Dubé, D. Graph Layout for Domain-Specific Modeling. Master's thesis, School of Computer Science, McGill University, Montreal, Canada, 2006. `http://moncs.cs.mcgill.ca/people/denis/files/thesis_HREF.pdf`.

[Egy06] Egyed, A. Instant consistency checking for the UML. In *ICSE '06: Proceeding of the 28th International Conference on Software Engineering*, pages 381–390, New York, NY, USA, 2006. ACM.

[Erw01] Erwig, M. Inductive graphs and functional graph algorithms. *J. Funct. Program.*, 11(5):467–492, 2001.

[EWH07] Erche, M, Wagner, M, and Hein, C. Mapping visual notations to MOF compliant models with QVT relations. In *SAC '07: Proceedings of the 2007 ACM symposium on Applied computing*, pages 1037–1038, New York, NY, USA, 2007. ACM.

[FGM+07] Foster, J. N, Greenwald, M. B, Moore, J. T, Pierce, B. C, and Schmitt, A. Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem. *ACM Trans. Program. Lang. Syst.*, 29(3):17, 2007.

[GGL05] Grunske, L, Geiger, L, and Lawley, M. A Graphical Specification of Model Transformations with Triple Graph Grammars. In Hartman, A and Kreische, D, editors, *ECMDA-FA*, volume 3748 of *LNCS*, pages 284–298. Springer, 2005.

[GM07] Garcia, M and Möller, R. Certification of Transformation Algorithms in Model-Driven Software Development. In Bleek, W.-G, Räsch, J, and Züllighoven, H, editors, *Software Engineering 2007*, volume 105 of *GI-Edition LNI*, pages 107–118, 2007. `http://www.sts.tu-harburg.de/~mi.garcia/pubs/2007/se2007/GarciaMoeller.pdf`.

[HJA05] Hoffmann, C. M and Joan-Arinyo, R. A Brief on Constraint Solving. Unabridged; abridged version in CAD&A, 2005. `http://www.cs.purdue.edu/homes/cmh/distribution/papers/Constraints/ThailandFull.pdf`.

[HLM+06] Hu, Z, Liu, D, Mei, H, Takeichi, M, Xiong, Y, and Zhao, H. A Compositional Approach to Bidirectional Model Transformation. Technical Report METR 2006-54, The University of Tokyo, Bunkyo-Ku, Tokyo, Oct 2006. http://www.keisu.t.u-tokyo.ac.jp/research/techrep/data/2006/METR06-54.pdf.

[KL05] Kiselyov, O and Lämmel, R. Haskell's overlooked object system. Draft; Submitted for publication; online since 30 Sep. 2004; Full version released 10 September 2005, http://homepages.cwi.nl/~ralf/OOHaskell/paper.pdf, 2005.

[KS06] Königs, A and Schürr, A. MDI - a Rule-Based Multi-Document and Tool Integration Approach. *Journal of Software & System Modeling*, 5(4):349–368, December 2006.

[Lei06] Leitner, J. Verifikation von Modelltransformationen basierend auf Triple Graph Grammatiken, March 2006. Diplomarbeit. TU-Berlin und Universität Karlsruhe (TH).

[LHG07] Liu, N, Hosking, J. G, and Grundy, J. C. MaramaTatau: Extending a Domain Specific Visual Language Meta Tool with a Declarative Constraint Mechanism. In *VL/HCC*, pages 95–103. IEEE Computer Society, 2007.

[LHT07a] Liu, D, Hu, Z, and Takeichi, M. Bidirectional interpretation of XQuery. In *PEPM '07*, pages 21–30, New York, NY, USA, 2007. ACM. http://www.ipl.t.u-tokyo.ac.jp/~liu/PEPM2007.pdf.

[LHT+07b] Liu, D, Hu, Z, Takeichi, M, Kakehi, K, and Wang, H. A Java Library for Bidirectional XML Transformation. *JSSST Computer Software*, 24(2):164–177, May 2007. http://www.ipl.t.u-tokyo.ac.jp/~hu/pub/jssst-cs06-liu.pdf.

[MB05] Marković, S and Baar, T. *Proc of the 8th Intnl Conf MoDELS 2005*, volume 3713 of *LNCS*, chapter Refactoring OCL Annotated UML Class Diagrams, pages 280–294. Springer Verlag, October 2005.

[MC02] Marriott, K and Chok, S. S. QOCA: A Constraint Solving Toolkit for Interactive Graphical Applications. *Constraints*, 7(3-4):229–254, 2002.

[MHN+07] Matsuda, K, Hu, Z, Nakano, K, Hamana, M, and Takeichi, M. Bidirectionalization transformation based on automatic derivation of view complement functions. In Hinze, R and Ramsey, N, editors, *ICFP*, pages 47–58. ACM, 2007. http://www.ipl.t.u-tokyo.ac.jp/~hu/pub/icfp07.pdf.

[Pie06] Pierce, B. C. The Weird World of Bi-Directional Programming. Invited talk at ETAPS, 2006. http://www.cis.upenn.edu/~bcpierce/papers/lenses-etapsslides.pdf.

[Pro05] Prochnow, S. H. KIEL: Textual and Graphical Representations of Statecharts. http://rtsys.informatik.uni-kiel.de/~rt-kiel/kiel/documents/talks/oberseminar-0511-spr/talk.pdf, Nov 2005.

[SBP07] Sen, S, Baudry, B, and Precup, D. Partial Model Completion in Model Driven Engineering using Constraint Logic Programming. In *INAP'07 (International Conference on Applications of Declarative Programming and Knowledge Management)*, Würzburg, Germany, 2007.

[Ste07] Stevens, P. Bidirectional Model Transformations in QVT: Semantic Issues and Open Questions. In *MoDELS*, volume 4735 of *LNCS*, pages 1–15. Springer, 2007.

[XLH+07] Xion, Y, Liu, D, Hu, Z, Zhao, H, Takeichi, M, and Mei, H. Towards Automatic Model Synchronization from Model Transformations. *22nd IEEE/ACM International Conference on Automated Software Engineering*, pages 164–173, November 2007. http://www.ipl.t.u-tokyo.ac.jp/~xiong/papers/ASE07.pdf.