

# Integrating Data from Disparate Sources: A Mass Collaboration Approach

Robert McCann, Alexander Kramnik, Warren Shen, Vanitha Varadarajan, Olu Sobulo, AnHai Doan

University of Illinois, USA

{rlmccann, kramnik, whshen, varadara, sobulo, anhai}@cs.uiuc.edu

The rapid growth of distributed data at enterprises and on the WWW has fueled significant interest in building data integration systems. Such a system provides users with a uniform query interface (called *mediated schema*) to a multitude of data sources, thus freeing them from manually querying each individual source.

Figure 1 illustrates a data integration system that helps users find houses on the real-estate market. Given a user query over the mediated schema, the system uses a set of *semantic mappings* to translate it into queries over the local schemas of the data sources. Next, it executes the queries using wrapper programs attached to the sources, then combines and returns the results to the user.

Today, constructing such a data integration system requires the *system builder* to execute a series of *tasks*, such as finding data sources (e.g., on the Web), creating the mediated schema, constructing wrappers, and matching the mediated schema with the schemas of the sources. Sources often change over time. Thus, after the system is deployed, the builder must monitor it continuously, to detect and repair system components (e.g., wrappers, mappings) that become broken due to changes.

The above tasks are well-known to be very labor intensive. Hence, the problem of automating them has received much attention and numerous semi-automatic tools have been developed (e.g., [5]). However, current integration tools still have limited accuracy. Thus, even with their help, the system builder must still spend an enormous amount of labor executing these tasks. This in turn has incurred exorbitant costs of ownership for data integration systems, and severely limited their deployment in practice. Today, at enterprises, where data integration is frequently a must, it is carried out at a tremendous cost, often at 35% of the IT budget [2]. On the Web, where data integration systems can greatly simplify the search for information, there are currently few such systems and at limited scales. Indeed, large-scale or long-running data integration systems often cannot be built because the construction and maintenance workload quickly overwhelms the builder (or even a team of builders).

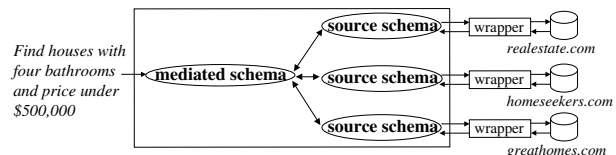


Figure 1. A data integration system in the real estate domain.

## The MOBS Project

To address the above problems, in the MOBS (Mass Collaboration to Build Systems) project at the University of Illinois we are developing solutions that learn from the multitude of users in the integration environment to improve the accuracy of integration tools. The improved accuracy in turn can significantly reduce the workload of the system builder. In developing MOBS we address the following key challenges:

**Obtaining User Participation:** The first challenge of MOBS is to secure user participation. We show that in many data integration scenarios, including intra-organization, inter-organization, and online communities, the system builder can naturally recruit participants. For instance, within an organization the future users of the system are often willing to help, especially given that they are the data experts.

In addition to recruiting *willing users*, MOBS can also obtain user participation via “*payment*” schemes, in which users “pay” to use certain services. For example, in certain settings the builder can design an integration system such that whenever a user of the system poses a query, he or she must “pay” for it, by answering a simple question before being allowed to see the query result. The answers to such questions are then used to help the builder maintain and expand the system. This scheme is reminiscent of similar “payment” schemes for online news services (e.g., *salon.com*) and for eliminating “freeloaders” from peer-to-peer systems.

**Learning from User Participation:** In the next step, given a semi-automatic tool  $T$  for an integration task  $K$ , we consider how to modify the tool to learn from the above users. Since most such users are “data integration illiterate”,

we modify  $T$  to ask them relatively simple questions, which have low cognitive load and can be answered quickly. The questions can help  $T$  in many aspects, such as gathering additional training data, soliciting simple domain constraints that  $T$  can utilize, and verifying intermediate as well as final predictions that  $T$  makes. By utilizing answers to these questions,  $T$  can increase its accuracy, thereby reducing the builder's workload on task  $K$ . The following example illustrates the approach.

Consider building an integration system such as the one in Figure 1. A builder may begin by deploying a tool which crawls the Web to find query interfaces into real-estate databases. Current tools however often return many false positives, forcing the builder to sift through a large number of forms to find desired query interfaces. To reduce the number of false positives, such a tool can be modified so that when it discovers a form  $F$ , instead of immediately showing  $F$  to the builder, it shows  $F$  to the users and asks, "Is  $F$  a query interface into real-estate listings?" Intuitively, if enough users say "no", then the tool can conclude with high confidence that  $F$  is not a real-estate query interface. It can then drop  $F$ , removing unnecessary work for the builder.

If enough users say "yes", then the tool presents  $F$  to the builder. After verifying that  $F$  is indeed about real-estate listings, the builder may employ another tool to construct a wrapper for the data source (represented by)  $F$ . Current wrapper tools are frequently brittle, in part because they often must search a huge space of possible wrappers, and make decisions with insufficient information [1]. In such cases, a tool can ask users questions such as "Is this text fragment a part of the data or a part of the wrapper template?", then use the answers to significantly cut down the search space and build more accurate wrappers.

Next, the builder may want to match the schema of source  $F$  with the mediated schema  $M$ . Here, the employed matching tool can be modified to learn from the users in many ways. For instance, it can learn simple domain integrity constraints. From examining the values of attributes `lot-area` and `house-size` of  $M$ , it can ask the question "Is `lot-area` always greater than `house-size`?" A confirmative answer from the users will result in an integrity constraint that the tool can use to disambiguate matches. The constraint can also be re-used in subsequent matching tasks. As another example, the tool can ask users to verify its matches. Suppose it predicts that `house-size` matches `lot-size`, then the match can be passed to the users to be verified.

Finally, once the system has been deployed, the builder may employ a tool to detect and repair broken wrappers. Several such tools have been developed (*e.g.*, [3]). The tools however have a high rate of false alarms, thus making the monitoring task of the builder very labor intensive. To reduce this workload, the tools can be modified so that a non-

urgent false alarm will first be verified by the users (*e.g.*, by showing the output of the wrapper and asking if it is displaying garbage data). Only when enough users say "yes" is the alarm presented to the builder.

We have considered the tasks of source discovery and schema matching (including both 1-1 and complex matching), and developed solutions that modify tools to these tasks to learn from users.

**Combining User Answers:** In the final challenge, since users can be ignorant or downright malicious (especially for Internet applications), it is important that we be able to solicit multiple answers for each question  $Q$  (that a tool asks), then merge the noisy answers into a correct answer for  $Q$  with high probability. We have developed a solution to this problem, which employs questions with known answers to evaluate the reliability of each user, then combines user answers based on their reliability. We formulate the solution in a probabilistic setting based on dynamic Bayesian networks, and provide theoretical guarantees for many common integration cases.

We have evaluated MOBS with extensive real-world and simulation experiments that demonstrate the utility of the approach. On the integration tasks of source discovery and schema matching, MOBS improves tool accuracies by 9 - 60%. The accuracy gain in turn reduces the builder workload by 29 - 88%. These experiments show that users have low workload, that they can answer questions quickly, and that their answers are useful. The experiments also show that we can construct some simple, ongoing systems on the Web that require very little workload from the system builder. Our initial work on MOBS is presented in [4], and the work described above is reported in detail in an upcoming technical report.

For future work, we plan to extend MOBS to other tasks (*e.g.*, wrapper construction, system maintenance), and provide more extensive real-world evaluation. We are also studying settings in which data integration systems can be designed so that they can learn from traces of user activities to improve integration accuracy.

## References

- [1] V. Crescenzi, G. Mecca, and P. Merialdo. RoadRunner: Towards automatic data extraction from large web sites. In *VLDB*, 2001.
- [2] C. Knoblock and S. Kambhampati. Tutorial on information integration on the web. In the *Nat. Conf. on AI (AAAI)*, 2002.
- [3] K. Lerman, S. Minton, and C. A. Knoblock. Wrapper maintenance: a machine learning approach. In *Journal of AI Research*, 2003.
- [4] R. McCann, A. Doan, A. Kramnik, and V. Varadarajan. Building data integration systems via mass collaboration. In *SIGMOD WebDB*, 2003.
- [5] E. Rahm and P. Bernstein. On matching schemas automatically. In *VLDB Journal*, 10(4), 2001.