# *iVuBlender*: A Tool for Merging Incomplete and Inconsistent Views

Mehrdad Sabetzadeh Steve Easterbrook
Department of Computer Science, University of Toronto, Canada
Email: {mehrdad,sme}@cs.toronto.edu

## Abstract

*View merging is an important activity in any conceptual modeling language. It is often desirable to combine a set of views to gain a unified perspective, to test hypotheses about how views are related, or to perform various types of analysis. A major challenge for view merging is toleration of incompleteness and inconsistency: views may be inconclusive, and may have conflicts over the concepts being modeled or how they are structured. Drawing on the theory developed in our earlier work [6, 5], we present a view merging tool, called iVuBlender, that allows for explicit modeling of incompleteness and inconsistency and provides a framework for interconnecting and merging incomplete and inconsistent views.*

## 1 Introduction

View merging is a core activity in model management. The existing work on view merging [3, 4] assumes that views are consistent prior to merging them. However, for most applications that involve multiple stakeholders, views are unlikely to be consistent a priori. Hence, considerable effort may be needed to detect and repair inconsistencies before computing the merges.

In [5], we proposed a framework for merging incomplete and inconsistent graph-based views. We introduced a formalism, *annotated graphs*, which incorporates a systematic annotation scheme capable of modeling incompleteness and inconsistency. We use structure-preserving maps to capture relationships between views, modeled as annotated graphs. Using ideas from category theory, we provided a general algorithm for merging views w.r.t. arbitrary view interconnections. In this paper, we describe our implementation, and illustrate it with a simple schema merging example.

## 2 Methodology

We augment views with an annotation scheme to model incompleteness and inconsistency: we annotate each view element to denote the *degree of knowledge* available about the element. The annotations are drawn from a *knowledge order*, which is a partially ordered set specifying the levels of knowledge, and the possible ways in which this knowledge can grow. A useful knowledge order is Belnap's 4-valued logic [2]. Figure 1 presents a variant of this, $\mathcal{K}$. Labeling an element with ! means that the element has been

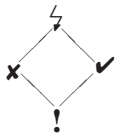

**Figure 1. The knowledge order $\mathcal{K}$**

*proposed* but it is not known if the element is indeed well-conceived; ✘ means the element is ill-conceived and hence *repudiated*; ✔ means the element is well-conceived and hence *affirmed*; and ϟ means there is disagreement as to whether the element is well-conceived, i.e. the element is *disputed*. An upward move in $\mathcal{K}$ denotes growth in the amount of knowledge: ! denotes inconclusiveness; ✘ and ✔ denote the desirable (i.e. conclusive) amounts of knowledge; and ϟ denotes an inconsistency, i.e. too much knowledge – we can infer something is both ill-conceived and well-conceived.

To merge a set of views, we need to know how they are interconnected. To express the relationships between views, we use structural mappings. For graph-based views, a natural choice for such mappings is graph homomorphism – a structure-preserving map describing how one graph is embedded into another. To have sound embeddings, we need to ensure that knowledge is preserved across mappings. For example, if we have already decided an element in a view is *affirmed*, it cannot then be embedded in another view such that it is reduced to just *proposed*, or is changed to a value not comparable to *affirmed* (i.e. *repudiated*).

Our view merging algorithm is based on a category-theoretic concept called *colimit* [1]. Given a family of interconnected views, computing the colimit yields a new view combining the given views w.r.t. their relationships as declared by the mappings.

## 3 Implementation

We have implemented a Java tool, called *iVuBlender*, for merging requirements views. The tool consists of two components: (1) a merge library for annotated graphs; and (2) a Swing-based front-end that allows users to graphically express their views, specify the view relationships, and compute merges. We have used the merge library for merging $i^*$ models, ER diagrams, and state-machines. Currently, the front-end only supports ER diagrams and is restricted to just the knowledge order in Figure 1 for annotating view elements. For future versions, we plan to develop a larger collection of graphical widgets to capture the visual syntax

of richer modeling formalisms such as $i^*$, and provide support for defining arbitrary knowledge orders.

## 4 Example

Figures 2 through 5 illustrate an example in which a pair of database schemata, `modelA` and `modelB`, are merged w.r.t. their overlaps as specified by a third schema `modelC`. First, the three schemata are created. Figure 2 shows one of these schemata (`modelB`). The color of each element represents the annotation for the element: if not colored, the element is at the proposed level; if colored blue, it is affirmed; if colored magenta, it is repudiated; and if colored red, it is disputed. For simplicity, we have only used the proposed and affirmed levels in our example.

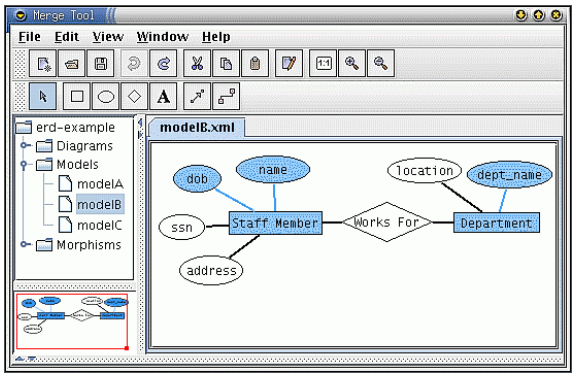


**Figure 2. View delineation**

In the second step, the desired mappings between the schemata are declared. In this example, two mappings, `mapCA` and `mapCB`, specify how the common part `modelC` is represented in each of `modelA` and `modelB`. Interconnections between the schemata are hypothesized using an interconnection diagram, as shown in figure 3. To specify each mapping, the respective source and target schemata are shown side-by-side. An element correspondence is established by first clicking on a source schema element and then on the corresponding target schema element. Figure 4 shows how the `Employee` element from `modelC` is mapped to the corresponding element in `modelA`.

The final step is to compute the merge – a new schema expressing the union of `modelA` and `modelB`, such that their overlap, `modelC`, is included only once. The result is shown in Figure 5. To ensure that the computed merge has a proper layout, a fully automatic layout algorithm is applied to it before it is presented to the user.

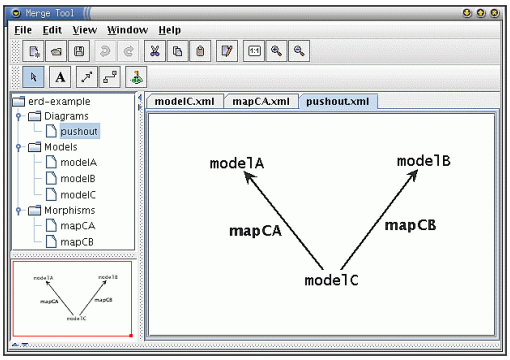


**Figure 3. View Interconnections**

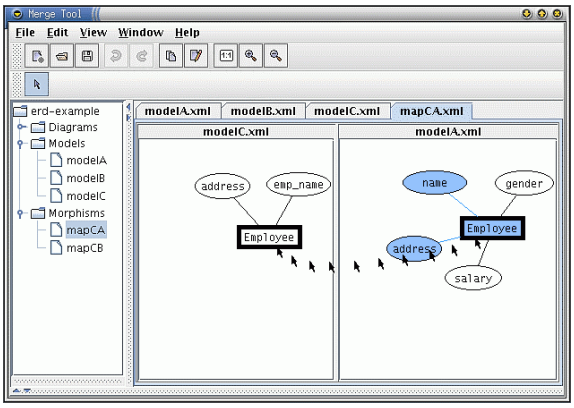


**Figure 4. Creating a mapping**

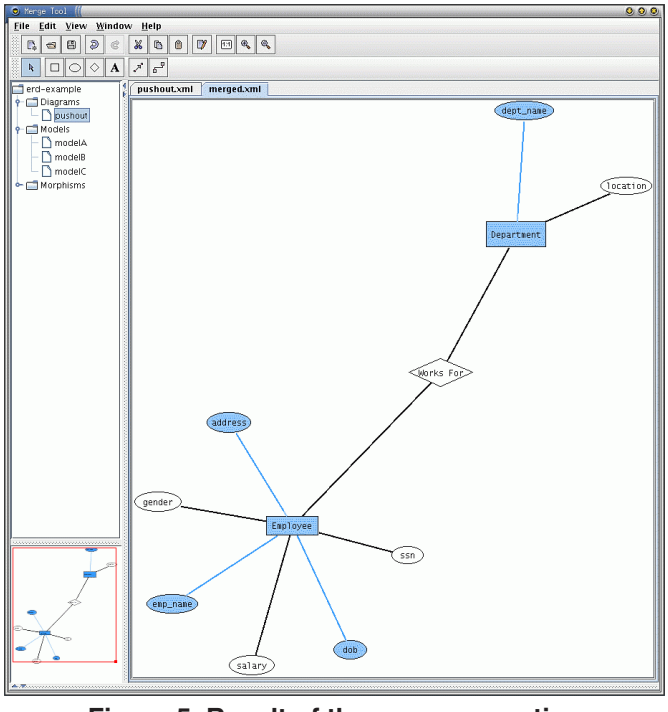


**Figure 5. Result of the merge operation**

## References


[1] M. Barr and C. Wells. *Category Theory for Computing Science*. Les Publications CRM Montréal, third edition, 1999.

[2] N. Belnap. A useful four-valued logic. In *Modern Uses of Multiple-Valued Logic*, pages 5–37. Reidel, 1977.

[3] H. Ehrig et al. A combined reference model- and view-based approach to system specification. *Intl. J. of Software Eng. and Knowledge Eng.*, 7(4):457–477, 1997.

[4] R. Pottinger and P. Bernstein. Merging models based on given correspondences. In *VLDB*, pages 862–873., 2003.

[5] M. Sabetzadeh and S. Easterbrook. An algebraic framework for merging incomplete and inconsistent views. In this proceedings.

[6] M. Sabetzadeh and S. Easterbrook. Analysis of inconsistency in graph-based viewpoints. In *ASE*, pages 12–21, 2003.