Supporting Agility in MDE Through Modeling Language Relaxation

Rick Salay and Marsha Chechik

University of Toronto
Toronto, Canada
{rsalay, chechik}@cs.toronto.edu

Abstract. Agility requires expressive freedom for the modeler; however, automated MDE processes such as transformations require models to conform to strict constraints (e.g. well-formed rules). One way out of this apparent conflict is to allow a "relaxed" version of a modeling language to be used by modelers and then use tool support to "tighten" such models so that they are conformant to the original constraints. In this paper, we explore the issues of relaxation and tightening of modeling languages and discuss the possibilities for tool support.

1 Introduction

The problem of agility in MDE arises because graphical models have two very different kinds of users: humans and programs. Humans use models to express themselves and communicate with each other. Programs manipulate models to do analyses or to transform them into other models. These two types of users give rise to a modeling dilemma: humans want expressive freedom and can cope with relaxed rules while programs need models to conform to precise constraints. How can this agility conflict be reconciled?

In this paper, we propose a transformation-based framework for addressing the agility conflict for a given modeling language by meeting the needs of both kinds of users. Human needs are satisfied by relaxing the language to permit greater expressive freedom. Program needs are satisfied by defining a *tightening* transformation that converts the model in the relaxed language back into the original, more strict, language.

We limit our scope by focussing on supporting two kinds of agility: *omission agility* – allowing the modeler to omit information in the model in order to express uncertainty, irrelevance, etc., and *clarity agility* – allowing the modeler to express information in the model more concisely or differently to improve clarity. Although our scope is limited, the usefulness of these forms of agility is justified by work in the philosophy of language relating to human communication. For example, Grice defines a "cooperative principle" that gives four maxims that hold in effective human communication [4]: *quantity* – making the contribution as informative as is required but no more informative than required; *quality* – being truthful; *relation* – being relevant; and *manner* – being clear. Both types

of agility we handle address quantity, relation and manner, whereas quality (i.e., truthfulness) is an orthogonal issue and is independent of language relaxation.

The paper is structured as follows. In Section 2, we illustrate different aspects of the two agility types using five examples. In Section 3, we give a preliminary framework for language relaxation and tightening and show how it can address our examples. Section 4 explores possible tool support for the framework. We discuss related work in Section 5 and conclude in Section 6.

2 Language relaxation and tightening by example

A modeling language can be relaxed in several different ways. In this section, we explore some of these possibilities using the examples depicted in Figure 1 (A-E), referring to these as Examples A-E, respectively. All of the examples use the language of UML class diagrams (CD). In the discussion below, assume that the metamodel of CD consists of a *vocabulary* defining the element and relation types in the language and a set of *constraints* defining well-formedness – i.e., a well-formed model must conform to the constraints. For each example, we first state what the modeler is attempting to express and the type of agility required, then describe the relaxation aimed to achieve this and finally, introduce the tightening transformation required.

Example A. The modeler wants to express that she doesn't yet know what sits on the other end of the `controlledBy` association (omission agility). To do this, she weakens the well-formedness constraint that a binary association must have a class on both ends. The tightening transformation assigns a class to the target of the `controlledBy` association. Since there is choice here (i.e., an existing class or a new class), this choice must be resolved.

Example B. The modeler wants to express that in the parallel inheritance hierarchies, the classes `Car`/`Driver` and `Plane`/`Pilot` are the intended pairings with the `controlledBy` association (clarity agility). To do this, she uses the vertical alignment in the layout to indicate the correspondences. Note that neither the vocabulary nor the constraints of the concrete syntax are affected, but the expressive power of the language is extended by giving the spatial relation of vertical alignment a special meaning. The tightening transformation defines an OCL constraint for each occurrence of the vertical alignment of a pair of classes that extend `Vehicle`/`Operator` to enforce the intended constraint.

Example C. The modeler wants to indicate that she isn't sure which class should hold the `park`() operation (omission agility). To express this, she wants to link `park`() to both classes but to do that, it would have to be simultaneously contained in two boxes. This "physical" constraint, which enforces the well-formedness constraint that an operation is owned by one class, cannot be weakened unless the boxes are made to overlap. Instead, for clarity, she opts for extending the vocabulary to allow operations to be specified externally to a class, using an ellipse and linked to the class with a dashed line (clarity agility). The tightening transformation adds an operation to a class for each ellipse linked to

the class. Since in this example, two owners of the operation `park`() are specified and this violates a well-formedness constraint, there is a choice (i.e., which class is the owner?) that needs to be resolved.

Example D. To reduce clutter, the modeler wants to put the name of the class outside, but close to, its box (clarity agility). To do this, she weakens the constraint that the class name is inside the box at the top. The tightening transformation defines text close to a class box as being the name of the class. To operationalize it, the definition of "closeness" must be given.

Example E. The modeler wants to express the fact that certain classes are "connected" without being specific about the type of connection – it can be a generalization, an association, etc. (omission agility). To do this, she extends the vocabulary with a special dashed line to indicate this relation. The tightening transformation resolves the dashed line to one of the class diagram relations that can hold between classes. Since there is choice here, someone needs to make it.

3 Towards a framework for relaxation and tightening

Our ultimate goal is to develop a framework for the relaxation and tightening of modeling languages to address the agility conflict. In this section, we use the examples of Section 2 to discuss the characterizing features of relaxation and tightening that could be parts of such a framework.

The approach is given schematically in Figure 2. We assume that a modeling language has a transformation $c2a$ that generates the abstract syntax for a model expressed using its concrete syntax. Modeling agility is supported by allowing the modeler to relax the concrete syntax to a new syntax, as needed, to provide the required expressive power. Then, when the model must be used for MDE operations, the tightening transformation T that transforms the model back to the more strict concrete syntax is constructed. The composition $c2a \circ T$ takes the relaxed model to the original abstract syntax, making it amenable to MDE operations such as transformation and analysis.

The approach is motivated by the observation that human and program (MDE) users of models have different foci: humans deal with concrete syntax while MDE primarily deals with the abstract syntax of a model[1]. Thus, all of our examples are in concrete syntax. Correspondingly, our transformation-based approach to relaxation and tightening is centered around concrete syntax rather than abstract syntax. The focus on concrete syntax does not limit the expressive power of the language relaxation; on the contrary, it is greater than if the relaxation were applied to abstract syntax. While all user-relevant information from the abstract syntax is preserved in the concrete syntax, the reverse is not true, e.g., Example D in Figure 1. *One of the contributions of the present work is to bring attention to the fact that extending MDE to address human issues such as agility requires transformations on the concrete syntax.*

[1] Model editors and model layout algorithms are notable exceptions to this.

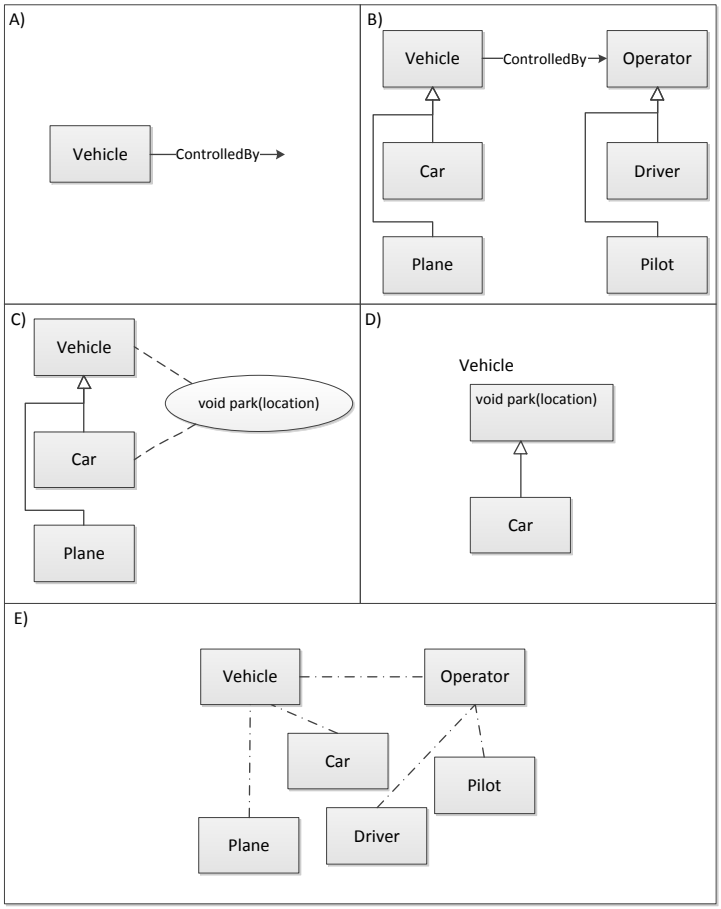

Fig. 1. Examples of language relaxation.

The motivation in Section 1 for limiting our scope to omission and clarity agility was to ensure that a tightening transformation T always exists (though it might not be necessarily unique). Relaxation to omit information can be tightened by adding back information; while relaxation to express information differently for clarity is tightened by defining an alternate expression in terms of native constructs in the original language.

3.1 Implementing relaxation and tightening

We now consider the ways in which elements of Figure 2 are affected by the relaxation and tightening process. The concrete syntax can be affected in two ways: *extending the vocabulary* (Examples C and E) or *weakening the well-formedness constraints* (Examples A, C and D). When the vocabulary is extended, the interpretation transformation `c2a` must be correspondingly broadened; but the

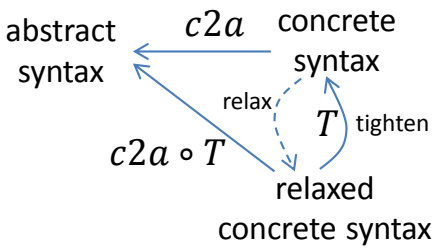

Fig. 2. Transformation-based approach to address model agility.

broadening of `c2a` may still be required even if the concrete syntax is unaffected – this is the case with Example B.

The language aspects involved in relaxation have corresponding tightening actions. Relaxing by extending the vocabulary requires tightening by redefining these extensions in terms of existing constructs. Relaxing by weakening constraints requires tightening by repairing the violations of the constraints that were weakened. In Section 4, we discuss these actions in more detail.

Support for agility. The examples in Figure 1 show how our approach applies to both clarity and omission agility. Clarity agility uses vocabulary extension in Example C and constraint weakening in Example D. In addition, Example B illustrates clarity agility when no language changes are made and only `c2a` is broadened.

Omission agility uses vocabulary extension in Examples C and E and constraint weakening in Example A. Furthermore, three different ways of omitting information are illustrated: *dropping information* (Example A), *providing alternatives* (Example C) and *using abstraction* (Example E).

Whenever omission agility is being addressed, choice may occur in the tightening process, and there are different ways to address this choice. One alternative is to elicit a decision from the modeler. Another possibility is to make a "systematic" decision (e.g., always create a new class in Example A). Yet another possibility is to defer the decision and keep all choices. We discuss this last possibility in Section 3.2.

Special characteristics of concrete syntax. The physical nature of concrete syntax makes it different from abstract syntax and this has two important implications for the framework. First, existing spatial relations that are "unused" can be appropriated for increasing expressiveness without changing the concrete syntax. This is the case with Example B where vertical alignment is given a meaning, and in Example D where closeness is given a meaning. Other relations that can be used are overlap, containment, horizontal alignment, clustering, radial alignment, etc. Second, some well-formedness constraints are enforced by the physical world, and so weakening them requires an alternative representation. This is what motivates the vocabulary extension in Example C.

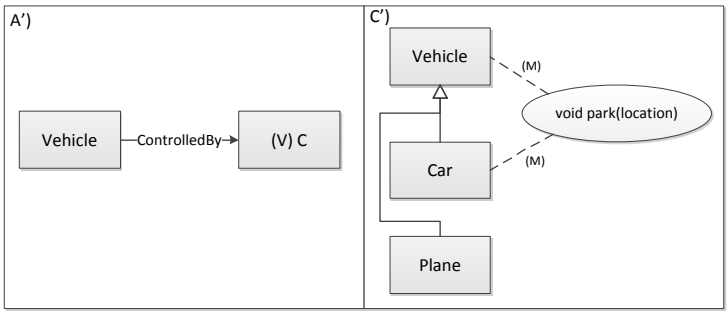

Fig. 3. Using partial modeling to express choice in Examples A and C from Figure 1.

3.2 Partial modeling

When tightening due to information omission yields a choice of alternatives, the modeler may not be comfortable having to choose, either because she doesn't yet know which choice is correct or because she wants to consider all alternatives. In this case, the technique of *partial modeling* allows the modeler to defer the decision and provides an alternative to tightening.

A partial model can express a set of possible models through the use of model annotations and is typically used to express model uncertainty. For example, Figure 3 shows the use of the *MAVO* partial modeling approach to express the choices due to the omission of information in Examples A and C of Figure 1, resulting in A' and C', respectively. The `V` annotation in Example A' means that the class `C` is a "variable" and so this represents a set of different models according to how the variable is instantiated. The `M` annotations indicate "maybe" so Example C' represents the set of models in which only one of the operation ownership relations exists. Due to lack of space, we omit a detailed description of *MAVO* partial modeling – interested readers are directed to [11]. The benefit of using partial models is that the annotations have formal semantics and thus partial models can be used in place of ordinary models in MDE operations such as property checking [11,2] and transformation [3].

4 Towards tool support

Our strategy relies on tool support. In this section, we discuss some of the possibilities for this in terms of existing technologies.

Relaxation. The relaxation tool may be a general drawing tool (e.g., Visio) with a predefined template for the concrete syntax to allow models expressed in the original language to be drawn. The relaxation mechanism of vocabulary extension is achieved by allowing other shapes to be drawn as well. The relaxation mechanism of constraint weakening is achieved by supporting an operating mode that does not enforce (selectable) constraints (e.g., see "soft validation" in

Section 5). Note that physical constraints imposed by the concrete syntax cannot be weakened, so these are addressed by vocabulary extension as in Example C.

Tightening. Constructing the tightening transformation is the more difficult part of the approach. There are two steps involved in the construction:

(1) Identify an occurrence of a language relaxation. Instances of vocabulary extension or constraint weakening (i.e., violation) are easy to detect automatically. Instances of broadening the interpretation may be impossible to detect without the modeler "pointing it out". One clue may be to detect occurrences of spatial relations (e.g., vertical alignment).
(2) Construct the appropriate tightening depending on the type of relaxation:
   – For *weakened constraints*, we must fix model inconsistencies relative to the original constraints. To do this, we can rely on existing computational approaches for computing *minimal model repairs*. The objective here is to search the space of possible changes to the model to find the minimal changes that fix a constraint violation. See Section 5 for a discussion of this work in the literature. If there is still a choice left after the repair process (i.e., there are several possible minimal repairs), then a strategy for dealing with choice must be followed, e.g., to elicit the decision from the modeler, follow a predefined choice policy or use a partial modeling mechanism as described in Section 3.2.
   – For *vocabulary extensions* and other interpretation broadening, a definition of the new elements/information in terms of constructs in the original language must be elicited from the modeler. Clearly, this requires the use of a transformation language for expressing this redefinition, and we rely on existing solutions for this, e.g., ATL[2].

5 Related Work

The use of relaxation to increase agility has been proposed in various contexts. There is a long tradition of work on relaxing the input method by allowing freehand sketching of models. See [8] for a recent example and [5] for a survey. Support for conversion of sketches to the "computer" form of the concrete syntax has been developed in commercial tools and explored in research (e.g., [1]). In contrast to this work, our focus is on agility through the relaxation of the concrete syntax rather than the input method.

Some modeling tools allow "soft validation" where the satisfaction of well-formedness constraints can be deferred until a more appropriate time when the modeler is finished with a unit of work (e.g., see Microsoft modeling tools[3]). This mechanism can be used as a limited form of language relaxation but it only addresses constraint weakening and not vocabulary extension.

[2] http://www.eclipse.org/atl/
[3] http://msdn.microsoft.com/en-us/library/bb245773.aspx

Metamodel relaxation has been used for purposes other than increasing the expressive power of models. For example, in [6], the authors propose a way of automatically constructing a transformation language from the concrete syntax of a modeling language. Since transformation rules must work with non-well-formed model fragments, the creation of the transformation language requires a relaxation of the original language. Both vocabulary extension and constraint weakening are used to achieve this. Further, since transformation languages have a different use than the original language they are based on, language modifications are required as well. In other work, Ramos et. al. [9] use model fragments as a way of specifying model patterns for pattern matching. They use constraint weakening to relax a metamodel so that model fragments (called "model snippets" here) become acceptable instances. Similarly, Levendovszky et. al. [7] use constraint weakening to construct metamodels that allow design patterns to be defined, while Sen et. al. [12] use constraint weakening to define metamodels of models containing partial knowledge in order to support transformation testing. In most of this work, the focus is on creating a metamodel that can accept model fragments as instances, with constraint weakening being the primary mechanism to achieve this. In our work, the goal is richer and hence vocabulary extension plays a larger role.

As discussed in Section 3, the issue of "model tightening" is dependent on mechanisms for model repair. Due to lack of space, we omit a thorough review of work in this area and instead only mention recent examples. Many approaches focus on attempting to formulate repair rules representing various change scenarios where specific repair actions are performed in response to detected changes, e.g., [13]. Others automatically infer the needed repairs directly from the well-formedness constraints and the violation, e.g., [10]. Many of these approaches also handle the elicitation of a decision from the user when a choice of multiple repairs is available. Our tightening transformations can work with either of these techniques.

6 Conclusion

Models are used by humans and programs in different ways, giving rise to what we have called *the agility conflict*: humans require expressive freedom while programs require strict conformance to constraints. In this paper, we outlined the beginnings of a framework to address the agility conflict with a focus on two types of agility: *omission agility* which gives the modeler the freedom to omit information, and *clarity agility* which allows the modeler the ability to rephrase information to improve clarity. Our approach involves relaxing the modeling language to support these types of agility and then constructing a tightening transformation to put the relaxed model back into a form that can be accepted by MDE processes. We explored the approach through a series of examples, discussing its characteristics and potential tool support. Our next steps are to further develop the theoretical details of this approach and prototype tool support for it.

References

1. Th. Buchmann. Towards Tool Support for Agile Modeling: Sketching Equals Modeling. In *Proc. of XM'12 Wksp*, pages 9–14, 2012.
2. M. Famelis, M. Chechik, and R. Salay. Partial Models: Towards Modeling and Reasoning with Uncertainty. In *Proc. of ICSE'12*, 2012.
3. M. Famelis, R.Salay, A. Di Sandro, and M. Chechik. Transformation of Models Containing Uncertainty. In *Proc. of MODELS'13*, 2013.
4. H. P. Grice. Logic And Conversation. In Cole et al., editor, *Syntax and Semantics 3: Speech arts*, pages 41–58. Elsevier, 1975.
5. G. Johnson, M. Gross, J. Hong, and E. Yi-Luen Do. Computational Support for Sketching in Design: a Review. *J. Foundations and Trends in HCI*, 2(1):1–93, 2009.
6. Th. Kühne, G. Mezei, E. Syriani, H. Vangheluwe, and M. Wimmer. Explicit Transformation Modeling. In *Proc. of MODELS'10*, pages 240–255, 2010.
7. T. Levendovszky, L. Lengyel, and T. Mészáros. Supporting domain-specific model patterns with metamodeling. *Software & Systems Modeling*, 8(4):501–520, 2009.
8. N. Mangano, A. Baker, M. Dempsey, E. Navarro, and A. van der Hoek. Software Design Sketching with CALICO. In *Proc. of ASE'10*, pages 23–32, 2010.
9. R. Ramos, O. Barais, and J. Jézéquel. Matching model-snippets. In *Model Driven Engineering Languages and Systems*, pages 121–135. Springer, 2007.
10. A. Reder and A. Egyed. Computing Repair Trees for Resolving Inconsistencies in Design Models. In *Proc. of ASE'12*, pages 220–229, 2012.
11. R. Salay, M. Famelis, and M. Chechik. "Language Independent Refinement using Partial Modeling". In *Proc. of FASE'12*, 2012.
12. S. Sen, J. Mottu, M. Tisi, and J. Cabot. Using models of partial knowledge to test model transformations. In *Theory and Practice of Model Transformations*, pages 24–39. Springer, 2012.
13. Y. Xiong, Z. Hu, H. Zhao, H. Song, M. Takeichi, and H. Mei. Supporting Automatic Model Inconsistency Fixing. In *Proc. of ESEC/FSE'09*, pages 315–324, 2009.