Theory and Practice of Networks of Models

Theory and Practice of Networks of Models

Perdita Stevens[1][0000−0002−3975−7612]

Laboratory for Foundations of Computer Science
School of Informatics
University of Edinburgh, UK
Perdita.Stevens@ed.ac.uk
http://homepages.inf.ed.ac.uk/perdita

Abstract. Separating concerns into multiple data sources, such as multiple models of a software system under development, enables people to work in parallel. However, concerns must also be re-integrated, and this gives rise to many interesting problems, both theoretical and practical. In my keynote talk I will discuss some of them; in this accompanying paper I give some background to my work in recent years and summarise some key points and definitions.

Keywords: bidirectional transformation · consistency maintenance · megamodel

1 Introduction

The power of information technology to benefit our lives comes from bringing together vast amounts of data, and complex behaviour based on that data, which unaided human brains could not handle. This, however, makes the development and maintenance of the information technology difficult for us: humans have to be able to direct it, without any one of us fully understanding it.

Separation of concerns, in Dijkstra's famous terminology [2], is fundamentally the only hammer we have to tackle this problem. Because no individual human can hold in mind *all* the information that is relevant to even a moderately-sized information system, it is essential for us to be able to parcel out that information so that the part of it that is necessary to each decision can, indeed, be held in mind and manipulated as necessary.

A related, but distinct, problem is the need to hold the data we care about in different places – on different computers, sometimes in different geographical locations, sometimes in different legal jurisdictions or on opposite sides of "Chinese walls". Sometimes it is impractical to colocate the data; sometimes it is positively undesirable, illegal or unethical [4].

We would not be considering, together, information that is held separately, if it there were no circumstance under which it is somehow related. Perhaps some computation is to be done that draws on data from two separate sources, thereby producing a third set of data. Whenever such a situation pertains, there is the possibility of getting a wrong, or nonsensical, answer – that is, it is possible for there to be inconsistencies between the data sources, and/or errors in

the relationships that are assumed or imposed on them. The imposition and maintenance of such relationships is called the *consistency maintenance problem*. It may arise in very simple guises: perhaps a datum is intended to have duplicate copies in two locations, so that if the copies differ at all, the sources are inconsistent. Or it may involve arbitrarily complex relationships.

In the simplest of cases, the dependencies between data sources flow in one direction only, producing an acyclic graph. For example, if one or more independent data sources are regarded as authoritative, and another is to be computed from them, then the only kind of inconsistency that can arise is if the computed data becomes out-of-date with respect to the sources. In such a case consistency can be restored by recomputing the outdated data. There may still be challenging problems of how to do so efficiently, but we do not encounter *bidirectionality*.

The essence of bidirectionality is given [6] by the following three features:

1. There is separation of concerns into explicit parts such that
2. more than one part is "live", that is, liable to have decisions deliberately encoded in it in the future; and
3. these[1] parts are not orthogonal. That is, a change in one part may necessitate a change in another.

The first point gives us data in separate places; the third tells us that inconsistency is a possibility; the second rules out mere recomputation as a solution.

Bidirectionality can be, and often is, attacked as a purely theoretical topic in which the first thing we do is to assume that there is a single consistent metadata framework into which all the data of interest fits; then the problem of maintaining consistency can be attacked within that framework. Unfortunately the real world is not so neat. We need to contend with

- data governance issues such as the need to control which data sources may be modified when – because our consistency maintenance framework may not always have authority to modify data in arbitrary ways
- legacy data sources – because we may not be allowed to reengineer a data source to conform to our chosen framework
- legacy programmed transformations or other consistency maintenance mechanisms between data sources – because it may not be desirable or practicable to recreate them
- the need to support fundamentally different notions of what it is for (even the same) data sources to be consistent – because we may need more or less stringent notions of consistency at different times, as shared meanings converge or diverge
- the need to support gradual adoption of automated consistency checking and maintenance – because in non-toy problems, big bang adoption of automated consistency maintenance will seldom be practical even if it is desirable.

[1] The original version says "the": but the key point is that live parts, which therefore cannot be simply overwritten by recomputation, should have dependencies between them

There is a saying that there is no problem in software engineering that cannot be solved by adding another level of indirection [2]. We shall see that this may be a case in point – although many problems remain to be solved.

2 Bidirectional transformations

Bidirectional transformation (bx, for short) is the term we use for any automated way to check and restore consistency between two or more data sources. A bidirectional transformation might be written in a specialist bidirectional transformation language, or in a conventional (unidirectional) programming language; in the latter case it may in fact be a collection of programs, each doing part of the job. For example, there might be one Java program that looks at two sources and returns true or false depending on whether they are currently consistent, while another Java program take the same two sources and return a modified version of the first, modified in such a way as to bring it into consistency with the second, and yet a third changes the second source instead.

Most work on bidirectional transformations has involved just two data sources, for example, a source (containing all the information we know about) and a view (containing only a subset of the information). (This situation is the familiar *view-update problem* from databases.) Even with this restriction, there are many choices of setting to be made. Should the bx operate purely on the data sources themselves? Should it have access to additional information, such as intensional edits that give clues as to what a user who changed one source was trying to achieve? These can be very helpful when trying to make "corresponding" changes to the other data source. Or should it maintain and use trace information (such as the correspondence graph used by triple graph grammars), specifying at a fine-grained level which parts of one data source correspond to which parts of another? How should the decision be made about when to invoke the transformation? Under what circumstances should the transformation be allowed to fail, and what should happen if it does – e.g. should it then make no changes at all to the data sources, or should it improve consistency even if it cannot perfectly restore it? Etc. In this work we use the simplest possible formalisation, in which only the data sources themselves are available to the consistency restoration process. This has the practical advantage that it does not rely on changes to the data sources being made with a consistency-restoration-aware tool.

Up to now, mindful of audience, this paper has used the term "data source" – from now on we shall use the term "model", in accordance with the literature we are discussing, which is usually motivated by the needs of software development in general and model-driven engineering in particular. However, this is a distinction without a difference, since our notion of model is so general that it encompasses any data source. A point to note is that, as is conventional in the field, the term "model" sometimes means a specific collection of data with the values it takes at one instant, and sometimes means a conceptual grouping of

[2] See https://en.wikipedia.org/wiki/Fundamental_theorem_of_software_engineering.

data, encompassing all the values that it could take. We use capitals M etc. for the latter notion, which we also sometimes refer to as "model set" or "model space" when plain "model" seems likely to cause confusion. We use lowercase m etc. for the former and sometimes write "model instance" or "state of a model" to disambiguate.

3 Networks of models

In practice two models is frequently not enough: we must separate our information into more than two concerns. This raises several new problems.

The first is how to conceptualise consistency between our n concerns. There are basically two things we can do.

1. We can think of consistency as an n-ary relation, constraining all of the models simultaneously (for example, by specifying consistency using a set of constraints in which any constraint may mention any of the models). We can reify the consistency relation – the set of consistent tuples of states of all the models – as a model in its own right, which then conceptually contains *all* of the information, and becomes a central model, of which each of the original models is a view. (Figure 1.)

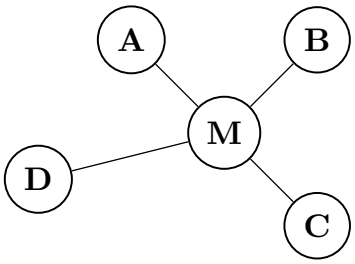

Fig. 1. Network with a central all-containing model: each $m \in M$ represents a globally consistent tuple of models (a, b, c, d) with $a \in A$ etc., and the derived consistency relation between A and M is that a' is consistent with (a, b, c, d) iff $a' = a$, etc.

2. We can embrace a network view, in which some collections (perhaps only pairs) of the n models have separately-specified consistency relations placed on them. The whole network is consistent iff all of these consistency relations hold. (Figure 2.)

The second view can of course be specialised to the first, provided there is no limit on the arity of the consistency relations used in the network. It becomes clearly different, however, if we constrain its relations to be binary, relating just two models each, so that the network is a graph (rather than a hypergraph).

Although the first view has an appealing simplicity, and may be fruitful when conditions permit, the second has practical advantages, as explored in [7,

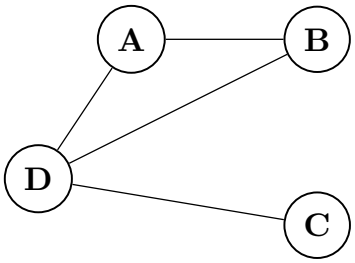

Fig. 2. Network without an added central model: consistency is represented directly in the relations between pairs of models.

8]. Although there exist n-ary relations that cannot be expressed as a conjunction of binary relations on pairs of their n places, these are cognitively difficult for humans to handle. Further, the need to handle a heterogeneous collection of models and transformations including legacy components strongly suggests that it is better to be able to work with a network of models.

Our first thought may be, then, to work with a network of models connected by binary bidirectional transformations. We may consider how to use sequences of the consistency restoration facilities of these bx to restore consistency in the network as a whole. Unsurprisingly, but unfortunately, naive approaches quickly run into problems. Given a fixed set of bx and model spaces, here are three problems that may easily arise.

1. There may be no completely consistent state of the network, e.g. because different consistency relations impose incompatible demands on the same model.
2. There may be such a state, but there may be no way to reach it using the provided bx in any sequence (see e.g. Example 5 of [7])
3. There may be many different such states, so that the overall state of the network which is reached after consistency restoration is sensitive to the precise sequence of consistency restorations chosen.

In [8] we start to explore the possibility of tackling these and other problems by adding another level of indirection (*builders*) to permit heterogeneous model and bx technologies to coexist, while simultaneously using an *orientation model* to tackle the data governance and other issues that make some sequences of consistency restoration functions acceptable and others not. In brief an orientation model records what models and relationships between them are present, and specifies which models may be modified by the consistency restoration process and which, by contrast, are currently to be considered *authoritative* and left alone; it may also specify that certain bx are to be applied only in a certain direction, thereby controlling which changes will "win" in certain conflict situations.

Taking advantage of the similarity between the problem of restoring consistency in a network of models, and the problem of incrementally building a

software system from a set of sources, we adapt the pluto build system of Erdweg et al. [3], together with its underlying theory and proofs of correctness and optimality, to this setting. In very brief, the key points are as follows.

- Each model which we may ever want updated by the consistency restoration process is equipped with a *builder* whose job is, on command, to manage the process of bringing this model into consistency with a given set of its neighbours.
- The builder might be very simple, e.g. it might simply invoke one or more legacy bx in a fixed sequence; or it might be arbitrarily complex, involving some legacy bx, some explicit modifications of the model, user interaction, search...
- What the builder has to guarantee is (a) that if it terminates successfully, then it has indeed brought its model into consistency with the specified neighbours (b) that it obeys certain rules about how it must record what information it uses, and which other builders it causes to be invoked.
- Given a set of builders that provides those guarantees, the MegaModelBuild adaptation of the pluto build system can be requested to build a particular model: it will invoke the correct builders in the correct order to guarantee appropriate correctness and optimality conditions hold.

Part of the information recorded by each builder pertains to which aspects of a model are important to its consistency with its neighbours. These *stamps* can be seen as specifying what changes to a model definitely do *not* break consistency. We turn next to considering various equivalences on models that support the restoration of consistency in networks of models.

4 Dissecting models and transformations

Several different notions of sub-models appear, when we need to reason about consistency in networks of models. Let us lay three of them on the table.

We need notation for a (set-based relational) bx: a bidirectional transformation $R : M \nrightarrow N$ between non-empty sets of models M and N is given by specifying a *consistency relation*, also by slight abuse of notation called $R \subseteq M \times N$, together a pair of functions

$$\overrightarrow{R} : M \times N \to N$$

$$\overleftarrow{R} : M \times N \to M$$

whose task is to enforce consistency. So, for example, if the current models (model instances, or states of the models) are $m \in M$ and $n \in N$, we say $R(m, n)$ holds iff m and n are to be deemed consistent according to the bx R. Given current models m and n, $\overrightarrow{R}(m, n)$ represents a new model from N, to be thought of as a version of n that has been updated in order to bring it into consistency with m.

We will assume that all bx mentioned are *correct* (consistency restoration really does restore consistency, e.g. $R(m, \overrightarrow{R}(m,n))$) and *hippocratic* (consistency restoration does not alter models that are already consistent, e.g. $R(m,n) \Rightarrow \overrightarrow{R}(m,n) = n$). We will sometimes restrict attention to bx that are *history-ignorant* (restoring consistency with a second model completely overwrites the effect of restoring consistency with a previous model, e.g. $\overrightarrow{R}(m', \overrightarrow{R}(m,n)) = \overrightarrow{R}(m',n)$).

4.1 Coordinate grid and history-ignorance

Definition 1. *(from [5][3]) The equivalence relations $\sim^M_{RF}$ and $\sim^M_{RB}$ on M, and $\sim^N_{RF}$ and $\sim^N_{RB}$ on N, are defined as follows:*

- $m \sim^M_{RF} m' \Leftrightarrow \forall n \in N. \overrightarrow{R}(m,n) = \overrightarrow{R}(m',n)$
- $m \sim^M_{RB} m' \Leftrightarrow \forall n \in N. \overleftarrow{R}(m,n) = \overleftarrow{R}(m',n)$

and dually,

- $n \sim^N_{RF} n' \Leftrightarrow \forall m \in M. \overrightarrow{R}(m,n) = \overrightarrow{R}(m,n')$
- $n \sim^N_{RB} n' \Leftrightarrow \forall m \in M. \overleftarrow{R}(m,n) = \overleftarrow{R}(m,n')$

It turns out (see [5]) that knowing the $\sim^M_{RF}$ and $\sim^M_{RB}$ class of a model m uniquely identifies it, and that in a precise sense these aspects of m separate the information which is relevant to N from that which is not. An important special case arises when these two kinds of information are sufficiently independent of one another. In that case the bx is history-ignorant and is *full* with respect to these equivalences; any choice of $\sim^M_{RF}$ and $\sim^M_{RB}$ class corresponds to a (necessarily unique) model.

4.2 Parts and non-interference

Definition 2. *(from [7]) 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 $p(c)$.*

This simple idea is related to, but more general than, the idea of a view: it may make sense to talk about parts, e.g. to talk about two model instances having the same value of a given part, even if it does not make sense to expect to update a part and have some notion of the right way to update the rest of the model.

Of course, given any bx R relating C to some other model, the set of $\sim^C_{RF}$ (rsp. $\sim^C_{RB}$) classes together with the quotient map onto it is an example of a

[3] Notation slightly adapted since we here want to refer to equivalences for multiple bx: in subscripts like RF and RB, R specifies the bx while F, B stand for forward, backward respectively

part relative to C. To say that two models have the same value of the $\sim^C_{RF}$ part is just another way of saying that they are $\sim^C_{RF}$ equivalent.

It turns out (see [7]) that we can use this idea to formalise the notion of different bx that both involve the same model caring about different parts of their common target, which is important when we want to consider whether it matters in which order we apply the bx.

Definition 3. *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*

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

Definition 4. *Given correct and hippocratic* $R : A \nleftarrow C$ *and* $S : B \nleftarrow C$*, 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 5. *Suppose we have correct and hippocratic* $R : A \nleftarrow C$ *and* $S : B \nleftarrow C$*. 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 both* $f_A(c_1) = f_A(c_2)$ *and* $f_{rest}(c_1) = f_{rest}(c_2)$ *then*

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

As suggested by the choice of terminology, we get

Theorem 1. *Let* $R : A \nleftarrow C$ *and* $S : B \nleftarrow C$ *be correct and hippocratic bx sharing a common target* C*. If there is a non-interfering A/B/rest decomposition of* C*, then* $\overrightarrow{R}$ *and* $\overrightarrow{S}$ *are non-interfering.*

(For proof, see [7])

A straightforward special case illustrates the connections between the notions in this subsection and in the previous one:

Corollary 1. *Let $R : A \nrightarrow C$ and $S : B \nrightarrow C$ be correct, hippocratic and history-ignorant bx sharing a common target C, and suppose that $\sim^C_{RF} \equiv \sim^C_{SB}$ and $\sim^C_{RB} \equiv \sim^C_{SF}$. Then $\overrightarrow{R}$ and $\overrightarrow{S}$ are non-interfering.*

The proof is routine: we take the A and B parts to be given by the equivalences and the rest part to be trivial. History ignorance ensures that the conditions of Definition 5 hold – which is not a surprise, because history ignorance indicates that C is morally a direct product of the part that is relevant to the other model with the part that is not.

4.3 Stamps, slices and optimality

The idea of a stamp on a model comes from the pluto framework where it is motivated by the wish to avoid unnecessary rebuilding of a target which, even though one or more of its sources have changed, does not need to be rebuilt because no source has changed in a way that is relevant to their target. It is originally given explicitly in terms of files and the file system, but for convenience we elide such details here, which causes a stamper to be formally almost identical to a part, although differently motivated.

Definition 6. *Given a model set M and a set S of stamp-values for M, a stamper is simply a function $s : M \longrightarrow S$.*

When a builder restores consistency to the model for which it is responsible, part of what it records is a current stamp value for each of the neighbouring models with which consistency was restored. The builder is responsible for the choice of stamper, but must use stamps fine enough to ensure that if a model changes without changing the stamp, consistency will not be lost. The framework uses this record to avoid invoking a builder that definitely has no work to do. The idea is that stamps should be quick to calculate and compare, and should reliably identify as many circumstances as possible when it is not necessary to do any consistency restoration work, such as applying a potentially expensive bx, because the non-change in the stamp guarantees that, even if the model has changed, it has not done so in a way that affects the model belonging to the builder that chose the stamp. The choice of stamper is an interesting problem: a very fine stamp (e.g. last-modified time) is cheap to use but it is clear that one can often do better in terms of saving unnecessary work. The use of the equivalences to provide and analyse stampers, and their use within networks of models, seems to hold promise in some circumstances, but is not explored here.

5 Conclusions

This paper has summarised the motivation behind, and some key points of, the author's recent work on networks of bidirectional transformations, and given pointers to further information. Much more remains to be done.

Acknowledgements

Conversations with too many people to list have informed this work, so let me just mention en masse the participants of Dagstuhl no. 18491 on *Multidirectional Transformations and Synchronisations* [1].

References

1. Anthony Cleve, Ekkart Kindler, Perdita Stevens, and Vadim Zaytsev. Multidirectional transformations and synchronisations (dagstuhl seminar 18491). *Dagstuhl Reports*, 8(12):1–48, 2018.
2. Edsger W Dijkstra. *Selected writings on Computing: A Personal Perspective*, chapter On the role of scientific thought, pages 60–66. Springer-Verlag, 1982.
3. Sebastian Erdweg, Moritz Lichter, and Manuel Weiel. A sound and optimal incremental build system with dynamic dependencies. In *OOPSLA*, pages 89–106. ACM, 2015.
4. Michael Johnson and Perdita Stevens. Confidentiality in the process of (model-driven) software development. In *Proceedings of the 7th International Workshop on Bidirectional Transformations, Bx 2018, co-located with 2nd International Conference on the Art, Science, and Engineering of Programming*. ACM, 2018.
5. Perdita Stevens. Observations relating to the equivalences induced on model sets by bidirectional transformations. *EC-EASST*, 049, 2012.
6. Perdita Stevens. Is bidirectionality important? In Alfonso Pierantonio and Trujillo Salvador, editors, *Modelling Foundations and Applications - 14th European Conference, ECMFA 2018, Held as Part of STAF 2018, Toulouse, France, July 25-29, 2018, Proceedings*, volume 10890 of *LNCS*, pages 1–11. Springer, 2018. Keynote paper.
7. Perdita Stevens. Maintaining consistency in networks of models: Bidirectional transformations in the large. *Software and System Modeling*, 19(1):39–65, 2019. In print Jan 2020. Online first, May 2019.
8. Perdita Stevens. Connecting software build with maintaining consistency between models: Towards sound, optimal, and flexible building from megamodels. *Software and System Modeling*, 2020. In press. Online first, March 2020.