Unity: Speeding the Creation of Community Vocabularies for Information Integration and Reuse

Ken Smith, Peter Mork, Len Seligman, Peter Leveille, Beth Yost, Maya Li, Chris Wolf
The MITRE Corporation
{kps, pmork, seligman, psl, bethyost, haoli, cwolf}@mitre.org

Abstract

Many data sharing communities create data standards ("hub" schemata) to speed information integration by increasing reuse of both data definitions and mappings. Unfortunately, creation of these standards and the mappings to the enterprise's implemented systems is both time consuming and expensive. This paper presents Unity, a novel tool for speeding the development of a community vocabulary, which includes both a standard schema and the necessary mappings. We present Unity's scalable algorithms for creating vocabularies and its novel human computer interface which gives the integrator a powerful environment for refining the vocabulary. We then describe Unity's extensive reuse of data structures and algorithms from the OpenII information integration framework, which not only sped the construction of Unity but also results in reuse of the artifacts produced by Unity: vocabularies serve as the basis of information exchanges, and also can be reused as thesauri by other tools within the OpenII framework. Unity has been applied to real U.S. government information integration challenges.

Keywords: data integration, vocabularies, user interfaces

1. Introduction

Large enterprises typically have many heterogeneous data sources, each of which may participate in multiple data integration efforts. The use of organizational or community data standards ("hub" schemata) can speed integration by increasing reuse of both data definitions and mappings. However, creation of such standards remains a largely manual process performed by highly skilled (and expensive) integration engineers; there are few existing tools to help. In addition, standards can suffer from irrelevance and disuse when their concepts do not coincide well (or the correspondence is not clear) with the concepts of existing community schemata.

This paper presents Unity, a novel tool for speeding the development of new data exchange standards and mappings, which ensures the new standard's concepts are well-connected to those of existing community schemata. We make the following contributions:

- Unity enables the creation of a *vocabulary*—i.e., a new standard schema, with mappings to existing schemata. It accomplishes this by extending schema matching technology: Unity performs an n-way match across existing schemata to infer the synonym sets (*synsets*) which form the basis of the vocabulary.
- Unity's novel user interface (UI) allows an integration engineer to perform human-in-the-loop refinement of automatically suggested synsets. Users can inspect and edit synsets concepts (and the "near misses") in multiple contexts and drill down to supporting evidence, a powerful improvement over current spreadsheet interfaces.
- Unity was implemented by reusing key software components available in the OpenII open source information integration framework [1]. There are two beneficial byproducts of this implementation strategy: 1) interoperability with other tools implemented in the same framework (e.g., Harmony [2] and Affinity [3]) and 2) the reuse of the vocabularies developed by Unity as thesauri to improve schema matching in Harmony.

The need for a tool like Unity arose from efforts to construct a vocabulary for communities of interest (COIs) in the U.S. Department of Defense, as described in [4]. A COI is a collection of parties whose information needs overlap significantly (e.g., all parties are concerned with meteorology), and who have an interest in sharing data. By constructing a vocabulary, a COI can more rapidly integrate and reuse information.

2. Vocabulary generation

2.1. Modeling preliminaries

We adopt a generic metamodel based on [5, 6] which allows us to ignore structural peculiarities across modeling languages. For example, we can generate a

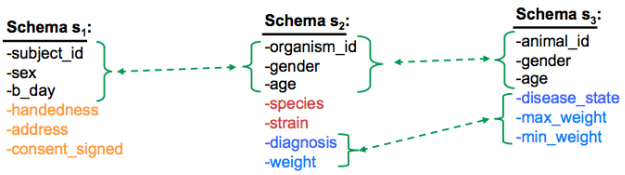

Figure 1: Sample schemata with inter-schema mappings

vocabulary for an exchange effort that includes both relational databases and XML messages.

In our metamodel a schema consists of *entities* (e.g., relations, classes, objects), *properties* of entities (e.g., columns, attributes), and *relationships* among entities (e.g., containment to support XML and other hierarchical models).

For the purposes of vocabulary generation, we further abstract the entities, properties and relationships in a schema into a set of *concepts*. Each concept consists of an identifier and a name: $c =< id, name >$. For example, schema s_1 in Figure 1 includes six concepts (only the concept names are shown), which are properties of a medical test subject entity.

To facilitate schema integration, our metamodel also includes mappings as first-class objects. A mapping is a binary relationship between the concepts in two schemata: $M_{s_i \leftrightarrow s_j} \subseteq s_i \times s_j$. Each mapping entry indicates a semantic correspondence between a concept in s_i and a concept in s_j. For example, in Figure 1, $M_{s_1 \leftrightarrow s_2}$ contains three entries, one of which indicates a correspondence between the concepts b_day and age.

2.2. Vocabulary definition and overview

The goal of Unity is to enable the rapid creation of a simple, yet powerful knowledge structure called a *vocabulary*. As illustrated in Figure 1, the schemata in a community often contain common concepts, which correspond to each other semantically, even though they may be designated differently in each schema. The degree to which concepts span schemata can vary widely, ranging from concepts common to *all* schemata in the set (e.g., gender in Figure 1) to concepts found in only a single schema (e.g., species).

Given a set of participants that needs to reuse and share information, let S be the set of individual schemata ($s_1 - s_3$ in Figure 1). We define a vocabulary for S as $V =< C, M >$ where:

- C is a new canonical schema that contains the union of all the unique concepts in S.
- M is a set of mappings that define the concept-level correspondences between the concepts in $s_i \in S$ and the concepts in C.

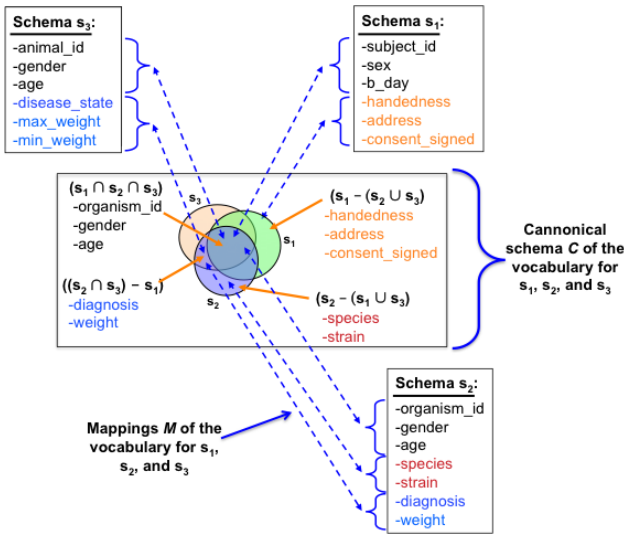

Figure 2: Sample vocabulary including a canonical schema and mappings that relate existing schemata to the canonical schema.

For example, in Figure 2, C contains ten concepts and M contains three mappings: $m_{s_1 \leftrightarrow C}$, $m_{s_2 \leftrightarrow C}$, and $m_{s_3 \leftrightarrow C}$. The first mapping contains six correspondences including (subject_id, organism_id) and (handedness, handedness).

Vocabularies enable the sharing and reuse of a group's information because they provide a "hub" for information exchanges. Unlike externally-defined standards, which may correspond poorly to a group's concepts, a vocabulary is *inductively* created from the schemata of the potential sharing participants (which could include an external standard), provides a shared language for that group, and pre-computes the mappings necessary to engineer any data exchange in that group. Thus, the cost, or "activation energy" of information reuse is significantly reduced by the presence of a vocabulary. In addition to facilitating information reuse, a vocabulary also provides a descriptive inventory of any group's shared concepts and is a useful tool for improving their alignment.

Generating a knowledge structure like a vocabulary (e.g., an information exchange standard) normally involves a lengthy manual process lasting from weeks to years. Unity is a novel system, based on schema matching and information visualization technologies, for rapidly and semi-automatically generating a vocabulary.

As shown in Figure 3, Unity generates a vocabulary in two stages. Unity first runs schema matching algorithms to compute a set of binary correspondences among the schemata in S. Each line in Figure 3B represents a mapping between a pair of source schemata, consisting of individual concept-to-concept correspondences. Second, Unity aggregates these individual correspondences across

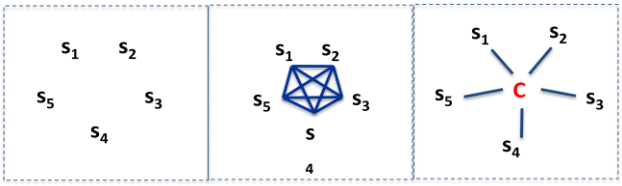

Figure 3: Example of vocabulary generation with five participants. A: Initial set of participants. B: Generation of source concept mappings using schema matching. C: Vocabulary generation.

all schemata in S to compute the set of concepts in C, retaining the correspondences back to the concepts in S as M. Each line in Figure 3C represents an element of M. As discussed in Section III, expert users can intervene in either step through an interactive visualization. For example, a subject matter expert could interactively refine the initial set of element level correspondences between schemata, prior to generating the vocabulary.

2.3. Algorithms for generating vocabularies

Given S, the first step is to compute a set of pair-wise semantic mappings among the *source concepts* (i.e., concepts in the context of their schema-of-origin) in S. For any two source concepts x and y, we generate a mapping $(x, y, sim(x, y))$ where $sim(x, y)$ denotes the similarity between concepts x and y. Similarity scores are determined using conventional schema matching techniques [7], such as: the edit distance between concept labels, the similarity between textual concept descriptions, the structural similarity of each concept's schematic context, and similarity via synonymy (based on a thesaurus). A similarity score of +1 indicates a perfect correspondence; a score of –1 indicates no possible correspondence between x and y; and a score of 0 indicates that we cannot discriminate between a correspondence and a non-correspondence on the basis of the available evidence.

By default, we generate all possible mappings between pairs of schemata, but the user can override this default by specifying any superset of a spanning tree. Each mapping is generated by invoking standard schema-matching algorithms [7] for every pair of source schemata for which a mapping does not already exist. Let us assume that no mappings exist and that the user has accepted the default behavior. If each schema contains n source concepts, then the complexity of the matching operation is $O(|S|^2 \times n^2) = O(m^2)$ where $m = |S| \times n$, the number of source concepts in S.

The second step is to derive the set of *vocabulary concepts*, which form C. We accomplish this task by repeatedly merging source concepts into clusters (i.e., *synsets*). Clustering algorithms rely on a metric for distance; we derive the distance between pairs of concepts as follows:

$$distance(x, y) = \begin{cases} \infty & if\ x \in s_i, y \in s_i \\ \infty & if\ sim(x, y) < 0 \\ 1 - sim(x, y) & else \end{cases}$$

Thus, when two concepts are in perfect correspondence, the distance between them is zero. Smaller correspondences generate larger distances. We set the distance to ∞ when the similarity is negative or the concepts are in the same schema to prevent those concepts from appearing in the same cluster.

Merging source concepts into clusters (based on a distance matrix) can be performed using any agglomerative clustering algorithm. Unfortunately, given m source concepts, most clustering algorithms have a computational complexity of $O(m^3)$.

For realistically scaled problems, this clustering step simply takes too much time. Instead, we utilize a practical optimization based on disjoint-set forests that reduces the complexity of this second step to below that of the first step. We begin with a disjoint-set forest containing one (singleton) tree for each source concept in S. Each tree represents a cluster and we maintain a bitmap that indicates which schemata have contributed concepts to that cluster.

We then sort the distances in ascending order (after eliminating all ∞ distances) and iterate over this sorted list. We find the trees for both x and y; if the intersection of the corresponding bitmaps is empty or the distance is 0, we merge the two trees. This requires us to update a) the disjoint-set forest and b) the bitmap for the merged tree.

Each find or merge operation is $O(A^{-1}(m))$ where A is the Ackermann function (effectively constant). Each bitmap operation is $O(|S|)$. Thus, given d positive correspondences, the complexity of this algorithm is bounded by the sorting operation: $O(d \times lg(d))$, assuming $lg(d) > |S|$. Note that in theory, d could be as large as m, but in practice, d usually varies linearly with n (schema matching algorithms rarely suggest that every concept is related to every other concept!).

Finally, for each tree in the disjoint-set forest, we create a new vocabulary concept $c \in C$. Then, for every source concept s in the tree, we create a correspondence between s and c. These source-vocabulary concept correspondences are aggregated to create the mappings in M.

3. User interface design

Unity transforms vocabulary generation from a manual process into a faster *semi-automated* process where algorithms perform large amounts of the initial work, which expert users then refine. User interaction occurs at

two points. First, a user can open an automatically generated match between two schemata and view and edit (e.g., accept/reject/augment) the matches, improving the quality of the input to synset generation algorithms. The graphical interface design of schema matchers is discussed elsewhere [2, 8]. The second interaction point is refining the initially generated set of synsets, which are then used to generate the final vocabulary. The remainder of this Section discusses how the Unity user interface (UI) enables expert users to more rapidly, productively, and accurately refine synsets than is currently possible.

3.1. Design criteria

Integration engineers traditionally receive candidate synsets in a tabular spreadsheet, and must perform tasks such as:

- Evaluate the quality of automatically generated synsets to determine if their concepts truly belong together (i.e., are semantically congruent).
- Split a synset into two or more synsets if the concepts do not belong together.
- Merge two or more synsets into a single synset if their concepts do belong together.
- Migrate an incorrectly placed concept from one synset to another.
- Create new, or entirely delete existing, synsets.

Interviews with expert users revealed shortcomings in the process enabled by a simple tabular interface, establishing three criteria underlying the design of Unity's UI:

1) *Explore synsets in multiple related contexts*: A tabular view displays synsets in a single context. Synsets and their concepts are best evaluated, however, when explored via multiple linked contexts, because each context (e.g., seeing a concept's placement in its schema-of-origin, as opposed to only in its synset) reveals further semantics.
2) *Edit synsets in a focused workspace:* Exploration across the entire set of synsets often reveals several smaller foci (e.g., similar synsets that may need to be combined). In a standard spreadsheet, the synsets a user wants to edit may be spread from the very top to the very bottom of the spreadsheet. It is valuable to pull "interesting" synsets out of the larger view, and into a focused workspace for edit operations.
3) *Enable drill down into synset evidence*: A tabular synset view reveals only the schema-of-origin (i.e., the column heading) for a given concept. When making edit decisions, it is useful to drill down to see the strength of matches within a synset, and to see the strength of matches for concepts that were *nearly* included in the synset.

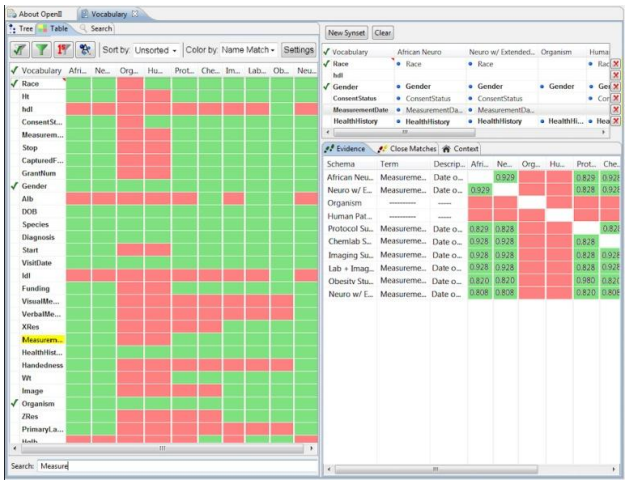

Figure 4: Synset inspection in the Unity user interface. On the left is the Exploration Pane. The upper right is the Workspace Pane, while the lower right is the Detail Pane.

3.2. Interface panes

As shown in Figure 4, the Unity interface consists of three related tabbed panes, each pane addressing one of the above design criteria. Panes and their tabs are interactively linked: actions taken in one can affect what is displayed in the others.

3.2.1. Exploration pane: The left Exploration pane enables users to explore synsets in several different contexts. Tabs provide three views: a Table view, a Tree view, and a Search view. Context sensitive menus enable users to relate concepts across views. For example, a synset concept can be highlighted in all three views at once.

The Exploration pane of Figure 4 shows the Table view, in which each row corresponds to a synset and each column corresponds to a schema. The Table view is similar to most tabular spreadsheet views, but provides sorting and filtering enhancements. A user can click on a column heading to sort alphabetically (as is standard) but they can also sort by the number of schemas participating in the synset. There are a number of filtering options—e.g., by the number of schemas participating in the synset (e.g., show only singletons) or by whether or not the synsets have already been marked as complete. Enhanced features include the ability to turn element labels on or off, and being able to color cells based on the quality of match to the canonical concept (green is an exact name match, yellow is a match that was not an exact name match, red is no match). Turning element text off and coloring on essentially transforms the Table view into a type of heat

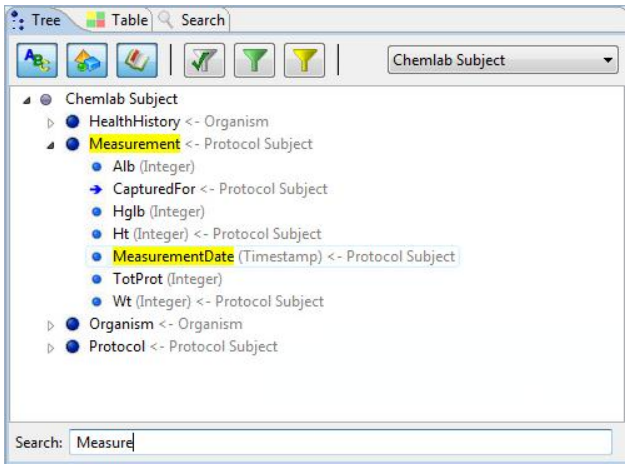

Figure 5: Tree view option within the Exploration Pane, showing synset concepts in their original schema context

map, providing a quick summary of schema alignment with the canonical schema across synsets.

The next tab in the exploration pane is the Tree view, which lays out a schema as a tree. A user can select a concept from the Table View and switch to the Tree View to see that concept in the context of its schema-of-origin, as illustrated in Figure 5. The Tree view is particularly useful for understanding additional semantics about a concept so a user can determine whether or not it truly belongs in a synset. It is not uncommon for multiple concepts in a schema to have the same minimally-informative name (e.g., ID) or for definitions to be missing. In such a situation, original schema context may be the only way to access their actual meaning and thus disambiguate concepts. For example, seeing the concept Tank appear in a plumbing supplies type in its schema-of-origin provides important semantics that a user needs in order to decide if that concept belongs in a particular synset. A simple search capability is available at the bottom of both the Tree and Table view which enables users to execute simple string searches and find specific concepts.

3.2.2. Workspace pane: The top right area is the Workspace pane. Users can drag a synset (i.e., row) from the Exploration pane and drop it into the Workspace pane. The Workspace pane in Figure 4 is populated with six synsets. Within the Workspace pane, users can edit any synset (dragging and dropping concepts among synsets), annotate a synset, and create new synsets.

When a user is content that a synset is correct, they can mark that synset as complete. Completed synsets appear with a check mark next to them. Note that back in the Exploration Pane, a user can hide synsets that have been marked as complete, allowing a user to get a feel for how many synsets are left to review.

3.2.3. Detail pane: The Detail pane appears on the bottom right and enables a user to drill down into evidence for the synset. The data in this pane is automatically populated when the user selects a synset (e.g., MeasurementDate in Figure 4) in the Workspace pane. For each concept in the synset, the Evidence tab shows the strength of the match with every other concept in the synset.

The Close Matches tab shows the user concepts which had high match scores with one or more concepts in the synset, but did not end up in the final synset.

4. Reuse within the OpenII framework

Unity was built within OpenII, which provided multiple opportunities for productive reuse. OpenII [1] is an integrated framework for performing information integration tasks and tool development. In contrast to commercial integration tools, OpenII is open source (http://openii.sourceforge.net/) and includes a set of interfaces and reusable components for solving integration problems. Including Unity, a growing set of tools are being contributed to the OpenII framework [3, 9].

In the following we discuss how building Unity within the OpenII framework was facilitated by reuse of OpenII components. We then illustrate how this strategy provided: a) interoperability with other tools developed in the same framework and b) enables the novel reuse of knowledge captured in vocabularies to enhance traditional schema matching.

4.1. OpenII basics

The OpenII framework provides:

- *Built-in Object Types*. OpenII recognizes and manages relevant object types for information integration, including: *schemata*, *mappings*, and *projects.*
- *Persistence*. Underlying an OpenII instance is a repository for serializing objects of these types.
- *Software Interface*. OpenII provides various programming interfaces (e.g., Java, web services) so tools can manipulate, serialize, and retrieve object instances.
- *User Interface*. An OpenII instance is an Eclipse application, and thus uses the familiar Eclipse interface for panes, tabs, etc.

4.2. Component reuse in Unity implementation

Like all OpenII tools, Unity relies on the OpenII repository. Its neutral metamodel enables users to import existing community schemata written in a variety of formats (e.g., relational, XML schemata, etc.) into a common persistent environment as OpenII *schema* objects.

To create a new vocabulary, Unity instantiates a new OpenII *project* object containing these community *schema* objects. Unity then generates the necessary pairwise mappings among *schema* objects by invoking the existing Harmony schema matcher, which utilizes identical OpenII object representations and interfaces. The resulting OpenII *mapping* objects are added to the *project* object.

If a needed *mapping* object already exists in the repository, it can be reused by simply importing it into the Unity *project* object, saving the cost of regenerating it. This is valuable reuse, in that mappings may have undergone significant expert refinement for previous integration tasks, which should not be wasted.

Unity generates synsets from these *mapping* objects via clustering, as described in Section II. Expert users can then refine these synsets via panes in the OpenII Eclipse UI, such as those shown in Figures 4 and 5. Unity uses the completed synsets to generate a new OpenII *schema* object consisting of a simple list of vocabulary concepts, and a new set of OpenII *mapping* objects between the new schema and the existing schemata.

Thus, the final vocabulary generated by Unity is easily persisted in the OpenII repository as standard and reusable OpenII objects.

4.3. Vocabulary reuse

One illustration of how Unity-generated vocabulary objects can be reused in OpenII as thesauri. Thesauri are important objects within the OpenII framework, because they are employed by the Harmony schema matchers to match concepts via synonymy. Harmony permits users to configure a composite matcher from a suite of individual "match voters," some of which rely on thesauri to identify possible semantic correspondences. Harmony thesauri can be trivially constructed from Unity vocabulary objects.

This particular reuse scenario, from a Unity-generated vocabulary to a Harmony thesaurus match voter, can improve the quality of data exchanges in a domain over time. Vocabulary generation involves the identification of semantically related concepts, some of which are domain specific and would never be found in a normal thesaurus. However, by using a *vocabulary-derived* thesaurus as a schema match voter (on schemata in a semantically congruent domain), Harmony can recognize the correspondence between such concepts. In this way, a feedback loop is created where the accuracy of matches within a domain can improve over time based on the generation of vocabularies, as discussed in [10].

This dynamic is illustrated in the following example. Consider ten hospitals which must align their internal schemata to exchange digital patient records. A Unity-generated vocabulary can be used to help generate executable data exchanges among the hospitals. When an eleventh hospital joins the group, additional schema matching is triggered (i.e., the 11^{th} schema must be matched to each of the other 10) to generate a new vocabulary. Match voters utilizing a thesaurus based on the Unity-generated vocabulary will generate high confidence scores for domain-specific synonyms discovered in the previous matching process which are *also* applicable in this subsequent process.

5. Related work

Many of the tools provided by OpenII are implementations of model management [11-13] operators. Unity, for example, is an implementation of the *merge* operator. Our implementation is based on [14], which describes how to resolve possible representational conflicts. We avoid these conflicts by abstracting schemata into a set of concepts, focusing instead on algorithmic efficiency.

Our work also leverages the rich literature on schema matching [7]. The algorithm presented in Section II is, in effect, an n-ary one-shot integration process [15], with the additional novelty that our integration process is based on first generating a set of binary pairwise schema mappings.

[16] noted the need for improvements in schema matching user interfaces. While [2] added useful filtering and focusing options for the user, it did not fundamentally alter the basic user interface paradigm—e.g., one schema on the left, a second on the right, and lines between them indicating correspondences. Our current work is the first of which we are aware that seriously tackles visualization of n-way matches, drill down into supporting evidence and near misses, and a convenient workspace that supports rapid merging, splitting, and other editing of synsets.

6. Conclusions and future work

This paper describes Unity, a tool for semi-automatic construction of vocabularies, which can serve as standards for information integration projects, from a set of community schemata. Vocabularies consist of a canonical schema whose concepts are related to synonymous concepts in community schemata. Unity relies on a novel combination of n-way schema matching to produce synsets, and context-preserving UI design to refine them.

Unity was constructed through extensive reuse of components in the open source OpenII information integration framework. Not only did this implementation strategy speed the development of Unity and provide compatibility with other tools in the OpenII framework, it enabled a novel reuse of the vocabulary artifacts produced by Unity as a domain-specific thesaurus to enhance future schema matching. Unity has been applied to U.S. Government information integration problems.

Unity faces the same limitations as other schema matching technology in terms of the kinds of matches it is able to identify: unless there are similarities in the names or descriptions of data elements (directly, or via some thesaurus), we will be unable to automatically detect a match. A second limitation is that the vocabulary Unity currently generates consists of an unstructured list of concepts. In the future, we plan to augment Unity to preserve structural relationships among concepts, to the extent possible.

References

1. Seligman, L., Mork, P., Halevy, A., Smith, K., Carey, M.J., Chen, K., Wolf, C., Madhavan, J., Kannan, A., Burdick, D.: OpenII: An Open Source Information Integration Toolkit. SIGMOD (2010)
2. Mork, P., Seligman, L.J., Rosenthal, A., Korb, J., Wolf, C.: The Harmony Integration Workbench. Journal on Data Semantics **11** (2008) 65–93
3. Smith, K., Bonaceto, C., Wolf, C., Yost, B., Morse, M., Mork, P., Burdick, D.: Exploring Schema Similarity at Multiple Resolutions. In: Elmagarmid, A., Agrawal, D. (eds.): Proceedings of the ACM SIGMOD International Conference on Management of Data. ACM, Indianapolis, IN (2010) 1179–1182
4. Smith, K., Morse, M., Mork, P., Li, M., Rosenthal, A., Allen, D., Seligman, L.J.: The Role of Schema Matching in Large Enterprises. Fourth Biennial Conference on Innovative Data Systems Research, Asilomar, CA (2009)
5. Atzeni, P., Torlone, R.: Management of Multiple Models in an Extensible Database Design Tool. In: Apers, P.M.G., Bouzeghoub, M., Gardarin, G. (eds.): Advances in Database Technology - EDBT'96, 5th International Conference on Extending Database Technology. Springer, Avignon, France (1996) 79–95
6. Atzeni, P., Gianforme, G., Cappellari, P.: A Universal Metamodel and Its Dictionary. Transactions on Large-Scale Data- and Knowledge-Centered Systems **1** (2009) 38–62
7. Rahm, E., Bernstein, P.A.: A Survey of Approaches to Automatic Schema Matching. The VDLB Journal **10** (2001) 334–350
8. Bernstein, P.A., Melnik, S., Churchill, J.E.: Incremental Schema Matching. In: Dayal, U., Whang, K.-Y., Lomet, D.B., Alonso, G., Lohman, G.M., Kersten, M.L., Cha, S.K., Kim, Y.-K. (eds.): Proceedings of the 32nd International Conference on Very Large Data Bases. ACM, Seoul, Korea (2006) 1167–1170
9. Chen, K., Madhavan, J., Halevy, A.: Exploring schema repositories with schemr. In: Çetintemel, U., Zdonik, S.B., Kossmann, D., Tatbul, N. (eds.): SIGMOD 2009. ACM, Providence, RI (2009) 1095–1098
10. Madhavan, J., Bernstein, P.A., Doan, A., Halevy, A.Y.: Corpus-based Schema Matching. Proceedings of the 21st International Conference on Data Engineering, ICDE 2005. IEEE, Tokyo, Japan (2005) 57–68
11. Bernstein, P.A.: Applying Model Management to Classical Meta Data Problems. First Biennial Conference on Innovative Data Systems Research, Asilomar, CA (2003)
12. Bernstein, P.A., Melnik, S.: Model Management 2.0: Manipulating Richer Mappings. In: Chan, C.Y., Ooi, B.C., Zhou, A. (eds.): Proceedings of the ACM SIGMOD International Conference on Management of Data. ACM, Beijing, China (2007) 1–12
13. Melnik, S., Rahm, E., Bernstein, P.A.: Rondo: A Programming Platform for Generic Model Management. In: Halevy, A.Y., Ives, Z.G., Doan, A. (eds.): Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data. ACM, San Diego, CA (2003) 193–204
14. Pottinger, R.A., Bernstein, P.A.: Merging Models Based on Given Correspondences. VLDB 2003, Proceedings of 29th International Conference on Very Large Data Bases. Morgan Kaufmann, Berlin, Germany (2003)
15. Batini, C., Lenzerini, M., Navathe, S.B.: A Comparative Analysis of Methodologies for Database Schema Integration. ACM Computing Surveys **18** (1986) 323–364
16. Bernstein, P., Melnik, S., Petropoulos, S., Quix, C. Industrial Strength Schema Matching, SIGMOD Record, Vol. 33, No. 4, December 2004