Model-versioning-in-the-large: algebraic foundations and the tile notation

Zinovy Diskin, Krzysztof Czarnecki, Michal Antkiewicz
Generative Software Development Lab
University of Waterloo, Canada
{zdiskin, kczarnec, mantkiew}@gsd.uwaterloo.ca

Abstract

Model-versioning-in-the-large is concerned with complex scenarios involving multiple updates and multiple replicas of a model. The paper introduces tile systems *as rephrasing of* double categories *in model versioning terms, and shows that the tile language enables a very general formalization of versioning concepts. The formalization makes the concepts amenable to algebraic analysis and provides a convenient notation for version system designers. It also allows one to formulate algebraic laws that a correct versioning system must or may want to satisfy.*

1 Introduction

The majority of work in model versioning has focused on what can be described as *versioning-in-the-small*: performing basic model versioning tasks on pairs of models. These tasks include computing and visualizing the differences between models (e.g., [8, 9, 12, 13]) and, for optimistic systems [10, 11], replica synchronization. The latter includes propagation of updates between replicas and conflict detection and reconciliation (e.g., [2, 14]).

Another perspective on model versioning is to consider compositions of the basic versioning tasks, which we refer to as *versioning-in-the-large*. Such compositions arise in versioning scenarios involving multiple model updates and multiple replicas. *Replicas* are concurrent copies of the same model maintained by individual developers or teams. They can be understood as trajectories of models in time. Points in the trajectories, that is, snapshots of the same replica at different time moments, are *versions*. Replicas and versions together form a two-dimensional space in which model versioning scenarios unfold. Our goal is to provide an adequate mathematical framework and a convenient notation for reasoning about this space and the scenarios populating it.

Our framework is based on the notion of *tile*, which can be understood as a two-dimensional delta that captures model differences across both replicas and versions (Fig. 1). Tiles are the basic blocks of versioning-in-the-large and their composition enables modeling of complex scenarios.

We call a universe of tiles closed under composition in both dimensions a *tile system*. Mathematically, the latter is a rephrasing of the notion of a *double category*, which has been studied in category theory since the sixties (e.g., [7]). As category theory is a powerful algebraic discipline, the language of tiles brings algebraic foundations and machineries to model versioning; it also provides a convenient diagrammatic notation for describing complex scenarios. We consider these two aspects of tile systems as the main contributions of this paper. Applying tiles to concrete model versioning problems is our ongoing project; some results will be presented in a forthcoming paper [5].

We demonstrate the utility of the tile framework for model versioning with two applications. The first one is the Pasting Lemma directly borrowed from category theory: it says that although a complex tile can be composed from its subtiles in different ways, the result of composition remains the same. The second application goes beyond mere rephrasing of a known result. We show that reconciliation can be understood as an operation on tiles, and its properties can be specified algebraically via universally valid equations. Particularly, we formulate the compatibility of reconciliation with tile compositions in precise algebraic terms.

2 Versioning-in-the-small: deltas are *tiles*

There are two sorts of deltas in optimistic versioning. Time-deltas are relationships between two versions of the same replica at different time moments. Replica-deltas are relationships between two replicas at the same time moment. We will refer to time-deltas as *updates* and to replica-deltas as *matches*. Versioning scenarios comprise both types of deltas; thus, their elementary unit is a square cell shown in Fig. 1.

Nodes A and B denote two replicas of the same model maintained by two developers, say, Ann and Bob. Horizontal arrow (match) m denotes relationship between the two versions of the replicas at the moment t. It can be under-

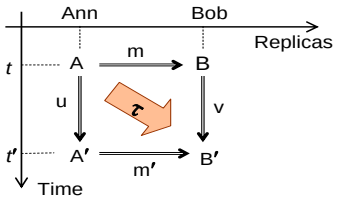

Figure 1. The space of model versioning

stood operationally as a specification of what is to be done to transform A to B. It can be also understood declaratively as a special relational structure specifying similarities and discrepancies (conflicts) between the models.

Ann and Bob work independently and at the moment t' we have two updated versions A' and B' with arrows u and v denoting the corresponding updates. Again, the updates can be understood either operationally as edit sequences, or declaratively as relational structures specifying what was kept unchanged and what was modified during the update.

The four deltas m, m', u, v are mutually related and we need a special data format for storing the cell. We call this format a *tile* and denote it by a directed block-arrow (τ in Fig. 1). We may interpret τ as an update of the match arrow m, or as a match between updates u and v. In fact, τ comprises both: it is a two-dimensional (2D) delta.

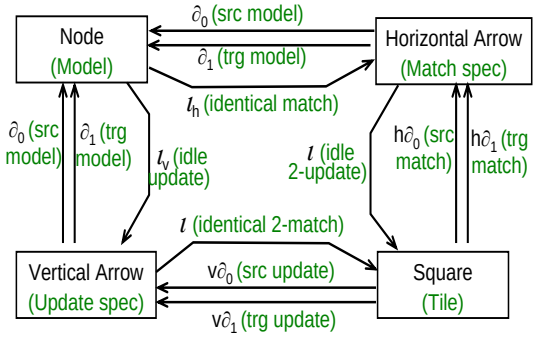

Figure 2. Metamodel of tile structure

Fig. 2 provides the metamodel of the structure of tiles. It defines four classes of objects constituting the tiles: *nodes*, *horizontal* and *vertical arrows*, and *squares* (we follow the terminology used in category theory). Sometimes we will refer to all these objects as *cells*: nodes are 0-cells, arrows (of both types) are 1-cells and squares are 2-cells. We will often write *h-* and *v-arrow* for, respectively, horizontal and vertical arrow.

Although the class names carry notational connotations (node, arrow), the classes refer to abstract semantic rather than syntactic objects. In the model versioning context, these objects obtain concrete semantic interpretations: nodes are models, h-arrows are matches, v-arrows are updates, and squares are tiles (2D-deltas). The cells constituting a tile are adjoint to each other in a special way, and their incidence relations are specified by directed ∂_i-associations ($i = 0, 1$) between classes; in Fig. 2 they are shown by straight arrows. In fact, ∂-arrows are totally defined functions (in the UML jargon, they are associations with multiplicities 1 for the directed ends and 0..* for the undirected ends). The ∂_0-arrows points to the source (src) and ∂_1-arrows to the target (trg) cells; optional prefixes 'v' and 'h' stand for, respectively, 'vertical' and 'horizontal' source or target. Bent ι-arrows point to identity/idle objects assigned to cells; they are neutral units for arrow and tile composition and will be explained later in Section 3. In the versioning context, given a model A, the identity h-arrow $\iota_h(A)$ is the identity relation on A interpreted as the identity match, and the idle v-arrow $\iota_v(A)$ is the idle update of A, i.e., an update that does nothing.[1]

We use the term *tile* in two senses: as a name for the abstract construct (square) and as a name for a 2D-delta, comprising two matches and two updates. To keep the framework general, we do not impose any specific restrictions on what matches, updates, and tiles are. The only requirement is that any cell of dimension n has uniquely defined $(n-1)$-dimensional source and target cells, and a uniquely defined identity cell of dimension $(n+1)$ as shown by the metamodel.

2.1 Example: relational tiles

Figure 3 presents a toy example illustrating the notion of tile. Models are very simple structures consisting of objects with attributes. Attributes have a name and a value. Moreover, each model in Fig. 3 has only one object. Model IDs are A, B, A', and B'. The primed IDs denote the updated versions. Model element IDs use letters P, Q for the objects and a, b with subscripts for the attributes.

In the specifications of matches, similarity or "sameness" of elements is denoted by $x{=}y$, whereas discrepancy or conflict is denoted by $x{\neq}y$. In the specifications of updates, preservation of elements is denoted by $x{\leftrightarrow}y$, whereas modification is denoted by $x{\rightsquigarrow}y$. Matches and updates are, in fact, binary relations between the models; these relations are compatible with the structure of the models. Using the set-theoretical notation, we may write $m \subset A \times B$ and $u \subset A \times A'$; however, keep in mind that m and u are structures (similar to A and B) rather than merely sets of pairs of elements. We also have the "sameness" sub-relations $m_{=} \subset m$ and $u_{\leftrightarrow} \subset u$. Further, $a{\neq}b$ says that the pair of elements $(a, b) \in m$ but $(a, b) \notin m_{=}$, and similarly $a{\rightsquigarrow}a'$

[1] Note that for any cell c of dimension $|c| = n$, we have $|\partial_i(c)| = n{-}1$ and $|\iota(c)| = n{+}1$. Correspondingly, ∂s are not defined for 0-cells (nodes) and $iota$s are not defined for 2-cells (squares). However, if higher dimensions are needed, we can define the notion of higher-dimensional tiles along the lines described above.

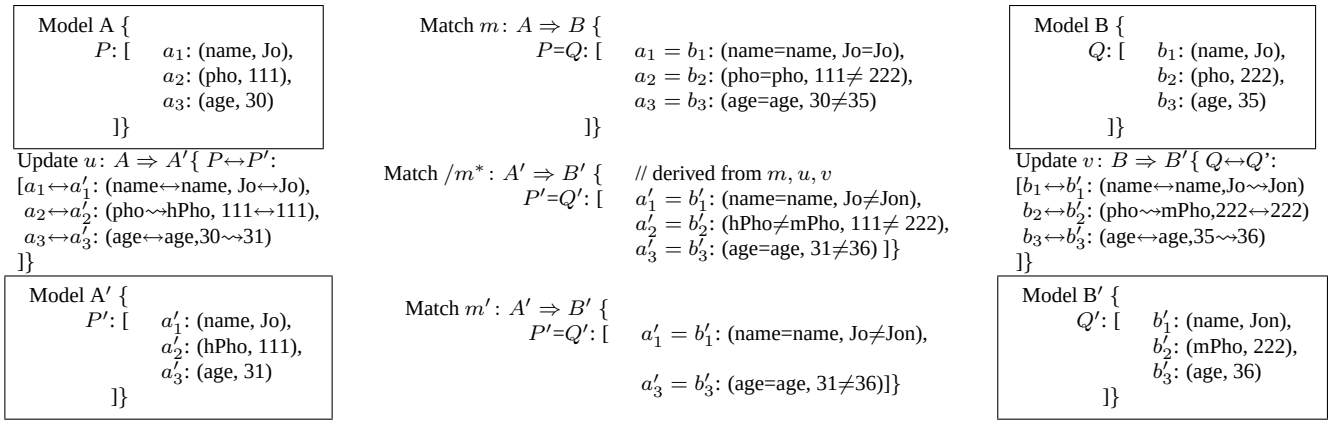

Figure 3. Example of relational tile

means $(a, a') \in u$ but $(a, a') \notin u_{\leftrightarrow}$.

Given the match m and the updates u, v, we can compute the match $/m^*$ between the updated models by relational composition as shown in the middle part of Fig. 3 (following UML, we denote derived elements by slash-prefix). In this derived match specification, attributes a'_2 and b'_2 appear to be the same but with naming and value conflicts. However, suppose that we know that "phone" in model A refers to *home* phone, whereas "phone" in B means *mobile* phone. Then it is not reasonable to consider these attributes to be the same, and we need to revise the match as shown in the bottom of Fig. 3: the pair a'_2=b'_2 is removed from match m' and, correspondingly, the two conflicts disappear (note the empty row). Overall, the tile can be seen as an update of match m: the conflict of names Jo≠Jon appears in m' because of the distinct updates $(u$: Jo↔Jo$)\neq(v$: Jo⇝Jon$)$, and the conflict of phone numbers disappears.

We stress that the match of the updated replicas is an independent (rather than derived) piece of data forming the tile. In fact, tile τ specified in Fig. 3 can be decomposed as shown in Fig. 4(a): we first compute match $/m^*$ and then revise it coming to match m'. The block-arrow Δm^* in Fig. 4(a) can be seen as a relation between matches $/m^*$ and m' (in fact, as a 2-relation since matches are themselves relations). In a similar way, we can decompose the tile horizontally if we wanted to view it as a revision of the derived update $/u^*$ towards update v.

Thus, all components of the tile are essential and neither one can be derived from others. The tile is a quadruple of binary relations having "same"-subrelations and compatible sources and targets; we call this data format a *relational* tile. Note that the abstract definition admits existence of different tiles with the same four boundary arrows; this generality is not used in our example.

2.2 2-Arrows and idle/identity arrows

Horizontal and vertical *2-arrows* are two important special cases of tiles. They are shown in Fig. 4(b1,b2). For tile (b1), vertical arrows are *idle* updates, i.e., updates that do nothing. In our example of relational tiles, their same-relations are full diagonals and the change-relations are empty. The tile then presents a revision of match specification between two replicas as we discussed in the previous section. Essentially, the bottom "semi-tile" in Fig. 4(a) is an abbreviation of a tile whose vertical arrows are idle updates, $1_{A'}: A' \Rightarrow A'$ and $1_{B'}: B' \Rightarrow B'$ for the left and right sides respectively. For tile (b2), horizontal arrows are *identity* matches that declare the two models to be the same. In our example of relational tiles, their same-relations are full diagonals and the conflict relations are empty. The tile then presents a revision of an update between the same two models A and A'.[2]

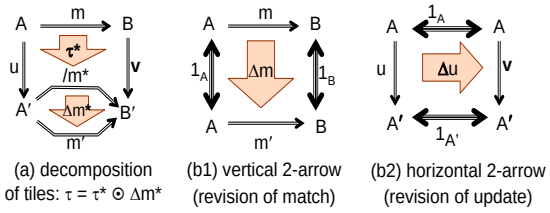

Figure 4. Examples of 2-arrows

If updates are understood operationally as edit sequences, then u and v may be two different sequences with the same result. If updates are understood declaratively as specifications of what is kept and what is changed, then there may be different relations u and v between the same two nodes. If even A and A' are processed with the same tool and their el-

[2] In our context, 'idle' and 'identity' are synonyms, but we prefer to use 'idle' for updates and 'identity' for matches.

ements have universally unique IDs, there is still a room for different update specifications between the same two models.[3]

3 Versioning-in-the-large: tile composition

The goal of this section is to give a precise definition of tile systems. We begin with motivating considerations using Fig. 5. Tiles τ and σ, for which $\mathsf{v}\partial_1(\tau) = \mathsf{v}\partial_0(\sigma)$, can be composed horizontally. The resulting tile AC' is denoted by $\tau \otimes \sigma$ and has the following components: $\mathsf{v}\partial_0(\tau \otimes \sigma) = u$, $\mathsf{v}\partial_1(\tau \otimes \sigma) = w$, $\mathsf{h}\partial_0(\tau \otimes \sigma) = m; n$ and $\mathsf{h}\partial_1(\tau \otimes \sigma) = m'; n'$. Here ; denotes sequential composition of both updates and matches.

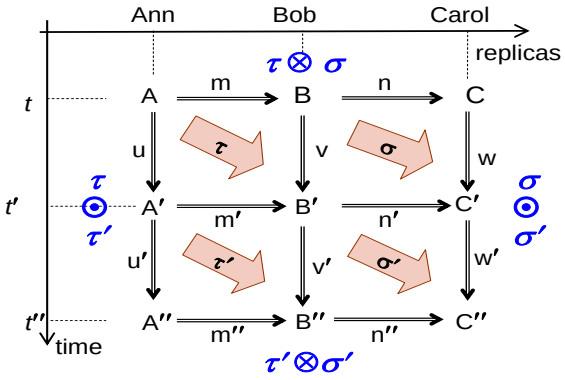

Figure 5. Composition of tiles

Thus, for horizontal tile composition $\otimes$, matches are composed sequentially, whereas updates are parallel. Similarly, tiles τ and τ' that have a common intermediate match m' can be composed vertically to produce tile AB''. The tile is denoted by $\tau \odot \tau'$ and has evident horizontal and vertical sources and targets: updates are composed sequentially, whereas matches are parallel.

Consequently, we have two binary operations on tiles: horizontal $\otimes$ and vertical $\odot$ composition. It is reasonable to require them to be associative (if tiles consist of binary relational structures, their composition is indeed associative). Moreover, both compositions have identity tiles shown in Fig. 6(a1,b1). These tiles serve as units (neutral elements), i.e., the following equalities hold for an arbitrary tile τ with the sides (u, v, m, m') as shown in Fig. 5, :

(1) $1_u \otimes \tau = \tau = \tau \otimes 1_v$ and $1^m \odot \tau = \tau = \tau \odot 1^{m'}$.

Such idle tiles are assigned to every h-arrow and to every v-arrow as specified by the bent ι-arrows in Fig. 2 (2-match means a match of updates and 2-update is a revision of match). Also, idle v- and h-arrows are assigned to every node. Further, formation of idle tiles is compatible with tile composition Fig. 6(a2, b2) and with identity arrows Fig. 6(ab).

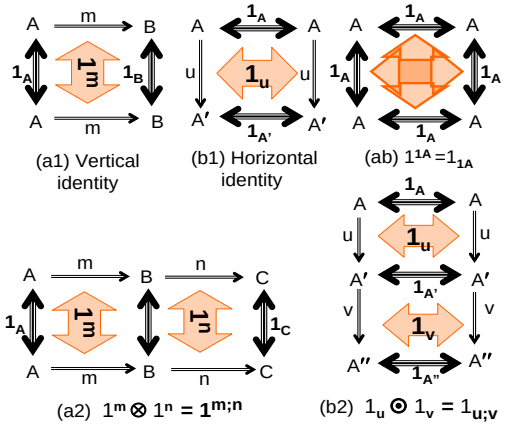

Figure 6. Identity/idle tiles

Finally, if we have four tiles with common intermediate matches and updates as shown in Fig. 5, we can compose them in two different ways. We can first compose two pairs of tiles horizontally, and then compose the results vertically coming to the tile $AC''_{(1)} = (\tau \otimes \sigma) \odot (\tau' \otimes \sigma')$. Alternatively, we may first compose vertically and then horizontally producing the tile $AC''_{(2)} = (\tau \odot \tau') \otimes (\sigma \odot \sigma')$. When tiles are composed from relational structures, it can be proved that $AC''_{(1)} = AC''_{(2)}$ [5]. Thus, for our tile systems, we should require the following *interchange law*:

(2) $(\tau \otimes \sigma) \odot (\tau' \otimes \sigma') = (\tau \odot \tau') \otimes (\sigma \odot \sigma')$.

Definition. An *(abstract) tile system* is a four-sorted algebraic structure consisting of *nodes*, *h-arrows*, *v-arrows* and *tiles*. Every h- and v-arrow is assigned two nodes called its *source* and *target*, and every node N is assigned *idle h-arrow* and *idle v-arrow* as shown by the metamodel in Fig. 2. These idle arrows are loops: $\partial_0(\iota_h(N)) = N = \partial_1(\iota_h(N))$ and $\partial_0(\iota_v(N)) = N = \partial_1(\iota_v(N))$.

Every tile is assigned two h-arrows (*h-source* and *h-target*) and two v-arrows (*v-source* and *v-target*) such that the incidence relations between the cells hold.[4] Every h-arrow is assigned *v-idle tile*, and every v-arrow is assigned *h-idle tile* (see the metamodel in Fig. 6), which are 2-loops.[5]

H-arrows can be composed; their composition is associative; and h-idle arrows act as units. V-arrows can be composed; their composition is associative; and v-idle arrows

[3] Suppose that Ann deleted some element a but later, after many editing steps, decided to restore it. Ann then creates a new element a' with the same name and all other attributes. However, element a' has a new ID and hence the "sameness" of a and a' must be explicitly declared in v.

[4] These relations are geometrically evident but for the formally minded reader, they are as follows: $\partial_i(\mathsf{h}\partial_i(\tau)) = \partial_i(\mathsf{v}\partial_i(\tau))$, $i = 0, 1$ and $\partial_i(\mathsf{h}\partial_j(\tau)) = \partial_j(\mathsf{v}\partial_i(\tau))$, $i = 0, 1$, $j = 0, 1$ and $i \neq j$.

[5] To make looping explicit, glue together the horizontal sides of tile (a1) and the vertical sides of tile (b1).

act as units. Tiles can be composed *horizontally* and *vertically*; both compositions are associative; and h- and v-idle tiles act as the respective units. Finally, for any four tiles related as shown in Fig. 5, the interchange law (2) holds. In terms of category theory, the definition above says that an abstract tile system is a *double-category* [7].

The main result on double categories providing algorithmic applications is the following lemma.

Pasting Lemma [3]. In any double category having so called *factorization*, composition of compatible tiles in any order gives the same result.

An illustrating example is shown in Fig. 7: to ease reading the equalities, the symbol of v-composition is omitted. Factorization means, roughly, that if a boundary arrow of a tile is composed (e.g., the left side of tile σ_3), then the tile can be presented as the composition of the corresponding smaller tiles ($\sigma_3 = \tau_{13}\tau_{23}$ not shown in the figure). The majority of tile systems appearing in mathematical practice do have factorizations [3]; this class also includes relational tile systems of model versioning [5].

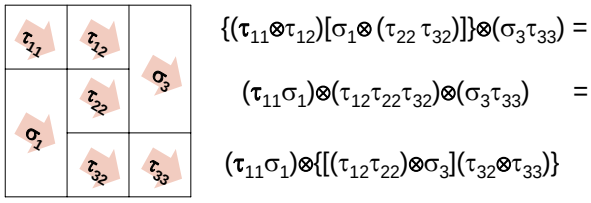

Figure 7. Application of Pasting Lemma

The distinction between sequential and parallel composition of one-dimensional deltas is not new, e.g., [2]. However, our analysis shows that in optimistic replication, composition of deltas is actually two-dimensional and mixed: either updates are composed sequentially while matches are parallel (vertical composition), or matches are composed sequentially while updates are parallel (horizontal composition).

4 Reconciliation as a tile operation

A typical situation of optimistic model versioning is shown in the upper-middle cell in Fig. 8 (cf. [6, Fig. 1]). An original model O is concurrently updated by two teams, which results in two different and possibly conflicting replicas A, B.

The upper-right cell in Fig. 8 rephrases the situation as a tile. Since new elements may be introduced in both replicas, a correct match $m\colon A \to B$ is not derivable from update specifications u and v. The reason is that potential "sameness" of the new elements across the replicas needs to be explicitly asserted. Hence, the match $m\colon A \to B$ presents an independent piece of input for reconciliation. On the other

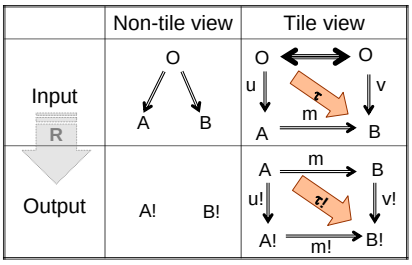

Figure 8. Reconciliation schemas

hand, the history of obtaining versions A, B from the ancestor O matters for conflict resolution; hence the reconciliation procedure should know the updates u and v. Thus, the input for the reconciliation operation is given by the entire tile.

A reconciliation policy (e.g., [6, 14]) allows some of the conflicts specified by m to be resolved automatically, whereas resolving the remaining conflicts may require human input. A common approach is to leave such conflicts for future reconciliation and to allow the replicas to *diverge*, i.e., to remain different after synchronization [6].

The lower-middle cell in Fig. 8 shows the divergent case of merging A and B, which results in partially reconciled and partially conflicting replicas $A!$ and $B!$. The corresponding tile is shown in the lower-right cell. In addition to $A!$ and $B!$, the tile also includes the updates $u!$ and $v!$ that relate the updated replicas $A!$ and $B!$ to A and B, respectively, and the non-identity match $m!\colon A! \to B!$. The update and reconciliation cycle can be repeated by applying new updates u' and v' to $A!$ and $B!$, and then applying reconciliation on the new tile $A!B!A!'B!'$. Thus, reconciliation can be viewed as an algebraic operation mapping tiles to tiles, $!\colon Tiles \to Tiles$, such that $\mathsf{h}\partial_0(\tau!) = \mathsf{h}\partial_1(\tau)$ for any tile τ. The latter condition means that reconciliation works vertically.

An important requirement on a reasonable reconciliation policy is its compatibility with horizontal tile composition, that is, the following equation should hold for any two horizontally adjacent tiles τ,σ (Fig. 5):

(3) $(\tau \otimes \sigma)! = \tau! \otimes \sigma!$

Another useful requirement is history independence, which is captured by the equation

(4) $(\tau \odot \tau')! = \tau'!$

that holds for any two vertically adjacent tiles τ,τ'. This equation can be seen as a general pattern for formulating the history independence laws for synchronization systems [4].

5 Application scenario

Figure 9 shows a simple example of using the tile notation. The diagram presents a precise formal specification: nodes are models and arrows are relations between them organized in tiles. At the same time, this formal specification

is intuitive and can be easily understood. On Monday, Ann and Bob started to work concurrently with the same model O (note the top identity h-arrow). On Tuesday, their replicas were automatically reconciled (as denoted by the vertical block-arrow of reconciliation operation). Bob's reconciliation result was copied to Carol (note the horizontal-identity 2-arrow $1_{v!}$), who continued to work concurrently with Ann and Bob. On Wednesday, Ann's and Bob's replicas were again reconciled, Bob continued to work with the model while Ann finished. On Thursday, Bob and Carol reconciled their replicas, which finished the week. When such scenarios comprise a big amount of tiles, Pasting Lemma turns out a useful result.

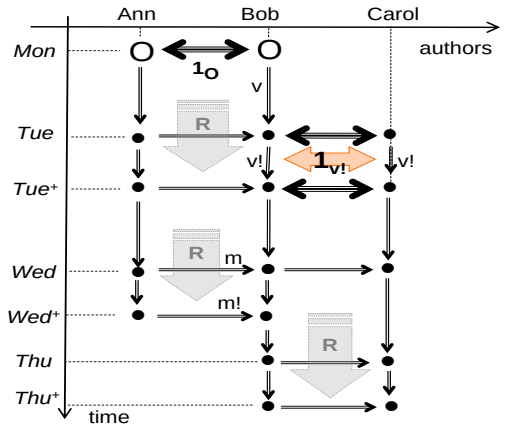

Figure 9. Use case of tile machinery

We can zoom into each element of the tile diagram to obtain a corresponding structure, such as a model, relation, or tile. Formally, it means that we have a mapping $\#\colon \mathcal{T} \times \mathcal{R} \to Tiles$ between two tile systems.[6] Given two replica IDs A, B and two time moments t, t', the value $\#(t, t', A, B)$ is a tile comprising four model versions with matches and updates between them.

Importantly, mapping # must be compatible with algebraic structure of tiles, i.e., vertical and horizontal composition and identities, and hence be a *tile system homomorphism*. In fact, a model versioning system *is* an implementation of a tile system homomorphism. This brief formulation encodes several important algebraic conditions that a correct implementation must satisfy.

6 Conclusion

The paper shows that the algebraic structure of tile system a.k.a double-category is well suited for formally modeling version management concepts in a very general and metamodel-independent way. Reformulation of versioning constructs in the tile language makes them amenable to algebraic manipulations and provides a convenient notation for versioning-system designers. Unexpectedly, this reformulation allowed us to revise some basic concepts, such as sequential and parallel composition of deltas, and make them more precise. Our reformulation also reveals several important algebraic laws that a correct versioning system should or may want to satisfy. The tile framework may also provide useful guidance in designing new algorithms for model versioning, particularly in the context of incremental updates.

Whereas our discussion focused on the two-dimensional space of model versioning, other aspects of versioning, such as model heterogeneity and metamodel evolution, or features [1], may bring new dimensions. This generalization may require the constructs and setting of higher-dimensional category theory, which would lead to a higher-dimensional algebra of model versioning.

[6] The source of # is a trivial partial-order tile system formed by Cartesian product of two posets: a poset of time moments $\mathcal{T}$, as usual for modeling time in distributed environments, and a poset $\mathcal{R}$ of replica IDs.

References

[1] D. Batory, M. Azanza, and J. Saraiva. The objects and arrows of computational design. In *MoDELS*, pages 1–20, 2008.

[2] A. Cicchetti, D. D. Ruscio, and A. Pierantonio. Managing model conflicts in distributed development. In *MoDELS*, pages 311–325, 2008.

[3] R. Dawson and R. Pare. General associativity and general compostion for double categories. *Cahiers de topologie et géométrie différentielle catégoriques*, 34:57–79, 1993.

[4] Z. Diskin. Algebraic models for bidirectional model synchronization. In *MoDELS*, pages 21–36, 2008.

[5] Z. Diskin, M. Antkiewicz, and K. Czarnecki. Declarative metamodel-independent definitions of model matches and updates: Relational tile systems. In preparation.

[6] J. N. Foster, M. B. Greenwald, C. Kirkegaard, B. C. Pierce, and A. Schmitt. Exploiting schemas in data synchronization. *J. Comput. Syst. Sci.*, 73(4):669–689, 2007.

[7] G. Kelly and R. Street. Review of the elements of 2-categories. In *Category Seminar, Sydney 1972/73*, Lecture Notes in Math., 420, pages 75–103, 1974.

[8] Y. Lin, J. Gray, and F. Jouault. DSMDiff: A Differentiation Tool for Domain-Specific Models. *European J. of Information Systems*, 16:349–361, 2007.

[9] D. Ohst, M. Welle, and U. Kelter. Differences between versions of UML diagrams. In *ESEC/FSE*, pages 227–236, 2003.

[10] Y. Saito and M. Shapiro. Optimistic replication. *ACM Comput. Surv.*, 37(1):42–81, 2005.

[11] G. L. Thione and D. E. Perry. Parallel changes: Detecting semantic interferences. In *COMPSAC*, pages 47–56, 2005.

[12] C. Treude, S. Berlik, S. Wenzel, and U. Kelter. Difference computation of large models. In *ESEC/FSE*, pages 295–304, 2007.

[13] Z. Xing and E. Stroulia. UMLDiff: an algorithm for object-oriented design differencing. In *ASE*, pages 54–65, 2005.

[14] Y. Xiong, D. Liu, Z. Hu, H. Zhao, M. Takeichi, and H. Mei. Towards automatic model synchronization from model transformations. In *ASE*, pages 164–173, 2007.