UQBE: Uncertain Query By Example for Web Service Mashup

Junichi Tatemura[1] Songting Chen[1] Fenglin Liao[2]
Oliver Po[1] K. Selcuk Candan[1] Divyakant Agrawal[1]
[1]NEC Laboratories America, 10080 North Wolfe Road, Suite SW3-350, Cupertino, CA 95014
[2]Department of Computer Science, University of California, Santa Barbara, Santa Barbara, CA 93016
{tatemura,songting,oliver,candan,agrawal}@sv.nec-labs.com,
fenglin@cs.ucsb.edu

ABSTRACT

The UQBE is a mashup tool for non-programmers that supports query-by-example (QBE) over a schema made up by the user without knowing the schema of the original sources. Based on automated schema matching with uncertainty, the UQBE system returns the best confident results. The system lets the user refine them interactively. A tuple in the query result is associated with *lineage* that is a boolean formula over schema matching decisions representing underlying conditions on which the corresponding tuple is included in the result. Given binary feedbacks on tuples by the user, which are possibly imprecise, the system solves it as an optimization problem to refine confidence values of matching decisions. The demo features graphical user interaction on the UQBE system, including querying and refinement.

Categories and Subject Descriptors: H.2.4 [Database Management]: Systems

General Terms: Design, Human Factors

Keywords: web service, uncertain query, query by example, data integration, lineage

1. INTRODUCTION

Mashups are web applications that combine information from several sources, typically provided through simple Web APIs (i.e. web services). Recently several development tools have been proposed to enable the user to compose a mashup without programming language such as JavaScript. However, we claim that such visual development tools still leave fundamental difficulties for non-programmers when they want to create a mashup from new data sources: The user needs to understand schema and semantics for each data source.

We cannot expect non-programmers would read API documents on data sources to compose a mashup (or query). It is easier for them to understand data semantics from concrete data (i.e., *by example*) rather than abstract specification on the schema. A problem is that concrete data (i.e. query results) is available *after* a query is composed and issued.

The UQBE (pronounced as "*you-cube*") is a mashup tool for non-programmers that supports query-by-example (QBE). Unlike traditional QBE systems, a query is represented on a schema made up by the user, who do not know the schema of the original sources. Based on automated schema matching with uncertainty, the UQBE system returns the best confident results and lets the user refine them interactively.

Our approach is along with the concept of dataspaces [5]: *pay-as-you-go* data management dropping the requirement of full semantic integration of the sources from the first step. Our main contribution compared to existing work on data integration with uncertainty [4] is interactive refinement through user feedbacks by leveraging lineages [3]. Whereas a lineage in [3] represents the source of uncertainty in data, ours represents the one in metadata (i.e., schema matching). Enabled by this technique, the demo features our unique approach to web service mashup through query-by-example.

2. MOTIVATING EXAMPLE

To illustrate our demo, we introduce the following example scenario, where a non-programmer wants to create a mashup over multiple data sources.

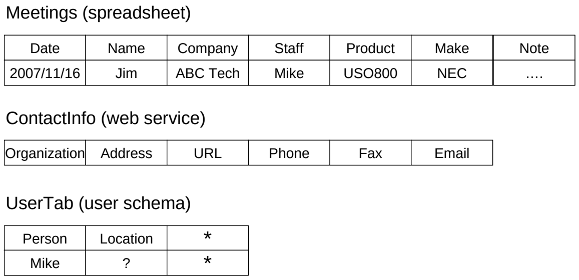
EXAMPLE 1. *A manager in a sales force department plans to visit Silicon Valley. She wants to know recent meeting activities of her group with customers around there. Among the group members, she is especially interested in Mike's meetings. In a shared repository, she found a spreadsheet file listing recent meetings. She wanted to visualize the data on Google Map. However she does not know how to write a script to plot data on the map. Besides, the contact information such as addresses of customers is only available through Web service API of the company's CRM system.*

Figure 1 shows the schema of data sources, a spreadsheet `Meetings` and a CRM web service `ContactInfo` as well as the user's schema on desirable output `UserTab`. The asterisk (*) in the user schema means the user wants any associated information as is. The tuple `(Mike,?,*)` in `UserTab` represents a query-by-example based on the user schema.

3. UNCERTAIN QUERY BY EXAMPLE

3.1 Data Sources

SIGMOD'08, June 9–12, 2008, Vancouver, BC, Canada.
Copyright 2008 ACM 978-1-60558-102-6/08/06 ...$5.00.



Meetings (spreadsheet)

Date	Name	Company	Staff	Product	Make	Note
2007/11/16	Jim	ABC Tech	Mike	USO800	NEC	....

ContactInfo (web service)

Organization	Address	URL	Phone	Fax	Email

UserTab (user schema)

Person	Location	*
Mike	?	*

Figure 1: Schema Example

The user chooses a source s_i from a source repository $S = \{s_i\}$. The current system assumes that each source has a relational schema, which may be automatically extracted from the content or explicitly given from a user who registers the source. Supported source types include web services and spreadsheets (See Section 5 for details). A web service is represented as a relation with limited access patterns [7].

A schema consists of a set of attributes. An attribute optionally has a type such as `Number`, `Time`, `Text`. A type information is useful for schema matching as well as inference of selection predicates in the query as described later. An attribute type is inferred from instance data or specified explicitly by the user who registers a source. Notice that, although schema is required for each source, the user does not have to understand it to issue a query.

3.2 User Query

A user query is a triple $Q =< \Gamma, \sigma, \pi >$ where source selection Γ, a user form σ, and output parameters π are defined as follows.

Source selection.

Data sources are selected and connected together as a graph $\Gamma =< V, E >$ where vertices V represent a set of selected sources $\{s_1, \cdots, s_n\} \subset S$, and edges $E = \{e_1, \cdots, e_k\}$ represent binary join relationships between two sources in V.

User form.

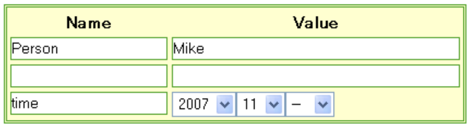
The user can specify selection criteria as attribute-value pairs $(< a_1, v_1 >, \cdots, < a_m, v_m >)$, which are interpreted as conjunction of predicates $p_i(a_i, v_i)$:

$$\sigma = p_1(a_1, v_1) \wedge \cdots \wedge p_n(a_m, v_m)$$

The semantics of p_i depends on the type of attribute a_i. Figure 2 illustrates an example of the user form, where two attribute-value pairs are specified in different format. By using a *date* selector for the value "`2007-11`," the user naturally specifies that the attribute "`time`" has `Time` type, and the system accordingly applies temporal range selection as a predicate. On the other hand, the value "`Mike`" is treated as *keyword* selection for `Text` data. Advanced users may further use *search options* in the same manner they do for typical Web search engines. For example, the value "`< 10`" is interpreted as an inequality predicate (<) with a `Number` value (10). The type of an attribute is also inferred through the schema matching process described later.

Output parameters.

The system supports various ways to visualize the query results, including a *table view*, a *map view*, and a *calendar view*. Each view needs a specific parameter to visualize a tuple except the most generic *table view*. For example, a map needs `Location`, and a calender needs `Date`. When the user chooses a specific view, the corresponding parameters are implicitly specified as a part of the user query. In general, the user specifies, explicitly or implicitly, a set of attributes to be used for output construction. $\pi = \{a_{m+1}, \cdots, a_{m+l}\}$.



Name	Value
Person	Mike
time	2007 11 –

Figure 2: User Form

In order to specify the query in Figure 1, the user can fill the user form with a pair <`Person`, "Mike"> and choose a map view (i.e., <`Location`,?>).

3.3 Uncertain Schema Matching

The query Q can be seen as a query-by-example $\{v_1, \cdots, v_m\}$ over a table with schema $\{a_1, \cdots, a_{m+l}\}$. In order to execute this as a concrete query, the system needs to identify mapping from the schema of multiple sources $\{s_1, \cdots, s_n\}$ to the user schema $\{a_1, \cdots, a_{m+l}\}$ given join relationships $\{e_1, \cdots, e_k\}$. Such mapping is given as a *schema match* $\mathcal{M} = \{\mu^1, \cdots, \mu^{m+l+k}\}$ where μ^i denotes an *attribute match* for the i-th element of $\{a_1, \cdots, a_{m+l}, e_1, \cdots, e_k\}$. In the example scenario in Figure 1, possible attribute matches for join between `Meetings` and `ContactInfo` are

$$\texttt{Meetings.Company} = \texttt{ContactInfo.Organization}$$
$$\texttt{Meetings.Make} = \texttt{ContactInfo.Organization}$$

Although the current system only supports a join with a single attribute pair, this framework can incorporate a join with multiple attribute pairs.

We use an automated schema matching engine that generates possible attribute matches M_i for every μ^i.

$$M_i = \{\mu^i_1, \cdots, \mu^i_{n_i}\}$$

Then an *uncertain schema match* is given as $\tilde{\mathcal{M}} = \{M_i\}$.

A *decision* on μ^i refers to choosing one μ^i_j from M_i, and we assume that decisions are mutually independent: for every $\mu_i \in M_a$ and $\mu_j \in M_b$, the probability $P(\mu_i \wedge \mu_j) = 0$ if $a = b, i \neq j$; $P(\mu_i)P(\mu_j)$ if $a \neq b$.

The schema matching engine also generates confidences on μ^i_j as a confidence function $c : \bigcup_{\tilde{\mathcal{M}}} M_i \to [0, 1]$ such that $\sum_{\mu \in M_i} c(\mu) = 1$. The confidence conceptually corresponds to the probability mentioned above, but does not necessarily represent the actual probability.

3.4 Uncertain Query and Lineage

We take a "possible world" approach similar to [4] for query execution under uncertain schema match $\tilde{\mathcal{M}}$. A (certain) schema match $\mathcal{M}$ is determined given decisions on all μ^i. It is then applied to rewrite Q into an SPJ query Q_j, which is called a *possible query*. Then the semantics of the uncertain query Q is given as union of all possible queries: $Q = Q_1 \cup \cdots \cup Q_N$ where $N = \prod_i |M_i|$.

Note that, in this context, *possible* does not mean *executable*. When an uncertain query is applied over web services, there can be a possible query that has no executable plan since a web service API restricts ways to access the data source. $\tilde{\mathcal{M}}$ did not take such cases in consideration in order to preserve independence between decisions. Interpretation on this issue in our possible world model is as follows: a possible query represents the user's possible intension which may or may not be satisfied. A possible but non-executable query is ignored at run time and does not cause failure of the entire uncertain query. It is our future work to determine if we should let the user know difference between failure and successful empty result of possible queries.

A tuple in the query result is associated with *lineage* (or provenance) that is a boolean formula over matchings $\{\mu\}$ representing underlying conditions on which the corresponding tuple is included in the result. In the Trio system [3], lineage is used to trace the uncertainty back to the source data. The UQBE system uses lineage to trace the uncertainty back to the metadata, i.e., the schema matches. We will show that such lineage plays a vital role to refine the query result based on users' interaction.

Given a lineage λ and the confidence scores $\{c(\mu)\}$, the score $s(\lambda)$ is derived in the same manner as deriving probability. For $\mu_i \in M_a$ and $\mu_j \in M_b$, scores are derived as follows:

$$\begin{aligned} \mathrm{s}(\mu_i) &= c(\mu_i) \\ \mathrm{s}(\mu_i \wedge \mu_j) &= \begin{cases} 0 & \text{if } a = b; \\ c(\mu_i)c(\mu_j) & \text{if } a \neq b; \end{cases} \\ s(\mu_i \vee \mu_j) &= s(\mu_i) + s(\mu_j) - s(\mu_i \wedge \mu_j) \end{aligned}$$

The query result is then ranked based on the lineage scores.

3.5 Approximate Top-K Query Processing

Similarly to [4], we are only interested in k most highly ranked tuples as a query result. Unlike [4], unfortunately, the number of possible queries is exponential. In practice, since a query is interactively composed by the user (i.e., the user can give feedback for disambiguation before the query becomes large), the number of uncertain matching will not be too large for *enumeration* of possible queries. However, *execution* of them would be still expensive.

In addition to k_t as the limit on the number of tuples, we take another number k_q as the limit on the number of possible queries to be taken in consideration. Adopting the previous algorithm [4], the query engine executes a possible query with higher score earlier and maintains the upper and lower bounds of the final score on each tuple in the intermediary data in order to capture an opportunity for early termination with k_t tuples. In our case, we may stop execution even earlier when the number of executed queries reaches k_q. As a result, top-k ranking may not be fully guaranteed.

Let us denote the exact top-k result as R_{k_t} and the approximate result as $R_{k_t}^{k_q}$. The query engine returns not only $R_{k_t}^{k_q}$ but also additional information on two other sets $\hat{R}$ and $\tilde{R}$, where $\hat{R}$ denotes a set of tuples that are guaranteed to be in R_{k_t}, $\tilde{R}$ denotes a set of tuples that may be in R_{k_t} and are not in $\hat{R}$, and $\hat{R} \subseteq R_{k_t}^{k_q} \subseteq \hat{R} \cup \tilde{R}$. $\tilde{R}$ is said to be *unbounded* if it is possible that a tuple in $(Q_{k_q+1} \cup Q_N) \setminus (Q_1 \cup Q_{k_q})$ (i.e. a tuple unseen in the current intermediary result) is in R_{k_t}. The query engine returns the numbers $|\hat{R}|$ and $|\tilde{R}|$ as indexes to represent uncertainty of $R_{k_t}^{k_q}$ with respect to the exact R_{k_t}.

4. INTERACTIVE REFINEMENT

4.1 Decision-level Feedback

When the UQBE system visualizes a tuple in the result, it utilizes the corresponding lineage to explain how the tuple is derived. The user can tell the system if the assumptions under the explanation (i.e., decisions on μ's) are correct. These feedbacks provide ground truth (1 or 0) on the confidence values $\{c(\mu)\}$. When there is a conflict (e.g. the user gives 1's on multiple μ's in the same M_i), more recent (newer) feedbacks are taken in order to let the user interactively explore possible queries.

4.2 Tuple-level Feedback

The user interaction for feedback on lineage explanation can be complicated in some cases (e.g., boolean conjunction/disjunction can be confusing for non-programmers). The system alternatively lets the user give an overall decision, either 1 (true) or 0 (false), on a tuple (i.e., lineage). Since a lineage is a boolean formula over matchings $\{\mu\}$, the refinement by this tuple-level feedback can be seen as the boolean satisfiabiliy problem. However, this problem can be very expensive and the user may not give satisfiable feedback. Instead, we see this refinement as an optimization problem.

Let a user feedback on a lineage λ be denoted as $t(\lambda)$ ($\in \{0, 1\}$). Given feedbacks $\{t(\lambda_j)\}$, the problem is to find the confidence values $\{c(\mu)\}$, such that $F = \sum |s(\lambda_j) - t(\lambda_j)|$ is the minimum, where $s(\lambda_j)$ is the score of λ_j derived from $\{c(\mu)\}$.

We adopt a simple and fast iterative algorithm, called *coordinate descent* [6]. Assume that F contains the confidence values of k distinct μ's, i.e., $F = f(c_1, \cdots, c_k)$. Given the search space $c_1 \times \cdots \times c_k$, coordinate descent applies the gradient descent algorithm for one coordinate at a time repeatedly until it reaches a local minimum. If we see only one confidence value c_i (i.e. a coordinate) as a variable, F becomes in a form of $a_i c_i + b_i$. Thus, coordinate descent is easily applicable with additional care of normalization of the confidence values.

One issue is that the number of user feedbacks is typically small. Moreover, the user is more likely to give negative feedbacks since the user spends time on interaction when unsatisfied with the result. This often results in over-discounting of the confidence values originally given by the schema matching engine. Our remedy is to interpret silence as agreement: if a tuple is shown to the user but does not get any feedback, the current score (typically less than 1) is used as a feedback value.

4.3 Preserving User Preferences

Besides the feedbacks described above, the user often modifies a query itself. The user may replace a spreadsheet with another to expect similar results on different data. The user may add another web service to the current query to associate further information. The confidence scores updated by the user feedbacks are maintained and reused for such modified queries as user preference in the current context.

5. SYSTEM ARCHITECTURE

Figure 3 illustrates the architecture of the UQBE system, which is implemented as middleware over web services. The entire system consists of three tiers connected through Web service APIs: the *client*, *mediator*, and *data sources.*

The Client UI, JavaScript code running on a Web browser, manages user interaction and visualizes query results on a table view, a map view, and a calendar view. It establishes a session with the mediator to let the user keep refining the query results. Within a session, the Query Execution Engine in the mediator preserves the query and feedbacks (as well as the confidence scores derived from the feedbacks). The Query Execution Engine consults with the Schema Matching Engine to rewrite the user query. Given user feedbacks, it reports the refined confidence scores back to the Schema Matching Engine, which accumulates these scores in the matching logs. It is our future work to make use of the matching logs to improve schema matching in later sessions.

The current system supports the following data sources: (1) a simple (REST-based) web service that returns a list of objects with attribute-value pairs as their child elements, and (2) a spreadsheet that has a tuple in each row. A spreadsheet must be made available through web service API known to the mediator. A web service must be registered on the Service Directory by one of the users, called a *publisher*. At registration time, mapping between the XML output and its relational representation is automatically identified, but the system also lets a knowledgeable publisher to correct the mapping. Note that other users except the publisher do not have to know such details. Once a directory is prepared, the users can browse and choose these data sources visualized as icons without looking into detailed schema.

The user can register a query in the Query Registry for later use. Once it is registered, any client application can call this query through Web service API. This Web service can also be used as a data source of another mashup query.

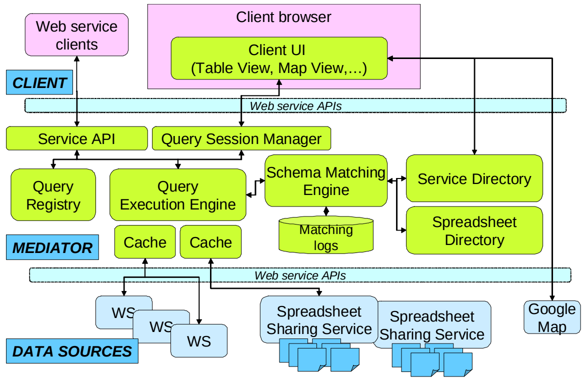
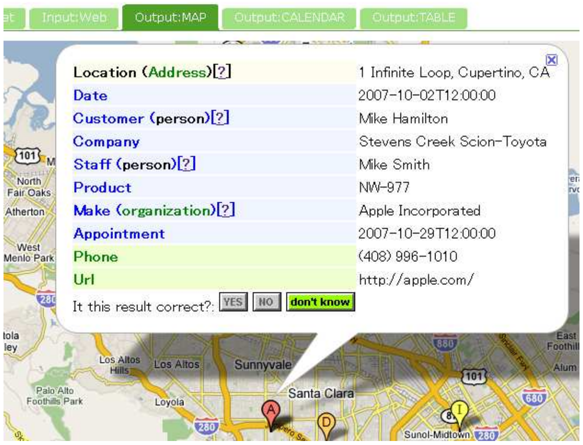


Figure 3: System Architecture

6. DEMO SCENARIO

The demo features user interaction on the UQBE system, including querying and refinement. Figure 4 shows a tuple visualized on a map for the query in Figure 1. Each icon corresponds a tuple in the result. The difference of their colors shows the difference of lineages, visualizing uncertainty in the results. A pop-up shows the content of a tuple. The color of each attribute indicates the original source. An attribute match is displayed as "`Customer (person) [?]`," for example. This means that the user query <`person`, "`Mike`"> matched with the attribute `Customer`.

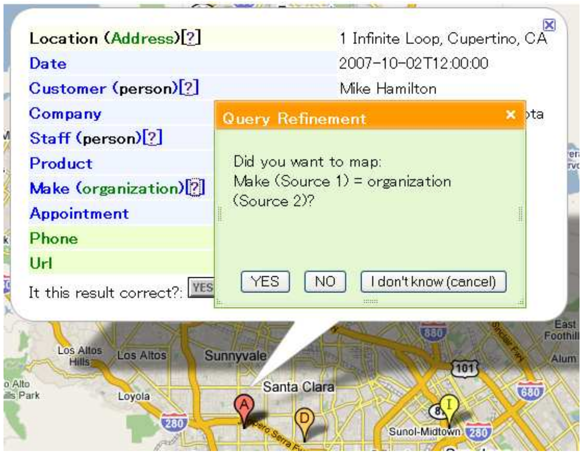
The user can give a tuple-level feedback by choosing "*yes*," "*no*," or "*don't know*" for each tuple. In this example, a data is plotted on the map using the address of the product company (`Make`) instead of the customer company (`Company`). If it does not look like an appropriate answer, the user can give "*no*" to this tuple. Alternatively, the user can give more detailed, decision-level feedbacks: Clicking `[?]` leads the user more detailed explanation in a dialog window (Figure 5), where the user can give "*no*" to the decision on mapping between `Make` and `organization` for a join condition. In either case, the user will get refined ranking results.



Location (Address)[?]	1 Infinite Loop, Cupertino, CA
Date	2007-10-02T12:00:00
Customer (person)[?]	Mike Hamilton
Company	Stevens Creek Scion-Toyota
Staff (person)[?]	Mike Smith
Product	NW-977
Make (organization)[?]	Apple Incorporated
Appointment	2007-10-29T12:00:00
Phone	(408) 996-1010
Url	http://apple.com/

It this result correct?: YES NO don't know

Figure 4: Map view: The map shows details of tuple `A` (the alphabetical order represents ranking). Attributes from two sources are colored differently.



Location (Address)[?]	1 Infinite Loop, Cupertino, CA
Date	2007-10-02T12:00:00
Customer (person)[?]	Mike Hamilton
Company	
Staff (person)[?]	
Product	
Make (organization)[?]	
Appointment	
Phone	
Url	

It this result correct?: YES

Query Refinement

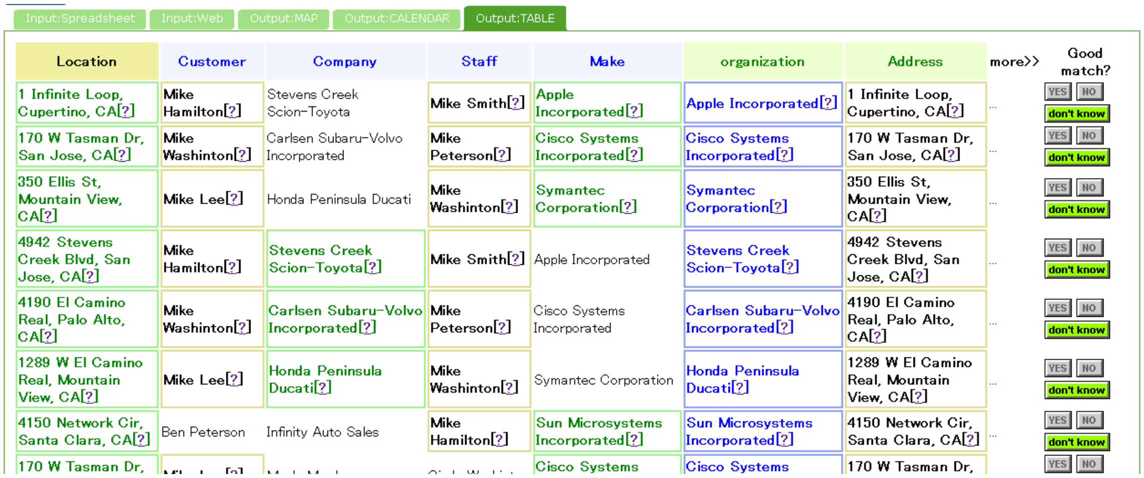
Did you want to map:
Make (Source 1) = organization
(Source 2)?

YES NO I don't know (cancel)

Figure 5: Decision-level feedback: the popped dialog window shows mapping between attributes ("`Make`" and "`organization`") in two sources, which is used as a join condition. The user can answer whether this mapping is correct.

After this feedback, the user adds another selection condition or replaces the spreadsheet with another one that contains similar data. The system reuses the feedbacks for the previous query as much as possible until the user resets the session or gives new feedbacks that conflicts with the old ones.

We notice that user interaction for decision-level feedbacks



Location	Customer	Company	Staff	Make	organization	Address	more>>	Good match?
1 Infinite Loop, Cupertino, CA[?]	Mike Hamilton[?]	Stevens Creek Scion–Toyota	Mike Smith[?]	Apple Incorporated[?]	Apple Incorporated[?]	1 Infinite Loop, Cupertino, CA[?]	...	YES NO don't know
170 W Tasman Dr, San Jose, CA[?]	Mike Washinton[?]	Carlsen Subaru–Volvo Incorporated	Mike Peterson[?]	Cisco Systems Incorporated[?]	Cisco Systems Incorporated[?]	170 W Tasman Dr, San Jose, CA[?]	...	YES NO don't know
350 Ellis St, Mountain View, CA[?]	Mike Lee[?]	Honda Peninsula Ducati	Mike Washinton[?]	Symantec Corporation[?]	Symantec Corporation[?]	350 Ellis St, Mountain View, CA[?]	...	YES NO don't know
4942 Stevens Creek Blvd, San Jose, CA[?]	Mike Hamilton[?]	Stevens Creek Scion–Toyota[?]	Mike Smith[?]	Apple Incorporated	Stevens Creek Scion–Toyota[?]	4942 Stevens Creek Blvd, San Jose, CA[?]	...	YES NO don't know
4190 El Camino Real, Palo Alto, CA[?]	Mike Washinton[?]	Carlsen Subaru–Volvo Incorporated[?]	Mike Peterson[?]	Cisco Systems Incorporated	Carlsen Subaru–Volvo Incorporated[?]	4190 El Camino Real, Palo Alto, CA[?]	...	YES NO don't know
1289 W El Camino Real, Mountain View, CA[?]	Mike Lee[?]	Honda Peninsula Ducati[?]	Mike Washinton[?]	Symantec Corporation	Honda Peninsula Ducati[?]	1289 W El Camino Real, Mountain View, CA[?]	...	YES NO don't know
4150 Network Cir, Santa Clara, CA[?]	Ben Peterson	Infinity Auto Sales	Mike Hamilton[?]	Sun Microsystems Incorporated[?]	Sun Microsystems Incorporated[?]	4150 Network Cir, Santa Clara, CA[?]	...	YES NO don't know
170 W Tasman Dr,				Cisco Systems	Cisco Systems	170 W Tasman Dr,		YES NO

Figure 6: Table view: each row represents a tuple. Values used in mapping are highlighted with corresponding colors. Attributes that are not used in mapping at all are omitted as "more>>."

is similar to debugging and designing of schema mapping with interactive tools such as SPIDER [2] and MUSE [1]. Especially, MUSE provides the user with data instances to disambiguate schema mappings. We share the same intuition with MUSE, that is, refinement of schema matching should be much easier to understand with concrete data (i.e., *by example*). The key difference is following: whereas these tools help a developer at a design time to complete schema mappings, our system helps a user at a query time to improve the results until the user is satisfied (along with the "*pay-as-you-go*" principle [5]). In our case, the query result is always available from the first step. A novice user may only give (possibly imprecise) tuple-level feedbacks and still get some improvement whereas an advanced user may get more precise results with decision-level feedbacks.

In a more general task, query results can be visualized in a table view as in Figure 6, where the user can give tuple-level and decision-level feedbacks in a similar manner.

After the user is satisfied with the result, she can save it in various forms including a dynamic spreadsheet (i.e., a materialized view that is periodically updated) and a new web service (the user's form becomes input parameters of the service).

7. FUTURE WORK

The system is being extended to support data extracted from HTML tables in web pages. We also plan to support mappings in nested relation in order to cover more complicated Web service data sources. In the current system, the user needs to choose data sources from a directory in order to form a query. It is our future work to eliminate this need and automatically discover data sources given a query.

Finally, we plan to conduct extensive user studies to validate the usefulness of the system. For example, the ranking scheme based on the possible world model [4] may not be the best way when the system supports interactive query refinement.

8. REFERENCES

[1] Bogdan Alexe, Laura Chiticariu, Renee J. Miller, and Wang-Chiew Tan. Muse: Mapping understanding and design by example. In *ICDE 2008*, 2008.

[2] Bogdan Alexe, Laura Chiticariu, and Wang-Chiew Tan. Spider: a schema mapping debugger. In *VLDB 2006*, pages 1179–1182, 2006.

[3] Omar Benjelloun, Anish Das Sarma, Alon Y. Halevy, and Jennifer Widom. Uldbs: Databases with uncertainty and lineage. In *VLDB 2006*, pages 953–964, 2006.

[4] Xin Luna Dong, Alon Y. Halevy, and Cong Yu. Data integration with uncertainty. In *VLDB 2007*, pages 687–698, 2007.

[5] Alon Halevy, Michael Franklin, and David Maier. Principles of dataspace systems. In *PODS 2006*, pages 1–9, 2006.

[6] Jorge Nocedal and Stephen Wright. *Numerical Optimization*. Springer, 2000.

[7] Utkarsh Srivastava, Kamesh Munagala, Jennifer Widom, and Rajeev Motwani. Query optimization over web services. In *VLDB 2006*, pages 355–366, 2006.