Traceability in Viewpoint Merging: A Model Management Perspective

Mehrdad Sabetzadeh Steve Easterbrook
Department of Computer Science
University of Toronto
40 St. George Street
Toronto, ON, Canada M5S 2E4
{mehrdad,sme}@cs.toronto.edu

ABSTRACT

Viewpoint merging is one of the core activities in viewpoints-based development. We may consolidate a set of viewpoints to unify different stakeholders' perspectives, to explore interactions between different parts of a problem, or to perform various types of analysis. Once viewpoints are merged, it is important to be able to determine how the merged viewpoint represents each viewpoint, and to track the assumptions involved in the merge. Building on the viewpoint merging framework in our earlier work [22], this paper proposes a systematic way to generate and represent the traceability information required for tracing the merged viewpoint elements back to their originating viewpoints, and to the merge assumptions related to the elements.

Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirements/Specifications

General Terms

Requirements Engineering, Model Management

Keywords

Requirements Elicitation, Viewpoint Merging, Traceability

1. INTRODUCTION

Viewpoints have been proposed as a way to structure and manage the process of requirements elicitation and specification. Viewpoints may be employed to specify different aspects of a problem, model competing perspectives on a single aspect, or describe various concerns as to how different problem constituents can interact. By separating the descriptions provided by different stakeholders, viewpoints make it possible to distinguish between the conceptual contributions of individual sources [6].

During requirements elicitation, viewpoints may be manipulated in various ways. Model Management techniques [15] aim to structure these manipulations using a number of generic operations. Some of the most notable of these operations are *merging*, *matching*, and *differencing* [3, 2, 15]. After a manipulation is carried out, we often need to know how the input artifacts to the manipulation process participated in producing the result. To achieve a satisfactory solution to this, operations must be capable of establishing proper traceability links between their output and their operands.

This paper is an attempt to devise a traceability mechanism for the merge operation. Viewpoint merging, also known as viewpoint integration, is regarded as one of the core activities in viewpoints-based development [5]. Large models are often constructed and accessed by manipulating individual viewpoints, but it is important to be able to integrate a set of viewpoints to gain a unified perspective, to explore interactions between different parts of a problem, or to perform various kinds of end-to-end analysis [22]. The need for traceability in viewpoint merging arises in at least two respects:

- *Origin traceability*: After a merge is completed, we may want to know where each element of the merged viewpoint came from. If it is important to capture individual contributions using separate viewpoints, then, it must be equally important to keep track of how these contributions are incorporated into the merged viewpoint.

- *Assumption traceability*: Requirements elicitation is an exploratory process where we can never be completely sure how concepts expressed in different viewpoints are related. Each merge is hypothesized based on a set of assumptions describing how the viewpoints relate to one another. If a particular set of assumptions results in an unacceptable merge, it may be because we made poor assumptions, or because there is an inconsistency between the given viewpoints. In either case, we need to be able to trace the unacceptable structures in the merged viewpoint back to the assumptions involved in creating them.

In our previous work [21, 22], we proposed a framework for merging incomplete and inconsistent graph-based viewpoints. We used structure-preserving maps to capture the relationships between viewpoints, and provided a general algorithm for merging viewpoints w.r.t. a given set of interrelations. We demonstrated the usefulness of our approach using examples drawn from the areas of formal verification and conceptual modeling.

In this paper, we discuss how origin and assumption traceability links can be derived during the merge process. To infer and unify viewpoint overlaps in a merge problem, our merge algorithm uses a construct which we refer to as a *unification graph* [22]. We show how this construct can be employed to automatically generate the required traceability information. Our proposed traceability mech-

anism is very general and scales to any number of viewpoints with arbitrary interrelations. Further, it does not rely on any formalism-specific information, and hence can be adapted to different modeling languages.

The problem of inconsistency management discussed in our earlier work is orthogonal to the traceability concerns addressed in this paper. To support incompleteness and inconsistency, we proposed knowledge annotations as a means to capture stakeholders' beliefs about viewpoint elements, and how these beliefs may evolve. Although not directly related to the main theme of this paper, knowledge annotations will be introduced briefly in Section 3 to provide the machinery we need for giving a non-trivial and yet meaningful viewpoint merging example.

2. RELATED WORK

Over the years, the term "viewpoint" has appeared in the literature with several different meanings. Viewpoints have been used to mean different classes of users [20], the contexts in which different roles are performed [6], to distinguish between stakeholder terminologies [24], and to encapsulate knowledge about a system into loosely-coupled objects [8]. A survey and comparison of the existing viewpoints-based approaches can be found in [5]. Our interpretation of viewpoints falls closely in line with the emerging trends in model management where viewpoints are employed to capture conceptual data gathered from disparate sources into *independent* but *interrelated* units. To describe viewpoint interrelations, explicit mappings must be defined between them [2].

The ability to trace requirements back to their human sources is one of the most important traceability concerns in software development [12]. To this end, *contribution structures* [11] have been proposed as a way to facilitate cooperative work among teams and to ensure that the contributions of involved parties are properly accounted for throughout the entire development life-cycle. The notion of origin traceability in our work tries to address a similar problem in the context of viewpoint merging by providing support for tracing the merged viewpoint elements back to the viewpoints where they originated.

The importance of establishing traceability links between artifacts and the assumptions involved in creating them has been emphasized in design rationale [9, 14] and design traceability [7]. However, the focus of the work has been mainly on assumptions that relate upstream and downstream artifacts. Our work, instead, focuses on requirements elicitation which is an entirely upstream activity. We discuss the nature of the relationships between viewpoints produced during elicitation, and propose an approach for keeping track of how each assumption made about viewpoint interrelations affects the merge.

Viewpoint merging is an interdisciplinary subject which has been studied in several areas including Databases where it is known as schema merging [4, 19, 16], Software Engineering [17], Requirements Engineering [21, 25], and The Semantic Web [18]. None of these, however, address traceability specifically. The general problem of composing different aspects of a system and determining how individual aspects affect the overall system has been noted in [13]. But, the work focuses on a very specific issue which is tracking how global quality concerns are addressed during composition – a subject which is orthogonal to what we discuss in this paper.

In our earlier work [22], we proposed a notion, called *stakeholder traceability*, which, in essence, is similar to origin traceability discussed in this paper. The motivation behind this notion was to make it possible for multiple stakeholders to contribute to a single viewpoint. Traceability was limited to finding which stakeholders contributed to each element of the merged viewpoint irrespective of *where* the contributions were made. In the traditional viewpoints-based approaches, a stakeholder can have several viewpoints; however, each viewpoint typically belongs to just one stakeholder. Consequently, tracing the merged viewpoint elements back to the viewpoints where they came from is potentially more useful. This is what we address by introducing origin traceability in this paper. The constructs we use to implement origin traceability and those we used in [22] to implement stakeholder traceability are independent; therefore, if necessary, both traceability notions can be used simultaneously without causing any interference to the other.

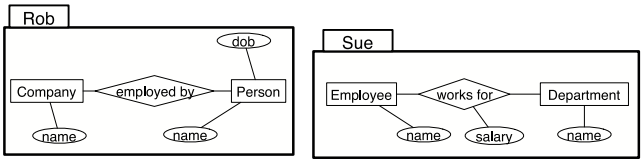

Figure 1: Initial stakeholders' viewpoints

3. MOTIVATING EXAMPLE

To highlight the traceability problem in viewpoint merging, consider the following example: Rob and Sue want to gather and consolidate the requirements for a payroll database. An analyst, Jack, will help them with requirements elicitation and identification of interrelations between their perspectives. Viewpoints are captured using Entity Relationship Diagrams (ERD's, for short). To model stakeholders' beliefs, an annotation is attached to each viewpoint element:

1. **!** : used when an element is *proposed* but is not yet known to be appropriate (or inappropriate) for sure;
2. ✔ : used when an element is conclusively *appropriate*;
3. ✘ : used when an element is conclusively *inappropriate*.

For convenience, "proposed" (**!**) is treated as a default annotation for all viewpoint elements, and only the remaining annotation values are shown.

Once Rob and Sue provide their initial viewpoints (Figure 1), Jack will merge them to create a unified perspective. To carry out the merge, Jack needs to specify how Sue's and Rob's viewpoints are related. Initially, the two viewpoints may appear to be non-overlapping because of terminology differences; but, after further analysis, Jack identifies some straight-forward correspondences: Employee in Sue's viewpoint is probably the same entity as Person in Rob's. Consequently, the name attribute of Employee would be the same as that of Person. To describe the correspondences, Jack creates a new viewpoint, **Connector1** (Figure 2), containing only the elements that are in common between Rob's and Sue's viewpoints. He then chooses appropriate names for the elements in **Connector1** and specifies how the viewpoint is embedded into each of the stakeholders' viewpoints (Figure 2). We usually refer to shared viewpoints as *connectors* because they are used to describe the correspondences between other viewpoints. Notice that even if Rob used the term Employee instead of Person in his viewpoint, defining a connector would still be necessary because our merge framework does not rely on naming conventions to describe the desired unifications – all correspondences must be identified explicitly prior to the merge operation.

Merging Rob's and Sue's viewpoints w.r.t. **Connector1** yields a viewpoint like Figure 3. For naming the elements of this viewpoint, we have assumed that the naming choices in the connector (which

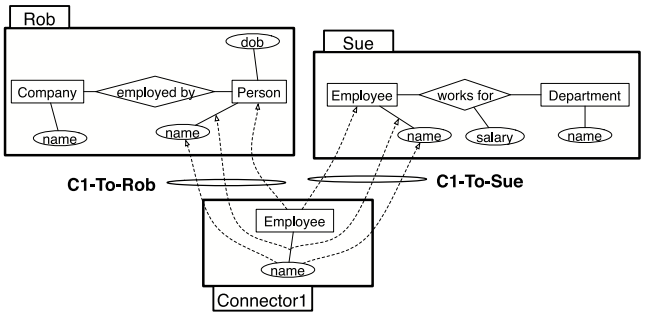

Figure 2: Viewpoint interrelations

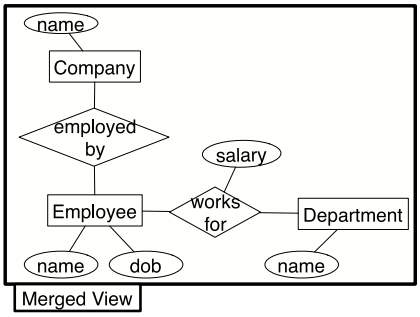

Figure 3: The merged viewpoint

happen to favor Sue in this example) take precedence over those in the stakeholders' viewpoints.

The merge in Figure 3 may be sufficient from an operational point of view (e.g. to produce a SQL script for creating a database), but as we argued in the introduction, we often need the flexibility to distinguish between the *origins* of different contributions after the merge is completed. Obviously, the above merge lacks the traceability information required for determining where each of its elements came from: we cannot tell which elements appeared only in Rob's viewpoint, which appeared only in Sue's, and which were shared among both stakeholders, and hence appeared in all viewpoints including **Connector1**.

For a simple merge scenario like the one outlined above, knowing where the elements of the merged viewpoint are coming from might be our only traceability concern. In this scenario, we did not have to worry about keeping track of the *assumptions* about correspondences between the viewpoints: all assumptions were localized to the mappings **C1-To-Rob** and **C1-To-Sue**. Therefore, if we later needed to check why, for example, Person in Rob's viewpoint was unified with Employee in Sue's, we could easily find the chain of correspondences that brought about the unification: Person(**Rob**) = Employee(**Connector1**) by **C1-To-Rob**, and Employee(**Connector1**) = Employee(**Sue**) by **C1-To-Sue**.

As viewpoints evolve over time and merge scenarios become more complex, identifying the assumptions behind each unification may be no longer trivial. To illustrate this, consider the following: When the above merge (Figure 3) is shown to Sue, she notices Company, an entity she had not discovered in her original viewpoint. She decides to add the entity to her viewpoint, but, she prefers to call it Corporation. She also adds an aggregation link from Corporation to Department to relate the two entities. Further, she deems the employed by relationship inappropriate in the light of the existence of the works for relationship in her viewpoint. To capture all this, she creates an *evolution* of her original viewpoint, **Sue Evolved** (Figure 4); and, with the help of Jack, establishes the required correspondences through a new connector, **Connector2**,

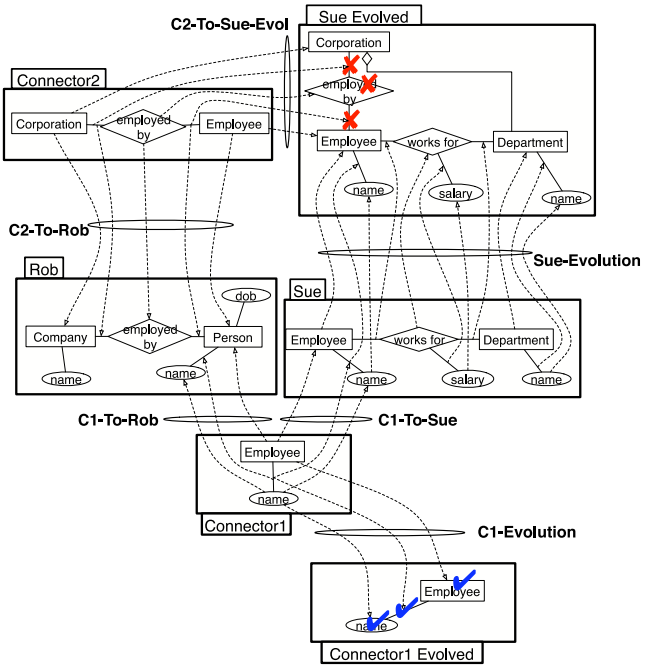

Figure 4: New viewpoint interrelations

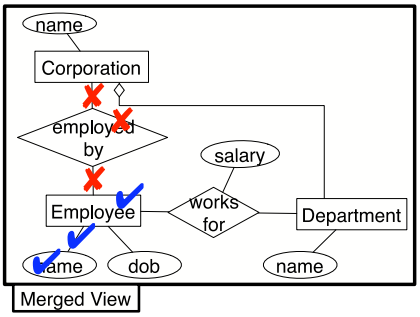

Figure 5: The new merged viewpoint

and two mappings, **C2-To-Rob** and **C2-To-Sue-Evol** (Figure 4). At the same time, Jack, who has now become certain about the correctness of the elements in **Connector1**, wants to confirm his beliefs by evolving **Connector1** into a conclusive state in which elements are annotated with ✔. He does so by introducing a new viewpoint **Connector1 Evolved**. The merge scenario and the resulting merged viewpoint are respectively shown in Figures 4 and 5.

As can be seen in Figure 4, the assumptions about correspondences between viewpoints are scattered among several mappings. For example, the unification of Company in **Rob** and Corporation in **Sue Evolved** involves **C2-To-Rob** and **C2-To-Sue-Evol**; and the unification of Person's name attribute in **Rob** and Employee's name attribute in **Sue Evolved** involves **C1-To-Rob**, **C1-To-Sue** and **Sue-Evolution**. More interestingly, the unification of Employee in **Rob** and Employee in **Sue Evolved** can be traced to two different correspondence chains, the first involving **C1-To-Rob**, **C1-To-Sue** and **Sue-Evolution**; and the second involving **C2-To-Rob** and **C2-To-Sue-Evol**.

To be able to keep track of the correspondence assumptions involved in each unification, we need to know the details of the interrelations among the input viewpoint elements that are unified to form an element of the merged viewpoint.

The traceability mechanism that we propose in this paper addresses both origin and assumption traceability by generating the required traceability links during the merge process.

4. OUR APPROACH

Our viewpoint merging framework hinges on three abstractions: *viewpoints*, *mappings*, and *interconnection diagrams*.

A *viewpoint* is delineated by a directed graph whose elements are annotated with values denoting the stakeholders' beliefs about the elements. We have already seen several examples of viewpoints in Section 3. In [22], we describe how stakeholders' beliefs can be formalized using *partially ordered sets*, and how capturing these beliefs allows for toleration of incompleteness and inconsistency. There, we also show how viewpoints can be equipped with a typing mechanism for differentiating between different kinds of nodes (e.g. ERD attributes, entities, etc.), and different kinds of edges (e.g. ERD has-a, is-a, etc.) in the modeling formalism being used.

A *mapping* expresses an admissible way to interconnect a pair of viewpoints by showing how the contents of one viewpoint potentially map onto those of another. Mappings preserve the graphical structure of viewpoints and respect stakeholders' belief annotations. A detailed treatment of the constraints that viewpoint mappings should satisfy can be found in [22]. In our motivating example in Section 3, **C1-To-Rob**, **C1-To-Sue**, **Sue-Evolution**, **C2-To-Rob**, **C2-To-Sue-Evol**, and **C1-Evolution** are all valid mappings.

An *interconnection diagram* articulates a merge hypothesis, and is given by a set of viewpoints and a set of mappings between them. We describe merge hypotheses explicitly because: Firstly, we may want to do merges in which only a subset and not necessarily all of the existing viewpoints participate. This requires that we specify which viewpoints are involved in each merge. Secondly, we may have several competing versions of mappings between the participating viewpoints. This makes it necessary to specify explicitly which mappings are to be used for interconnecting the viewpoints. Figures 6(a) and 6(b) respectively show the interconnection diagrams for Figures 2 and 4. If we later chose to repeat the merge scenario of Figure 4 without capturing the evolution of **Connector1**, we would hypothesize the merge using the interconnection diagram shown in Figure 6(c).

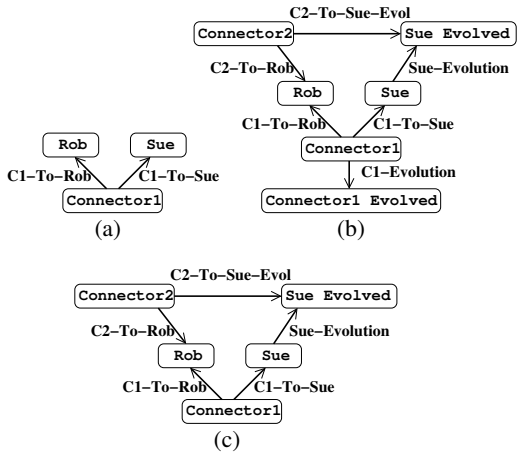

Figure 6: Interconnection diagrams

Our viewpoint merging algorithm is based on an algebraic concept called *colimit* [1]. Given an interconnection diagram, computing the colimit results in a new viewpoint combining the viewpoints in the diagram w.r.t. their interrelations as described by the mappings in the diagram. The general intuition behind colimits is that they put structures together with nothing essentially new added, and nothing left over [10]. This principle works irrespective of the details of viewpoints and mappings.

SET-MERGE $(S_1, \ldots, S_n, f_1, \ldots, f_k)$:

- ▷ Let U be an initially discrete graph with node-set $S_1 \uplus \ldots \uplus S_n$;
- ▷ For every function $f_i \quad (1 \leq i \leq k)$:
  - ▷ For every element a in the domain of f_i:
    - ▷ Add to U an undirected edge between the elements corresponding to a and $f_i(a)$;
- ▷ Let P be the set of the connected components of U;
- ▷ Return P as the result of the merge operation.

Figure 7: Algorithm for merging sets

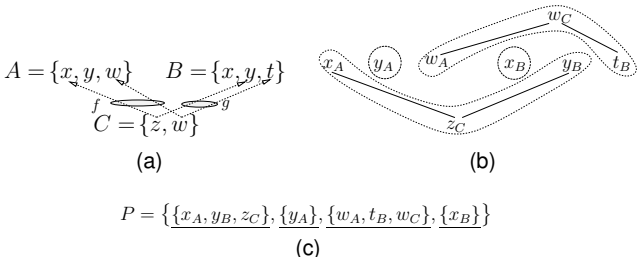

Figure 8: Set merging example

At the core of our viewpoint merging algorithm is a simple algorithm for merging sets. Once equipped with this set-merging algorithm, we can merge viewpoints by treating each as being composed of a node-set and an edge-set; and treating each viewpoint mapping as being composed of a node-set mapping and an edge-set mapping [21]. Belief annotations for the elements of the merged viewpoint are computed by a separate algorithm which is not needed for the purposes of this paper, and hence not discussed here (see [22] for details).

To merge a family of interrelated sets, we start with the disjoint union of the sets as the largest possible merged set, and refine it by grouping together elements that get unified by the interrelations. We construct the disjoint union by subscripting the elements of each given set with the name of the set and then taking the union. To identify which elements should be unified, we construct a *unification graph*, a graphical representation of the symmetric binary relation induced on the elements of the disjoint union by the interrelations. We then combine the elements that fall in the same connected component of the unification graph. Figure 7 shows the general algorithm for merging a family of sets $S_1, \ldots, S_n$ interconnected by functions $f_1, \ldots, f_k$.

Figure 8 shows an example of merging sets: 8(a) shows the interconnection diagram; 8(b) shows the induced unification graph and its connected components; and 8(c) shows the merged set.

To assign a name to each element of the merged set in Figure 8(c), we combined the names of all the elements in A, B, and C that are mapped to it. For example, "$\{x_A, y_B, z_C\}$" indicates an element that *represents* x of A, y of B, and z of C. A better way to name the elements of the merged set is assigning *naming priorities* to the input sets. As we noted in Section 3, it usually makes sense to give connectors a higher priority in determining the names of the elements in the merge. In this example, the merged set would be written as $P = \{z_C, y_A, w_C, x_B\}$ if we gave C a higher priority.

Another technicality worth mentioning is that in the example shown in Figure 8, the elements in each set are uniquely identifiable by their names. This is not necessarily the case in general because we may have unnamed or identically-named, but distinct

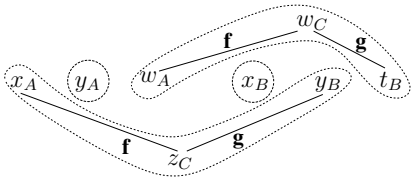

Figure 9: Extended unification graph

elements. For example, in Section 3, the edges in all viewpoints were unnamed and the "name" node appeared more than once in the node-sets of each **Rob**, **Sue**, **Sue Evolved**, and the merges in Figures 3 and 5. To avoid ambiguity, our implementation of the merge framework [23] uses *Global Identifiers* (GId's) rather than names to differentiate between the elements of a viewpoint.

Unification graphs, as introduced earlier in this section, immediately yield the required origin traceability links: For a given merge problem, the set of nodes in each connected component of the unification graph constitutes the origin information for the corresponding merged element.

However, the current notion of unification graph cannot be used to find the correspondence chain(s) involved in creating each merged element. This is because we did not keep track of *which* mapping induced each of the edges in the unification graph. To address this problem, we label each edge in the unification graph with the name of the mapping that gave rise to it. Figure 9 shows the extended unification graph for the merge problem in Figure 8. This extended construct embodies the required origin information as well as the mapping assumptions behind each merged element.

We can now demonstrate how to merge the ERD's in the motivating example and simultaneously derive the required traceability information. Figure 10 shows the (extended) unification graph for merging the node-sets of the viewpoints in Figure 4. Each connected component in this graph corresponds to one node in the merged viewpoint shown in Figure 5. As an example, we have explicitly shown in Figure 10 the connected component corresponding to the name attribute of Employee.

To support traceability, we store in each merged viewpoint element a reference to the connected component that gave rise to the element. Figures 11(a)–(c) respectively show the traceability information that we keep for three representative elements of the merged viewpoint: Corporation, Employee, and Employee's name. In each case, the stored traceability information makes it possible to trace the merged viewpoint element back to its origins, and to find the assumptions involved in the corresponding unification. If we later want to see why, for example, Employee in **Sue Evolved** was unified with Person in **Rob**, we find the *paths* between Employee (**Sue Evolved**) and Person (**Rob**) in Figure 11(b). There are two paths connecting the two elements, one involving **C2-To-Sue-Evol** and **C2-To-Rob**; and the other involving **Sue-Evolution**, **C1-To-Sue**, and **C1-To-Rob**. Each path is a correspondence chain unifying the two elements.

To avoid clutter, we chose not to show the element GId's in Figure 11. However, we should emphasize that GId's are kept as part of the traceability information to alleviate ambiguity – as stated earlier, an element may not be uniquely identifiable by its name.

5. TOOL SUPPORT AND DISCUSSION

We have implemented a Java tool, *iVuBlender*, for merging viewpoints. The tool consists of a generic merge library, and a front-end that allows users to graphically express their viewpoints, specify viewpoint interrelations, and compute merges. We have used the

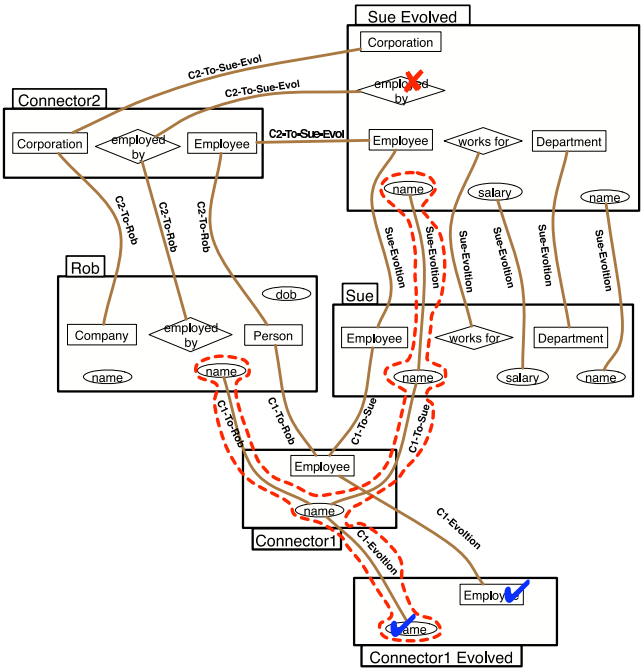

Figure 10: Unification graph for the node-sets in Figure 4

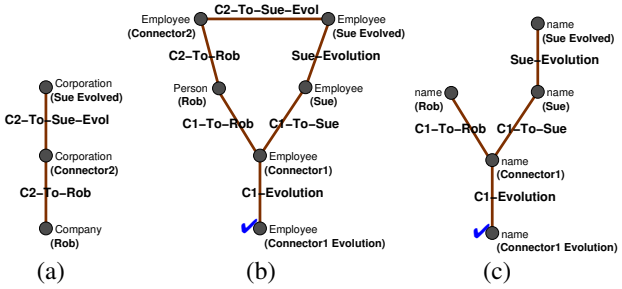

Figure 11: Examples of traceability information

merge library for merging various graph-based notations including ERD's, state-machines, and requirements goal models. A brief description of the tool can be found in [23]. We are now extending our merge library so that it can store the required traceability information in the merges. We are also looking at ways to visualize the traceability information in the front-end, and to provide support for viewpoint-to-viewpoint navigation based on this information.

Recently, we asked a number of subjects to independently conduct uncontrolled experiments on *iVuBlender*. We also used the tool in a number of controlled experiments each involving multiple subjects. The desire to trace the elements of the merged viewpoint back to their sources was raised as a concern by several subjects in both types of experimentation. The problem was more pressing in our controlled experiments where the involved parties had no line-of-sight to other parties' viewpoints. The issue of tracking down the merge assumptions was raised only by a few of the subjects who already had some background in viewpoint merging. None of the involved subjects were given any prior information about the intent of our study.

Our study yielded another interesting observation: some subjects had difficulty matching their cognitive image of their contributions to what they actually saw in the merged viewpoint. This was mainly because, in most cases, the merged viewpoint turned out to be much more detailed than the individual subjects' view-

points. A second and less obvious problem was that the subjects found the automatically-generated layout of the merged viewpoint significantly different from that of their own. We anticipate that addressing both of these problems will rely on storing appropriate traceability links in the merges.

6. CONCLUSION

We discussed the problem of traceability in viewpoint merging. We considered two different traceability notions, one for tracing the elements of the merges back to their sources, and another for tracking down the viewpoint interrelation assumptions behind each unification. We called the former notion *origin traceability* and the latter *assumption traceability*. We proposed a systematic approach to generating and representing the information required for supporting these notions. A major advantage of our approach is that it is independent of the details of the modeling language being used and hence, can be applied to various modeling notations.

The work reported here can be carried forward in many ways. Our ongoing work includes extending our viewpoint merging tool to generate and visualize traceability links, and to support model-to-model navigation using these links. We are also conducting further experiments to surface other potential concerns regarding traceability in viewpoint merging. Our future work includes adding support for adaptive filtering and layout of the merges based on available traceability data and stakeholders' preferences.

Acknowledgments. We thank Shiva Nejati for helpful discussions and careful comments on an earlier draft of this paper. Financial support was provided by NSERC, Bell University Labs (BUL), and MITACS.

7. REFERENCES

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

[2] P. Bernstein. Applying model management to classical meta data problems. In *Proceedings of the 1st Conference on Innovative Data Systems Research*, 2003.

[3] P. Bernstein, A. Halevy, and R. Pottinger. A vision for management of complex models. *SIGMOD Record*, 29(4), 2000.

[4] P. Buneman, S. Davidson, and A. Kosky. Theoretical aspects of schema merging. In *Proceedings of the 3rd International Conference on Extending Database Technology*, 1992.

[5] P. Darke and G. Shanks. Stakeholder viewpoints in requirements definition: A framework for understanding viewpoint development approaches. *Requirements Engineering*, 1(2), 1996.

[6] S. Easterbrook. Domain modeling with hierarchies of alternative viewpoints. In *Proceedings of the 1st International Symposium on Requirements Engineering*, 1993.

[7] A. Egyed. A scenario-driven approach to traceability. In *Proceedings of the 23rd International Conference on Software Engineering*, 2001.

[8] A. Finkelsetin, J. Kramer, B. Nuseibeh, L. Finkelstein, and M. Goedicke. Viewpoints: A framework for integrating multiple perspectives in system development. *International Journal of Software Engineering and Knowledge Engineering*, 2(1), 1992.

[9] G. Fischer, A. Lemke, R. McCall, and A. Morch. Making argumentation serve design. *Design rationale: concepts, techniques, and use*, 1996.

[10] J. Goguen. A categorical manifesto. *Mathematical Structures in Computer Science*, 1(1), 1991.

[11] O. Gotel and A. Finkelstein. Contribution structures. In *Proceedings of the 2nd International Symposium on Requirements Engineering*, 1995.

[12] O. Gotel and A. Finkelstein. Extended requirements traceability: results of an industrial case study. In *Proceedings of the 3rd International Symposium on Requirements Engineering*, 1997.

[13] D. Gross and E. Yu. Dealing with system qualities during design and composition of aspects and modules: an agent and goal-oriented approach. In *1st International Workshop on Traceability in Emerging Forms of Software Engineering*, 2002.

[14] T. Gruber and D. Russell. Generative design rationale: beyond the record and replay paradigm. *Design rationale: concepts, techniques, and use*, 1996.

[15] S. Melnik. *Generic Model Management: Concepts and Algorithms*, volume 2967 of *Lecture Notes in Computer Science*. Springer-Verlag, 2004.

[16] S. Melnik, E. Rahm, and P. Bernstein. Rondo: a programming platform for generic model management. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, 2003.

[17] T. Mens. A state-of-the-art survey on software merging. *IEEE Transactions on Software Engineering*, 28(5), 2002.

[18] N. F. Noy and M. Musen. Prompt: Algorithm and tool for automated ontology merging and alignment. In *Proceedings of the 7th National Conference on Artificial Intelligence*, 2000.

[19] R. Pottinger and P. Bernstein. Merging models based on given correspondences. In *Proceedings of the 29th International Conference on Very Large Data Bases*, 2003.

[20] D. Ross. Applications and extensions of SADT. *IEEE Computer*, 18(4), 1985.

[21] M. Sabetzadeh and S. Easterbrook. Analysis of inconsistency in graph-based viewpoints: A category-theoretic approach. In *Proceedings of the 18th International Conference on Automated Software Engineering*, 2003.

[22] M. Sabetzadeh and S. Easterbrook. An algebraic framework for merging incomplete and inconsistent views. In *Proceedings of the 13th International Requirements Engineering Conference*, 2005.

[23] M. Sabetzadeh and S. Easterbrook. iVuBlender: A tool for merging incomplete and inconsistent views. In *Proceedings of the 13th International Requirements Engineering Conference*, 2005. Tool Demo Paper.

[24] R. Stamper. Social norms in requirements analysis: an outline of measur. *Requirements engineering: social and technical issues*, 1994.

[25] S. Uchitel and M. Chechik. Merging partial behavioural models. In *Proceedings of the 12th International Symposium on Foundations of Software Engineering*, 2004.