Bidirectional Transformations In The Large

Citation for published version:
Stevens, P 2017, Bidirectional Transformations In The Large. in 2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS). pp. 1-11, 2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems, Austin, United States, 17-22 September. DOI: 10.1109/MODELS.2017.8

Digital Object Identifier (DOI):
10.1109/MODELS.2017.8

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS)

Bidirectional Transformations in the Large

Perdita Stevens
Laboratory for Foundations of Computer Science
University of Edinburgh
UK
Email: Perdita.Stevens@ed.ac.uk

***Abstract*—The model-driven development of systems involves multiple models, metamodels and transformations, and relationships between them. A bidirectional transformation (bx) is usually defined as a means of maintaining consistency between "two (or more)" models. This includes cases where one model may be generated from one or more others, as well as more complex ("symmetric") cases where models record partially overlapping information. In recent years binary bx, those relating two models, have been extensively studied. Multiary[1] bx, those relating more than two models, have received less attention. In this paper we consider how a multiary consistency relation may be defined in terms of binary consistency relations, and how consistency restoration may be carried out on a network of models and relationships between them. We relate this to megamodelling and discuss further research that is needed.**

I. Introduction

Model-driven development (MDD) has achieved some success; but it has not yet transformed software development, and a transformation is sorely needed. The demand for software, and especially for changes to software, outstrips the availability of skilled software engineers who can build the software and effect the changes. Communication between stakeholders (who know what changes are required) and software engineers (who can effect the changes) is a bottleneck which today's agile development methods cannot fully overcome.

MDD's key aim is the separation of concerns into models, so that people can work with models that record all and only the information they need to make their decisions. These models are related by *model transformations*. Because circumstances do not usually permit finishing work on one model before starting work on another, these often have to be *bidirectional transformations* (bx). Here, despite the name which indicates where most attention has so far been directed, a bidirectional transformation may in fact relate any number of models; it is a means of restoring consistency between them. (The term *model synchronisation* is also used, although sometimes this is reserved for the case that both of two models are changed [1], [2].) Note that restoring consistency typically requires using information about the current state of all the models, not just regenerating one. In most formalisms, a bx may be used in several modes, e.g. to check whether models are consistent, or to hold a given collection of models fixed while restoring consistency by modifying the rest. The ideal is that a single artefact (e.g. a triple graph grammar (TGG) [3] or a QVT-R transformation [4]) records the bx developer's decisions about how to carry out all of these tasks, so as to avoid duplicating information. Much research has been done on the properties that a bx should have, such as how to ensure that the way it carries out its several tasks is coherent. The most basic of these properties are *correctness* (when a bx restores consistency, the resulting models are, indeed, consistent) and *hippocraticness* (if the models are already consistent, then restoring consistency changes nothing).

Let us imagine a world in which the Bx community has achieved its aims. We have developed powerful, usable bx languages which are well-supported by tools, and are taught to every undergraduate. Mainstream software is typically developed by cooperating groups of experts in all relevant fields. Each group of experts works with a model precisely adapted to their needs: it records all and only the information they need to have in mind to make their decisions. Ultimately, deployed software is produced automatically from these models.

What kind of bx do we have in such a setting, and how tightly must an organisation control exactly when the bx are used to restore consistency between models? If there are n models (including the deployed software system itself), we may argue, consistency is ultimately an n-ary relation: but we are unlikely to specify it as such, because to do so is tantamount to designing the complete software, and if we knew how to do that, we would not need the models in the first place. More likely, what we will have is a network of bx – some bought off-the-shelf, others developed for the project – each relating some of the models.

This, however, raises many questions which have not yet been addressed. The aim of the paper is to begin to make progress on them (though we will not finish!). They include:

- Do we limit the notions of consistency that can be expressed, if we insist that consistency of a collection of models is expressed in terms of consistency of pairs of those models? Under what assumptions? Does it matter?
- How can we talk about a collection of models, connected by bx? What does it mean to restore consistency of such a collection?
- Under what circumstances can we restore consistency of a collection of models related by bx? How?
- What flexibility do we have in varying our consistency restoration procedure? When are we guaranteed to get the same result, regardless of how we do it?

[1]An unusual word, but has the advantage over n-ary that it makes clear there is no fixed arity n

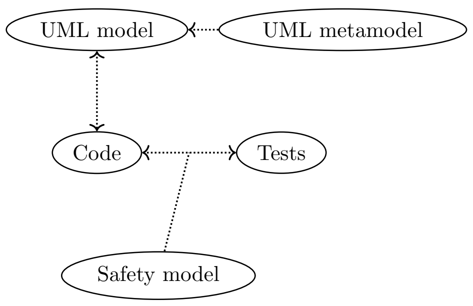


Fig. 1: Models with informal relationships

- What if we cannot fully restore consistency to the collection of models after one set of changes, before more changes begin to happen?

Paper organisation: After an example (Section II), Section III discusses how the desired consistency between models is expressed, and the implications of limiting ourselves to expressing consistency between just two models at a time. Before we turn to how consistency is restored, Section IV suggests we sometimes need to tolerate inconsistency; Sections V, VI discuss networks of bx and consistency restoration in them. Section VII discusses how this work fits with ideas of megamodelling. We discuss much related work along the way, but Sections VIII and IX mention further relevant work. Section X concludes.

Notation: We use R, S, etc. (sometimes subscripted) both for consistency relations and (later) for bx. Model sets are denoted M, N ... or A, B, ... (These may be the sets of all models in appropriate modelling languages, but in this paper we do not need to concern ourselves with modelling languages.) Models are denoted by lower case versions of the same letter, e.g. model n in model set N; for a collection, or tuple, of models we write $\overline{m}$, or $\langle m_k \rangle$ if we need to talk about the individual models.

II. Motivating example

We have alluded to a grand vision of how bx might be used in the future to move decisions about software wholesale out of the hands of software developers into those of stakeholders. Here, though, we use a more familiar example. Figure 1 shows what we may regard as a (small) megamodel, such as may arise in today's development. All the relationships may be formalised as bx. The UML model is consistent with the UML metamodel iff it conforms to it; the relationship between the UML model and the code is standard roundtripping. The relationship between the code, the tests and the safety model is more interesting, and illustrates a case where megamodels may use non-binary relationships. Suppose that the safety model records (among other things) whether the system is considered safety critical; suppose that if it is, there is a coverage criterion that must be satisfied. That is, the *relationship* between the code and the tests depends on the safety model, or to put it another way, the consistency relationship between tests, code and safety model is ternary.

Remark

1) We do not necessarily expect that all the relationships between the models are recorded in the diagram: it is likely that there are also non-automated dependencies.
2) This example illustrates that models cannot necessarily be seen as simply views onto the eventual implemented system: the models may matter even beyond their influence on which systems are allowed. Here, it is possible for the system to be correct in its behaviour, yet unacceptable because it is not adequately tested for the safety criticality level.

III. Expressing multiary consistency relations

There are strong practical reasons for considering the consistency of models pairwise wherever possible: cognitively, it is hard enough to think about a binary consistency relation in order to specify it correctly. Does this impose unacceptable limits on expressivity? For a given, natural, multiary notion of consistency, and the restriction of this notion to pairs of models, it is easy to construct examples where any pair from three models is consistent but the collection of three is not (there is one in Appendix A of [5] for example). This might lead us to give up and conclude that binary consistency is not expressive enough, and that it will be essential for future bx languages to permit the expression of multiary consistency and its restoration. However, this may be deceptive, since in practical use of bx the bx developers have some flexibility in how to define the notion(s) of consistency, and may also have the option of adding additional models.

A. Pessimistic view

First let us lay out the sense in which binary (consistency) relations are not enough. Let $\{M_i : i \in I\}$ be a (finite and ordered, for convenience, though nothing will depend on this) collection of model sets, R a relation on all these model sets i.e. a subset of $\prod_i M_i$ (to be thought of as a consistency relation), and $\{R_{ij} : i < j \in I\}$ a set of binary relations on each distinct pair of model sets, $R_{ij} \subseteq M_i \times M_j$.

Definition 1. *R is* binary-defined *by $\{R_{ij} : i < j \in I\}$ if for every I-tuple $\overline{m}$ we have*

$$(\forall i, j . R_{ij}(m_i, m_j)) \Leftrightarrow R(\overline{m}).$$

R is binary-definable *if it is binary-defined by some set of binary relations $\{R_{ij} : i < j \in I\}$.*

Not every relation is binary-definable; here is a ternary counterexample.

Example 1. *Let a, a' be distinct elements of model set A, and similarly for B, C. Then consider $R = \{(a,b,c'),(a,b',c),(a',b,c)\}$. R is not binary-definable. For suppose it were binary-defined by R_{AB}, R_{BC}, R_{AC}. We would have $(a,b) \in R_{AB}$ because $(a,b,c') \in R$, and similarly $(b,c) \in R_{BC}$ and $(a,c) \in R_{AC}$. But then $(a,b,c) \in R$, which is a contradiction.*

A given binary-definable multiary relation can of course be binary-defined by many different sets of binary relations: the pairs that are projections from consistent tuples are forced to be in the binary relations, but other pairs can be added (e.g. adding (a,b) will make no difference to the resulting multiary relation, provided there's no c such that (a,c) and (b,c) are both in the respective relations). To formalise this, fix a collection $\{M_i : i \in I\}$ of model sets. Partially order multiary relations on $\prod_i M_i$ by subset inclusion, writing $R \subseteq S$. Partially order sets of binary consistency relations pointwise, writing $\{R_{ij} : i < j \in I\} \sqsubseteq \{S_{ij} : i < j \in I\}$ iff for every $i < j \in I$ we have $R_{ij} \subseteq S_{ij}$. It turns out that these orders are intimately related. Recall (e.g. from [6][2]) the standard definition

Definition 2. *A (monotone)* Galois connection *between partially ordered sets $(A, \leq)$ and $(B, \preceq)$ is a pair of monotone functions $L : A \to B$ and $U : B \to A$ such that for every $a \in A$ and $b \in B$ we have $L(a) \preceq b$ iff $a \leq U(b)$.*

L is the lower adjoint*, U the* upper adjoint*. An* order isomorphism *is a Galois connection in which L and U are inverse bijections.*

Many pleasant consequences follow from the existence of a Galois connection; the most immediately useful to us is that $UL : A \to A$ is a closure operator, i.e. for any $a, a' \in A$, we have

- $a \leq UL(a)$ (extensive)
- $a \leq a' \Rightarrow UL(a) \leq UL(a')$ (increasing) and
- $UL(UL(a)) = UL(a)$ (idempotent).

Dually $LU : B \to B$ is a kernel operator.

Theorem 1. *We have a Galois connection (but not an order isomorphism) between multiary consistency relations and sets of binary consistency relations, as follows:*

- *given $R \subseteq \prod_i M_i$, define a set gR of binary consistency relations $\{(gR)_{ij} : i < j\}$ by $(gR)_{ij}(m_i, m_j)$ iff there exists $\overline{m}$ extending (m_i, m_j) such that $R(\overline{m})$;*
- *given a set $\{R_{ij} \subseteq M_i \times M_j : i < j \in I\}$, define a multiary consistency relation $f(\{R_{ij}\})$ by $f(\{R_{ij}\})(\overline{m})$ iff for all $i < j$, we have $R_{ij}(m_i, m_j)$.*

Here f is the upper adjoint, g the lower. Thus gf is a kernel operator and fg a closure operator; the closed multiary consistency relations are exactly those that are binary-definable, viz., those in the image of f.

Proof: Straightforward, omitted for space reasons.

■

Example 2. *Returning to the example from Section II, let us say concretely that A is a set of Java systems, and B a set of JUnit test suites. Suppose the basic idea is that implementation $a \in A$ is consistent with $b \in B$ iff a passes all the tests in b. The safety model could be as simple as $C = \{\top, \bot\}$ recording whether or not the system is deemed safety-critical; suppose that if it is, then the relationship between systems and test suites needs to be more stringent: as well as all tests that exist having to pass, it is also required that there are enough tests to satisfy some coverage criterion.*

Recording all this as a ternary consistency relation R, we may have $R(a,b,\bot)$ but not $R(a,b,\top)$, because tests b do not give enough coverage of a in the safety-critical case; while at the same time, perhaps tests b are adequate for some other implementation even in the safety-critical case, and a can, in the safety-critical case, be adequately tested by some other test suite. The upshot is that (a,b), $(a,\top)$ and $(b,\top)$ all appear in the relevant binary relations of gR, and therefore $(a,b,\top) \in fgR$. Thus R is not closed, and hence not binary-definable.

[2]or from Wikipedia: https://en.wikipedia.org/wiki/Galois_connection

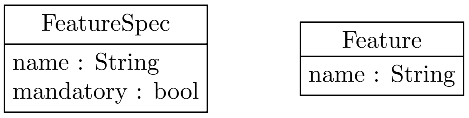


Fig. 2: Metamodels for a product specification (left) and configuration (right), from [7]

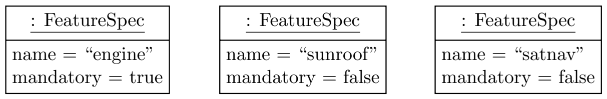


Fig. 3: Product line P

Now, in this example, one potential solution is obvious: development might proceed using the precautionary principle, imposing the coverage criterion in case it is required. We may ask: is this approach always reasonable? Our next example is borrowed from [7] (we will consider their paper in more detail in Section IX).

Example 3. *This example concerns a highly simplified product line. Figure 2 shows metamodels for product line specifications and product configurations respectively: a product line (e.g. P in Figure 3) specifies some features, which may be mandatory or not, while a configuration (e.g. (a), (b) or (c) in Figure 4) has some features. Consistency is defined over one specification and an arbitrary number of configurations; this collection of models is deemed consistent if all* and only *the mandatory features appear in all the configurations. Thus, the collection of models $\langle P, (a), (b) \rangle$ is consistent, since the mandatory feature* `engine` *is the only one to occur in all (both) configurations; however, the collection of models $\langle P, (a), (c) \rangle$ is not, because the non-mandatory feature* `sunroof` *occurs in all configurations, and therefore, according to our notably artificial rule, should be mandatory in the specification.*

Instantiating the Galois connection definitions, we see that the closure of the given multiary consistency relation is that in which the products in a tuple include at least *the features that are mandatory in the specification. However, unlike in the previous example, there is no natural closed (i.e. binary-definable) consistency relation* contained in *the one we really want, against which we could develop. For as soon as we can express specifications with two different sets of mandatory features, one contained in the other ($A \subset B$ say), together with*

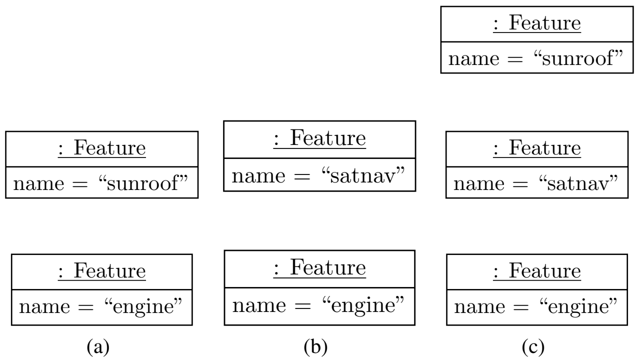


Fig. 4: Models $\langle P,(a),(b)\rangle$ consistent; $\langle P,(a),(c)\rangle$ inconsistent

all tuples of configurations that are permitted by each of those specifications, it will follow that our closed relation will also permit specification A together with tuples of products that all have the larger set B of mandatory features.

B. Optimistic view

Approach 1: add extra models: A related question has long been studied by the constraint satisfaction problem (CSP) community, under the name of *constraint networks*. Here, in place of our model sets, we take variables (each with a domain of allowed values), and in place of our consistency relations, we take constraints (which may have any arity, that is, constrain any number of variables; but two is a special case of considerable interest). The constraint network problem is: given a set of variables and a set of constraints, find an assignment of values to the variables that satisfies all the constraints. It is a classic result [8] that *any non-binary constraint network can be translated into a binary one with additional variables*. There are two standard ways to do this, known as the dual translation and the hidden variable translation. We refer to [9] for a detailed description and comparison of these approaches. In outline, the dual translation, which originates in the database community, produces a binary constraint network with one variable for each constraint in the original network. We call these c-variables to distinguish them from the variables of the original network. The allowed domain for the c-variable is precisely the set of tuples of values for the constrained variables that satisfy the constraint. Two c-variables are connected with a binary constraint if they have any constrained variables in common. In this case, the constraint says that they must agree on their shared variables. The hidden variable translation produces a constraint network that has more variables, but which can be more tractable because it involves a bipartite graph. It produces a network whose variable set is the union of the original set of variables and the set of new c-variables, as in the dual translation. Instead of adding constraints between c-variables, the hidden variable translation adds constraints between a c-variable and each of its constrained variables. These constraints say that the value of the original variable must agree with the value in the tuple of the c-variable.

The CSP community has done considerable work to understand the relative merits of these and other translations and algorithms on them. These may repay study. However, it does not follow that the bx problem is solved, for two reasons. First, it will often be impractical or unhelpful to add the extra models. In the limiting case of a single multiary consistency relation, either translation involves building a model set whose values are the correct tuples of all the original models; it is illusory that this would help. On the other hand, it might be a practical way to avoid working with, say, ternary relations on closely related models. Second, the aims of the two fields differ. In CSP, as in bx, the first task is to express a multiary consistency relation (constraint) in terms of binary ones. In CSP, however, the resolution task is to find a solution, by any means. If, in a bx world, we wish to insist that the resolution be effected by means of the consistency restoration functions of binary bx – which we may well wish to do, in order to permit the bx programmer to ensure appropriate least change properties, for example, and thus limit the surprise that a developer may experience when their model is changed by a bx [10] – this limits the resolution possibilities. We shall return to this point after introducing bx networks: see Example 5.

Approach 2: vary the consistency definition: In other contexts, we are comfortable with the idea that early decisions made about the development process may render some software systems – that would in fact meet their requirements – inaccessible via the chosen development process. For example, it might be perfectly possible to meet the requirements using functional programming, but if we settle instead on object oriented design, we thereby exclude that collection of correct solutions. Indeed, the development process can be seen as whittling down the solution space, repeatedly excluding some unsatisfactory solutions but perhaps also some satisfactory ones, until we end up at one, satisfactory, solution.

It is arguable, therefore, that the requirement to find a set of binary consistency relations such that *every* correct n-tuple arises from the conjunction of them is too stringent. We might instead ask for a *binary-implementable* consistency relation:

Definition 3. *R is binary-implemented by $\{R_{ij} : i < j \in I\}$ if for every I-tuple $\overline{m}$ we have*

$$(\forall i, j. R_{ij}(m_i, m_j)) \Rightarrow R(\overline{m}).$$

That is, we allow the binary relations to forbid pairs of models, even though these pairs could, in fact, be completed to entirely correct system implementations. We do this in order to “be on the safe side” because of not knowing what else is going on in the system. In Example 2, for example, we might decide to use the relation between Java systems and JUnit test suites that insists on the coverage criterion as well as on all tests passing, just in case it turns out that the system is safety critical according to the third model.

Our confidence that this might be a useful weakening of *binary-definable*, explaining how to represent a larger class of multiary relations in terms of binary relations, is shaken by the observation that every R may be binary-implemented by some set of binary relations, and that not all sets of implementing binary relations are useful for development – indeed *any* multiary relation is implemented by *any* unsatisfiable set of binary relations!

We next wonder whether we can make progress by excluding such pathological cases. The next theorem attempts to capture the intuition that this will not help: we will be vulnerable to back-forming the implementing binary consistency relations from a single tuple satisfying the multiary relation. In order to make the attempt, we make the relationship between the multiary consistency relation and the development process more explicit. We consider the requirements on the eventually implemented system as a predicate on equivalence classes of such systems; using equivalence classes assures us that the requirements are insensitive to whatever notion of trivial detail is appropriate. Other pathologies we exclude are that the chosen collection of models might be unsuitable or too weak to specify the eventual implemented system meaningfully.

Theorem 2. *Let $\mathscr{S}$ be a set of equivalence classes of systems, $\Phi \subseteq \mathscr{S}$ be those deemed to satisfy the requirements, and $\{M_i\}$ be a collection of model sets such that partial function $f : \prod_i M_i \rightharpoonup \mathscr{S}$ satisfies $f(\langle m_k \rangle) = S \Leftrightarrow \forall k.S \models m_k$. That is, our collection of models is sufficiently expressive to determine the system, up to our chosen notion of equivalence. Write $R(\overline{m})$ for $f(\overline{m}) \in \Phi$: then R is our desired multiary consistency relation.*

Suppose further that the requirements are satisfiable via the model sets, in the sense that $f(\prod_i M_i) \cap \Phi \neq \emptyset$; that is, R is satisfiable. Then R can be binary-implemented by a set of binary consistency relations $\{R_{ij} \subseteq M_i \times M_j\}$ that are simultaneously satisfiable.

Proof: Pick $g = \langle g_k \rangle \in R$ and define $R_{ij}(m_i, m_j)$ to be true iff $m_i = g_i$ and $m_j = g_j$. These satisfy the condition, because the antecedent is only true of g! These R_{ij} are simultaneously satisfiable (by g). ■

Is this a feature, or a bug? On the positive side, this result suggests that, given any requirements expressible as a multiary consistency relation, we can indeed replace the multiary relation by a set of binary relations and develop to these, which is convenient. On the negative side, being able to find *some* set of binary relations that fit the definition is not the same as being able to find a practically useful set to develop against, as Theorem 2's proof demonstrates.

Moreover, an essential consequence of being prepared to give up some development flexibility for the convenience of using binary relations is that we could end up – through simultaneous work being done on the various models – with a complete collection of models that is indeed correct according to the multiary relation, but which does not satisfy all of the implementing binary relations we chose. In bx terminology this may show up as the analogue of a hippocraticness failure: if the automated consistency restoration is done in terms of binary bx whose consistency relations do not all hold, then restoring consistency will change the models, even though no change is actually required.

Taken as a whole, this exploration justifies the decision not to start with a multiary consistency relation and derive from it a set of binary consistency relations to develop against, but rather, to trust that we can produce a sensible collection of binary consistency relations, which binary-*define* a good-enough closed multiary consistency relation via Theorem 1. The rest of the paper will be concerned with how the consistency restoration functions of the binary bx can be used to restore consistency in a network of models related by such binary bx.

IV. How do we best manage and tolerate inconsistency?

We will shortly go on to consider networks of models connected by bx and how we may restore consistency in such a network. But first, a disclaimer is in order. From a theoretical point of view it is tempting to imagine that each time a change is made to a model, all other models are *instantaneously* brought into consistency with the changed model, so that at any moment all models are consistent, being views of a notional Ur-model. But this will not do: even in the case of just two models, it is too disruptive to people working with a model to have it changed under their noses. As soon as the task of consistency maintenance is distributed, temporary inconsistency is unavoidable. Moreover inconsistency between models is understood to be valuable in many ways; it may record as-yet-unresolved conflicts in the real world, for example, or it may be a consequence of not acting prematurely on changes that are being made to a model speculatively in the process of actually doing design. There is a literature on this topic; in [5] for example, the authors argue persuasively, based on case studies at NASA, that inconsistency management should be a core activity in software development, and that the main danger arises not from inconsistency between models but from *unrecognised* inconsistency. Space precludes going further here.

V. Networks of bx

Imagine a (hyper)graph with model sets (thought of as types) as nodes and consistency relations (later: bidirectional transformations) as (hyper)edges.

Definition 4. *A* transformation context *C is a (hyper)graph whose nodes $N \in \mathscr{N}$ represent model types and whose labelled (directed or undirected) (hyper)edges represent relations between them. Self-loops and multiple (hyper)edges between the same node sets, even with the same labels, are not excluded (they may be needed to talk about relations between several models of the same type, for example). A* network *or* instance *I in a transformation context is a (hyper)graph in which nodes represent models, typed with model types from $\mathscr{N}$, and (hyper)edges can only exist between models whose types have (hyper)edges between them with the same label. (That is, there is a (hyper)graph morphism $I \longrightarrow C$ preserving labels.) An (hyper)edge between some models labelled with a relation (binary case: $n \overset{T}{\longleftrightarrow} m$) is* consistent *if the models are consistent according to the relation. An instance is* consistent *if all its (hyper)edges are consistent.*

An authority instance *provides, further, a non-empty subset of the models (the* authority set*) which are to be deemed* authoritative*, i.e. must not be changed. If I is an authority instance then instance J is a* resolution *of I if every authoritative model is the same in both, and J is consistent.*

Having a set of models that must not be changed generalises the usual sitation in binary bx, in which the two consistency restoration functions each hold one model fixed and modify the other to restore consistency. This is, arguably, easier to manage than the alternative framing where both (all) models may be modified simultaneously – especially when consistency restoration is to be done automatically, without a human to resolve conflicts. In the general setting, the same network might have different authority sets at different times, e.g., a model that has just been reviewed might be placed in the authority set so that it is not changed, while others that are to be reviewed in future, after consistency restoration, are not. Or the authority set might be fixed, e.g. if there is a fixed workflow of consistency restoration.

A. Focusing on binary networks of relational bx

Now we are in a position to think about how resolutions are found. To make any progress on what is and is not possible, we will need to make some assumptions about what kind of steps can be taken towards resolving a network.

Therefore, although it is interesting to consider networks including bx that synchronise two or more models by modifying them all, for the rest of the paper we shall limit ourselves to networks of ordinary binary state-based relational bx. That is, each edge in a transformation context, and hence in a network, will be labelled with a bx defined as usual by three components: $R : M \leftrightarrows N$ is defined by a consistency relation $R \subseteq M \times N$ (by the usual slight abuse of notation), and two consistency restoration functions $\overrightarrow{R} : M \times N \to N$ and $\overleftarrow{R} : M \times N \to M$. These will always be assumed to be correct and hippocratic.

If an edge is directed, $M \xrightarrow{R} N$, this means that the direction of consistency restoration along this edge has been decided. Note this is not to say that R is a unidirectional transformation – the result of applying it will be to change the value of the target model, but the resulting value may depend on the previous values of the models at *both* ends of the edge. However, a unidirectional transformation $f : M \to N$ may indeed be seen as a special case of this ($f(m,n)$ iff $n = f(m)$, $\overrightarrow{f}(m,n) = f(m)$, $\overleftarrow{f}$ undefined, which is not a problem as the notation indicates we have already decided not to use it), and we can include these without needing to notate them differently.

We assume that as part of the definition of the graph, each edge has a fixed ordering of its ends corresponding to the argument order of its bx: note this can be different from the chosen direction of application.

Definition 5. *To* orient *an edge is to choose a direction of application; to orient an authority instance is to do this for every edge.*

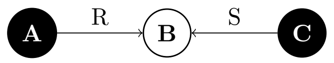


Fig. 5: Not all authority instances have resolutions

Definition 6. *Let I be an authority instance. A* resolution step *is (e,d), where e is an edge in I and $d \in \{\rightarrow, \leftarrow\}$ a direction, compatible with the orientation of e (if any). It modifies one node of I by applying the bx represented by edge e in direction d: we write $(e,d) : I \mapsto I'$. It is required to be permitted by I's authority set, that is, not to modify an authority model.*

A resolution path *is a sequence of resolution steps $I \mapsto \ldots J$ such that J is consistent. We may identify a resolution path by giving a list $\langle (e_i, d_i) \rangle$.*

Definition 7. *I is* resolvable *if there exists a resolution path $I \mapsto J$.*

Definition 8. *I is* confluent *if it is resolvable and moreover any two resolution paths $I \mapsto J$ and $I \mapsto J'$ must have $J' = J$.*

We give confluence as a property of the instance, not just of two paths, because the practical concern is how much management control over the exercise of consistency restoration is required. If the instance is confluent individual developers can use individual bx to restore consistency "locally" as and when they like – the order in which bx are used will not matter to the global result.

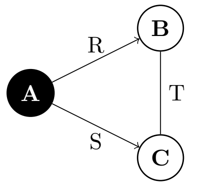
Unfortunately, while it is easy to define these properties, it is far from easy to cause them to hold. We will have to impose further conditions in order to ensure any of these (increasingly stringent) desirable properties of an authority instance hold:

- it has a resolution (i.e. there is some collection of models that satisfies all the relations);
- it is resolvable (i.e. starting from the given collection of models, some resolution path, i.e. some order of application of the bx's consistency restoration functions, leads to a consistent collection of models);
- it is confluent (i.e. there is a unique consistent result of any resolution path).

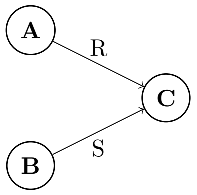
We will use blob diagrams to illustrate authority instances; black filled circles will represent models that are in an authority set, i.e. may not be changed, and white circles will represent models that may be changed. We will always lay them out so that the first argument to the bx is above and/or to the left of the second argument; an arrow on the edge represents that a particular consistency restorer has been chosen. Note that edges out of nodes in the authority set will need to be directed out of those nodes, since by definition the authority set nodes cannot be altered.

Example 4. *Figure 5 will represent an authority instance with no resolution, if $\{b \in B : R(a,b)\} \cap \{b \in B : S(b,c)\} = \emptyset$.*

The next example demonstrates how it may be impossible to resolve a bx network using the bx consistency restoration functions, even though a consistent set of models exists. Such



(a) Not all instances that have resolutions are resolvable



(b) Some consistency restorers with common target commute

Fig. 6

examples explain why the bx problem is not merely a subproblem of the constraint network problem.

Example 5. *Consider the network shown in Figure 6(a).*

Let $A = \{a\}$, $B = \{b_1, b_2\}$, $C = \{c_1, c_2, c_3\}$. Suppose that the current states of the models are (a, b_1, c_1), and that $\{a : A\}$ is the authority set. Define R, S and T as follows:

$R(a,b_1)$, $\neg R(a,b_2)$; $\overrightarrow{R}(a,\_) = b_1$

$S(a,c_1)$, $S(a,c_2)$, $\neg S(a,c_3)$; $\overrightarrow{S}(a,\_) = c_1$

$T(b_1,c_2)$, $T(b_1,c_3)$, $T(b_2,c_1)$,

$\neg T(b_1,c_1)$, $\neg T(b_2,c_2)$, $\neg T(b_2,c_3)$;

$\overrightarrow{T}(b_1,\_) = c_3$, $\overrightarrow{T}(b_2,\_) = c_1$

$\overleftarrow{T}(\_,c_1) = b_2$, $\overleftarrow{T}(\_,c_2) = b_1$, $\overleftarrow{T}(\_,c_3) = b_1$

Here _ stands for any model, e.g. $\overrightarrow{R}$ with first argument a returns b_1 regardless of its second argument. Note that $\overleftarrow{R}$ and $\overleftarrow{S}$ are not relevant, so we have not defined them; since a is authoritative we could not apply them.

Now, there is a resolution, viz. (a, b_1, c_2). However, it is easy to see that there is no resolution path that reaches this (or any other) solution: since $R(a,b_1)$ and $S(a,c_1)$ already hold, the only consistency restorers that make any difference are $\overrightarrow{T}$ and $\overleftarrow{T}$. Applying $\overrightarrow{T}$ will break the consistency according to S, and the only way to fix this will return us to the original state. Applying $\overleftarrow{T}$ will break the consistency according to R, and similarly, the only next step is back to where we started.

Supposing that a network is resolvable, it is also clear that only in rather special circumstances will it be confluent; generally the presence of non-bijective transformations will preclude this. Indeed it is easy to construct on Figure 6(a) an example where the choice between using $\overrightarrow{T}$ and $\overleftarrow{T}$ prevents confluence, and on Figure 5 an example where the choice of which edge to use first does so.

A possible reaction to all this is to give up on allowing bx developers to specify how consistency is restored: we could allow them only to express what consistency relation must hold, and then adapt, from the CSP community, techniques for restoring these constraints as necessary. Such an approach has been explored (see for example [11], [12], and from a different angle, [13]). However, developers care not only about consistency but also about how it is restored, and we know [10] that simple notions of which restoration is best are not always correct. Perhaps interesting hybrid approaches are available; but here we will assume that consistency must be restored using the consistency restorers that are defined as parts of the binary bx that specify consistency.

B. Non-interference

Informally, non-interfering transformations do not care about[3] the same aspects of their common target model, so their applications commute. Consider a fragment of a network as illustrated in Figure 6(b).

Definition 9. *Consistency restorers $\overrightarrow{R} : A \times C \to C$ and $\overrightarrow{S} : B \times C \to C$ are* non-interfering *if for all $a \in A, b \in B, c \in C$ we have*

$$\overrightarrow{S}(b, \overrightarrow{R}(a,c)) = \overrightarrow{R}(a, \overrightarrow{S}(b,c))$$

This is a strong condition – too strong to expect it to hold for every pair of transformations with a common target – but does nevertheless arise. Let us first look at a trivial algebraic case, and some formal consequences, before returning to consider another class of situations where we may be able to take advantage of this property.

Trivially, if C can be factored into $C_A \times C_B$ such that $\overrightarrow{R}$ only modifies C_A (in a way depending only on C_A) and $\overrightarrow{S}$ only modifies C_B (in a way depending only on C_B), then $\overrightarrow{R}$ and $\overrightarrow{S}$ are non-interfering.

Non-interference gives by correctness and hippocraticness

Lemma 1. *Whenever $\overrightarrow{R}$ and $\overrightarrow{S}$ are non-interfering and $R(a,c)$, then for all b we have $R(a, \overrightarrow{S}(b,c))$ ("once consistent, always consistent"). It follows that for any $a \in A$ and $b \in B$ there exists $c \in C$ such that both $R(a,c)$ and $S(b,c)$ hold.* ■

One may informally think of the common target of non-interfering consistency restorers as being able to be factored into two separate parts for the two transformations; but as in practice this factoring might not be reasonable to present, it may be more useful to be able to work the other way. Non-interference allows us to stick the two transformations together into one, thereby reducing the number of edges in our network.

Lemma 2. *Suppose we have correct and hippocratic $R : A \leftrightarrows C$ and $S : B \leftrightarrows C$ with $\overrightarrow{R}$ and $\overrightarrow{S}$ non-interfering, as before.*

We can define $RS : A \times B \leftrightarrows C$ by

- *$RS((a,b),c)$ iff $R(a,c) \wedge S(b,c)$*
- *$\overrightarrow{RS}((a,b),c) = \overrightarrow{R}(a, \overrightarrow{S}(b,c)) = \overrightarrow{S}(b, \overrightarrow{R}(a,c))$ by non-interference.*
- *$\overleftarrow{RS}((a,b),c) = (\overleftarrow{R}(a,c), \overleftarrow{S}(b,c))$*

Then RS is correct and hippocratic. ■

Moreover we may do this in the context of a network of bx, gluing edges that were incident on A or B to the new $A \times B$ and adjusting their bx in the obvious way, and this preserves other non-interference relations as we would hope: the only point to be careful of is that if $T : D \leftrightarrows C$ then $\overrightarrow{T}$ needs to be non-interfering with *both* $\overrightarrow{R}$ and $\overrightarrow{S}$, in order to be non-interfering with $\overrightarrow{RS}$. Details omitted.

[3]This is the informality: it turns out that care is needed not to confuse the aspect of the model that the bx's behaviour depends on, and the aspect it modifies, which might not be the same.

(The converse is false, however: given correct and hippocratic $RS : A \times B \not\rightleftharpoons C$ we might not be able to factor it into correct and hippocratic $R : A \not\rightleftharpoons C$ and $S : B \not\rightleftharpoons C$, because RS might have "twisting" behaviour on $A \times B$.)

A pragmatic case of non-interference: Note that the definition of non-interference is (of course) sensitive to the *amount of choice* there is for the instantiating models – that is, to the model sets – as well as to the *actual behaviour* of the consistency restorers. In some important cases, there is some discretion over how these sets are chosen and formalised. In the example of Section II, consider the question of whether there is interference between the two consistency restorers which target the UML model. The consistency relation between the UML model and the UML metamodel is conformance; if the UML metamodel changes, restoring consistency in the direction of the UML model is updating the model for the new metamodel version (presumably, making minimal semantic changes). But what is the model set from which the UML metamodel is drawn, and hence, the set of values of that type that must be considered in the definition of non-interference?

If we take this to be, say, the set of all models conforming to MOF, then these two consistency restorers will not be non-interfering. For, however clever the UML model updater is, there will be some ways in which the UML metamodel could be replaced by a different MOF metamodel which will require the meaning of the UML model to change, in the sense that a previously consistent value of the Code will no longer be consistent with the updated UML model. That is, these two consistency restorers will interfere.

However, we might pragmatically take the point of view that, even though it is desirable to have the UML metamodel as part of our megamodel and be able to make use of a consistency restorer to update the UML model if we have to accept a new UML metamodel version, we can predict that all future versions of the UML metamodel will be very similar to the current one. That is, when we think about the domain of the consistency restorer between the UML metamodel and the UML model, we might be content to have in mind a very small space of possible values for the metamodel – so small, perhaps, that we predict any resulting updates to the UML model will have no effect on the semantics of that model in terms of its consistency with the code. That is precisely what is needed to say that these consistency restorers would be non-interfering.

On the other hand, turning to the two consistency restorers that target the Code, one having the UML model as source and one having the Tests as source, it does not seem to be possible to make such an argument. Intuitively, these two consistency restorers will interfere.

We will explore the implications of these situations in Section VI.

VI. Resolving networks

Suppose we have a network of binary bx and a (connected) authority instance we wish to resolve. We have already shown that there might not be a solution; that if there is it might not be reachable via the consistency restoration functions of the bx; and that if a solution is reachable in that way, it might not be unique. What positive results can we find?

It is natural to turn our thoughts towards trees. We will need a root from which to start the resolution process. If our network includes any already-oriented edges (e.g. arising from unidirectional transformations or from decisions already made about which direction should be used), there may be models that (though not initially designated authoritative) cannot be modified and need special treatment:

Step 0: If any node is not reachable, from any authority model, by a path in the network (respecting any oriented edges), add it to the authority set.

Next we simplify the situation by creating a single root, the *supersource*:

Step 1: If the authority set contains more than one node, add a single authority node which is connected (by an edge with a notional universal consistency relation) to each node in the authority set.

Step 2: Check any edges between two nodes in the authority set. If any of these are not consistent, give up. If any edge between two nodes in the *original* authority set is inconsistent, of course, there is no resolution.

Now, if the resulting network is a tree, the situation is particularly pleasant. Step 3 then applies:

Step 3: If the network (after Steps 0-2) is a tree, then orient it, always away from the root towards the leaves.

Lemma 3. *If Step 3 applied, its result is resolvable.*

Proof: Start at the root and apply the bx in turn until all leaves are reached. ■

However, note that it is still not necessarily confluent. For there are resolution paths that do not apply the bx systematically starting from the root, and these will not necessarily give the same result. For example, consider the fragment of the Section II example comprising the UML model, the Code and the Tests, with consistency restoration functions in that direction (model to code to tests), and suppose the UML model is in the authority set. Suppose that in the initial state, the UML model includes, and the Tests test, a class Foo which (perhaps because it has been deleted) does not appear in the Code.

If we restore consistency first by updating the Code with respect to the UML model, and then by updating the Tests with respect to the Code, the expected outcome is that class Foo reappears in the Code and (under natural assumptions) corresponding tests remain in the Tests.

However, if instead we first restore consistency between Code and Tests toward Tests, presumably tests of Foo will be deleted. If we go on to restore consistency between UML model and Code, towards Code, then class Foo will be reinstated in the Code; but even when we go on to restore consistency (again, having changed Code) towards Tests, the original tests that mentioned class Foo cannot be fully restored, because information about them that was only present in the Tests has been lost.

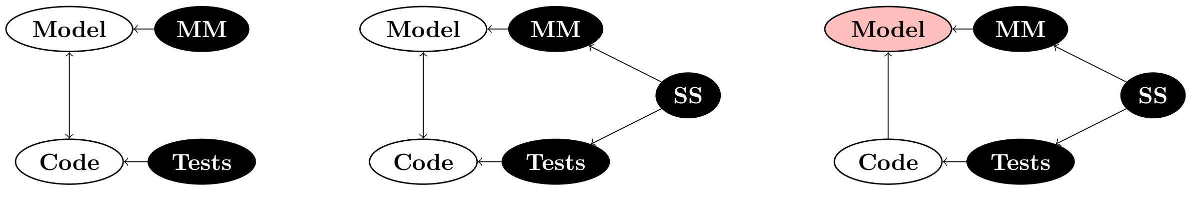


Fig. 7: Resolving a simple network

This is a familiar problem: the consistency restoration between Code and Tests is not *history ignorant*.

Definition 10. *$\overrightarrow{R}$ is history ignorant if for all $a, a' \in A, b \in B$, we have $\overrightarrow{R}(a, \overrightarrow{R}(a', b)) = \overrightarrow{R}(a, b)$. Dually for $\overleftarrow{R}$.*

Finally we get a positive result on confluence:

Lemma 4. *If Step 3 applied, and all the bx in the network have history ignorant consistency restoration functions in the relevant direction of orientation, then the network is confluent.*

Proof: Induction on branch length, making use of correctness and hippocraticness as well as history ignorance. ■

Note that outside this special setting, it is also not reasonable to expect to find a (non-exhaustive) algorithm that guarantees that if there is a resolution path, the algorithm finds it. For the existence of such a path might be dependent on a quirk of the behaviour of any one of the bx, such that finding it required applying that particular bx at a particular stage.

We can use non-interference, where available, to give us positive results beyond trees. Suppose we carried out Steps 0-2 but Step 3 did not apply.

Step 3′: Identify any nodes n that are not leaves, but have the property that all the consistency restoration functions that the network permits to be directed at n are non-interfering. If there is a set S of such nodes, such that deleting the nodes in S would turn the network into a tree, then pick one and colour the nodes in S.

Lemma 5. *If Step $3'$ applies, then the network is resolvable. Moreover if all the bx are history ignorant, then the network is confluent.*

Proof: Orient the network as in Step 3, and resolve the network as in Lemma 3, treating the coloured nodes as though they were leaves. Non-interference ensures (via Lemma 1) that performing all consistency restorations towards the coloured nodes does restore all the consistency relations, and moreover that it does not matter in which order these are carried out. ■

Figure 7 informally illustrates, on the assumption that the restoration functions into Model are non-interfering, as discussed earlier.

There is more that could be done in this direction; in particular if removing nodes with non-interfering edges does not result in a tree as required by Step $3'$, one could consider unfolding the network into a tree by duplicating the ongoing section of the graph. This is delicate and beyond the scope of this paper, however.

VII. Megamodelling

Megamodelling is a term applied first by Bézivin et al. to the practice of regarding models themselves, and their relationships, as objects of study. They write “A megamodel is a model of which at least some elements represent and/or refer to models or metamodels” [14]. Megamodelling is better seen as a mental discipline than as a technology. Megamodels are often, in practice, informal: one draws a diagram whose nodes are models, and adds (usually binary) relationships between the models, whose nature is explained in natural language (often stylised, with relationships taken from a fixed set e.g. “conforms to”, “instance of”, but usually not given a formal definition). It is possible, without loss of generality, to interpret these relationships as consistency relationships, as in our example: the current state of one model is consistent with the current state of the other model, if and only if the desired relationship does in fact hold. E.g., if a model is related to a metamodel via “conforms to”, we may interpret this as a consistency relation between the set of possible models and the set of possible metamodels, where a model is consistent with a metamodel if and only if it conforms to it.

Within this discipline we may conceive multiary transformations, and especially work on the resolution of networks of bx, as a contributing technology. For example, the megamodelling notation used in [15] has relationships between models that include “equals”, “contains” and, most tellingly, “overlaps”. “Overlaps” represents what we see as a general consistency relation; it is defined as “the content of both artifacts is overlapping, but might be represented differently; for example the word document overlaps in content with the HTML table”. One small part of the megamodel described there (Fig. A13 of [15]) includes nine models, all related to one another by one or more binary “overlaps” relationships.

VIII. Multiary variants of triple graph grammars

In [16] Königs and Schürr introduced the notion of *multi-graph grammars* (MGGs), a variant on the more familiar triple graph grammar (TGG) idea in which two graphs are related via a correspondence graph, and the coevolution of all three graphs is specified by a collection of TGG rules. The MGG is a straightforward extension of the TGG idea; the MGG rules specify how all the graphs, including a single correspondence graph, evolve simultaneously. The authors point out that this is impractical, and show how to derive binary operational rules from the MGG. They do not discuss, though, the semantic relationship between the derived rules and the original, or the issues that may arise from applying them over a network of models as we have considered here.

Trollmann and Albayrak build on this and other work to propose an extension to TGGs based on *graph diagrams* [17], [18]. A graph diagram is in our terms essentially a consistent network, but expressed in categorical terms which we did not need, in order to interoperate with the TGG literature. This interoperation is the main contribution of the work: the authors point out that they have not yet addressed many of the matters that concern us here.

IX. Other related work

A different approach, related to the CSP work discussed in Section III-B, is to take a program repair view of consistency restoration. Here constraints are expressed over all models, any of which may be modified; see for example [19].

Garcia [20] used a bidirectionalization approach based on [21] to define what he called a Declarative Model-View-Controller architecture. Macedo et al. [7] considered some issues that arise in taking seriously QVT-R's claim [4] to be able to maintain multiary consistency relations. As remarked, our Example 3 is borrowed from their paper. They note that QVT-R's insistence on a semantics in which *for all* valid bindings in all but one of the models, there must *exist* a valid binding in the one remaining ("target") model leads to expressivity restrictions, essentially because the *for all* antecedent may be trivially satisfied in the case where there is no valid binding of the other models. To remedy this, they give extra expressivity by introducing "checking dependencies": basically a way to say that one model depends on a subset (not all) of the other models. That is, they widen the class of multiary consistency relations expressible to those definable by a Horn-clause-like set of subrelations.

A pragmatic approach to pairwise model-merging is described in [22], where a set of models is merged by repeatedly identifying and merging the most similar pair; might this be useful in bx more generally? It would be interesting to explore resolving a network by identifying the closest to consistent edges and fixing these first. It will not always give the best result, in the same way that a naive metric-based least-change property is not always desirable [10]; e.g. when a single model is out of line, it may either be in error, or be a desirable first step towards a better consistent state.

There are several large areas of mathematics and computer science from which we might be able to learn, beyond what we have touched on here. One is the study of flow networks and transportation algorithms (as covered at undergraduate level in [23] for example). From that field we picked up the useful basic tool of adding a supersource, in Section VI. Perhaps thinking of changes as flowing round a network has more to offer. Returning to the issue of tolerating inconsistency, we note that we have not addressed what happens if, while we are in the process of restoring consistency across a network, further changes to models are taking place. Distributed algorithms experts understand such situations [24]. Perhaps the ideas of partial transformations with a well-founded partial order of consistency levels, as in [25], could play a role.

X. Conclusion

This paper has attempted to demonstrate some of the consequences of using bx in the large, to relate more than two models; we wanted to show how, eventually, bx may be an important tool in the automated management of development described by megamodels. This seems essential if MDD is to have a transformative effect on software development.

We started by considering carefully the senses in which multiary bx, are or are not, formally required in order to be able to express consistency of sets of more than two models; this is a topic that has caused some confusion in the Bx community. We hope to have clarified it by demonstrating that one obtains different answers to the question depending on one's assumptions about, for example, whether it is permitted to add extra models. In the process, we made some connections with the mature field of constraint solving. We went on to consider the difficult question of how consistency may be restored in a network of binary bx. Here our findings were mostly negative: this suggests that in practice, it will usually be necessary to manage the consistency restoration process, e.g. specify a resolution path not just an authority instance, rather than relying on confluence. However, we provided an algorithm and positive results for special cases. Finally we related the work to megamodelling and other parts of the literature, and pointed at some further possibly fruitful areas.

Aficionados of bx will have noticed that this paper has not discussed trace links, deltas or related issues: we have written in terms of the simplest, state-based relational, notion of bx as consistency restorers. There are several reasons for this. First, the simplicity of this approach enables us to get started. Second, in a megamodelling environment, which by its nature involves a disparate collection of tools, assumptions about tools maintaining information beyond the models themselves may be difficult to sustain, at least in a uniform way. Most importantly, however, once we work in a network setting, we can reframe a more expressive relationship between models, embodied in a witness structure such as a set of trace links, as a multiary relationship between the models and a trace model, where these are themselves related by simple relational consistency. Thus, we may argue, at bottom, everything can be seen as state-based and relational. This does not preclude, of course, value being obtained from working away from the bottom in some cases.

Acknowledgement

I thank Anthony Anjorin, Robin Bradfield, Jeremy Gibbons, James McKinna and the anonymous reviewers for helpful comments and discussion.

References

[1] Z. Diskin, H. Gholizadeh, A. Wider, and K. Czarnecki, "A three-dimensional taxonomy for bidirectional model synchronization," *Journal of Systems and Software*, vol. 111, pp. 298–322, 2016.

[2] Y. Xiong, H. Song, Z. Hu, and M. Takeichi, "Synchronizing concurrent model updates based on bidirectional transformation," *Software and System Modeling*, vol. 12, no. 1, pp. 89–104, 2013.

[3] A. Schürr and F. Klar, "15 years of triple graph grammars," in *ICGT*, ser. LNCS, vol. 5214. Springer, 2008, pp. 411–425.

[4] OMG, "MOF2.0 query/view/transformation (QVT) version 1.3," OMG document formal/2016-06-03, 2016, available from www.omg.org.

[5] B. Nuseibeh, S. M. Easterbrook, and A. Russo, "Making inconsistency respectable in software development," *Journal of Systems and Software*, vol. 58, no. 2, pp. 171–180, 2001. [Online]. Available: http://dx.doi.org/10.1016/S0164-1212(01)00036-X

[6] B. A. Davey and H. A. Priestley, *Introduction to lattices and Order*. CUP, 2002.

[7] N. Macedo, A. Cunha, and H. Pacheco, "Towards a framework for multidirectional model transformations," in *Bx*, ser. CEUR Workshop Proceedings, vol. 1133. CEUR-WS.org, 2014, pp. 71–74. [Online]. Available: http://ceur-ws.org/Vol-1133/paper-11.pdf

[8] F. Rossi, C. J. Petrie, and V. Dhar, "On the equivalence of constraint satisfaction problems," in *ECAI*, 1990, pp. 550–556.

[9] F. Bacchus, X. Chen, P. van Beek, and T. Walsh, "Binary vs. non-binary constraints," *Artif. Intell.*, vol. 140, no. 1/2, pp. 1–37, 2002. [Online]. Available: http://dx.doi.org/10.1016/S0004-3702(02)00210-2

[10] J. Cheney, J. Gibbons, J. McKinna, and P. Stevens, "On principles of least change and least surprise for bidirectional transformations," *Journal of Object Technology*, vol. 16, no. 1, pp. 3:1–31, 2017.

[11] K. Lano, "Constraint-driven development," *Information & Software Technology*, vol. 50, no. 5, pp. 406–423, 2008.

[12] K. Lano and S. Y. Tehrani, "Verified bidirectional transformations by construction," in *PAME/VOLT@MODELS*, ser. CEUR Workshop Proceedings, vol. 1693. CEUR-WS.org, 2016, pp. 28–37.

[13] N. Macedo and A. Cunha, "Least-change bidirectional model transformation with QVT-R and ATL," *Software and System Modeling*, vol. 15, no. 3, pp. 783–810, 2016.

[14] J. Bézivin, F. Jouault, and P. Valduriez, "On the need for megamodels," in *Proc. OOPLSA/GPCE workshop: Best Practices for Model-Driven Software Development*, 2004.

[15] R. Hebig, H. Giese, K. Batoulis, P. Langer, A. Z. Farahani, G. Yao, and M. Wolowyk, *Development of AUTOSAR standard documents at Carmeq GmbH: a case study*. Universittsverlag Potsdam, 2015.

[16] A. Königs and A. Schürr, "MDI: A rule-based multi-document and tool integration approach," *Software and System Modeling*, vol. 5, no. 4, pp. 349–368, 2006. [Online]. Available: http://dx.doi.org/10.1007/s10270-006-0016-x

[17] F. Trollmann and S. Albayrak, "Extending model to model transformation results from triple graph grammars to multiple models," in *ICMT*, ser. LNCS, vol. 9152. Springer, 2015, pp. 214–229.

[18] ——, "Extending model synchronization results from triple graph grammars to multiple models," in *ICMT*, ser. LNCS, vol. 9765. Springer, 2016, pp. 91–106.

[19] Y. Xiong, Z. Hu, H. Zhao, H. Song, M. Takeichi, and H. Mei, "Supporting automatic model inconsistency fixing," in *FSE'09*. ACM, 2009, pp. 315–324. [Online]. Available: http://doi.acm.org/10.1145/1595696.1595757

[20] M. Garcia, "Bidirectional synchronization of multiple views of software models," in *DSML*, ser. CEUR Workshop Proceedings, vol. 324, 2008, pp. 7–19. [Online]. Available: http://ceur-ws.org/Vol-324/paper1.pdf

[21] K. Matsuda, Z. Hu, K. Nakano, M. Hamana, and M. Takeichi, "Bidirectionalization transformation based on automatic derivation of view complement functions," in *ICFP*. ACM, 2007, pp. 47–58.

[22] M. Boubakir and A. Chaoui, "A pairwise approach for model merging," in *MISC'16*. Springer, 2016, pp. 327–340. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-33410-3_23

[23] D. C. Luenberger, *Linear and Nonlinear Programming*. Addison Wesley, 1984.

[24] N. Lynch, *Distributed algorithms*. Elsevier, 1996.

[25] P. Stevens, "Bidirectionally tolerating inconsistency: Partial transformations," in *FASE*, ser. LNCS, S. Gnesi and A. Rensink, Eds., vol. 8411. Springer, 2014, pp. 32–46.