# The Harmony Integration Workbench

Peter Mork, Len Seligman, Arnon Rosenthal, Joel Korb, and Chris Wolf

The MITRE Corporation
McLean, VA, USA
{pmork,seligman,arnie,jkorb,cwolf}@mitre.org

**Abstract.** A key aspect of any data integration endeavor is determining the relationships between the source schemata and the target schema. This schema integration task must be tackled regardless of the integration architecture or mapping formalism. In this paper, we provide a task model for schema integration. We use this breakdown to motivate a workbench for schema integration in which multiple tools share a common knowledge repository. In particular, the workbench facilitates the interoperation of research prototypes for schema matching (which automatically identify likely semantic correspondences) with commercial schema mapping tools (which help produce instance-level transformations). Currently, each of these tools provides its own ad hoc representation of schemata and mappings; combining these tools requires aligning these representations. The workbench provides a common representation so that these tools can more rapidly be combined.

## 1 Introduction

Schema integration is an integral aspect of any data integration endeavor. The goal of this paper is to organize the strategies and tools used in schema integration into a consistent framework. Based on this framework, we propose an open, extensible, integration workbench to facilitate tool interoperation.

We view the development of a data integration solution to consist of three main steps: schema integration, instance integration and deployment. This paper focuses on schema integration, which generates a transformation that translates source instances into target instances.

Schema integration first involves identifying, at a high level, the semantic correspondences between (at least) two schemata, data models, or ontologies, a task we refer to as *schema matching*. Second, these correspondences are used to establish precise transformations that define a *schema mapping* from the source(s) to the target.

Researchers have built many systems to semi-automatically perform schema matching [1, 2]. Schema mapping tools generally provide the user with a graphical interface in which lines connecting related entities and attributes can be annotated with functions or code to perform any necessary transformations. From these mappings, they synthesize transformations for entire databases or documents. These tools have been developed by commercial vendors (including Altova's MapForce, BEA's AquaLogic, and Stylus Studio's XQuery Mapper) and research projects (such as Clio [3], COMA++ [4] and the wrapper toolkit in TSIMMIS [5]).

Currently, an integration engineer can choose to embrace a specific development environment. The engineer benefits from the automated support provided by that vendor but cannot leverage new tools as they become available. The alternative is to splice together a number of tools, each of which has its own internal representation for schemata and mappings. In one case, we needed four different pieces of software to transform a mapping from one tool's representation into another.

By adopting an open, extensible workbench, integration engineers can more easily leverage automated tools as they become available and choose the best tool for the problem at hand.

## 1.1 Contributions

First, we discuss the information likely to be available to integration engineers: 1) contrary to conventional wisdom, many real-world schemata are well documented, so linguistic processing of text descriptions is important, 2) in several real-world scenarios, schema integration must be performed without the benefit of instance data, and 3) domain values are often available and could be better exploited by schema matchers.

Second, we establish a task model for schema integration based on a review of the literature and tools and on observations of engineers solving real-world integration problems. We have presented our task model to three experienced integration engineers to verify that the model includes all of the subtasks they have encountered.

The task model is important because it allows us to make comparisons: Among integration problems, we can ask which of the tasks are unnecessary because of simplifying conditions in the problem instance. Among tools, we can ask what each tool contributes to each task and quantify the impact in realistic settings.

Third, we describe how the task model and pragmatic considerations guide the development of a specific integration tool, in our case Harmony, a prototype schema matcher, which bundles a variety of match algorithms with a graphical user interface.

Fourth, we articulate the need for data integration among schema integration tools—our community can benefit in insight and utility by practicing what we preach. We propose a candidate collection of interfaces that constitute an integration workbench, which allows multiple integration tools to interoperate and provides a common knowledge repository for schemata and mappings. One outcome of the integration workbench is that integration engineers can more easily choose which match algorithms (or suites thereof [6]) to use when solving real integration problems.

In this expanded version of our previous work [7], we add two new contributions: Our fifth contribution is to demonstrate the integration workbench by describing how several schema integration tools can be instantiated within the workbench. We introduce a general model for matching tools that accounts not only for the extent to which the available evidence suggests the existence of a semantic correspondence (as is traditionally done), but also the *amount* of evidence. Thus, the results generated by multiple matching tools can be combined based on the amount of evidence considered by each approach. In our experiences, the resulting match scores correspond more closely to the intuitions of integration engineers about the "goodness" of a match than traditional methods.

Our sixth contribution is a discussion of the lessons we have learned from our users' experiences with the integration workbench. We describe how the modular architecture allows these users to utilize Harmony to meet their specific schema integration needs including situations in which they have needed to introduce new tools to accomplish their tasks. We conclude by describing how the integration workbench simplifies integration of our schema matching tools with a commercial schema mapping tool (BEA's AquaLogic) that generates global-as-view (GAV) [8] mappings.

### 1.2 Outline

This paper is organized as follows: Section 2 contains our observations regarding schema integration efforts performed on behalf of the federal government. In Section 3 we describe a task model for integration problems. In Section 4 we present design desiderata based on the task model and describe how the Harmony schema matching tool addresses these desiderata. Section 5 describes the interfaces that constitute the integration workbench. In Section 6 we describe a set of schema matching tools that we have plugged into the integration workbench. Section 7 describes the lessons we have learned from interviewing our users about their experiences with Harmony. Finally, we discuss related work in Section 8 and future work in Section 9.

## 2 Integration in Large Enterprises

Conventional wisdom suggests that schema matching should focus on data instances because instances are common and documentation is sparse (or even incorrect). Whereas these phenomena may be observed in some settings, particularly web-based sources, it is often not the case for schemata developed for or by the US federal government (or, we suspect, other large enterprises).

From the perspective of an integration engineer, data instances may be extremely hard to obtain (the data exist, but are not available to the engineer) for at least two reasons.

- **Security/sensitivity**: Data instances are often more sensitive than their corresponding schemata—e.g., in defense applications, an integration engineer may have access to schemata but may lack sufficient clearances to access instances. Sometimes, an agency that owns the data is willing to share them with another agency, but not with the contracting integration engineers responsible for developing the initial mappings. Wider release of schema information is less problematic.
- **Integrating to a future system:** One may begin creating important mappings to and from a new system, even before it has any data or running applications. For example, the U.S. Federal Aviation Administration developed a mapping of some of its systems to a conceptual model for the new European Air Traffic Control System before that system was implemented or had any instance data. As a general phenomenon, when one builds a data warehouse, the mappings from data sources are the actual means for populating it.

Thus, we have observed that it is not safe to assume that instance data will be available to integration tools. Instead, schema integration tools must use whatever information is available. Instance data, thesauri, etc. are sometimes available and sometimes not.

While instance data are often unavailable, we have found that many government (and probably many other enterprises') schemata are well documented. Evidence for this claim will now be presented.

We obtained a collection of 265 conceptual (ER) models from the Department of Defense metadata registry (which contains schemata only, no instances!). This repository contains 13,049 elements (entities or relationships) and 163,736 attributes. As indicated in Table 1, the vast majority of these items contain a definition of roughly one sentence.

**Table 1.** Frequency and length of documentation in the DoD Metadata Registry

| Item | Item Count | # With Definition | % With Definition | Word Count | Words/ Item | Words per Definition |
|---|---|---|---|---|---|---|
| Element | 13,049 | 12,946 | ~99% | 143,315 | ~11.0 | ~11.1 |
| Attribute | 163,736 | 135,686 | ~83% | 2,228,691 | ~13.6 | ~16.4 |
| Domain | 282,331 | 282,128 | ~100% | 1,036,822 | ~3.67 | ~3.68 |

This registry also explicitly enumerates domain values for which documentation is also available. A domain introduces a list of codes, each of which has particular semantics. A domain is a reusable schema construct that can be referenced by multiple attributes. For example, a common domain is the list of two-character state codes (such as VA or MD). In a shipping order, this domain might be referenced by both the shipping entity and the billing entity. Domains and their associated documentation facilitate schema integration even in the absence of instance data. Unfortunately, this documentation is often lost when a logical schema is converted into SQL. The standard approach is to store each coding scheme in its own relation, and each code as a string or integer value, *sans* documentation.

This approach is good for referential integrity, but bad for integration efforts. A better solution would be to define semantic domains for each coding scheme so that integration tools could more easily identify domain correspondences. In fact, when we asked integration engineers to describe how they approach an integration problem, a recurring pattern emerged. They first identify obvious top-level entity correspondences. But then, instead of proceeding to sub-elements or attributes, they then manually inspect the domain values to find correspondences. From this low-level, they then work their way up the schema hierarchy to attributes, sub-elements, and finally back to top-level entities. Our task breakdown is designed to support this pattern.

## 3 Task Model for Data Integration

To better understand how schema integration tools assist an integration engineer, we enumerated the subtasks involved in schema integration. We started with a task model that we created and that was acceptable to 147 survey participants familiar with schema integration from a research or practical perspective [9]. We extended that model to include the subtasks addressed by a variety of systems ([4, 5, 10-16]) and then presented it to three experienced integration engineers for validation. Based on their feedback, we extended the model to include subtasks not directly supported by any system.

At a high level, we consider 13 fine grained integration tasks, grouped into five phases: schema preparation, schema matching, schema mapping, instance integration and finally system implementation. During schema preparation, the source and target schemata are identified so that a set of correspondences can be identified during the matching phase. These semantic correspondences are formalized in the third phase as explicit logical mappings. Once schema integration is complete, instance integration reconciles any remaining discrepancies. In the final phase the integration solution is deployed.

In this section, we describe each phase in detail and describe how we evaluated the task model's completeness. Throughout this section we refer to the following terms: a schema is a collection of schema elements, each of which is either an entity or an attribute. An entity represents a collection of related instances, and an attribute represents a relationship between an entity and another entity or a datatype. An instance belongs to a particular entity and it instantiates values for that entity's attributes. In many cases, the ultimate goal of data integration is to transform source instances into valid target instances.

## 3.1 Schema Preparation

The first phase of schema (or data) integration captures knowledge about the source and target schemata, to facilitate the subsequent matching and mapping phases. It identifies the target schema, and organizes the source schemata. The specific subtasks are:

**1) Obtain the source schemata.** This step gathers available documentation and imports the source schemata into the integration platform. If the source schemata are not in a format compatible with the platform, this step also includes any necessary syntactic transformations.

**2) Obtain or develop the target schema.** If performed, this step is analogous to the previous step. In many cases, the target schema is defined by the problem specification (e.g., translate data into the following message format). In other cases, the target schema must be developed based on the queries to be supported, or to combine the data from multiple sources. This step is optional because the target schema may be derived from the correspondences identified among the source schemata, as is assumed in [11].

In both cases, one may enrich the schemata, e.g., by defining coding schemes as domains, or documenting constraints that are not documented in the actual system, either because the system does not support the needed constructs, or because nobody took the time to do so. Thus, the integration platform may enable richer descriptions than the underlying systems. One also needs a means to keep the metadata in synch as the actual systems change.

## 3.2 Schema Matching

The second phase establishes high-level correspondences among schema elements. There is a semantic correspondence between two schema elements if instances of one schema element imply the existence of corresponding instances of the other [17]. We avoid a more precise definition of a semantic correspondence because the nature of a correspondence depends on the overall goal of schema integration. For example, in the case of a data exchange system, these correspondences imply the existence of a logical transformation that can convert instances of the source element into instances of the target. However, when the integration goal is to generate a consensus vocabulary for a particular community, a semantic correspondence may indicate that the set of source instances overlaps with the set of target instances (i.e., their intersection is non-empty).

If a target schema has been identified, these correspondences establish relationships between each source schema and the target. As noted in [11], in the absence of a target schema, correspondences can also be established between pairs of (or across sets of ) source schemata.

For example, to publish data stored in a relational database into an XML message format, some correspondences indicate that tuples from the source relation will be used to generate XML elements. Additional correspondences indicate which attributes will be used to generate data values. For example, multiple relations might correspond to a single element because a join is needed to populate the element's attributes, or a single relation may correspond to multiple elements to match nesting present in the target.

**3) Generate semantic correspondences.** This step determines which schema elements loosely correspond to the same real world concepts. These correspondences establish a weak semantic link in that they indicate that instances of one element can be used to generate instances of the other.

Whereas this phase consists of a single step, we consider matching to be its own phase because of its importance and the research attention it has received. The exact transformations implied by a correspondence are detailed in the mapping phase.

## 3.3 Schema Mapping

The schema mapping phase establishes, at a logical level, the rules needed to transform instances of the source schemata into instances of the target. The mappings must generate results that adhere to the target schema (or the target must be modified to reflect accurately the transformed data).

These mappings are often expressed as queries expressed in a language applicable to the source or target schema. For example, in [8] mappings are expressed as Datalog queries and in [18] mappings are expressed using XQuery (even though the source schema is relational). However, in [19] the mappings are expressed in SQL even though the transformed data are expressed as XML.

The first four subtasks below establish piecemeal transformations, and are not performed in a particular order. Each transformation indicates the precise mechanism by which source data is used to generate target data. Note that at times these transformations cross the schema/instance boundary [20]. Once transformations have been established for each schema element, they are aggregated into a logical mapping and verified.

**4) Develop domain transformations.** For each pair of corresponding domains, a transformation must be developed that relates values from the source domain to values in the target domain. In the simplest case, there is a direct correspondence (i.e., no transformation is needed). However, it is often the case that an algorithmic transformation must be developed, for example, to convert from feet to meters, or from first- and last-name to full-name. In the most detailed case, the transformation can best be expressed using a lookup table (e.g., to convert from one coding scheme to a related coding scheme). Context mediation techniques can then be applied [21, 22].

**5) Develop attribute transformations.** The previous step handled the case where the same property was encoded using different domains. This step deals with properties that are different but derivable. Sometimes one provides a transformation from source to target values, either scalar (e.g., Age from Birthdate), or by aggregation (e.g., AverageSalaryByDepartment from Salary). Other transforms we have seen include pushing metadata down to data (e.g., to populate a type attribute or time-stamp), and populating a comment (in the target) to store source attribute information that has no corresponding attribute. Finally, it may be necessary to convert a single attribute into a composition of attributes (in the local-as-view (LAV) [8] formalism) or vice versa for GAV.

**6) Develop entity transformations.** The next step is to determine the structural transformations necessary to generate instances of the target schema. In the simplest case, a direct 1:1 mapping can be established. Alternatively, multiple entities may need to be combined to generate a single target entity. This combination may require a join operation if the source schema vertically partitions information across multiple entities or a union operation if the source schema horizontally partitions information across entities that are subclasses of the target entity. Additionally, a single entity may need to be split into multiple entities (e.g., based on the value of some attribute), which effectively elevates data in the source to metadata in the target.

**7) Determine object identity.** For each entity in the target, the next step is to determine how unique identifiers will be generated. In the simplest case, explicit key attributes in the source can be used to generate key values in the target. This may include populating implicit keys (such as those inherited from a parent entity), or correctly establishing parent/child relationships (such as in a nested meta-model). For arbitrarily assigned identifiers (such as internal object identifiers), Skolem functions are commonly employed (see, for example, [3]).

These four subtasks interact with schema matching because establishing transformations is an iterative process. For example, in the first pass, we might establish a transformation from Professor to Employee (since instances of the former are also instances of the latter). While working on the Course/Grade sub-schema, we might realize that, in some cases, Students are also Employees. This new insight requires us to refine the Employee mapping. In other words, the previously identified correspondences may be both imprecise and incomplete.

The remaining mapping subtasks produce an executable mapping.

**8) Create logical mappings.** The next step is to aggregate the piecemeal mappings, which all concern individual elements, into an explicit mapping for entire databases or documents. Humans may need to specify additional information (e.g., to distinguish join from outerjoin) before automated tools can sew the pieces together. In most cases, this requires writing a query (over the source schemata) that generates instances of the target schema, although in LAV [8] the source schemata are expressed as views over the target schema.

**9) Verify mappings against target schema.** If the integration task included a specific target schema, the final step is to verify that the transformations are guaranteed to generate valid data instances (i.e., all constraints are satisfied). In some cases, the only solution may be to modify the target schema to remove constraints that we cannot satisfy. If a target schema is not specified, the final step is to generate the target schema based on the logical mappings.

### 3.4 Instance Integration

At this point, the tasks involved in schema integration are complete, and we turn our attention to instance integration.

**10) Link instance elements.** Two source instances (with different unique identifiers) may represent the same real-world object. This subtask merges these instances into a single instance or creates an association between the instances. See [23] for an overview of the algorithms involved.

**11) Clean the data.** This subtask removes erroneous values from instances. A value may be erroneous because it violates a domain constraint or because it contradicts information from a more reliable source. For example, we may know that a person should have a single value for the height attribute, but the available sources might provide differing values for this attribute value. See [24] for more information about this subtask.

### 3.5 System Implementation

Finally we are ready to develop and deploy a system that addresses operational constraints—factors external to schema and instance elements. Examples include determining the frequency and granularity of updates and the policy that governs exceptional conditions.

**12) Implement a solution.** In this phase the system developers must first gather any operational constraints and then design an integration system that satisfies these constraints. The significance of the operational constraints on real-world integration systems is stressed by the integration engineers who have reviewed the task model. For example, operational constraints such as the volume of data involved, the freshness of results, and security factors strongly influence whether a federated database or data warehouse should be developed.

**13) Deploy the application.** This step does not receive much research attention, but ease of deployment is an important concern. Many of the commercial data integration tools place particular emphasis on this subtask. Once deployed, system engineers must maintain the application, but a task model for application maintenance exceeds the scope of this paper.

This task model guided our development of the Harmony schema matching tool.

## 4 Harmony

Harmony is a schema matching tool that combines multiple match algorithms with a graphical user interface for viewing and modifying the identified correspondences. The architecture for Harmony is shown in Fig. 1. Harmony's contributions include adding linguistic processing of textual documentation to conventional schema match techniques, learning from the input of a human in the loop, and GUI support for removing clutter and iterative development, as discussed in following sections.

Harmony currently supports XML schemata, entity-relationship schemata from ERWin, a popular modeling tool, and will soon support relational schemata. Schemata are normalized into a canonical graph representation.

The Harmony match engine adopts a conventional schema integration architecture [6, 25-27]. It begins with linguistic preprocessing (e.g., tokenization, stop-word removal, and stemming) of element names and any associated documentation. Then, several *match voters* are invoked, each of which identifies correspondences using a different strategy. For example, one matcher compares the words appearing in the elements' definitions. Another matcher expands the elements' names using a thesaurus. For each [source element, target element] pair, each match voter establishes a confidence score in the range (–1, +1) where –1 indicates that there is definitely no correspondence, +1 indicates a definite correspondence and 0 indicates complete uncertainty.

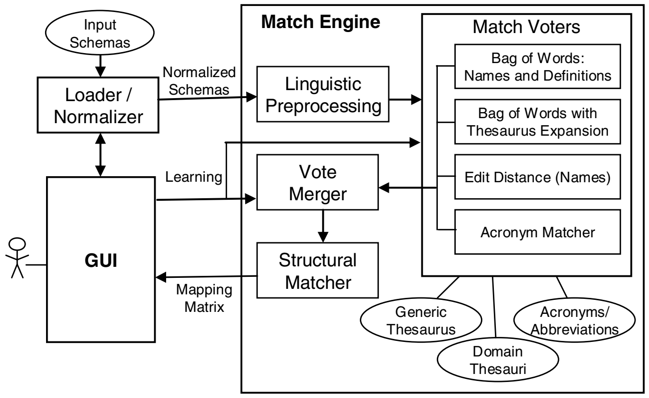

**Fig. 1.** Architectural Overview of Harmony

Given *k* match voters, the vote merger combines the *k* values for each pair into a single confidence score. The vote merger weights each matcher's confidence based on its magnitude—a score close to 0 indicates that the match voter did not see enough evidence to make a strong prediction.

A version of similarity flooding [28] adjusts the confidence scores based on structural information. Positive confidence scores propagate up the schema graph (e.g., from attributes to entities), and negative confidence scores trickle down the schema graph. Intuitively, two attributes are unlikely to match if their parent entities do not match.

Finally, these confidence scores are shown graphically as color-coded lines connecting source and target elements. The GUI provides various mechanisms for manipulating these lines, based on our design desiderata.

## 4.1 Design Goals

The statistics presented in Section 2 suggest that schema matching algorithms should not assume the absence of usable documentation. Many of the candidate matchers in the Harmony engine perform natural language processing and comparisons on this documentation. In our experience, these matchers have good recall, although their precision is less impressive.

The task model in Section 3 suggests additional design desiderata. First, the integration engineer needs to be able to focus at different levels of granularity. For example, a common first step is to establish correspondences among conceptual sub-schemata. In the air traffic flow management domain, these sub-schemata might include facilities (airports and runways), weather, and routing. Note that the hierarchical and decomposable nature of XML Schema makes it easier to identify sub-schemata.

After establishing these high-level correspondences, the integration engineer focuses on one sub-schema at a time and delves into the details of the domains appearing in that sub-schema. The engineer wants to be distracted neither by correspondences pertaining to other sub-schemata nor those at intermediate levels of granularity.

A related goal is that the software tools must support iterative refinement. This desideratum is one of our motivations for developing the integration workbench described in Section 5. If data cannot flow freely among components, the engineer has little control over the order in which tasks will be completed.

The final desideratum is that all sub-tasks involved in schema integration must be supported. The commercially available tools naturally take this requirement more seriously than do research tools, such as Harmony. Whereas it is an interesting research problem to identify semantic correspondences, this contribution alone does not greatly assist the integration engineer. Because Harmony by itself does not currently support schema mapping, we defer further consideration of this desideratum to Section 5. We now consider how Harmony addresses the remaining desiderata.

## 4.2 Filtering

The Harmony GUI supports a variety of filters that help the integration engineer focus her attention. These filters are loosely categorized as link filters and node filters. A link filter is a predicate that is evaluated against each candidate correspondence to determine if it should be displayed. A node filter determines if a given schema element should be *enabled*. An enabled element is displayed along with its links; a disabled element is grayed out and its links are not displayed.

Harmony currently supports three link filters. First, a confidence slider filters links based on the confidence assigned to a link by the Harmony engine. Only links that exceed the slider-set threshold are displayed. Links that were drawn by the integration engineer, or were explicitly marked as correct, have a confidence score of +1. Similarly, links explicitly rejected have a score of –1.

The second filter determines if a link should be displayed based on whether it is human-generated or machine-suggested. The final filter displays those links with maximal confidence for each schema element (usually a single link, but ties are possible).

The node filters include a depth filter and a sub-tree filter. The former enables only those schema elements that appear at a given depth or above. For example, in an ER model, entities appear at level 1, while attributes are at level 2. In XML schemata, arbitrary depths are possible. Thus, using this filter, the engineer can focus exclusively on matching entities.

The sub-tree filter enables only those elements that appear in the indicated sub-tree. For example, this filter can be used to focus one's attention on the 'Facility' sub-schema. By combining these filters, the engineer can restrict her attention to the entities in a given sub-schema.

### 4.3 Iterative Development

Harmony supports iterative refinement through two mechanisms. First, the engineer can rerun the Harmony engine, which can learn from her feedback. Second, the engineer can mark sub-schemata as complete. We now describe these two mechanisms.

When the Harmony engine is invoked after some correspondences have been explicitly accepted or rejected (i.e., set to +1 or –1), this information is passed to the engine and used in two ways. First, each candidate matcher can learn from the user's choices and refine any internal parameters. For example, a matcher that weighs each word based on inverted frequency increases or decreases word weight based on which words were most predictive. Second, the vote merger weights the candidate matchers based on their performance so far. Learning new weights must be done carefully, though. Each candidate matcher focuses on a particular form of evidence, such as elements' names. If the engineer based her first pass on exactly that form of evidence, the corresponding candidate matcher will appear overly successful.

In addition to accepting and rejecting specific links, the engineer can mark a sub-tree as complete. This action has several effects. First, it accepts every link pertaining to that sub-tree as accepted (if currently visible), or rejected (otherwise). Once a link has been accepted or rejected, the engine will not try to modify that link. This ensures that links do not mysteriously disappear or appear should the user subsequently invoke the Harmony engine.

Second, it updates a progress bar that tracks how close the engineer is to a complete set of correspondences. This feature was introduced at the request of integration engineers working on large schema integration problems that involve several dozen iterations.

Once all schema elements have been marked as complete, the final set of correspondences could be used to guide the generation of a more detailed mapping. Harmony provides neither a mechanism for authoring code snippets, nor a code generation feature; these would duplicate commercial capabilities. Instead, we are developing the integration workbench to couple our matching tools (and GUI) with commercially-available mapping products.

## 5 Integration Workbench

Our attempts to integrate Harmony with other schema integration tools revealed a key barrier to interoperability. Whereas schema integration experts trumpet the advantages of a modular, federated architecture that presents a unified view of multiple data sources, we (as a community) have not applied that same insight when we develop our own systems and tools.

As a concrete example, we recently received a collection of XML files from a colleague. Each file described a schema mapping between a source and target schema.

However, before we could use these files, we needed to transform them into a structure compatible with Harmony. To effect this transformation we used one tool to reverse engineer the schema assumed by our colleague. We then matched that schema to the Harmony schema (using Harmony). We recreated the match in AquaLogic to generate a suitable XQuery for transforming a single XML file. Finally, we wrote a Perl script to apply the XQuery to each XML file. A modular architecture would facilitate tool interoperability.

While some vendors (such as IBM and BEA) may be moving in this direction internally to support integration of their own tools, they have not published their approaches or interfaces. There are obvious advantages to user organizations and small software companies to developing a *standard* framework for combining schema integration tools. We propose the following as a way to initiate discussion that could lead toward development of such a standard.

At the core of our workbench proposal is an integration blackboard, which is a shared knowledge repository. Mediating between the blackboard and the various schema integration tools is a workbench manager. The manager provides several services including transaction management, event services and query evaluation. The following sections describe the blackboard and manager.

## 5.1 Integration Blackboard

The integration blackboard (IB) is a shared repository for information relevant to schema integration that is intended to be accessed by multiple tools, including schemata, mappings, and their component elements. We propose using RDF [29] for the IB, because: 1) it is natural for representing labeled graphs, 2) one can use RDF Schema to define useful built-in link types while still offering easy extensibility, 3) it is vendor-independent, and 4) it has significant development support.

The basic contents of the IB are schema graphs and mapping matrices (an approach also taken in [25]). However, in RDF, any element can be annotated; we use this feature to enrich the graphs and matrices with additional information. We predefine certain annotations using a controlled vocabulary (these terms appear in sans serif).

### 5.1.1 Schemata

The IB represents a schema as a directed, labeled graph. The nodes of this graph correspond to schema elements. In the relational model, these elements include relations, attributes and keys. In XML, they include elements and attributes.

The edges of a schema graph correspond to structural relationships among the schema elements. These edges are object properties whose subject and object are both schema elements. For example, in the relational model contains-table edges are used to link a database to the tables it contains. Tables are linked to attributes via contains-attribute edges. In XML, elements are linked to sub-elements via contains-element edges, and to attributes via contains-attribute edges. For many schema languages, the edge-types are specified by the modeling language, but with ontologies they are extensible.

Whereas schema elements can be annotated arbitrarily, we identify three edge labels of particular importance to schema importing and matching utilities: name, type

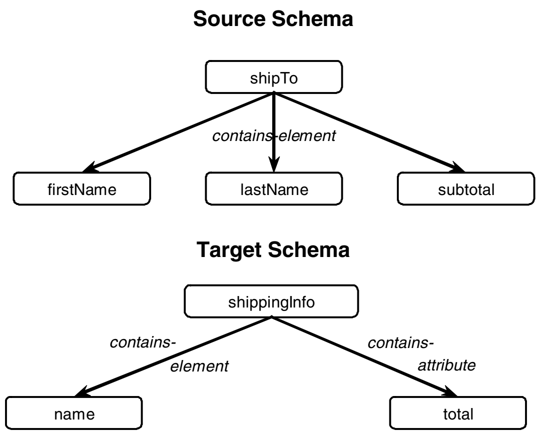

**Fig. 2.** Sample schema graphs

and documentation. Import tools populate these metadata so that they can be used by schema matchers to identify potential correspondences.

Sample schema graphs appear in Fig. 2. In the next section we present a sample mapping from the source schema to the target schema.

### 5.1.2 Mappings

Inter-schema relationships can be represented conceptually as a *mapping matrix*. This matrix consists of headers (describing source and target elements) plus content (a row for each source element and a column for each target element). Note that whereas the structure can easily be interpreted as a matrix, we store this matrix using RDF.

| code= `let $shipto := $purchOrd/shipTo return <shippingInfo total = "{ data($shipto/subtotal) * 1.05 }"> { for $fName in $shipto/firstName, $lName in $shipto/lastName return <name>{ concat($lName, concat(", ", $fName)) }</name> } </ShippingInfo>` | **shippingInfo** is-complete=`false` code= | **name** is-complete=`false` code= `concat($lName, concat(", ", $fName))` | **total** is-complete=`false` code= `data($shipto/subtotal) * 1.05` |
|---|---|---|---|
| **shipTo** is-complete=`false` variable=`$shipto` | confidence=`+0.8` user-defined=`false` | confidence=`-0.4` user-defined=`false` | confidence=`-0.6` user-defined=`false` |
| **firstName** is-complete=`true` variable=`$fname` | confidence=`-1` user-defined=`true` | confidence=`+1` user-defined=`true` | confidence=`-1` user-defined=`true` |
| **lastName** is-complete=`true` variable=`$lname` | confidence=`-1` user-defined=`true` | confidence=`+1` user-defined=`true` | confidence=`-1` user-defined=`true` |
| **subtotal** is-complete=`true` variable=`$shipto/subtotal` | confidence=`-1` user-defined=`true` | confidence=`-1` user-defined=`true` | confidence=`+1` user-defined=`true` |

**Fig. 3.** Sample mapping matrix in which every component has been annotated

For example, the mapping matrix for the schemata in Fig. 2 contains four rows and three columns, as shown in Fig. 3. Each cell in the mapping matrix describes a potential correspondence between a source element and a target element.

Mapping elements are also annotated. First, each cell is annotated with confidence-score, which ranges from –1 (definitely not a match) to +1 (definitely a match), and is-user-defined. This latter annotation is true for any correspondence provided by the user (for example, by drawing a link between two elements), and the associated confidence-score is either +1 or –1 (for rejected links). When a match algorithm is executed, is-user-defined is false, and the confidence-score falls in the range (–1,+1).

Each row is further annotated with a variable-name. Each column is annotated with code that references these names. Finally, the matrix as a whole has a code annotation, which represents the mapping from source to target. Additional annotations are possible; for example, Harmony annotates rows and columns with is-complete to track progress. The relationship between these annotations and the mapping matrix appears in Fig. 3.

#### 5.1.3 Integration Blackboard Enhancements

We currently assume that the blackboard captures information about the source and target schemata, as well as the current state of the mapping that relates the source(s) to the target. Future goals include the following.

- The blackboard should maintain a library of mappings, partly to facilitate mapping reuse, but also as a resource for some matching tools.
- Schemata inevitably change; the blackboard should track schemata across versions.
- Mappings are also refined over time, especially once they are tested on real data. The blackboard should maintain mapping provenance.
- Based on Section 4.2, the blackboard should allow contextual information, such as focus on a particular subschema, to be shared across tools.
- The blackboard should be shared across multiple workbench instances.

### 5.2 Workbench Manager

All interaction with the IB occurs via the workbench manager, which coordinates matchers, mappers, importers, and other tools. The manager provides several services: First, it provides transactional updates to the IB. Second, following each update, it notifies the other tools using an event. Third, the manager processes ad hoc queries posed to the IB.

A single-user version of the workbench architecture appears in Fig. 4. Ultimately, we envision there to be one IB for each community of interest—i.e., a set of stakeholders “who must exchange information in pursuit of their shared goals, interests, missions, or business processes” [30]. Each integration engineer would have her own instance of the integration workbench containing a single manager and multiple tools.

#### 5.2.1 Tools

We focus on four kinds of tools: loaders, matchers, mappers and code-generators. The first two tools support the first two phases of schema integration. Given the complexity of schema mapping, we separate out steps 4)–7), in which the mapping is produced piecemeal, from steps 8) and 9), in which code is generated.

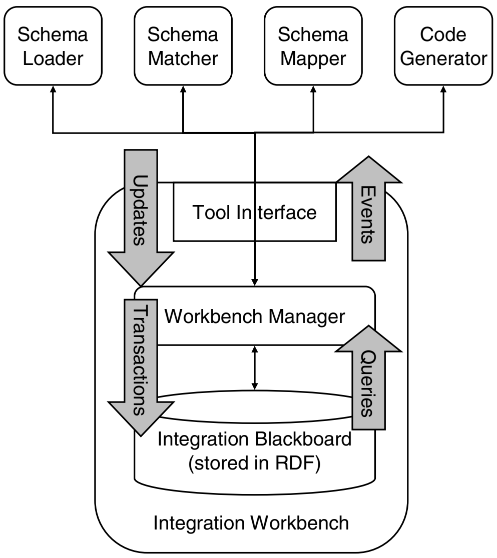

**Fig. 4.** Workbench Architecture

Loaders are used during schema preparation to parse a schema from a file, database or metadata repository (including ancillary information such as definitions from a data dictionary) into the internal representation used by the IB. When the user invokes a loader, that tool places the new objects in the IB, which extends the mapping matrix accordingly and advises the other tools via an event.

Schema matching can be performed manually, as is the case for most commercial tools, or semi-automatically. (Harmony supports both approaches.) A match tool updates the cells of the mapping matrix. When correspondences are generated automatically, all of the interactions with the IB are wrapped in a transaction; no events are generated until the mapping matrix has been updated.

Schema mapping can also be performed manually or automatically [31], although we are not aware of any commercial automatic mapping tools. A mapping tool updates the code associated with each column. Both matchers and code generators may need to listen for these events to update their internal state.

Finally, a code-generator assembles the code associated with each column into a coherent whole. Thus, the code-generator must understand how to assemble code snippets based on the structure of the target schema graph (e.g., Clio [3]).

This enumeration of tools is by no means complete. Another tool might attempt to enforce domain-specific constraints on the mapping matrix. Or, a tool might annotate a schema with information culled from external documentation. All that is required is that a tool implements the tool interface.

The tool interface defines two methods. First, a tool must provide an invoke method. The implementation of this method might launch a GUI (for mapping), invoke a match algorithm, or display a file selection dialog (to load). Second, when the workbench starts, each tool has the option of implementing an initialize method. Generally, this is done when a tool needs to register for events.

### 5.2.2 Events

Tools generate events whenever they make any change to the contents of the IB. The workbench manager propagates these events to allow any tool to respond to the update. A different type of event is generated for each major component of the IB so that a tool can register for only those events relevant to that tool.

A schema loader generates a *schema-graph event* when it imports a schema into the workbench. Any tool with a GUI listens for these events and refreshes the display.

A *mapping-cell event* is generated when a user manually establishes a correspondence. Multiple such events are triggered by an automatic matching tool. A mapping tool can listen for these events to propose a candidate transformation, such as a type conversion.

Conversely, when a mapping tool establishes a transformation, it generates a *mapping-vector event*. Match tools listen for these events to synchronize the mapping cells with the updated row or column. A code generation tool similarly listens for these events to synchronize the assembled mapping. The code generation tool, in turn, generates a *mapping-matrix event* when the user manually modifies the final mapping.

Additional interactions are possible, but generally speaking, a tool listens for events immediately upstream or downstream in the task model. It is necessary to listen in both directions given the iterative behavior described in Section 4.3 and illustrated by the schema matching tools we have developed.

## 6 Sample Schema Matching Tools

Our research has focused on the development of two types of schema matching tools. The role of a match voter is to consider some source of evidence to generate a match score for a particular (source element, target element) pair. The match score is a function of the amount of evidence observed that suggests the pair of elements match (the positive evidence) and the total amount of evidence available. The standard approach for generating a match score is to compute the ratio of positive evidence to total evidence.

However, this approach ignores the fact that, as the amount of evidence increases, the impact of that evidence is greater. In this section, we first formalize the roles of positive and total evidence. We then describe how to apply this theory to various match voters.

Within the integration workbench, multiple match voters might be available. The role of a vote merger is to combine the match scores generated by a suite of match voters into a single confidence score to be stored in the mapping matrix. To derive a confidence score the vote merger assigns a weight to each match voter and combines the match scores based on the amount of evidence observed by each match voter.

In this section we describe each component in greater detail. As a preliminary, we briefly describe how we normalize the available documentation. We then describe how *any* match voter can compute a match score based on numeric scores for positive and total evidence. Next, we describe a specific match voter in which the evidence is based on the extent to which the words appearing in the schema documentation for two elements overlap. Finally, we describe how the scores generated by multiple match voters can be combined into a single value.

## 6.1 Text Normalization

For some match voters, several pre-processing strategies are required. First, we tokenize all text strings in the source and target schemata, splitting those phrases that are not divided by spaces into distinct words. Because of the frequency with which upper-case letters are used to indicate word boundaries (sometimes called CaMeL case), whenever an upper-case letter is immediately followed by a lower-case letter, we break the text into separate words at that boundary (e.g., 'firstName' becomes 'first Name'). Tokenization also removes all punctuation. Following tokenization, the text contains only letters, numbers and white-space.

Second, we replace all capital-letters with lower-case letters. Third, we remove plural suffixes and verb conjugations. For example, 'reading books' becomes 'read book'. Fourth, we remove any words that appear on a pre-defined list, (such as 'a' and 'for'). These *stop-words* are too common to be useful for linguistic processing. We refer to the output of these four steps as *normalized* text.

During pre-processing, we also count the frequency of each normalized word appearing anywhere in a source or target schema element. Generally speaking, words that are rarely used are more significant that words that appear frequently. The word frequency function *freq(wd)* maps each word *wd* to the number of times it appears in normalized text:

$$freq(wd) \rightarrow N \tag{1}$$

The weight associated with each word is inversely proportional to the number of times it appears in the source and target schemata. In the ideal case, a word appears exactly once in the source and once in the target, or twice total. Based on this observations, the weight function *wt(wd)* is:

$$wt(wd) = \frac{2}{freq(wd)} \tag{2}$$

As an ongoing example, we will consider two schema elements drawn from the domain of military tracking. In this domain, it is important to know how a particular set of coordinates were obtained so that human experts can gauge the reliability of the information. Hence, we will consider source element $s$ = "How: provides a hint about how the coordinate was obtained," and target element $t$ = "TargetSource: indicates how the latitude and longitude were obtained." Whereas these elements are not identical, they are similar in nature and should be matched (and ultimately mapped) to one another.

After normalization, these elements are simplified to "how provide hint about how coordinate obtain" and "target source indicate how latitude longitude obtain," respectively. For simplicity, let us assume "how" appears sixteen times in the source and target schemata and that the remaining words appear twice each. Thus, the *wt*("how") = 0.125 and *wt*(*wd*) = 1 otherwise. We will return to this example in section 6.3 when we describe our bag-of-words match voter. But first we describe (in abstract terms) our match score framework from the perspective of positive and total evidence.

## 6.2 Match Scores

Our match score framework expects that each match voter will assign a single score to each pair of source and target elements. This match score is generated by considering some collection of evidence (*toe* for total observed evidence) of which a subset suggests a correspondence between the pair of elements (*poe* for positive observed evidence). For example, in the preceding example, the total evidence consists of the words used to describe *s* and *t* and the positive evidence consists of the words they share. Other sources of evidence might include the datatypes assigned to these elements or the data values used to instantiate them. In this section we describe how any match voter can combine *toe* and *poe* to generate a match score that ranges from –1 to +1.

The intuition behind these match scores is that a score of 0 should indicate that, based on the observed evidence, the likelihood of a match is impossible to determine. As the ratio of positive evidence to total evidence increases, the match score should increase. For a fixed evidence ratio, as the total evidence increases, the match score should also increase.

Based on this intuition, we can establish some theoretic bounds. If there is an infinite amount of positive evidence, the match score should equal +1. However, if there is no positive evidence, but an infinite amount of total evidence, the match score should equal –1. Finally, if there no evidence (of either type), the match score should be 0.

Formally, for a given (source element, target element) pair, let *poe* represent the amount of positive observed evidence, and *toe* represent the total observed evidence. However, before observing this evidence, there is some small probability *x* that two elements (chosen at random) match. Thus, we must factor in this prior probability to calculate the (combined) positive evidence *pe* and total evidence *te*.

$$pe = x + k \times poe \tag{3}$$

$$te = 1 + k \times toe \tag{4}$$

In equations (3) and (4), *k* is a scaling factor that indicates how much we want to weigh the observed evidence. Now, we calculate the evidence ratio *er* as the ratio of positive evidence to total evidence.

$$er = \frac{pe}{te} \tag{5}$$

The weighted evidence ratio *wer* scales the evidence ratio from the interval [0, 1] to the interval [1, e]. When the weighting factor *j* is one, this is a linear transformation. Large values of j generate a sub-linear transformation.

$$wer = er^{1/j}(e-1)+1 \tag{6}$$

The evidence factor *ef* measures the amount of evidence considered by mapping the positive evidence from the interval [0, ∞) to the interval [e, 1].

$$ef = (1+pe)^{1/pe} \tag{7}$$

The match score *ms* is the natural log of the ratio between *wer* and *ef*.

$$ms = \ln\left(\frac{wer}{ef}\right) \tag{8}$$

**Table 2.** Relationship between evidence (positive and total) and match scores for extreme values. The final column provides insight into equations (3)–(8).

| $pe$ | $te$ | $er$ | $wer$ | $ef$ | $ms$ | = |
|---|---|---|---|---|---|---|
| 0 | $\infty$ | 0 | 1 | $e$ | –1 | $\ln\left[\frac{1}{e}\right]$ |
| 0 | 0 | 1 | $e$ | $e$ | 0 | $\ln\left[\frac{e}{e}\right]$ |
| $\infty$ | $\infty$ | 1 | $e$ | 1 | 1 | $\ln\left[\frac{e}{1}\right]$ |

Finally, the match score is guaranteed to fall in the interval (–1, +1) as demonstrated by a limit analysis (see Table 2) as the positive and total evidence approach 0 and positive infinity. All that remains is to determine suitable values for the parameters *j*, *k*, and *x*. We choose *x* such that in the absence of direct evidence, the match score evaluates to 0. Whereas we have not found a closed solution for *x* in terms of *j*, for certain values of *j* we have observed the following:

$$x \approx e^{\frac{-\ln j}{1.5}} \text{ when } j \geq 7 \tag{9}$$

The values of the remaining two parameters depend on the match voters under consideration. Generally speaking, *j* controls how much positive evidence is required for *ms* to generate a match score greater than zero, and *k* amplifies the observed evidence. We recommend suitable values for these parameters for match–voters based on algorithms developed for measuring the similarity between two natural language documents.

## 6.3 Sample Linguistic Match Voters

The preceding section described a match voter at an abstract level. We now turn our attention to specific match voters based on natural-language processing (NLP). In this section, we describe how to quantify the observed evidence for NLP-based match voters. We then establish reasonable values for the constants described above.

In the domain of document retrieval, one strategy for determining the similarity of two documents is to determine the extent to which the pair of documents has words in common. We apply this approach to schema matching by treating each schema element as a document. For a given schema element, the corresponding document contains the normalized text appearing in the element's documentation and name[1]. This document is then reduced to a bag-of-words (i.e., a set of words in which a given word can appear multiple times). The evidence represented by bag-of-words $B_s$ is computed as follows, where the weight function was defined in equation (2), above.

$$ev(B) = \sum_{wd \in B} wt(wd) \tag{10}$$

[1] Because of the importance of an element's name, we actually add that normalized text to the document twice.

**Table 3.** Relationship between evidence (positive and total) and match scores for three different values of positive observed evidence.

| $B_s \cap B_t$ | *poe* | *toe* | *pe* | *te* | *er* | *wer* | *ef* | *ms* |
|---|---|---|---|---|---|---|---|---|
| {"how", "obtain"} | 1.125 | 10.125 | 3.6 | 32 | 0.11 | 2.4 | 1.5 | 0.44 |
| {"how"} | 0.125 | 10.125 | 0.59 | 32 | 0.019 | 2.2 | 2.2 | –0.019 |
| {} | 0 | 10.125 | 0.22 | 32 | 0.0068 | 2.0 | 2.5 | –0.19 |

For a given (source-element, target-element) pair, the positive evidence is based on the intersection of the corresponding bags, and the total evidence is based on the union.

$$poe(s,t) = ev(B_s \cap B_t) \tag{11}$$

$$toe(s,t) = ev(B_s \cup B_t) \tag{12}$$

In our ongoing example ("How" vs. "TargetSource"), the positive observed evidence is based on the bag {"how", "obtain"} and the total observed evidence on the bag {"about", "coordinate", "hint", "how", "how", "indicate", "latitude", "longitude", "obtain", "provide", "source", "target"}. Given the previously assigned word weights, *poe*(*s*,*t*) = 1.125 and *toe*(*s*,*t*) = 10.125.

Harmony also supports the inclusion of evidence external to the source and target schemata. A second match voter uses a bag-of-words augmented with a thesaurus. For each word in $B_s$, if that word appears in the thesaurus, its synonyms are added to the bag. Once the bags have been augmented with synonyms, the weight function in equation (2) must be re-evaluated. Otherwise, the thesaurus-based bag-of-words match voter is identical to the normal bag-of-words match voter.

All that remains is to establish values for *j* and *k*. In our experience, *j*=20 seems to work well in practice. Given the trade-off between precision and recall, we prefer to err on the side of recall because it is easier for an integration engineer to reject false matches, than to identify false non-matches. We found *k*=3 to work well for the basic bag-of-words matcher, and *k*=1 to work well when using a thesaurus. The intuition behind using a smaller *k* is that we expect to see more total evidence with the thesaurus, and therefore do not need to amplify the effect of the observed evidence.

To illustrate how the total and positive observed evidence is used to calculate a match score, we will return to our ongoing example in which the positive observed evidence value is 1.125. Let us also consider similar scenarios in which the common words are {"how"} and {}. (This example assumes *k*=3, but *j* is set to 10 because the documentation strings are so short). Table 3 shows how the match scores are derived in each of these scenarios. We have deliberately chosen *j* and *k* such that the match score will be relatively large whenever a pair of elements share even a small number of uncommon words. Moreover, very low scores cannot be generated without a huge amount of total evidence. In our experience, based on real-world schemata, positive evidence is a much stronger indicator than negative evidence. By incorporating this intuition into our match scores, multiple sources of evidence can be combined by the vote merger.

## 6.4 Vote Merger

Within Harmony, several match voters are run in parallel, each of which generates a match score for each pair of source and target elements. In this section we describe how to combine these values into a single score for each pair. We begin by describing how to merge match scores assuming each match voter were also to return an evidence score in addition to a match score (for each pair). We then describe how to merge match scores without imposing this additional requirement.

The vote merger is responsible for combining multiple match scores into a single confidence value. This combination is based on multiple factors including the weight assigned to each match voter, the amount of evidence available to that match voter, and the positive evidence observed by that match voter. For each (source element, target element) pair, the match voter generates a single confidence value.

The basic vote merging algorithm is simply a weighted average of the match scores generated by each match voter. If we assume that the weight of a given match voter *v* is *wt*(*v*), and that the weight associated with the evidence observed by that match voter is $ew_v$, then the confidence score is the weighted average of match scores as follows, where *V* is the set of all match voters.

$$conf = \frac{\sum_{v \in V} wt(v) \times ew_v \times ms_v}{\sum_{v \in V} wt(v) \times ew_v} \tag{13}$$

When the weights associated with each match voter are equal, the confidence score is simply the weighted average of the match scores, based on $ew_v$. Thus, we need to determine how to compute evidence weights.

In general, the evidence weight needs to scale from zero (in the absence of evidence), to one (given infinite evidence). Thus, any function that maps *te* to the interval [0, 1] fulfills this conditions. For example, the following function is an analogue of equation (7).

$$ew = (1 + \frac{1}{te})^{te} \tag{14}$$

Note that equation (14) requires that we preserve multiple values for each match voter. However, the match score calculated in equation (8) is close to zero when there is little total evidence, and close to ±1 when the amount of total evidence is large. Given this observation, we use the absolute value of the match score as the evidence weight. Assuming equal match voter weights, the confidence score simplifies to the following.

$$conf = \frac{\sum_{v \in V} | ms_v | \times ms_v}{\sum_{v \in V} | ms_v |} \tag{15}$$

Intuitively, equation (15) uses the match score returned by each match voter as its weight. This simplification works because a match score of zero indicates insufficient evidence to determine if the source element and target element match[2]. A score close

[2] As a special case, if the denominator is zero, the confidence score generated is also zero.

to ±1 indicates strong evidence either in support of a match, or against a match. Thus, by scaling each match voter as described in the previous section, we can easily merge match scores based on the strength of each match score.

In our ongoing example, the bag-of-words match voter generated a match score of 0.44. The bag-of-words with thesaurus match voter generated a match score of 0.55, and a match voter based on the edit distance between "how" and "targetsource" generated a match score –0.21 (the schema element names share only the letter "o"). Based on equation (15), the final confidence score is 0.38. The bag-of-words match voters are weighted more heavily because their match scores are more decisive. Recall that the match voters can generate large positive scores more easily than large negative scores. Given the behavior of the match voter, as long as any match voter suggests a match, the final confidence score will likely be positive.

We have incorporated all three match voters and the vote merger into the Harmony integration workbench, along with the Harmony GUI. Our customers and colleagues have been using this package for roughly one year. In the next section, we report on their experiences with the tool suite.

## 7 User Experiences

We released the original version of Harmony (including the GUI, match engine, and integration workbench) in November 2006. Since that time, the package has been used to support several government projects. We followed up with a half-dozen Harmony users to assess the extent to which Harmony has met their needs. In this section, we describe the lessons learned from these interviews. We first enumerate the questions that we have asked. We then provide a summary of these users' interactions with Harmony, both in terms of the GUI and the integration workbench. We conclude the section with a description of how the integration workbench has simplified the integration of Harmony with BEA's AquaLogic tool.

### 7.1 Background

We contacted several Harmony users, of which a half-dozen provided feedback on the tool suite. In each interview we asked the following questions.

- What can you tell us about the schemata in your application domain?
- What (if any) were the benefits of using Harmony over manual integration or other tools?
- Of which UI features were you aware, and which did you use?
- What issues or limitations did you experience?
- Did you interact with the integration workbench? If so, how difficult was it to use this framework?

In almost all cases, the schemata in question were very large, containing several thousand distinct schema elements. In one case, the schemata were OWL ontologies, one of which contained nearly 100,000 concepts.

The application domains ranged widely. For example, one scenario involved mapping XML message formats to a smaller "community of interest vocabulary"—i.e., a set of terms with text definitions all directly connected to a root node. The goal of this project was not to create an executable mapping, but instead to establish a data dictionary describing the elements common in the domain, including alternate formulations of these elements. A second scenario involved mapping the same set of source schemata to a collection of target schemata using just the match engine (i.e., without human intervention) to determine which target schema best covered the source schemata. A third scenario involved aligning a small highly-technical ontology with a large general-purpose ontology. The goal of this project was to merge the technical ontology into the general-purpose ontology to provide better domain coverage. Note that none of these projects were trying to generate executable transformations to generate target instances from source instances, which is the typical motivating application for schema integration research.

## 7.2 Match Engine and GUI Experiences

Not surprisingly for a research prototype, the Harmony match engine was unable to handle source and target schemata containing thousands of schema elements. Because the match engine evaluates a confidence score for every possible [source element, target element] pair, the match engine was unable to generate a complete mapping matrix in less than 24 hours (and in some cases would run out of memory). This limitation stresses the importance of match algorithms that do not need to consider all possible pairs (e.g., [32]).

As a workaround, in all but one case, the users identified external tools that could partition the schemata into smaller, more manageable, pieces. From their experiences, we can draw two conclusions. First, new tools should be added to the workbench that can partition a schema into smaller sub-schemata. Second, the GUI should make it clear that only the nodes currently selected (e.g., using the sub-tree filter) would be fed to the match engine. Only one user was aware of this strategy for handling large schemata.

The users with whom we spoke did use most of the GUI filters to explore the mapping matrix. In particular, we heard that the sub-tree filter was very helpful in focusing one's attention on a particular context. This feature allowed the integration engineer to verify the proposed matches, specifically within that context because the validity of a match depended on the context.

To identify these contexts, the integration engineer used a combination of the depth filter and the confidence filter. The depth filter eliminated the low-level details, leaving only high-level concepts used to establish a context. The confidence filter identified those contexts for which good matches could quickly be identified. Thus, our intuition that schema matching is an iterative process in which the integration engineer alternates between high-level and detailed views of the problem was validated.

## 7.3 Integration Workbench Experiences

In three cases, the users with whom we spoke modified Harmony directly. In the first case, a new match voter was created that parallelized the generation of match scores. This match voter also discarded any score that fell below a user-defined threshold to avoid the overhead of maintaining these scores in the blackboard. Once implemented, it was trivial to add this new match voter to Harmony, largely because the interactions between a match voter and the workbench were well-established.

To support ontology alignment, the GUI was extended to introduce an additional mapping cell annotation: relationship indicates the nature of the relationship between the source schema element and target schema element. This annotation could be used to indicate that the source element was more specific than (a subclass of), equivalent to, or compatible with the target element. In effect, this annotation made each mapping cell a reified relationship linking the source to the target. It took the integration engineer roughly 40 hours to extend the GUI and to link the new tool into the integration workbench. The integration engineer responsible indicated that he was quite pleased to see that the workbench correctly saved and loaded the new annotations along with the built-in annotations.

Finally, to determine which target schema best covered the source schemata, the integration engineers needed a new tool to display a mapping matrix in summary form. This tool generates a pie chart that indicates the percentage of source elements for which a good match (confidence score ≥ 0.75), weak match (≥ 0.25), or no match was found. Implementing this tool and linking it into the integration workbench took an integration engineer roughly 20 hours.

Although our experiences are limited, we believe that the integration workbench has proven to be an effective mechanism for adding new tools to the suite. This capability is particularly important because many of our users were not interested in generating executable code. In fact, several of them reported that they were unable to use commercial schema integration tools because the only possible end product generated by these tools is executable code. Our users' needs were more varied than could be supported by off-the-shelf tools.

However, we recognize that in many cases, the goal of schema integration is to generate an executable mapping. Towards that end, we have teamed with BEA to integrate the Harmony match engine with BEA's AquaLogic tool via the integration workbench.

## 7.4 Matching + Mapping

In [33] we describe our efforts to combine the Harmony match engine with BEA's AquaLogic tool, which we summarize here. Briefly, AquaLogic "employs a declarative foundation to enable a user to design, develop, deploy, and maintain a framework that understands both the logical and semantic heterogeneity of data sources." In the context of the integration workbench, AquaLogic provides a graphical interface so that an integration engineer can manually indicate semantic correspondences. The tool automatically proposes mapping snippets (largely type-conversions) based on the semantic correspondences. It then assembles these snippets into an executable transformation, optionally deploying this transformation in a service-oriented architecture.

Given the potential synergy between Harmony and AquaLogic, we have begun a joint effort to combine these tools. In the resulting product, the Harmony match engine will propose candidate matches; AquaLogic is responsible for providing a graphical user interface and for generating mappings/transformations. Moreover, given a library of source schemata and a target schema, Harmony can suggest source schemata that are likely to be relevant.

To make Harmony accessible to AquaLogic, we needed to implement two new functions. The first takes, as input, a source schema element and a target schema element, and computes the mapping matrix for the corresponding schema sub-trees. The function returns the top *k* [target element, confidence] pairs for each source element such that the confidence score exceeds some threshold. (The intention is to limit the amount of information presented to the user.) Implementing this functionality using the integration workbench required only four lines of code: 1) invoke the Harmony match engine, 2) determine which confidence scores to compute, 3) filter out any results that do not exceed the confidence threshold, and 4) add the top *k* matches for each source element to the result.

The second new function allows AquaLogic to indicate which correspondences have been accepted by the integration engineer. This tells Harmony which mapping cells should not be modified by future invocations of the match engine and could potentially be used to tune the algorithmic parameters. Implementing this method required three lines of code: 1) iterate over the set of manually identified matches, 2) lookup the corresponding cell of the mapping matrix, and 3) update the confidence score for that cell.

At this time, BEA is extending their graphical interface to display the results generated by the Harmony match engine. However, the ease with which the necessary information could be extracted from the blackboard via the integration workbench offers further proof that the workbench is an effective mechanism for integrating schema integration tools and that our task model correctly captures activities common to data integration.

## 8 Related Work

The data integration task model is an extension [7] of our prior work presented in [9]. The improved model includes additional subtasks addressed by real integration systems and identified as being important by three experienced integration engineers. A task model of schema integration also appears in [11], but that work predates the data integration industry and does not benefit from the insights of practitioners.

In [34], Haas describes a task model similar to ours. In her model, Haas includes four basic tasks: First, the integration engineer must understand the schemata (subtasks 1–2, above). Second, the integration engineer must standardize the underlying sources. This includes establishing a standard schema that specifies the syntax, structure and semantics of the information (subtasks 3–9). She also emphasizes the importance of determining how to a) identify information that pertains to the same subject (subtask 10) and b) handle missing or inconsistent information (subtask 11). Third, the developers must specify the execution engines and produce the executable (subtask 12). Finally, the solution must be executed (subtask 13).

Of the tasks pertaining to schema integration (subtasks 1–9), most of the research, including our own, has focused on subtask 3, schema matching (e.g., [4, 6, 25-28]). Overviews of the common approaches appear in [1] and [2]. Based on Rahm and Bernstein's hierarchy [1], the match engine (as a whole) is a composite matcher that composes the vote merger with a structure-level matcher. The vote merger, in turn, is a hybrid matcher that combines the match scores generated by a collection of element-level linguistic matchers (the match voters). However, we are aware of only one prior schema matching algorithm that exploits textual definitions [35], which uses a simple approach based on a commercial information retrieval tool. Harmony adds more sophisticated linguistic pre-processing (e.g., stemming), a thesaurus and scoring algorithms that consider the amount of evidence available. In [36] similar linguistic pre-processing techniques to ours are used, but are applied only to element names. In addition, instead of doing bag-of-word comparisons across elements of different schemata, they use natural language techniques to translate each name into a logical formula and then compare the logical formulae to perform match. This approach is complementary with techniques in use in Harmony.

Harmony provides the first GUI that supports an iterative development cycle. This GUI is the first to allow the integration engineer to filter the match results based on a variety of criteria.

The integration workbench is far from the first schema integration toolkit to adopt a modular architecture. A similar approach is used by both schema matching prototypes such as COMA++ [4] and Protoplasm [25] and commercial schema mapping tools such as those offered by IBM and BEA. For example, Protoplasm allows the integration engineer to string together match voters and vote mergers in arbitrary ways. This modularity allows the research group or commercial entity to adapt or extend their software. However, the integration workbench that is proposed in this paper is unique in that it is based on a common blackboard using open standards so that independently developed tools can interoperate.

## 9 Conclusions and Future Work

Data integration is a widely researched problem. However, we described ways in which enterprise data integration differs from the situations usually encountered in the research literature (e.g., documentation is widely available, instance data less so). Other pragmatic comments discussed how best to represent coding schemes so they can be leveraged by integration tools.

We also enumerated the subtasks involved in data integration, partitioned to reflect the behavior of integration engineers and the support provided by existing tools. This task analysis is intended to guide tool development and to enable comparisons across tools and integration problems.

Based on our observations and task modeling, we identified important design goals for integration tools. Specifically, we articulated the need to support all of the tasks involved in schema integration. One approach to meeting this need is to bring multiple tools to bear.

Unfortunately, assembling several tools to solve a particular integration problem is daunting. Our community needs to adopt the principle of assembling systems from modular components and integrating existing components. To facilitate tool interoperation, we proposed an open, extensible integration workbench. This architecture provides a unified view of schemata and mappings so that integration tools can more easily communicate. We believe that both tool vendors and database researchers benefit from this arrangement. We hope that this proposal will generate discussion that ultimately could lead to standards (e.g., for mapping matrices) for data integration tool interoperation.

Since our overarching goal is to improve the lives of integration engineers, our next task is to perform a usability analysis of the Harmony integration suite. We will measure the extent to which software tools save time on each of the schema integration subtasks.

## Acknowledgements

We would like to thank Ken Samuel for his work developing the Harmony match voters and vote merger, Mike Carey and Sachin Thatte for their input, Jeff Hoyt and Michael Morse for their editorial comments and the reviewers for their feedback.

## References

[1] Rahm, E., Bernstein, P.A.: A Survey of Approaches to Automatic Schema Matching. The VDLB Journal 10, 334–350 (2001)

[2] Shvaiko, P., Euzenat, J.: A Survey of Schema-Based Matching Approaches. Journal on Data Semantics 4, 146–171 (2005)

[3] Miller, R., Hernández, M.A., Haas, L.M., Yan, L., Ho, C.T.H., Fagin, R., Popa, L.: The Clio Project: Managing Heterogeneity. SIGMOD Record 30, 78–83 (2001)

[4] Aumueller, D., Do, H.H., Massmann, S., Rahm, E.: Schema and ontology matching with COMA++. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, MD (2005)

[5] Hammer, J., Garcia-Molina, H., Nestorov, S., Yerneni, R., Bruenig, M.M., Vassalos, V.: Template-Based Wrappers in the TSIMMIS System. In: Proceedings ACM SIGMOD International Conference on Management of Data, Tucson, AZ (1997)

[6] Doan, A., Domingos, P., Halevy, A.Y.: Learning to Match the Schemas of Databases: A Multistrategy Approach. Machine Learning 50, 279–301 (2003)

[7] Mork, P., Rosenthal, A., Seligman, L.J., Korb, J., Samuel, K.: Integration Workbench: Integrating Schema Integration Tools. In: InterDB 2006 Second International Workshop on Database Interoperability, Atlanta, GA (2006)

[8] Ullman, J.D.: Information Integration Using Logical Views. In: Afrati, F.N., Kolaitis, P.G. (eds.) ICDT 1997. LNCS, vol. 1186. Springer, Heidelberg (1997)

[9] Seligman, L.J., Rosenthal, A., Lehner, P.E., Smith, A.: Data Integration: Where Does the Time Go? IEEE Database Engineering Bulletin 25, 3–10 (2002)

[10] Ashish, N., Knoblock, C.A.: Wrapper Generation for Semi-structured Sources. SIGMOD Record 26, 8–15 (1997)

[11] Batini, C., Lenzerini, M., Navathe, S.B.: A Comparative Analysis of Methodologies for Database Schema Integration. ACM Computing Surveys 18, 323–364 (1986)

[12] Cluet, S., Delobel, C., Siméon, J., Smaga, K.: Your Mediators Need Data Conversion! In: SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, Seattle, WA (1998)

[13] Florescu, D., Levy, A.Y., Mendelzon, A.O.: Database Techniques for the World-Wide Web: A Survey. SIGMOD Record 27, 59–74 (1998)

[14] Pan, A., Raposo, J., Álvarez, M., Hidalgo, J., Viña, Á.: Semi-Automatic Wrapper Generation for Commercial Web Sources. In: Engineering Information Systems in the Internet Context, Kanazawa, Japan (2002)

[15] Papakonstantinou, Y., Gupta, A., Garcia-Molina, H., Ullman, J.D.: A Query Translation Scheme for Rapid Implementation of Wrappers. In: Ling, T.-W., Vieille, L., Mendelzon, A.O. (eds.) DOOD 1995. LNCS, vol. 1013. Springer, Heidelberg (1995)

[16] Popa, L., Velegrakis, Y., Miller, R., Hernández, M.A., Fagin, R.: Translating Web Data. In: VLDB 2002, Proceedings of 28th International Conference on Very Large Data Bases, Hong Kong, China (2002)

[17] Fagin, R., Kolaitis, P., Miller, R., Popa, L.: Data Exchange: Semantics and Query Answering. In: Calvanese, D., Lenzerini, M., Motwani, R. (eds.) ICDT 2003. LNCS, vol. 2572. Springer, Heidelberg (2003)

[18] Fernandez, M.F., Tan, W.-C., Suciu, D.: SilkRoute: Trading between Relations and XML. In: Ninth International World Wide Web Conference, Amsterdam, The Netherlands (2000)

[19] Rys, M.: Bringing the Internet to Your Database: Using SQL Server 2000 and XML to Build Loosely-Coupled Systems. In: Proceedings of the 17th International Conference on Data Engineering, Heidelberg, Germany (2001)

[20] Wyss, C.M., Robertson, E.L.: Relational Languages for Metadata Integration. ACM Transactions on Database Systems 30, 624–660 (2005)

[21] Goh, C.H., Bressan, S., Madnick, S.E., Siegel, M.: Context Interchange: New Features and Formalisms for the Intelligent Integration of Information. ACM Transactions on Information Systems 17, 270–293 (1999)

[22] Sciore, E., Siegel, M., Rosenthal, A.: Using Semantic Values to Facilitate Interoperability Among Heterogeneous Information Systems. ACM Transactions on Database Systems 19, 254–290 (1994)

[23] Koudas, N., Sarawagi, S., Srivastava, D.: Record Linkage: Similarity Mesaures and Algorithms. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, Chicago, IL (2006)

[24] Johnson, T., Dasu, T.: Data Quality and Data Cleaning: An Overview. In: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San Diego, CA (2003)

[25] Bernstein, P.A., Melnik, S., Petropoulos, M., Quix, C.: Industrial-Strength Schema Matching. SIGMOD Record 33, 38–43 (2004)

[26] Do, H.H., Rahm, E.: COMA - A System for Flexible Combination of Schema Matching Approaches. In: VLDB 2002, Proceedings of 28th International Conference on Very Large Data Bases, Hong Kong, China (2002)

[27] Madhavan, J., Bernstein, P.A., Rahm, E.: Generic Schema Matching with Cupid. In: VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases, Rom, Italy (2001)

[28] Melnik, S., Garcia-Molina, H., Rahm, E.: Similarity Flooding: A Versatile Graph Matching Algorithm. In: Proceedings of the 18th International Conference on Data Engineering, San Jose, CA (2002)

[29] Brickley, D., Guha, R.: RDF Vocabulary Description Language 1.0: RDF Schema. In: World Wide Web Consortium (W3C®) (2003)

[30] Stenbit, J.P.: Department of Defense Net-Centric Data Strategy (2003)

[31] Ilyas, I.F., Markl, V., Haas, P.J., Brown, P., Aboulnaga, A.: CORDS: Automatic Discovery of Correlations and Soft Functional Dependencies. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, Paris, France (2004)

[32] Mork, P., Bernstein, P.A.: Adapting a Generic Match Algorithm to Align Ontologies of Human Anatomy. In: Proceedings of the 20th International Conference on Data Engineering, ICDE 2004, Boston, MA (2004)

[33] Carey, M.J., Ghandeharizadeh, S., Mehta, K., Mork, P., Seligman, L.J., Thatte, S.: AL$MONY: Exploring Semantically-Assisted Matching in an XQuery-Based Data Mapping Tool. In: International Workshop on Semantic Data and Service Integration, Vienna, Austria (2007)

[34] Haas, L.M.: Beauty and the Beast: The Theory and Practice of Information Integration. In: Schwentick, T., Suciu, D. (eds.) ICDT 2007. LNCS, vol. 4353. Springer, Heidelberg (2007)

[35] Clifton, C., Housman, E., Rosenthal, A.: Experience with a Combined Approach to Attribute-Matching Across Heterogeneous Databases. In: Data Mining and Reverse Engineering: Search for Semantics, IFIP TC2/WG2.6 Seventh Conference on Database Semantics (DS-7), Leysin, Switzerland (1997)

[36] Giunchiglia, F., Shvaiko, P., Yatskevich, M.: S-Match: an Algorithm and an Implementation of Semantic Matching. In: Bussler, C.J., Davies, J., Fensel, D., Studer, R. (eds.) ESWS 2004. LNCS, vol. 3053. Springer, Heidelberg (2004)