Software & Systems Modeling (2020) 19:39–65
https://doi.org/10.1007/s10270-019-00736-x

SPECIAL SECTION PAPER

Maintaining consistency in networks of models: bidirectional transformations in the large

Perdita Stevens[1]

Received: 15 June 2018 / Revised: 13 May 2019 / Accepted: 15 May 2019 / Published online: 29 May 2019

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 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. In particular, we consider the circumstances under which we can prove *non-interference* between several bidirectional transformations that impact on the same model and how the use of a more refined notion of consistency can help in cases where this is not possible. In the process, we develop an abstract theory of parts of a model that are read or modified by a bidirectional transformation. We relate the work to megamodelling and discuss further research that is needed.

Keywords Model-driven development · Bidirectional transformation · Consistency · Megamodel · Model decomposition · Non-interference

1 Introduction

Model-driven development (MDD) has achieved some success; but it has not yet transformed software development, and a transformation is badly 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. Just to give a few examples, the development of a software system might involve: a design model in UML; the UML metamodel; a database schema; some code; a test suite; a safety model; a model recording the user's navigation through the user interface; etc. Each model is adapted to the needs of its users, so that they can work with maximum efficiency and effectiveness. However, the contents of the models are not independent: a decision which one stakeholder makes, and records in their model, may necessitate a change in another model, so that a suitable consistency relationship between them is maintained, in order that the eventual software system produced may correctly incorporate the decisions recorded in all the models. For example, if a class is deleted from a UML model, tests of that class may need to be deleted from a test suite. It is possible to maintain all the relationships between the models manually, using human communication between the stakeholders. However, this manual process is painful, expensive and error-prone. It may result in much of the benefit of separation of concerns being lost.

In MDD, models are, as far as possible, related instead by *model transformations*.[1] Because circumstances do not

Communicated by Mr. Vinay Kulkarni and Prof. Dr. Jeff Gray.

✉ Perdita Stevens
perdita@inf.ed.ac.uk

[1] Laboratory for Foundations of Computer Science, University of Edinburgh, Edinburgh, UK

[1] Note that, just as we use a broad notion of "model" that includes metamodels, code, etc., our notion of "transformation" can cover the

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 an automated 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 [12,37].) Note that restoring consistency typically requires using information about the current state of all the models, not just regenerating one. This is because typically, each model includes some information which is held in it and nowhere else. Such information cannot, by definition, be generated from the other models. Therefore, when a model needs to be changed to bring it into consistency with changes in other models, its own current state has to be part of the input to the consistency restoration process, so that important information unique to it is not lost in the process.

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) [29] or a QVT-R transformation [26]) records the bx developer's decisions about how to carry out all of these tasks, so as to minimise the duplication of 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 in our imagined future world is a collection of models, somehow related by bx, each relating some of the models. Some of these bx may be bought off-the-shelf, while others are developed specially for the project.

This, however, raises many questions which have not yet been addressed. The aim of the paper is to begin to address them, though we will not complete that undertaking in this one paper. They include:

– Granted that we do not expect a single bx to relate all the models, how many models do we expect a bx to relate? Do we limit the notions of consistency that can be expressed, if we insist that consistency of the whole 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? Can we model this as a network whose nodes are models and whose (hyper)edges are bx? What does it mean to restore consistency of such a network?
– 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?
– What if we cannot fully restore consistency to the collection of models after one set of changes, before more changes begin to happen?

The main contributions of this paper are:

- We clarify the senses in which multi-directional transformations, are or are not, formally required in order to be able to express consistency of collections of more than two models. We demonstrate that this depends on whether it is permitted to add extra models or refine the consistency relation. In the process, we make some connections with the mature field of constraint solving.
- We consider the restoration of consistency in a network of models connected by binary bx. We demonstrate that consistency restoration may be impossible to achieve at all, or impossible to achieve using the consistency restoration functions of the network's binary bx, and that even if it can be achieved, different ways of achieving it may give different results. We study some special cases where positive results are obtainable.
- We study in detail the situation in which two bidirectional transformations may act on the same model without interfering with one another. We give two equivalent sets of sufficient conditions for this.
- In doing so, we formally define a notion of *part* of a model, which gives a vocabulary for talking about what a bx reads and what it modifies. Our definition of part

Footnote1 continued
automatic maintenance of any relation between models: e.g. conformance between a model and a metamodel.

is very general, capturing not only structural parts, but other ways to identify some, but not all, of the information contained in a model: formally a part is an equivalence relation on a set of models.
- We show, perhaps counter-intuitively (it was counter to the author's initial intuition, anyway), that the conditions we showed were sufficient for non-interference are *not* also necessary.
- We discuss the need to tolerate inconsistency at times and the role played by partial bx in doing so.
- We relate the work to megamodelling and other parts of the literature and discuss how our findings, especially the negative ones, motivate future work in that direction.

1.1 Paper organisation

After an example (Sect. 2), Sect. 3 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. (The reader who is already happy with that limitation could skip Sect. 3.) Before we turn to how consistency is *restored*, Sect. 4 suggests we sometimes need to tolerate inconsistency and discusses the implication for restoring consistency in the presence of many models. Section 5 formalises networks of bx and introduces key concepts of restoring consistency in such networks. Section 6 concerns the important property of non-interference between several bx that impact the same model, and Sect. 7 goes on to give sufficient conditions for this to hold and discusses why these are not also necessary conditions. Section 8 is about how to discuss less ideal cases where non-interference may not hold, making use of work on bidirectional transformations that may not perfectly restore consistency but may nevertheless have useful properties. Section 9 considers some situations in which consistency restoration in networks is feasible. Section 10 discusses various strands of related work, beyond what has been covered along the way. Section 11 concludes.

This paper is an extended version of [32], which was presented at MODELS'17. Sections 7 and 8 are new; examples, proofs, explanation and discussion have been added throughout.

1.2 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.

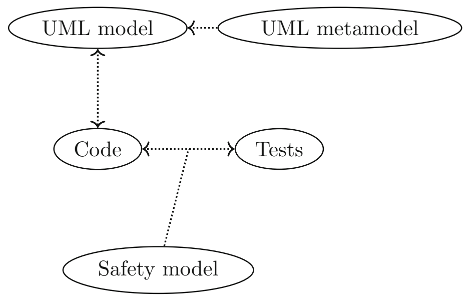


Fig. 1 A small megamodel

As "bidirectional transformation", abbreviated **bx**, has hitherto been widely used to include artefacts aiming to maintain consistency between more than two models, we now need to be careful about terminology. When it relates two models, a bx maintains a *binary* consistency relation, that is, a relation on two model sets. When we want to talk specifically about bx that relate more than two models, we will use the term multi-directional transformation, abbreviated **multx**. A multx maintains a *multiary* consistency relation, that is, a relation on an arbitrary number of model sets. We prefer the term *multiary* to *n-ary* because it makes clear there is no fixed arity n.

2 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 shown as dotted lines between the models in the diagram 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. Let us say that if it is safety critical, there is a coverage criterion that must be satisfied; perhaps, that every line of the code must be exercised by some test. 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. These are not recorded in the diagram because there is no advantage to including them: if the changes to one model are somehow correlated with the changes to another, but the correlation is not formalised in any way, then any automated restoration of consistency in the network cannot take it into consideration. Instead it must just take both models as it finds them.

Remark 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.

In the context of this example, the questions from Sect. 1 appear as:

– Is it essential to represent the ternary relationship between Code, Tests and Safety model explicitly? Or can we replace it, somehow, by binary relationships? (Sect. 3.)
– We have informally drawn the example as a network, but do any surprises arise when we formalise this? What does it mean to restore consistency in the whole network? Is it reasonable to conflate restoring consistency across all of the edges shown in our diagram, with restoring the whole collection of models to a consistent state? (Sect. 5.)
– Will we always be able to restore consistency in this network, regardless of the decisions taken by the owners of the different models? (Clearly no!) What guarantees can we get? (Sect. 5.)
– 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? (Sect. 9.)
– What if we cannot fully restore consistency to the collection of models after one set of changes, before more changes begin to happen? (This question is not answered in this paper, but see Sects. 8 and 11.)

3 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 [24] 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.

3.1 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 the cartesian product $\prod_i M_i$ (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 is 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 the cartesian product $\prod_i M_i$ 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 [11][2]) the standard definition

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

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

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

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

- $b \geq LU(b)$
- $b \geq b' \Rightarrow LU(b) \geq LU(b')$ and
- $LU(LU(b)) = LU(b)$.

Further, $ULU = U$.

We are now ready to give a precise relationship between the two most obvious ways of connecting multiary relations with sets of binary relations:

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 Monotonicity of g: if $R \subseteq S$ and $(gR)_{ij}$ holds of (m_i, m_j) then there exists $\overline{m}$ extending (m_i, m_j) such that $R\overline{m}$ holds; but then $S\overline{m}$ holds so $(gS)_{ij}$ holds of (m_i, m_j).

Monotonicity of f: if $\{R_{ij} : i < j \in I\} \sqsubseteq \{S_{ij} : i < j \in I\}$ and $f(\{R_{ij}\})$ holds of $\overline{m}$ then each R_{ij} holds of corresponding (m_i, m_j), so S_{ij} holds of (m_i, m_j), so $f(\{S_{ij}\})$ holds of $\overline{m}$.

We next have to show that for any multiary relation R and set of binary relations, say $\{S_{ij} : i < j \in I\}$, we have

$$gR \sqsubseteq \{S_{ij} : i < j \in I\} \tag{*}$$

iff

$$R \subseteq f(\{S_{ij} : i < j \in I\}).$$

For the forward direction, assume (*) holds, take any $\overline{m}$ satisfying $R(\overline{m})$, and for any $i < j$ consider the pair (m_i, m_j) drawn from m. We have to show that $S_{ij}(m_i, m_j)$ holds. But, since $\overline{m}$ is an example of a tuple extending (m_i, m_j) such that $R(\overline{m})$ holds, this follows from (*).

For the reverse direction, assume that $R \subseteq f(\{S_{ij} : i < j \in I\})$. We have to show (*), i.e. that for every $i < j \in I$ we have $(gR)_{ij} \subseteq S_{ij}$. Assume $(gR)_{ij}(m_i, m_j)$ holds, so that there exists $\overline{m}$ extending (m_i, m_j) such that $R(\overline{m})$. By assumption, $f(\{S_{ij} : i < j \in I\})(\overline{m})$ holds; in particular, $S_{ij}(m_i, m_j)$ as required.

The rest of the statement follows from the existence of the Galois connection. More explicitly, now that we have shown we have a Galois connection, we know that

$$fgf = f \tag{+}$$

It follows that the set of closed multiary relations, i.e. fgY (where Y is the set of all multiary relations), is identical with the image of f, i.e. fX, where X is the set of all collections of binary relations. For, $gY \subseteq X$ so $fgY \subseteq fX$, i.e. a multiary relation being closed implies that it lies in the image of f. Conversely, if R is in the image of f, say $R = f(\{R_{ij}\})$, then $fgR = fgf(\{R_{ij}\}) = f(\{R_{ij}\})$ by (+). This is R, so R is closed as required.

To see that we do not have an order isomorphism, observe that f is not a bijection: for example, the set of empty binary relations maps to the empty multiary relation, but so does the set of binary relations comprising some empty relations and just one non-empty relation. □

Why is it useful to show that this Galois connection exists? One reason is that it provides reassurance that we are not making a wrong choice of how to connect multiary relations and sets of binary relations. There are two obvious ways to form such a connection, formalised as g and f in the preceding theorem. On the one hand, we can do what g does. That is, given a multiary relation, we can define a family of binary relations, by considering a pair of elements to be related if, and only if, it can be extended to a tuple that lies in the given multiary relation. On the other hand, we can do what f does. That is, given a family of binary relations, we can define a multiary relation by conjoining the given binary relations on all pairs of elements drawn from a tuple. On the face of it, these might be unrelated notions, and which approach we should chose might be an empirical question. The existence of a Galois connection between the two, which we have just proved, demonstrates that we may, rather, think of these two approaches as complementary expressions of one idea, so there is no need to make a choice between them. In particular, it shows that the binary-definable multiary relations have a special status, as the closed elements of the set of multiary relations, relative to this Galois connection. This brings us to the second reason why it is useful to exhibit a Galois connection: it enables us to call on standard results about Galois connections (such as those listed just before the theorem) when we want to show whether or not a given multiary relation is binary-definable. The following example illustrates this.

Example 2 Returning to the example from Sect. 2, 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.

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 [21] (we will consider their paper in more detail in Sect. 10).

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 Fig. 3) specifies some features, which may be mandatory or not, while a configuration (e.g. (a), (b) or (c) in Fig. 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 every one of 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.

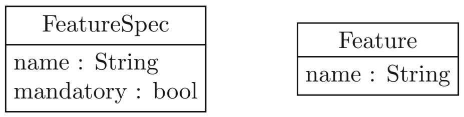


Fig. 2 Metamodels for a product specification (left) and configuration (right), from [21]

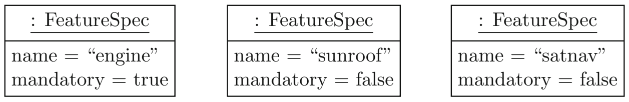


Fig. 3 Product line P

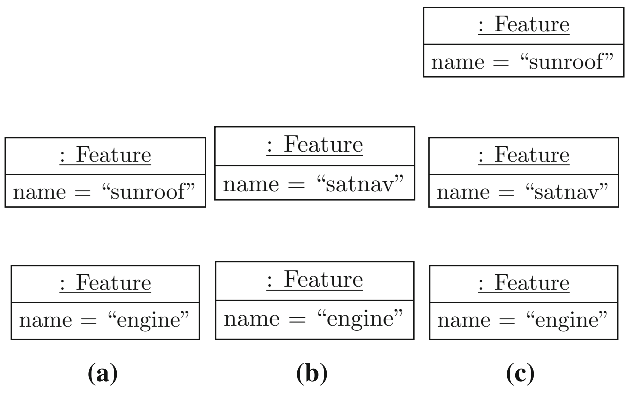


Fig. 4 Models $\langle P, (a), (b)\rangle$ consistent; $\langle P, (a), (c)\rangle$ inconsistent

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 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.

3.2 Optimistic view

3.2.1 Approach 1: add extra models

One approach we may try, as we seek to investigate whether we can somehow express arbitrary consistency relationships between models using only binary bx, is to add one or more extra models for the specific purpose of helping to express the overall consistency. We know from the preceding subsection that we cannot express all multiary relations between model sets using binary relations over exactly the same model sets, but perhaps it might be worth incurring the cost of creating and maintaining one or more extra models, either automatically or manually, if it would allow us to express what we need using formalisms that can only express binary consistency relations? In a specific example, it is not hard to develop the intuition that this can be done. Considering the ternary relation between Code, Tests and Safety model in the example of Sect. 2, we could imagine building a new, single, model (call it CodeTestsSafety) that incorporated the information from all three models. Then, we can think about the binary consistency relation between Code and CodeTestsSafety, which specifies exactly that these two models agree on the Code information. Similarly for the other two models. Our intuition is that this should express our intentions correctly: that is, that our original ternary relation should hold if, and only if, all three of the new binary consistency relations hold with the new model. We could generalise and formalise this construction, but fortunately, another community has already done so.

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 [28] 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 [2] 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. By contrast, in the MDD world, we typically care not only *that* consistency is restored, but also *how*; for example, we may want to ensure appropriate least-change properties, in order to limit the surprise that a user of a model may experience when their model is changed by a bx [7]. The programmer of each individual bx makes appropriate choices of resolution behaviour as they program the bx, so it is reasonable to insist that resolution in the network of models be effected by means of the consistency restoration functions of these binary bx. This is a constraint on the resolution possibilities. We shall return to this point after introducing bx networks: see Example 5.

3.2.2 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!

This is unsatisfactory, because what we were ideally looking for, when we defined "binary implementable", was a guarantee that, if we started with a binary implementable multiary relation and then found a collection of binary relations that implemented it, then we could sensibly develop against the implementing binary relations. We accept that the process of replacing the multiary relation by a collection of implementing binary relations may exclude *some* of the solutions, just as making other architectural decisions, such as programming language paradigm, excludes some solutions. However, we would hope that the binary relations would still allow enough flexibility that a wide class of solutions is still available—that is, a large subset of the set of all tuples satisfying the original multiary relation should also satisfy, pairwise, the implementing binary relations. Otherwise, the implementing binary relations will not be useful to develop against.

That any multiary relation is implemented by any unsatisfiable set of binary relations contradicts that hope, but arguably in a pathological and uninteresting way. We next wonder whether we can make progress by tweaks to the definition of "binary implementable", such as excluding unsatisfiable sets of binary relations.

The next theorem attempts to capture the intuition that this is not practical: whatever reasonable exclusions we try to make, we will still be vulnerable to back-forming the implementing binary consistency relations from a single tuple satisfying the multiary relation. That is, if there is any way at all to satisfy the multiary relation, there is a collection of implementing binary relations that permits *only* that one solution. That collection is not useful to develop against: it amounts to requiring that each model owner should guess the one model that is acceptable to the implementing binary relations. The existence of such a non-useful set of implementing binary relations does not rule out that binary-implementable is a useful notion, but it demonstrates that it is false that *every* set of implementing binary relations is useful.

To make this formal, we need to make the relationship between the models, the multiary consistency relation and the eventually implemented system, more explicit. We need two key properties. First, between them, the models are supposed to capture all the important decisions taken during development. That is, together, the whole collection of models should *determine* the eventually implemented system (up to some equivalence relation, since we do not expect the models to determine every last trivial detail of the system). Second, the multiary consistency relation is supposed to capture the requirements on the eventually implemented system. That is, once we agree that a given system is correctly modelled by a given collection of models, it should be the case that the system satisfies the requirements, if, and only if, the collection of models satisfies the multiary consistency relation.

We formalise this in terms of:

– a set $\mathcal{S}$ of equivalence classes of systems (we use equivalence classes simply to record that two systems may differ only in details that do not need to be modelled and do not affect whether the system satisfies the requirements)
– a collection of model sets M_i (for i in some index set I)
– a collection of satisfaction relations between (equivalence classes of) systems and models, capturing when the model models the system, i.e. the decisions recorded in the model have been incorporated into the system, i.e. the model is true of the system. We write $S \models m_i$, where $S \in \mathcal{S}$ is an equivalence class of systems and $m_i \in M_i$ is a model.

In the following theorem, we impose conditions in an attempt to exclude trivial and pathological situations. We forbid unsatisfiable relations; we also exclude situations in which the models fail to record enough information to know whether the resulting system meets the requirements or not. (Notice, though, that the function f in the theorem statement that takes us from a collection of models to a class of systems must be partial, because some collections of models may contradict one another, i.e. not correspond to any system at all, whether or not meeting the requirements.)

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. If we did not do this, we would still get the same result—the equivalence plays no technical role in the theorem—but we might wonder whether trivial details in the system, that we should have abstracted away, accounted for the result. Writing in terms of equivalence classes assures us that this is not the case.

It turns out that, despite all these precautions, our tweaked notion of binary implementable is still essentially vacuous, and that an implementing set of binary relations still might not be a useful set of relations to develop against:

Theorem 2 *Let $\mathcal{S}$ be a set of equivalence classes of systems, $\Phi \subseteq \mathcal{S}$ be those deemed to satisfy the requirements, and $\{M_i\}$ be a collection of model sets such that partial function $f : \prod_i M_i \rightharpoonup \mathcal{S}$ 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 any tuple of models $g = \langle g_k \rangle \in R$, which we know we can do because R is satisfiable. Now, we will use this tuple to define a family of binary consistency relations as required. The binary relation R_{ij} between M_i and M_j is defined by: for any model $m_i \in M_i$ and $m_j \in M_j$, we define $R_{ij}(m_i, m_j)$ to be true iff $m_i = g_i$ and $m_j = g_j$. That is, the *only* way to satisfy this binary relation is to pick exactly the models that were in the tuple we chose up front. The set of binary consistency relations $\{R_{ij} \subseteq M_i \times M_j\}$ defined in this way satisfies the condition in Definition 3, because the antecedent is only true of the tuple g itself! Moreover, these R_{ij} are simultaneously satisfiable (by g). □

Is this really a problem, or might it be considered a feature? 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. Now, the existence of a non-useful set of implementing binary relations does not, of course, prove the non-existence of a useful set, but it does mean that we cannot have what we hoped for, namely a guarantee that we could reasonably replace the multiary relation by any implementing set of binary relations we could find.

There is another potential disadvantage of taking the view that we should be prepared to give up some development flexibility and work with a binary-implementing set of binary consistency relations, instead of a multiary consistency relation that expresses the true requirements. As work is done, simultaneously, on all of the models, it could happen that at a certain moment, the collection of all the models is indeed correct according to the multiary relation that captures the true requirements—but that it does not satisfy all of the implementing binary relations we chose to develop against.

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.

However, there are also arguments that we need not worry about this. It is a normal part of development that, sometimes, work must be done which is necessary to adhere to the chosen development process and norms, not, strictly speaking, in order to meet the requirements. Moreover, if in practice consistency is restored frequently, following small changes, we may expect that the evolution of the models will follow paths that mean irritating "hippocraticness failures" are rare. The informal justification for this intuition is that, if the bx are written so as to satisfy a reasonable *least change* property [7], then these frequent changes effected by the bx will themselves tend to be small, and the overall set of models will never get far away from a set that is consistent according to the binary bx.

3.3 Discussion

Drawing the work of this section together, it is time to ask: should we conclude that future languages for programming bidirectional transformations must allow the programmer to express multi-directional transformations, or that they need not?

As we have seen, a language that does not have multiary syntax does not permit the expression of all possible consistency relations between a given set of models. However, we considered two ways that a bx developer faced with an inexpressible consistency relation may get round the problem. S/he may add an extra model, related to the original ones by binary bx; or s/he may refine the requirements, in effect making development decisions, to give a more stringent consistency relation that can be expressed. Each of these approaches has its disadvantages, but it seems reasonable to argue that between them they would make it practical to use a language without multiary syntax.

Finally, it is worth considering the task that has to be done by the person, or people, who are specifying consistency. Intuitively, it seems difficult to think about multiary relations, without resorting to considering how they restrict to binary relations, and as we have seen, this may lead to mistaken thinking about non-binary-definable relations. So perhaps, a formalism involving requiring people to specify multiary relations that may not be binary-definable might tend to lead to errors due to cognitive difficulties. This is only a hypothesis, which could be investigated empirically; but it should be considered, because language features which are error-prone are often better omitted from the language. In fact, in the author's opinion, this will probably turn out to be the strongest argument for working with binary consistency relations.

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.

4 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 all-encompassing model, which, even if it is not reified, notionally contains all the data from all models. 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 [24] 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.

A particularly interesting question, once we move away from the idea of instant and complete restoration of consistency, is whether a "push" or "pull" approach—or perhaps some third way—should be used. To illustrate the difference, consider a network of models that is currently in an inconsistent state, e.g. because one or more of the models has changed since an earlier time when all the models were consistent. Should our ideal tooling expect, in normal operation, that these changes will be rippled outwards to surrounding models, eventually restoring consistency to the whole collection as a direct result of the changes? This is what we refer to as a "push" model. Or, should the other models be, as far as possible, unaffected unless and until someone wants to use one of them, at which point the changes are "pulled" through as a result of a request for this model to be brought up to date? The "push" model has the advantage that consistency may be restored throughout the network, soon after the changes that require it, which may be important if, in practice, the restoration reveals problems with the changes, that require human attention. It is the model that we generally assume in this paper. The "pull" model has the (dual) advantage that it does not do work of updating models that turns out to be not needed. This, for example, is likely to be beneficial when many small changes are made to the same model in quick succession: it reduces disruption to the other models if they are only updated when required, and can then incorporate the effects of a whole sequence of changes. On the other hand, potentially disruptive changes cannot be completely avoided, either way. It is not possible to limit the changes to *only* the model whose update is requested, because models that sit in the network between the changed and the requested model may also be affected. Albeit in a context with rather different concerns to those of this paper, [33] begins work on consistency restoration in networks with a pull model. In the end, each choice has pros and cons; which works best is a matter that will have to be settled empirically.

4.1 Multi-view modelling

The need to tolerate inconsistency is, arguably, the fundamental problem encountered in the various approaches referred to as multi-view modelling. In some varieties of multi-view modelling, the idea is that there is an explicit single all-encompassing model that includes all the information from all the models. Each of the models that is actually used in development is then a view onto this model. In [30], for example, SysML profiles are used to allow SysML to be used as the language of the all-encompassing model. Theoretically, this is a very appealing approach. However, the authors point out that "a general framework must also take into account the work flow process, to allow consistency to be evaluated and reestablished periodically but not enforced all the time", and, though they consider that their approach is flexible enough to permit the development of such a framework, they do not undertake it. By contrast, work on RM-ODP, such as [4], takes the views as the primary artefacts. They serve as requirements on an all-encompassing model, which will only exist if the views are all consistent.

A recent and useful survey of the field of multi-view modelling is [8]. Based on its systematic literature review, [8] summarises by saying "there is a lack of support for semantic consistency management" which needs to be remedied: this paper is a step in that direction.

5 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 $N \in \mathcal{N}$ 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 $\mathcal{N}$, 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 situation 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. It is possible to centralise and record the decisions about which models are currently authoritative, and if we wish other decisions such as in which direction to apply a bx, by means of an *orientation model* [33].

5.1 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 $\overrightarrow{R} : M \times N \rightarrow N$ and $\overleftarrow{R} : M \times N \rightarrow M$. Except in Sect. 8, these will always be assumed to be correct and hippocratic.

Definition 5 A bx R is *correct* if for all $m \in M$ and $n \in N$ we have $R(m, \overrightarrow{R}(m, n))$, and dually.

Definition 6 A bx R is *hippocratic* if for all $m \in M$ and $n \in N$ we have $R(m, n) \Rightarrow \overrightarrow{R}(m, n) = n$, and dually.

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 $f : M \rightarrow N$ 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 7 To *orient* an edge is to choose a direction of application; to orient an authority instance is to do this for every edge.

Definition 8 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 $I \mapsto \ldots \mapsto J$ such that J is consistent. We may identify a resolution path by giving a list $\langle (e_i, d_i) \rangle$.

Definition 9 I is *resolvable* if there exists a resolution path $I \mapsto J$.

Definition 10 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, then 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 incident on nodes in the authority set will need to be directed out of those nodes, since by definition the authority set nodes cannot be altered.

5.1.1 Existence of resolution

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

Real-life analogues of this schematic formal example will occur whenever it is possible for two models to embody decisions that have contradictory implications for a third model. Consider the example from Sect. 2, and suppose that the UML model includes a class `Customer`, but the Tests include no tests for such a class. If the consistency relations between the UML model and the Code, and between the Code and the Tests, both specify that the models must agree on what classes there are, then there is no possible value for the Code that will be consistent with both the UML model and the Tests. So if they are taken to be authoritative, then an attempt to change Code to restore consistency in the instance must fail.

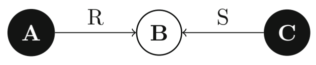
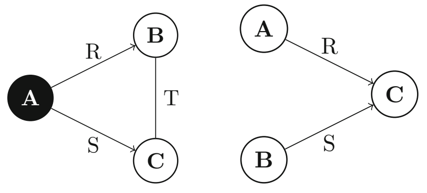


Fig. 5 Not all authority instances have resolutions



(a) Not all instances that have resolutions are resolvable

(b) Some consistency restorers with common target commute

Fig. 6 **a** Not all instances that have resolutions are resolvable. **b** Some consistency restorers with common target commute

5.1.2 Resolvability

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

Example 5 Consider the network shown in Fig. 6a.

Let $A = \{a_0, a_1\}$, $B = \{b_1, b_2\}$, $C = \{c_1, c_2, c_3\}$. Suppose that the current states of the models are (a_1, b_1, c_1), and that $\{a_1 : A\}$ is the authority set. Define R, S and T to be correct and hippocratic bx with restoration functions defined by the following tables. Here, the restoration function's name is shown in the top left, its first argument in the leftmost column, its second argument in the top row, and the result of evaluating the restoration function on those arguments in the corresponding place in the rectangle. For example, $\overrightarrow{R}(a_0, b_1) = b_1$.

$\overrightarrow{R}$	b_1	b_2
a_0	b_1	b_2
a_1	b_1	b_1

$\overleftarrow{R}$	b_1	b_2
a_0	a_0	a_0
a_1	a_1	a_0

$\overrightarrow{S}$	c_1	c_2	c_3
a_0	c_1	c_2	c_3
a_1	c_1	c_2	c_1

$\overleftarrow{S}$	c_1	c_2	c_3
a_0	a_0	a_0	a_0
a_1	a_1	a_1	a_0

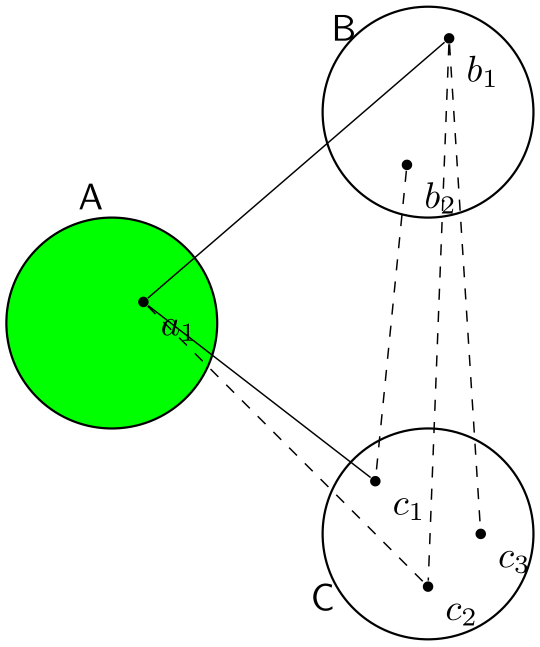


Fig. 7 Illustration for Example 5

$\overrightarrow{T}$	c_1	c_2	c_3
b_1	c_3	c_2	c_3
b_2	c_1	c_1	c_1

$\overleftarrow{T}$	c_1	c_2	c_3
b_1	b_2	b_1	b_1
b_2	b_1	b_1	b_1

Note that we can read off the consistency relations from the tables, using the fact that the bx are to be correct and hippocratic; for example, we see that $R(a_1, b_1)$ must hold, since $\overrightarrow{R}(a_1, b_1) = b_1$. Figure 7 illustrates: here, solid lines indicate consistent pairs in the current configuration, while dashed lines indicate other consistent pairs. a_0, which is consistent with everything, is omitted for clarity (its role in the example is simply to make it possible to define $\overleftarrow{R}$, $\overleftarrow{S}$ that make the transformations correct and hippocratic; since $a_1 \in A$ is authoritative, we can never move to it in any case).

Now, there is a resolution, viz. (a_1, b_1, c_2). However, it is easy to see that there is no resolution path that reaches this (or any other) solution: since $R(a_1, b_1)$ and $S(a_1, c_1)$ 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.

The next example is merely a more practical dressing up of Example 5: readers who have got the point should skip it.

Example 6 Consider another variant of the example from Sect. 2, in which we do not consider the Safety model or Metamodel, but we do have three bx relating Model, Code and Tests.

1. Suppose the bx CodeTests between Code and Tests maintains a consistency relation that insists (among other things that do not concern us) that there must be a test in Tests for every class in Code. Consistency restoration in the direction of Tests will (somehow) generate an appropriate set of tests.
2. Suppose the bx CodeModel between Code and Model maintains (among other conditions that do not concern us) that any sequence diagram in the Model that is *not* placed inside a package named "TestCases" must correspond to a public method in the Code. Consistency restoration in the direction of Model will delete any sequence diagram violating this.
3. Suppose the bx ModelTests between Model and Tests maintains the consistency relation that there must be a sequence diagram in the Model corresponding to every test in Tests (but that the consistency relation says nothing about the name of the package in which such sequence diagrams must reside). Consistency restoration towards Model will generate any necessary sequence diagrams, placing them in a package named "Generated". Consistency restoration towards Tests will delete any tests that lack sequence diagrams.

Now, suppose that Code is authoritative, and consistent with both the Model and the Tests—in fact, let us supposed that the tests are exactly those that CodeTests would generate from the current Code—but that there are no sequence diagrams for the tests in the model. Only applying the bx ModelTests will change anything. But, if we apply it towards Tests, tests will be deleted (because they lack sequence diagrams) breaking the consistency between Code and Tests; if we now apply CodeTests (towards Tests, which is the only option because Code is authoritative), we will be back where we started. On the other hand, if we apply ModelTests towards Model, sequence diagrams will be generated and placed in a package which is not named "TestCases" (it is instead named "Generated"), breaking the consistency between Model and Code. If we now apply CodeModel (towards Model, the only option as Code is authoritative), the offending sequence diagrams will be deleted, and we will be back where we started.

To a human, the solution is obvious: we should generate sequence diagrams as above, but then rename their package "Generated" to "TestCases", at which point all three consistency relations will be satisfied. This, however, is not behaviour which is achievable through the consistency restoration functions of the bx as described. As in Example 5, there is a resolution, but there is no resolution path that reaches it.

Confluence 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 Fig. 6a an example where the choice between using $\overrightarrow{T}$ and $\overleftarrow{T}$ prevents confluence, and on Fig. 5 an example where the choice of which edge to use first does so.

5.1.3 Consequences

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 [16,17], and from a different angle [20]). However, developers care not only about consistency but also about how it is restored, and we know [7] 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.

6 Non-interference

Intuitively, the main difficulty that arises in networks of models connected by bidirectional transformations is that several bidirectional transformations may affect, and hence impose constraints on, the same model. As we have seen, sometimes this will result in no solution being available. However, intuitively, we expect that in many cases, no such problem will arise: for example, it may be that the transformations do not care about the same aspects of their common target model, so their applications commute. (We are being deliberately informal here: as we shall see, 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.) Let us consider this situation more carefully. Consider a fragment of a network as illustrated in Fig. 6b.

Definition 11 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 some trivial cases, and some formal consequences, before returning to consider another class of situations where we may be able to take advantage of this property, and then exploring exactly when it holds more carefully.

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.

Even more trivially, if C contains only one model, then *any* $\overrightarrow{R}$ and $\overrightarrow{S}$ targetting C are non-interfering: both sides of the equation in Definition 11 must always evaluate to this single model, so they must be equal!

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.* □

It is tempting to think informally of the common target of non-interfering consistency restorers as being able to be factored into two separate parts for the two transformations. However, as, even if such a factoring exists (and we shall see in the next section that it might not) this factoring may not be reasonable to present in practice. So 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 $R : A \nleftrightarrow C$ and $S : B \nleftrightarrow C$ with $\overrightarrow{R}$ and $\overrightarrow{S}$ non-interfering, as before.*
We can define $RS : A \times B \nleftrightarrow C$ 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.

Proof Immediate from the definitions. □

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 $Q : D \nleftrightarrow C$ then $\overrightarrow{Q}$ needs to be non-interfering with *both* $\overrightarrow{R}$ and $\overrightarrow{S}$, if it is to be non-interfering with $\overrightarrow{RS}$. Details omitted.

The converse of Lemma 2 is false, however: given correct and hippocratic $T : A \times B \nleftrightarrow C$ we might not be able to factor it into correct and hippocratic $R : A \nleftrightarrow C$ and $S : B \nleftrightarrow C$—that is, there might not be any correct and hippocratic R and S such that $T = RS$—because RS might have "twisting" behaviour on $A \times B$. The following example illustrates.

Example 7 Let $A = \{a, a'\}$, $B = \{b, b'\}$ and $C = \{c, c'\}$, and let T be any correct and hippocratic bx $T : A \times B \nleftrightarrow C$ having the following consistency relation:

T	c	c'
(a, b)	$\top$	$\bot$
(a', b)	$\bot$	$\top$
(a, b')	$\bot$	$\top$
(a', b')	$\top$	$\bot$

(The behaviour of $\overrightarrow{T}$ is determined by the consistency relation; there is a free choice of whether to set $\overleftarrow{T}((a, b), c')$ to (a', b) or to (a, b'), and three other similar free choices.)

Now T cannot be the bx RS produced by combining any pair of correct and hippocratic bx $R : A \nleftrightarrow C$ and $S : B \nleftrightarrow C$ as in Lemma 2. For the fact that $T((a, b), c)$ forces $R(a, c)$ and $S(b, c)$, and the fact that $T((a', b'), c)$ forces $R(a', c)$, so in the combination $T = RS$ we would have $T((a', b), c)$, which is false.

6.1 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 Sect. 2, 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.

7 Sufficient conditions for non-interference

It is natural to ask whether we can formalise the intuition that non-interfering bx care about different parts of their common target model. It turns out, as so often in the study of bx, that the question is more interesting than at first appears.

First, what is a "part" of a model? Sometimes there is a natural notion of part, but often things are more blurred. The first thing to notice is that in this context the notion of part actually needs to be defined over a set of models, not just for an individual model: the conceptual importance of the notion of part is that we can talk about the "same" part of several different models or versions of a model.

Next, we probably do not want to limit ourselves to a notion of part that relates closely to the notion we use for physical objects, where we might talk, ultimately, about certain collections of *adjacent* molecules. Our "parts", by contrast, may have semantic reality, rather than geometric/spatial reality. For example, a bx that translates all the strings in a model will normally be non-interfering with one that restructures it without touching the strings. That is, we might like "all the strings" to count as a part, as well as some suitable notion of "the structure".

A very general notion—capturing not just structural parts, but arbitrary characteristics—is that a part of a model is represented by an equivalence relation on the set of all possibilities from which that model is drawn. Then two models are equivalent, if and only if they are the same in the part that the

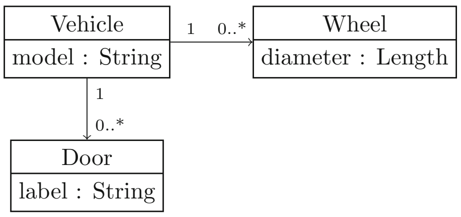


Fig. 8 A metamodel for vehicles

equivalence represents. It is convenient to use the quotient map rather than using the equivalence directly.

Definition 12 Let C be a set of models. A *part* relative to C is a set P together with a surjective function $p : C \to P$. The value of that part in a particular model $c \in C$ is $p(c)$.

Using this formulation helps us to be precise about our intentions – for example, when we think about the strings within a model as being a part, do we or do we not distinguish between multiple copies of the same string, and just one? We can make either choice, but they will yield different part maps p. Let us illustrate with a toy example.

Example 8 Let C be the set of all models that conform to the metamodel shown in Fig. 8. Such a model contains an arbitrary number of `Vehicle` elements, each of which may have some `Wheels` and some `Doors`, without restriction as to multiplicity. Each `Vehicle`, `Wheel` and `Door` carries with it one attribute, as shown, so that the Strings in a model are the strings which occur as values of the `model` attribute of a `Vehicle` or as the `label` attribute of some `Door`.

There are many different ways to define parts, relative to C—later, we shall discuss how to do this systematically based on the set of metaclasses in the metamodel, but we are not obliged to take that approach. Here are some simple examples:

$$p_1 : C \to \mathrm{Set}\langle\mathrm{String}\rangle$$

where p_1 applied to a particular model $c \in C$ returns the set of strings comprising the values of all the attributes of type `String`. Then to say that two models c_1 and c_2 are equivalent according to part p_1 is to say that they contain the same strings. Note that, according to this particular part definition, they might not contain these strings as values of the same attributes: swapping the values of attributes `model` and `label` does not modify the value of the part. This might be appropriate if, say, what we are concerned with is the cost of translating all the String values into a different language. We have huge flexibility to define parts that make the distinctions we wish to make. For example, we could define

$$p_2 : C \to \mathrm{Bag}\langle\mathrm{String}\rangle$$

if for some reason we care about the multiplicity with which strings occur in the model, but still not about exactly where. (Perhaps we plan to order stickers with the strings on them and want to know how many stickers must be printed with each string.)

Remark 3 While in a bx mindset, it is immediately tempting to see a part as a view and to ask whether the model and the part are related by an asymmetric lens. The answer is: sometimes. Considering Example 8 again, we might find it natural to define a part `wheels`, the value of this part, on a particular model, being the set of all wheels of vehicles in the model. This makes sense, whether the vehicles represented are cars, bicycles or something else. However, it does not seem natural to want to be able to make an arbitrary change to the set of wheels[3] and then update the full model including its `Vehicle` and `Door` elements to match. For example, it does not make sense to put bicycle wheels on a car! That is, we should not expect every part to give rise to an asymmetric lens, although some parts may do so.

Next we formalise the idea of bx R and S caring about different parts of their common target model C.

Definition 13 Given correct and hippocratic $R : A \leftrightarrows C$ and $S : B \leftrightarrows C$, an *A/B/rest* decomposition of C is a triple of parts:

- $f_A : C \to C_A$
- $f_B : C \to C_B$
- $f_{rest} : C \to C_{rest}$

such that the parts determine the whole, that is, if $f_A(c_1) = f_A(c_2)$ and $f_B(c_1) = f_B(c_2)$ and $f_{rest}(c_1) = f_{rest}(c_2)$, then $c_1 = c_2$.

Definition 14 Suppose we have correct and hippocratic $R : A \leftrightarrows C$ and $S : B \leftrightarrows C$. An A/B/rest decomposition (f_A, f_B, f_{rest}) of C is *non-interfering* if:

1. $\overrightarrow{R}$ only ever modifies the f_A part: that is, for all $a \in A$ and $c \in C$ we have
$$f_B(c) = f_B(\overrightarrow{R}(a, c))$$
$$f_{rest}(c) = f_{rest}(\overrightarrow{R}(a, c))$$
and dually, $\overrightarrow{S}$ only ever modifies the f_B part.
2. $\overrightarrow{R}$'s behaviour does not depend on anything $\overrightarrow{S}$ might modify: that is, for all $a \in A$ and for all $c_1, c_2 \in C$, if

[3] Even if we have carefully specified the codomain to include, not all sets of wheels, but only sensible sets, e.g. ruling out sets that contain exactly one bicycle wheel.

both $f_A(c_1) = f_A(c_2)$ and $f_{rest}(c_1) = f_{rest}(c_2)$ then

$$f_A(\overrightarrow{R}(a, c_1)) = f_A(\overrightarrow{R}(a, c_2))$$

Remark 4 Condition 1 amounts to saying that, for any $c \in C$, the subset $C_{R,c} = \{c' \in C : f_B(c') = f_B(c) \wedge f_{rest}(c') = f_{rest}(c)\}$ is an (R-)subspace in C, in the sense of Definition 10 of [31]: if the users of model $c \in C$ decide to stay within the subspace, bx R will never move them outside it, regardless of what the users of model $a \in A$ decide to do.

As suggested by the choice of terminology, we get

Theorem 3 *Let $R : A \nleftrightarrow C$ and $S : B \nleftrightarrow C$ be any correct and hippocratic bx sharing a common target C, as usual. If there is a non-interfering A/B/rest decomposition of C, then $\overrightarrow{R}$ and $\overrightarrow{S}$ are non-interfering.*

Proof Suppose there is a non-interfering A/B/rest decomposition given by f_A, f_B, f_{rest}. We have to show that for any $a \in A, b \in B, c \in C$,

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

which we will do by showing that f_A, f_B, f_{rest} all equate the two sides, then using the stipulation that the parts determine the whole. By condition 1 of Definition 14 we have

$$\begin{aligned} f_{rest}(\overrightarrow{R}(a, \overrightarrow{S}(b, c))) &= f_{rest}(\overrightarrow{S}(b, c)) \\ &= f_{rest}(c) \\ &= f_{rest}(\overrightarrow{R}(a, c)) \\ &= f_{rest}(\overrightarrow{S}(b, \overrightarrow{R}(a, c))) \end{aligned}$$

Also by condition 1, we have $f_A(\overrightarrow{S}(b, c)) = f_A(c)$, so we may take $c_1 = c$ and $c_2 = \overrightarrow{S}(b, c)$ in condition 2, giving

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

But condition 1 also gives $f_A(\overrightarrow{S}(b, \overrightarrow{R}(a, c))) = f_A(\overrightarrow{R}(a, c))$ so we have

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

Dually,

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

Then since the parts determine the whole, we are done. □

The A/B/rest decomposition seems to capture the essence of our intuitive understanding that the two transformations may be responsible for modifying different parts of their shared model, while there may be other parts that neither of the transformations modify. However, it is not obvious how it could be made use of in a practical setting: how are the parts of that decomposition to be determined?

In an MDD setting, it is natural to think about transformations written in terms of metamodels, such that it is straightforward to determine statically (at least in an over-approximation) when a given transformation could ever depend on, or modify, some instance of a particular metaclass within a model. Our next attempt gives a characterisation that is better adapted to that setting: it uses a potentially larger set of parts, which may, for example, correspond to the metaclasses in a metamodel, as we shall shortly illustrate. To allow us to capture the information about which parts are modified by a bx, and which play a role in determining the bx's behaviour, we introduce the idea of tagging parts. It turns out that our new characterisation is equivalent to the existence of an A/B/rest decomposition.

Definition 15 Let $R : A \nleftrightarrow C$ be a bx and let

$$\mathcal{D} = \{p_i : C \to P_i : i \in I\}$$

be a collection of parts with respect to C, for some index set I.

An A-*tagging* of $\mathcal{D}$ is a function

$$\text{tags}_A : I \to \mathcal{P}(\{A\text{-read}, A\text{-modify}\})$$

To interpret this tagging intuitively, one must imagine that $\overrightarrow{R}$ is modifying the element of C (in-place). Then the intention is that tagging a part A-read indicates that the behaviour of $\overrightarrow{R}$—the modifications it makes, to any part—may vary depending on the value of this part, while tagging it A-modify captures that $\overrightarrow{R}$ may modify this part. All four combinations of the two tags are permitted; for example, a part which is given neither tag should stay the same on application of any $\overrightarrow{R}(a, \_)$, and moreover, its value must not influence the effect of any application of $\overrightarrow{R}(a, \_)$ on any *other* part. Next, we formalise this.

Definition 16 Let $R : A \nleftrightarrow C$ be a bx, $\mathcal{D} = \{p_i : C \to P_i : i \in I\}$ a collection of parts, and tags_A an A-tagging, as in Definition 15. We say tags_A is a *well-tagging* for R if for any $a \in A$, $c, c_1, c_2 \in C$ and part $p_i \in \mathcal{D}$:

1. if $A\text{-modify} \notin \text{tags}_A(i)$ then $p_i(c) = p_i(\overrightarrow{R}(a, c))$;
2. if $A\text{-modify} \in \text{tags}_A(i)$ and $p_i(\overrightarrow{R}(a, c_1)) \neq p_i(\overrightarrow{R}(a, c_2))$, then there must be some part $p_j \in \mathcal{D}$ such that
   (a) p_j is tagged A-read, i.e. $A\text{-read} \in \text{tags}_A(j)$, and
   (b) $p_j(c_1) \neq p_j(c_2)$.

Here, Condition 1 stipulates that $\overrightarrow{R}$ only ever modifies parts that are tagged to show that it might do so. Condition 2 says, informally, that any change $\overrightarrow{R}$ makes must be justified by something it reads. It is perhaps easier to apprehend in the contrapositive form: if two models c_1 and c_2 are identical in all the parts that $\overrightarrow{R}$ is claimed to read, then any differences that do exist between them must be such as to be obliterated by consistency restoration with *any* element of A. Strictly speaking, since we have not yet imposed the condition that an element be determined by its parts, we should rather say that any differences *that are visible using this collection of parts* must be such as to be obliterated.

Definition 17 Let $R : A \nLeftrightarrow C$ and $S : B \nLeftrightarrow C$ be bx and let

$$\mathcal{D} = \{p_i : C \to P_i : i \in I\}$$

be a collection of parts with respect to C, for some index set I as before. Let tags_A, tags_B be well-taggings of $\mathcal{D}$ for R, S, respectively. We write tags for the obvious joint tagging, given by

$$\text{tags}(i) = \text{tags}_A(i) \cup \text{tags}_B(i).$$

Then, $\mathcal{D}$ is a *tagged decomposition* if the parts determine the whole: that is, if, for some c_1 and c_2, we have $p_i(c_1) = p_i(c_2)$ for all $i \in I$, then $c_1 = c_2$.

A tagged decomposition is *non-interfering* if:

1. no part that is tagged A-modify is also tagged B-modify or B-read;
2. dually, no part that is tagged B-modify is also tagged A-modify or A-read.

As begun at the end of this definition, from here on we will usually elide the index set and talk about the parts themselves being tagged; no confusion should arise.

It is straightforward to show:

Theorem 4 *There is a non-interfering tagged decomposition iff there is a non-interfering A/B/rest decomposition.*

Proof Suppose there is a non-interfering A/B/rest decomposition given by $f_A : C \to C_A$, $f_B : C \to C_B$, $f_{rest} : C \to C_{rest}$. To get our tagged decomposition we simply take these as the three parts and tag f_A with {A-modify, A-read}, f_B with {B-modify, B-read}, and f_{rest} with {A-read, B-read}. This is a well-tagging for R: Conditions 1 and 2 of Definition 16 follow from those of Definition 14 (making use of the contrapositive form of Condition 2). Dually, it is a well-tagging for S. The parts determine the whole because they did so in the A/B/rest decomposition. That the tagged decomposition is non-interfering as required by Definition 17 is clear from its definition.

Conversely, suppose that there is a non-interfering tagged decomposition given by $\mathcal{D} = \{p_i : C \to P_i\}_{i \in I}$ and tags. Define

$$C_A = \prod_{i:A\text{-modify}\in\text{tags}(i)} P_i$$

$$C_B = \prod_{i:B\text{-modify}\in\text{tags}(i)} P_i$$

$$C_{rest} = \prod_{i:\{A\text{-modify},B\text{-modify}\}\cap\text{tags}(i)=\emptyset} P_i$$

with the corresponding projection maps. (The non-interference condition in Definition 17 tells us that each P_i occurs in just one of the three products, so we are simply taking a possibly coarser abstraction.) The parts determine the whole in the A/B/rest decomposition because they did in the tagged decomposition.

Condition 1 of Definition 14 follows from the fact that the non-interference condition of Definition 17 says that none of the constituent parts of C_B or C_{rest} were tagged A-modify (and dually).

For Condition 2 of Definition 14, take c_1 and c_2 satisfying $f_A(c_1) = f_A(c_2)$ and $f_{rest}(c_1) = f_{rest}(c_2)$. From the definition above, if $p_i(c_1) \neq p_i(c_2)$, it follows that p_i must be tagged B-modify. But then by Definition 17, p_i is not tagged A-read. That is, c_1 and c_2 cannot differ in any part tagged A-read. The contrapositive of Condition 2 of Definition 16 then tells us that $\overrightarrow{R}(a, c_1)$ and $\overrightarrow{R}(a, c_2)$ cannot differ in any part tagged A-modify; hence by the definition above $f_A(\overrightarrow{R}(a, c_1)) = f_A(\overrightarrow{R}(a, c_2))$ as required. □

Corollary 1 *If there is a non-interfering tagged decomposition, then $\overrightarrow{R}$ and $\overrightarrow{S}$ are non-interfering.*

The trivial cases with which we began can easily be given such decompositions. If $C = C_A \times C_B$, with $\overrightarrow{R}$ concerned only with C_A and dually, then we will take projection onto C_A (rsp. C_B) as p_A (rsp p_B); there is no remainder, so C_{rest} can be a singleton set. If C is a single point, then all the parts can be maps to singletons.

7.1 Non-interference and metamodel-based definitions

We motivated the definition of a tagged decomposition as being more convenient than the A/B/rest decomposition for working with models and bidirectional transformations defined in terms of metamodels. Here, we sketch what we had in mind. For clarity, we will illustrate using the toy metamodel introduced in Example 8. For the sake of concreteness, we may take this to be an EMOF metamodel [25], though in fact, any reasonable metamodelling formalism will do.

Suppose the set C of models is defined to be the models defined by some metamodel MM, with a set of metaclasses $\mathcal{M}$. In the example, $\mathcal{M} = \{\text{Vehicle, Door, Wheel}\}$. Then, for each metaclass $M \in \mathcal{M}$, we could define a part p_M that, given a model m, returns the collection of model elements in m that are instances of metaclass M. In the example, these parts are:

$$p_{\text{Vehicle}} : C \rightarrow \text{Set}\langle\text{Vehicle}\rangle$$
$$p_{\text{Door}} : C \rightarrow \text{Set}\langle\text{Door}\rangle$$
$$p_{\text{Wheel}} : C \rightarrow \text{Set}\langle\text{Wheel}\rangle$$

Thus, any model (conforming to the metamodel) can be given as argument to each of these three parts. In effect, this splits the model up into the sets of its elements of different metaclasses. It is important to understand that an element of a metaclass will be given with all the data it owns, according to the metamodelling formalism, so that, despite the splitting, no information is lost: this is what makes it possible for such a set of parts to determine the whole, as required. For example, suppose our model c includes an element (`myBicycle` say) of metaclass `Vehicle`. As we see from the metamodel, metaclass `Vehicle` owns several properties, including the attribute `model` and the association ends that link a `Vehicle` to the collections of its `Wheels` and `Doors`. The values of these properties, in the element `myBicycle`, are part of the data of `myBicycle` considered as `Vehicle`. In an implementation, this means the result of applying p_{Vehicle} to a model is a set of `Vehicles` each of which is represented as a data structure including its properties, e.g. its own `xmi:id`, the string value of its `model` attribute (say "Cannondale XYZ"), and the sets of `xmi:ids` that record the owned association end properties. The data structure will not, however, include any data that is not owned by this element, e.g. it will not include the value of the `diameter` attribute of any `Wheel` object.

In equivalence relation terms, two models are the same according to the part p_M if an observer who can only examine model elements that are instances of M could not tell the difference between them.

Remark 5 The attentive reader may have noticed that we have brushed under the carpet the fact that, if we have a fully specified metamodel, not all sets of elements of a given metaclass may actually occur as the images of legal models; we mentioned the problem of having a set of `Wheels` that included exactly one bicycle wheel, in Remark 3. This apparent problem is trivial, however; the wish to correspond to the equivalence relation notion was the only reason for including "surjective" in Definition 12, so the fact that we may need to define our set of collections rather artificially in order to ensure surjectivity should not worry us.

As we have seen in the example above, some care is needed over properties, in order that this collection of parts will determine the whole. We may suppose, as we did in the example and as is normal in MOF-based languages, that each model element comes together with an identifier that is unique inside the model, and with its properties, including the identifiers of other model elements that are directly navigable from it.

That sorted, we may define, generically, a tagged decomposition of C, provided that we can give well-taggings for the incident bx.

This tagging should, in a reasonable bx language, be doable by static analysis of the bx transformation along with the metamodels on which it operates. Consider, as usual, a bx $R : A \nsim C$. A metaclass—strictly, its corresponding part—will need to be tagged A-write whenever $\overrightarrow{R}$ is capable of modifying some instance of the metaclass (including any of its properties: for example, deleting a model element to which this element can directly navigate, and tidying up by deleting its identifier from the properties of this element, will count as a modification). For example, the part p_{Vehicle} would need to be tagged A-write if $\overrightarrow{R}$ might modify the set of `Vehicles` in a model, including by changing any property of any of them, e.g. by deleting a `Wheel` object so that the corresponding association end property of some `Vehicle` would have to change. However, it would not need to be tagged A-write if the bx were only capable of changing the `diameter` of existing `Wheel` objects, because such changes would be invisible to a viewer who can only inspect `Vehicle` elements; they do not alter the value of the part p_{Vehicle} applied to any model. A metaclass will need to be tagged A-read if some instance of it may influence the behaviour of the bx, e.g. if it is mentioned in a pattern that must be matched.

As usual, tagging that is done as a result of static analysis would probably have to over-approximate what tags are required; depending on the expressiveness of the bx formalism, it would very likely be necessary (for decidability) to allow it to tag a metaclass A-write even if, in operation, no instance of that metaclass would ever be modified. Furthermore, matters that would need to be considered carefully, in a full description of this technique in a specific formal setting, include specialisation of metaclasses (we would normally expect that every submetaclass would have to inherit the tags of its parents) and the effects of any global constraint checking.

Nevertheless, it seems reasonable to be optimistic that this approach might permit the formalisation and automation of the kind of common-sense reasoning that people who operate with several tools on the same model naturally do—"it doesn't matter what order I use these tools in, because they don't care about the same things"—permitting some cases of non-interference to be detected automatically and easing the task of maintaining consistency in a network of models.

Fig. 9 Bx that are non-interfering, but have no non-interfering A/B/rest decomposition

$\overrightarrow{R}$	c_A	c_1	c_2	c_B
a_0	c_A	c_1	c_2	c_B
a_1	c_1	c_1	c_2	c_B
a_2	c_2	c_1	c_2	c_B

$\overleftarrow{R}$	c_A	c_1	c_2	c_B
a_0	a_0	a_0	a_0	a_0
a_1	a_0	a_1	a_1	a_1
a_2	a_0	a_2	a_2	a_2

$\overrightarrow{S}$	c_A	c_1	c_2	c_B
b_0	c_A	c_1	c_2	c_B
b_1	c_A	c_1	c_2	c_1
b_2	c_A	c_1	c_2	c_2

$\overleftarrow{S}$	c_A	c_1	c_2	c_B
b_0	b_0	b_0	b_0	b_0
b_1	b_1	b_1	b_1	b_0
b_2	b_2	b_2	b_2	b_0

$$\begin{tikzcd}
& c_1 & \\
c_A \arrow[ur, "a_1"] \arrow[dr, "a_2"'] & & c_B \arrow[ul, "b_1"'] \arrow[dl, "b_2"] \\
& c_2 &
\end{tikzcd}$$

Fig. 10 Actions of A and B on C

7.2 Necessary conditions for non-interference?

We have now discussed two equivalent kinds of "non-interfering decomposition" and shown that the existence of such a decomposition is sufficient to ensure non-interference. Note, however, that we have not shown that *every* pair of non-interfering bx can be demonstrated to be so by exhibiting a non-interfering decomposition. There is a good reason for this: despite our initial intuition, it is not true. Here is a small example (not quite the smallest that exists, but this one has an easy-to-explain intuition).

Example 9 Let $A = \{a_0, a_1, a_2\}$, $B = \{b_0, b_1, b_2\}$, and $C = \{c_A, c_1, c_2, c_B\}$. Let $R : A \not\asymp C$ and $S : B \not\asymp C$ be defined by their forward and backward transformations given in Fig. 9.

Here, $\overleftarrow{R}$ and $\overleftarrow{S}$ are included only to show that it is indeed possible to complete $\overrightarrow{R}$ and $\overrightarrow{S}$ to correct and hippocratic bidirectional transformations. (The role of a_0, b_0, which are consistent with all elements of C, is just to ensure this.)

Figure 10 illustrates. Here, we elide explicit mention of R and S in favour of showing A and B acting on C: we show an arrow from c to c' labelled with an a to indicate that $\overrightarrow{R}(a, c) = c'$, and similarly an arrow from c to c' labelled with a b indicates that $\overrightarrow{S}(b, c) = c'$. Missing arrows indicate that consistency already holds: for example, there is no arrow labelled b out of c_A, because, as can be seen from the tables above, $\overrightarrow{S}(b, c_A) = c_A$ (that is, $S(b, c_A)$ holds).

A reasonable intuition for this example is players A and B wish to jointly make a choice between two options represented by c_1 and c_2. The models in sets A, B represent the players' preferences, if any. a_0 means that A has no preference, and cannot choose; a_1 means she prefers option 1, a_2 that she prefers option 2. Similarly for B. In model C, c_1 and c_2 represent the choice having been made; c_A means that it is A's job to make the choice, while c_B means that it is B's job. The players are very polite and protocol-respecting: they never make a choice unless it is their job to do so, and if a player has the job of choosing, but cannot do so, no choice is ever made. Thus, $\overrightarrow{R}$ and $\overrightarrow{S}$ are non-interfering: regardless of what the models are originally, at most one of the transformations will change it, so there is no possibility that two consecutive changes fail to commute; or to put it another way, the players never compete to impose their choice!

As this intuition suggests, we cannot factor C into parts that are the separate concerns of $\overrightarrow{R}$ and $\overrightarrow{S}$. Their non-interference arises, rather, from the transformations' respect for a protocol about the circumstances in which they will change a part of the model, than from a separation of the parts they will change. Summarising formally, we get

Theorem 5 *The existence of a non-interfering A/B/rest decomposition of C (or, equivalently by Theorem 4, of a non-interfering tagged decomposition of C) is a sufficient, but not a necessary, condition for $R : A \not\asymp C$ and $S : B \not\asymp C$ to be non-interfering.*

Proof Sufficiency is Theorem 3. To show that necessity does not hold, let $A = \{a_0, a_1, a_2\}$, $B = \{b_0, b_1, b_2\}$, and $C = \{c_A, c_1, c_2, c_B\}$. Let $R : A \not\asymp C$ and $S : B \not\asymp C$ be defined by their forward and backward transformations given in Fig. 9. Then, R and S are non-interfering, yet there is no non-interfering A/B/rest decomposition of C.

We have already observed that they are non-interfering: formally, the range of each of $\overrightarrow{R}$ and $\overrightarrow{S}$ is the set $\{c_1, c_2\}$, while both c_1 and c_2 are fixed by every $\overrightarrow{R}(a, \_)$ and $\overrightarrow{S}(b, \_)$. Therefore, none of the equations in Definition 11 can fail; that is, $\overrightarrow{R}$ and $\overrightarrow{S}$ are non-interfering.

To see that no non-interfering A/B/rest decomposition exists, observe that, if (f_A, f_B, f_{rest}) were such a decomposition, then condition 1 of Definition 14 would force $\{c_A, c_1, c_2\}$ to be mapped to the same point by f_A and similarly, would force $\{c_B, c_1, c_2\}$ to be mapped to the same point by f_B. Condition 1 would also force the whole of C to be mapped to a single point by f_{rest}. But then none of f_A, f_B, f_{rest} could distinguish c_1 from c_2; that is, the parts could not determine the whole. Therefore, no non-interfering A/B/rest decomposition of C can exist. □

Before exploring the implications of these situations in Sect. 9, we briefly consider using a more refined notion of consistency.

8 More refined notions of consistency

In [31], we considered notions of consistency beyond a simple true/false distinction, and these ideas are particularly useful in the context of networks of bx, in two main ways. First, they contribute to modelling how consistency can be restored in phases: for example, if one phase of a bx does not guarantee to restore consistency perfectly, but does guarantee to eliminate certain kinds of inconsistency (missing elements, say), then it may be useful to have a way to specify precisely what it does promise. In our setting, as we shall see, this gives us the means to talk about situations in which two restoration functions must be applied to the same model, in order to make it consistent with both of two related models. Second, they provide a foundation for handling situations in which consistency cannot be fully restored, giving guarantees of some good behaviour even then and allowing helpful feedback of what was, and was not, achieved. For example, we might be happy to allow a bx to make some changes automatically, but we might want other changes to be made only with human oversight; in such a case, the bx might not be able to restore consistency fully, yet it might be helpful to have terminology for precisely what it has done. Or, in a network setting, we might need to talk about what has been achieved, even when perfect consistency in the network has not been restored.

Let $(\Lambda, \leq_\Lambda)$ be a partial order, called the consistency structure, having a top element $\top$ ("perfectly consistent"; all $\lambda \leq_\Lambda \top$). A pair of models, rather than simply being considered consistent or not, may be assigned a consistency level $\lambda \in \Lambda$.

Formally, let a partial bidirectional transformation $R : M \leftrightarrow N$ over Λ be defined by specifying

– a consistency indicator $R : M \times N \to \Lambda$ that says how consistent a given pair of models is
– consistency restoration functions $\overrightarrow{R} : M \times N \to N$ and $\overleftarrow{R} : M \times N \to M$.

The usual setting is of course subsumed: it has consistency structure containing just $\bot < \top$. The consistency restoration functions are still total functions, but they may not fully restore consistency; for example, $\overrightarrow{R}(m, n)$ might return n' such that $R(m, n') = \lambda < \top$. (Of course, the consistency restoration function can always return its argument of the appropriate type, so it is no restriction to require it to be a total function.)

Definitions of properties of bx that rely solely on equations between the results of consistency restoration functions, such as Definition 11, apply without modification to partial bx. Other property definitions need slight notational adjustment:

Definition 18 A partial bx R is *correct* if for all $m \in M$ and $n \in N$ we have $R(m, \overrightarrow{R}(m, n)) = \top$, and dually.

Definition 19 A partial bx R is *hippocratic* if for all $m \in M$ and $n \in N$ we have $R(m, n) = \top \Rightarrow \overrightarrow{R}(m, n) = n$, and dually.

That is, as usual, a bx is *correct* if it always restores perfect consistency, and *hippocratic* if it never makes any change to a perfectly consistent model.

The paper [31] considered more refined properties of bx, useful when fully correct and hippocratic behaviour is not achievable. It considered several reasons why this might be so, each of which might apply in a network setting. A perfectly consistent model might not exist; or it might not be possible to find one (say, within feasible computation constraints); or we might not want to authorise an automated bx to make certain changes to a model that would be required to restore perfect consistency, even though we find it useful to allow it to make some (perhaps "less dangerous" or "less controversial") changes that improve the consistency situation somewhat.

First, a bx is called *as correct as possible* (ACAP) if the model it produces is always a *candidate* with respect to its arguments. That is, while $\overrightarrow{R}(a, b)$ might give b' such that $R(a, b') <_\Lambda \top$—it fails to restore consistency perfectly—if R is ACAP, it guarantees that no b'' exists with $R(a, b'') >_\Lambda R(a, b')$, i.e. R does the best that can be done under the circumstances. Formally:

Definition 20 Given $m \in M$, the set of $\overrightarrow{R}$ candidates with respect to m is $\{n' \in N : R(m, n') \text{ is maximal}\}$. That is, n' is a candidate iff $R(m, n') = \lambda \in \Lambda$ such that there does not exist any $n'' \in N$ with $R(m, n'') >_\Lambda \lambda$. Dually for $\overleftarrow{R}$.

Definition 21 A bx R is *as correct as possible* (ACAP) if it always returns a candidate.

Often, this property cannot be achieved. However, it is a useful property in the case where consistency imposes a constraint on the argument that cannot be modified (a in this case). In some such cases, we may be able to model the restoration of consistency between a and b that requires modifying both a and b as a two-stage process, in which first an ACAP $\overrightarrow{R}$ is used to update b with respect to a, giving b', and then an ACAP $\overleftarrow{R}$ is used to update a with respect to b', giving a' satisfying $R(a', b') = \top$.

Second, we call a bx *as hippocratic as possible* (AHAP) if it has the property that it makes no change to a model unless it strictly increases the consistency level:

Definition 22 A bx R is *as hippocratic as possible* (AHAP) if its consistency restoration functions return exactly their argument of appropriate type, unless returning something strictly more consistent. That is,

$$\overrightarrow{R}(m,n) = n \ \vee\ R(m, \overrightarrow{R}(m,n)) > R(m,n)$$

and dually.

By contrast with ACAP, the AHAP property can reasonably be imposed. Thinking about practical applications: provided that we have access to an implementation of the consistency function, any bx can be modified so as to be AHAP, by wrapping it, as follows. Given a pair (a, b) of arguments to a consistency restoration function $\overrightarrow{R}$, first calculate the consistency level $\lambda = R(a, b)$; then calculate $b' = \overrightarrow{R}(a, b)$ and find $\lambda' = R(a, b')$; finally, return b' if $\lambda' > \lambda$, otherwise, return b.

In our network setting, this seems a reasonable response to the problem of having multiple bx targetting a single model, where the bx are not necessarily non-interfering. Consider again the situation in Fig. 6b, where each of $\overrightarrow{R}$ and $\overrightarrow{S}$ is correct and hippocratic, but this time they are not necessarily non-interfering. We may define $RS : A \times B \not\rightleftharpoons C$ over $\Lambda = \{\bot, \lambda_R, \lambda_S, \top\}$ (where λ_R and λ_S are incomparable, both greater than $\bot$ and less than $\top$) by

–

$$\begin{aligned} RS((a,b),c) = &\top \text{ iff } R(a,c) \wedge S(b,c)\\ &\lambda_R \text{ iff } R(a,c) \wedge \neg S(b,c)\\ &\lambda_S \text{ iff } \neg R(a,c) \wedge S(b,c)\\ &\bot \text{ iff } \neg R(a,c) \wedge \neg S(b,c)\end{aligned}$$

–

$$\begin{aligned}&\overrightarrow{wR}((a,b),c)\\ &\quad = \overrightarrow{R}(a,c) \text{ if } RS((a,b), \overrightarrow{R}(a,c)) > RS((a,b),c)\\ &\qquad c \text{ otherwise.}\end{aligned}$$

Similarly

$$\begin{aligned}&\overrightarrow{wS}((a,b),c)\\ &\quad = \overrightarrow{S}(b,c) \text{ if } RS((a,b), \overrightarrow{S}(b,c)) > RS((a,b),c)\\ &\qquad c \text{ otherwise.}\end{aligned}$$

Then

$$\overrightarrow{RS}((a,b),c) = \overrightarrow{wS}((a,b), \overrightarrow{wR}((a,b),c)$$

(note that in this definition $\overrightarrow{R}$ is applied first; there is an obvious variant, which we might call SR, where $\overrightarrow{S}$ is applied first).

– $\overleftarrow{RS}((a,b),c) = (\overleftarrow{R}(a,c), \overleftarrow{S}(b,c))$

It is immediate that

Lemma 3 *$\overrightarrow{RS}$ is AHAP. If in addition $\overrightarrow{R}$ and $\overrightarrow{S}$ are non-interfering, then this definition of RS is equivalent to the one given in Lemma 2, and both are equivalent to SR.*

Illustrating the difficulty in ensuring that a bx is ACAP, note that $\overrightarrow{RS}$ does not necessarily have that property, if $\overrightarrow{R}$ and $\overrightarrow{S}$ are not non-interfering. By correctness of R, S, it does ensure that the final consistency level is not $\bot$. However, it is possible that a c' exists that satisfies both $R(a, c')$ and $S(b, c')$, and yet which is not reachable using the consistency restoration functions provided.

Example 10 Returning once again to a variant of the example from Sect. 2, let us take $\overrightarrow{R}$ to be the restoration of consistency from Metamodel towards Model, and $\overrightarrow{S}$ to be the restoration of consistency from Code towards Model. We earlier pointed out that these might or might not be non-interfering, depending on the range of possibilities for Metamodel. Then, $\overrightarrow{wR}$ changes the Model to be consistent with the Metamodel, *but only if doing so does not break previously existing consistency between Model and Code; if it does, it leaves the Model alone*. Similarly, $\overrightarrow{wS}$ changes a Model to be consistent with the Code, unless doing so would break existing consistency with the Metamodel, in which case it does nothing. When these are combined into $\overrightarrow{RS}$, we get a consistency restorer which first tries to update the Model to conform to the Metamodel and then tries to update it with respect to the Code; but in neither step does it actually do anything if restoration of the kind of consistency it aims at would break the other kind of consistency. It is AHAP, in that it will make no change unless it gives an overall improvement in the consistency lattice, i.e. restores a previously lacking consistency between Model and one of Metamodel and Code, without breaking either previous consistency. It is not ACAP, as it might very well be that there is a Model consistent with both the Metamodel and the Code which is not reached by this simple-minded procedure. Generally speaking, some “intelligence” (scare quotes) is required to combine the effect of several bx, which the composition we described does not provide.

If we are in the situation where the consistency restorers $\overrightarrow{R}$ and $\overrightarrow{S}$ are non-interfering, the situation is greatly simplified: we then know that applying one of these consistency restorers will not break the other kind of consistency, so then we are just saying that we first restore conformance of the Model to the Metamodel and then restore consistency of the Model to the Code, ending up (by Lemmas 3 and 2) with a Model which is consistent with both Metamodel and Code.

In the next section, we will turn to some cases where networks can be resolved.

9 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, even 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 so need special treatment.

Step 0: Consider each non-authority node n. If there is no authority node from which n can be reached via a path in the network (respecting any oriented edges), then add n 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 4 *If Step 3 applied, its result is resolvable.*

Proof Since Step 3 applied, the network is a tree, with all edges oriented away from the root, towards the leaves. (Note that there cannot be any edges that were oriented in the opposite direction, for then their sources would have been unreachable in Step 1 and would have been added to the authority set.) Therefore, it suffices to start at the root, and apply the bx in turn, until all leaves are reached. Correctness of the individual bx tells us that this resolves the network. □

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 Sect. 2 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 towards 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.

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

Definition 23 $\overrightarrow{R}$ 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 5 *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 6 *If Step $3'$ applies, then the network is resolvable. Moreover, if all the bx are history ignorant, then the network is confluent.*

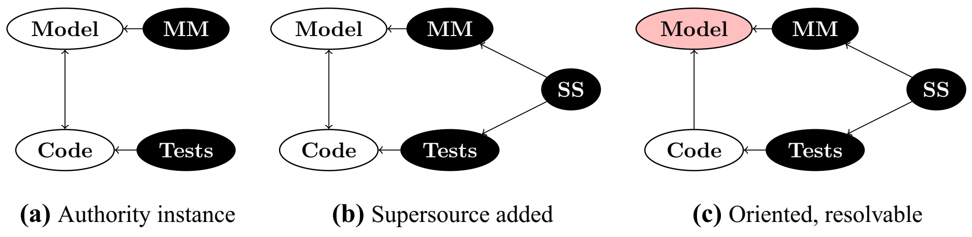


Fig. 11 Resolving a simple network

Proof Orient the network as in Step 3 and resolve the network as in Lemma 4, 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 11 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. We suspect that in practical applications, consistency in a network that deviates much from being a tree is better handled with a "pull" model than with the "push" model that is implicit in seeking confluent resolution of the whole network. This observation motivated ongoing work whose beginning is reported in [33].

10 Related work

10.1 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" [3]. Megamodelling is better seen as a mental discipline than as a technology. Megamodels are often, in current 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. For example, 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 multi-directional transformations and especially work on the resolution of networks of bx, as a contributing technology. For example, the megamodelling notation used in [14] 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 artefacts 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 [14]) includes nine models, all related to one another by one or more binary "overlaps" relationships.

The networks of models, connected by binary bx, discussed in this paper *are* megamodels, but by making explicit the binary bx between the models, we open up the possibility to treat the megamodel formally and uniformly. This is the sense in which this work is a contributing technology to the study of megamodels. For example, building explicitly on the conference version ([32]) of the present paper and using a delta-based formalism, Anjorin et al. in [1] described a collection of case studies from Siemens AG where this approach is useful.

One problem in megamodelling is knowing when to stop. A transformation may itself be regarded as a model that forms part of a megamodel; it may be related to the models it transforms. In fact, this can be useful, as we can then consider, explicitly, changes to the transformation itself; our binary consistency relations may then be between the transformation and its transformed models, and we may use mechanisms such as the Quick Fixes of Cuadrado et al. [10] to restore this consistency as necessary. The desired relationship between the transformed models, that it is the job of the transformation to maintain, needs to be *implemented by* our binary consistency relations. Both the application of Quick Fixes to the transformation and the application of the transformation to the models may be seen within our framework. We may write: $R(m_1, M_1, T, m_2, M_2)$ if and only if m_1 conforms to M_1 and is consistent, according to T which relates metamodels M_1 and M_2, with m_2 which conforms to M_2.

The English explanation of this multiary consistency relation shows how to implement it using binary consistency relations, but much of the practical difficulty of MDD is embodied in the choices that have to be made about how to restore this consistency when it is lost.

In this paper, we have demonstrated that the circumstances in which strong conditions such as confluence hold are very limited, so it makes sense to explore mechanisms for supporting humans in managing decisions about the choice of resolution path, and indeed of authority set. Early work in that direction is reported in [33].

10.2 Multiary variants of triple graph grammars

In [15], 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* [34, 35]. 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.

10.3 Further related work

Ideas similar to non-interference have been studied before, but we are not aware of any very closely related approach. Our use of tags is, at least superficially, reminiscent of that in [23]. The setting of that work differs so much from ours that it is hard to make precise comparisons, but roughly, in a functional programming context, they replace values (in their correlate of our models) by values that are tagged either O or U, and then use the tagging to enable certain compositions of asymmetric lenses. Their elements that are O-tagged are similar to our parts that are guaranteed *not* to be tagged Read, while their elements that are U-tagged may or may not be tagged Read. There does not seem to be any correlate of our Modify tags, however, perhaps because of the functional setting.

The concepts of sequential and parallel independence, extensively studied in the graph grammar community (see, for example, [5,27]) also seem to be somewhat related, at least in the aims of enabling certain changes to be made in either order. The technical results from that field are so deeply embedded in the graph grammar setting that at present I do not see how to make a fruitful connection, but perhaps there is the potential for future work here.

Turning to the problem of restoring consistency to a collection of many models, a different approach, related to the CSP work discussed in Sect. 3.2, is to take a program repair view. Here, constraints are expressed over all models, any of which may be modified; see for example [36].

Garcia [13] used a bidirectionalisation approach based on [22] to define what he called a Declarative Model-View-Controller architecture. Macedo et al. [21] considered some issues that arise in taking seriously QVT-R's claim [26] 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 suggest a modification giving 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 [6], 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 (for example, making use of a refined notion of consistency such as we discussed in Sect. 8) and fixing these first. Clearly, it will not always give the best result, by the same argument that shows a naive metric-based least-change property to be not always desirable [7]; 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. Probably, a mixture of theoretical and pragmatic approaches will be required.

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 [18] for example). From that field we picked up the useful basic tool of adding a supersource, in Sect. 9. 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 [19]: it would be interesting to try to adapt ideas from that field to this one.

11 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 multi-directional transformations, 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. Indeed, this finding was the starting point for a separate ongoing thread of work whose beginning is reported in [33]. Here, however, we provided an algorithm and positive results for special cases.

We also looked in detail at situations in which two bidirectional transformations may act on the same model without interfering with one another; we gave two sets of sufficient conditions for this to be so, and in doing so, gave an elementary notion of the parts of a model that a bidirectional transformation reads and modifies.

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.

As discussions at a recent Dagstuhl meeting [9] demonstrated, ideas and questions formulated here in terms of models may also have a wider applicability to other situations in which consistency has to be managed in the face of change, and many questions remain to be investigated.

Acknowledgements I thank Anthony Anjorin, Robin Bradfield, Jeremy Gibbons, James McKinna, Meng Wang, and the anonymous reviewers, both of this paper and of [32], for helpful comments and discussion. Discussions at Dagstuhl no. 18491 on *Multidirectional Transformations and Synchronisations* were also very helpful.

References

1. Anjorin, A., Yigitbas, E., Leblebici, E., Schürr, A., Lauder, M., Witte, M.: Description languages for consistency management scenarios based on examples from the industry automation domain. Program. J. **2**(3), 7 (2018)
2. Bacchus, F., Chen, X., van Beek, P., Walsh, T.: Binary vs. non-binary constraints. Artif. Intell. **140**(1/2), 1–37 (2002)
3. Bézivin, J., Jouault, F., Valduriez, P.: On the need for megamodels. In: Proceedings of the OOPSLA/GPCE: Best Practices for Model-Driven Software Development workshop, 19th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (2004)
4. Boiten, E.A., Derrick, J.: From ODP viewpoint consistency to integrated formal methods. Comput. Stand. Interfaces **35**(3), 269–276 (2013)
5. Bonchi, F., Gadducci, F., Heindel, T.: Parallel and sequential independence for borrowed contexts. In: International Conference on Graph Transformations (ICGT'08), Lecture Notes in Computer Science, vol. 5214, Springer, pp. 226–241 (2008)
6. Boubakir, M., Chaoui, A.: A pairwise approach for model merging. In: Modelling and Implementation of Complex Systems (MISC'16), Springer, pp. 327–340 (2016)
7. Cheney, J., Gibbons, J., McKinna, J., Stevens, P.: On principles of least change and least surprise for bidirectional transformations. J. Object Technol. **16**(1), 3:1–31 (2017)
8. Cicchetti, A., Ciccozzi, F., Pierantonio, A.: Multi-view approaches for software and system modelling. In: Software and System Modeling (2019)
9. Cleve, A., Kindler, E., Stevens, P., Zaytsev, V.: Report from Dagstuhl seminar 18491, multidirectional transformations and synchronisations. Dagstuhl Reports vol. 8, p. 12 (2019)

10. Cuadrado, J.S., Guerra, E., de Lara, J.: Quick fixing ATL model transformations. In: ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS'15), IEEE Computer Society, pp. 146–155 (2015)
11. Davey, B.A., Priestley, H.A.: Introduction to Lattices and Order, CUP (2002)
12. Diskin, Z., Gholizadeh, H., Wider, A., Czarnecki, K.: A three-dimensional taxonomy for bidirectional model synchronization. J. Syst. Softw. **111**, 298–322 (2016)
13. Garcia, M.: Bidirectional synchronization of multiple views of software models. In: Workshop on Domain-Specific Languages (DSML'08), CEUR Workshop Proceedings, vol. 324, pp. 7–19 (2008)
14. Hebig, R., Giese, H., Batoulis, K., Langer, P., Farahani, A.Z., Yao, G., Wolowyk, M.: Development of AUTOSAR Standard Documents at Carmeq GmbH: A Case Study, Universitätsverlag Potsdam (2015)
15. Königs, A., Schürr, A.: MDI: a rule-based multi-document and tool integration approach. Softw. Syst. Model. **5**(4), 349–368 (2006)
16. Lano, K.: Constraint-driven development. Inf. Softw. Technol. **50**(5), 406–423 (2008)
17. Lano, K., Tehrani, S.Y.: Verified bidirectional transformations by construction. In: Joint Proceedings of the Second International Workshop on Patterns in Model Engineering and the Fifth International Workshop on the Verification of Model Transformation (PAME/VOLT'16), CEUR Workshop Proceedings, vol. 1693, CEUR-WS.org, pp. 28–37 (2016)
18. Luenberger, D.C.: Linear and Nonlinear Programming. Addison Wesley, Boston (1984)
19. Lynch, N.: Distributed Algorithms. Elsevier, Amsterdam (1996)
20. Macedo, N., Cunha, A.: Least-change bidirectional model transformation with QVT-R and ATL. Softw. Syst. Model. **15**(3), 783–810 (2016)
21. Macedo, N., Cunha, A., Pacheco, H.: Towards a framework for multidirectional model transformations. In: 3rd International Worshop on Bidirectional Transformations (Bx'14), CEUR Workshop Proceedings, vol. 1133, CEUR-WS.org, pp. 71–74 (2014)
22. Matsuda, K., Hu, Z., Nakano, K., Hamana, M., Takeichi, M.: Bidirectionalization transformation based on automatic derivation of view complement functions. In: International Conference on Functional Programming (ICFP'07), ACM, pp. 47–58 (2007)
23. Matsuda, K., Wang, M.: Applicative bidirectional programming with lenses. In: International Conference on Functional Programming (ICFP'15), ACM, pp. 62–74 (2015)
24. Nuseibeh, B., Easterbrook, S.M., Russo, A.: Making inconsistency respectable in software development. J. Syst. Softw. **58**(2), 171–180 (2001)
25. OMG.: Meta object facility version 2.5.1. OMG document formal/16-11-01 (2016). http://www.omg.org
26. OMG.: MOF2.0 query/view/transformation (QVT) version 1.3. OMG document formal/2016-06-03 (2016). http://www.omg.org
27. Plump, D.: Modular termination of graph transformation. In: Graph Transformation, Specifications, and Nets, Lecture Notes in Computer Science, vol. 10800, Springer, pp. 231–244 (2018)
28. Rossi, F., Petrie, C.J., Dhar, V.: On the equivalence of constraint satisfaction problems. In: 9th European Conference on Artificial Intelligence (ECAI '90), pp. 550–556 (1990)
29. Schürr, A., Klar, F.: 15 years of triple graph grammars. In: International Conference on Graph Transformations (ICGT'08), LNCS, vol. 5214, Springer, pp. 411–425 (2008)
30. Shah, A.A., Kerzhner, A.A., Schaefer, D., Paredis, C.J.J.: Multiview modeling to support embedded systems engineering in SysML. In: Graph Transformations and Model-Driven Engineering, LNCS, vol. 5765, Springer, pp. 580–601 (2010)
31. Stevens, P.: Bidirectionally tolerating inconsistency: Partial transformations. In: Gnesi, S., Rensink, A. (eds.) Fundamental Aspects of Software Engineering (FASE'14), LNCS, vol. 8411, Springer, pp. 32–46 (2014)
32. Stevens, P.: Bidirectional transformations in the large. In: ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS'17), IEEE, pp. 1–11 (2017)
33. Stevens, P.: Towards sound, optimal, and flexible building from megamodels. In: ACM/IEEE 21th International Conference on Model Driven Engineering Languages and Systems (MODELS'18), ACM (2018)
34. Trollmann, F., Albayrak, S.: Extending model to model transformation results from triple graph grammars to multiple models. In: International Conference on Model Transformations (ICMT'15), LNCS, vol. 9152, Springer, pp. 214–229 (2015)
35. Trollmann, F., Albayrak, S.: Extending model synchronization results from triple graph grammars to multiple models. In: International Conference on Model Transformations (ICMT'16), LNCS, vol. 9765, Springer, pp. 91–106 (2016)
36. Xiong, Y., Hu, Z., Zhao, H., Song, H., Takeichi, M., Mei, H.: Supporting automatic model inconsistency fixing. In: 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE'09), ACM, pp. 315–324 (2009)
37. Xiong, Y., Song, H., Zhenjiang, H., Takeichi, M.: Synchronizing concurrent model updates based on bidirectional transformation. Softw. Syst. Model. **12**(1), 89–104 (2013)

Perdita Stevens is Professor of Mathematics of Software Engineering in the Laboratory for Foundations of Computer Science at the University of Edinburgh, in Scotland.