RIDE: A Tool for Interactive Source Registration in Community-oriented Information Integration

Yannis Katsis Alin Deutsch Yannis Papakonstantinou Keliang Zhao
CSE Department
UC San Diego
{ikatsis, deutsch, yannis, kezhao}@cs.ucsd.edu

1. INTRODUCTION

Modern Internet communities need to integrate and query structured information. Employing current information integration infrastructure, data integration is still a very costly effort, since source registration is performed by a central authority which becomes a bottleneck. We propose the community-based integration paradigm which pushes the source registration task to the independent community members. This creates new challenges caused by each member's lack of a global overview on how her data interacts with the application queries of the community and the data from other sources. How can the source owner maximize the visibility of her data to existing applications, while minimizing the clean-up and reformatting cost associated with publishing? Does her data contradict (or could it contradict in the future) the data of other sources?

To facilitate autonomous source registration, we introduce Registration guIDE (RIDE), a visual registration tool that guides the source owner in the autonomous registration, assisting her in answering these questions. We start by first presenting the goals of the community members when registering their sources and then describing how RIDE helps the members achieve them.

1.1 Source Owner's Goals

When a source owner registers her source in an integration system, she wants to achieve the following:

Contribute to application query results. The ultimate motive of an owner registering her source is to make her data visible to relevant client applications that issue queries against the community's global schema (which we will call *target schema*, in keeping with the terminology of IBM's Clio [6, 4]). For example, a book retailer joining a community of bibliophiles wants her book ads to be visible to queries issued by a popular brokerage application.

Trade off self-reliance for cleaning cost savings. However while willing to contribute to the application queries the owner has two subgoals that conflict with each other:

On one hand, she wants to *publish everything asked by the query* so that her data is visible to the query, regardless of which other sources are present in the system (we call such a registration *"self-sufficient"*). For instance, if the query asks for ads of books with great reviews, the retailer may want to provide *both* book ads and their reviews so that her data are *always* visible to the query.

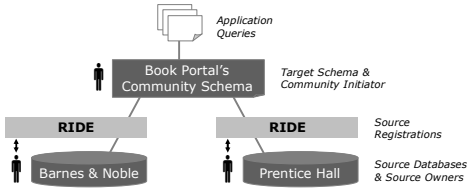

Figure 1: Community-Based Integration Architecture

On the other hand, she *wants to restrict the amount of published information* in order to minimize the cleaning effort associated with publishing. In that case she may willingly give up self-sufficiency, settling for a *"complementary"* registration that relies on other, trusted sources. For instance assume that the retailer collects third-party reviews in the form of text blurbs. Cleaning them up for publication (spell-checking, language censorship, etc.) is an expensive process requiring human involvement. Therefore the retailer needs to know whether another trusted source (e.g. a book publisher) provides reviews, so that she relies on this source for reviews and provides herself only book ads, thus saving the cleaning effort.

Avoid inconsistency. Since every owner registers her sources independently, she may easily register data that contradict those from other registered sources. For instance, a publisher and a retailer may list different authors for the same book (identified by its ISBN). Inconsistencies correspond to wrong data and should be avoided. It is therefore important for an owner to know when her registration may create inconsistencies in the integration system.

1.2 Our Solution: Registration guIDE

It is easy to see that achieving these goals requires an overview of the system, which a source owner does not have. For example, deciding which attributes can be borrowed from other sources to create a complementary registration requires understanding the registrations of all other sources currently registered in the system.

The tool. To overcome this lack of global overview and facilitate autonomous source registration, we propose Registration guIDE (RIDE), a visual tool that extends the classic schema mapping interface (as encountered in IBM Clio [6, 4], MS BizTalk Server [1] and Stylus Studio [3]) with a suggestion component that guides the source owner in the registration of her source. The suggestions assist the owner in achieving her goals detailed above. They allow her to negotiate the trade-off between two competing requirements: maximizing self-reliance for making her data visible to existing application queries, versus minimizing the data cleaning cost. In addition, RIDE helps the owner avoid inconsistency of her data with respect to data in other sources by issuing appropriate warnings.

The resulting architecture of a community-based integration system enabled by RIDE is shown in Figure 1. In such systems the

VLDB '08, August 24-30, 2008, Auckland, New Zealand

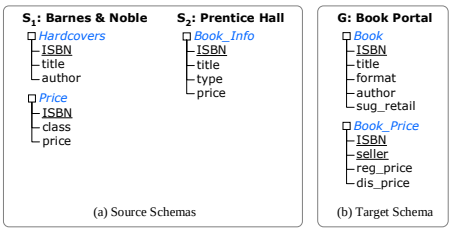

Figure 2: Source schemas and Target schema

```
SELECT  title, format, seller, reg_Price, dis_Price
FROM    Book, Book_Price
WHERE   Book.ISBN = Book_Price.ISBN AND Book.author = "Ullman"
```

Figure 3: Application query

community initiator starts by designing the target schema. After the initiation of the community, application developers register the queries (over the target schema) that their applications will issue during operation. When a community member wants to register her source into the community she only has to choose the application query to which she wants her source to contribute, as well as the desired self-reliance level and initiate RIDE.

To allow the owner to specify whether she prefers more self-reliance or less cleaning effort, self-reliance levels model coarse-grained trade-offs between these two conflicting goals. Higher levels require publishing more data fields, which yields less reliance on what other sources provide, but in exchange may involve more cleaning effort. With respect to a query Q over the target schema, a registration R can be (in decreasing order of self-reliance):

- *self sufficient* if it contributes answers to Q even if all other sources leave the community or
- *complementary* if it contributes to Q, but only in cooperation with registrations of other sources in the community.

Once the owner picks the desired self-reliance level, RIDE explains her the different ways in which she can provide the target attributes required by the query in order to achieve the selected level. As we will explain later, RIDE's suggestions correspond to even finer-grained trade-offs between self-reliance and cleaning effort. To make it easy to register a source, RIDE's interface allows a user to create a registration solely through visual actions (i.e. lines, constants). Its suggestions are also expressed visually by appropriately shading the corresponding visual components. The list of suggestions adapts to the owner's action at each interaction step, to include only attributes that are essential and that the owner is willing to provide.

In addition to making suggestions on how to provide a target attribute, RIDE also helps the owner avoid inconsistencies. To this end it issues warnings for two types of inconsistencies:

- *potential* inconsistency, which may occur for *some* contents of the source databases, and
- *definite* inconsistency, which will occur for *all* source databases.

Potential inconsistency is more of a conservative property checked at registration time, since whether it will actually occur at run-time will depend on the data in the source databases. Definite inconsistency on the other hand is a serious problem, since it will always appear at run-time regardless of the source data.

Examples of suggestions and inconsistency warnings are presented in Section 4. For formal definitions of the concepts used in the paper (including the self-reliance levels and the types of inconsistencies) as well as a detailed description of the algorithms powering the system the reader is referred to [5]. A demo of RIDE is available at **http://db.ucsd.edu/ride**.

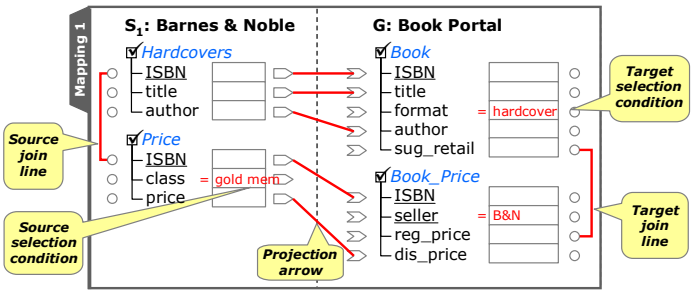

Figure 4: Barnes & Noble Registration

2. DEMONSTRATED SCENARIO

For our demonstration we will use "Bibliophilia"; an application for the bibliophiles' community that integrates book information from several sources. The community's target schema $\mathcal{G}$, shown in Figure 2b, consists of relations Book and Book_Price. Relation Book contains general information about a book, while Book_Price stores the regular and the discounted price (i.e., price for "Bibliophilia" members) at which sellers provide the book. Underlined attributes correspond to (composite) primary keys.

"Bibliophilia" contains an application query Q retrieving the title and format of books by Ullman together with their regular and discounted price as shown in Figure 3. In our demonstration we will show how two source owners can use RIDE to register their sources into the system so that they contribute to Q. Our sources are the bookstore *Barnes & Noble (B&N)* and the publisher *Prentice Hall (PH)*, with the schemas shown in Figure 2a.

3. RIDE FRONTEND

Mapping Specification. RIDE's front end resembles graphical interfaces of schema mapping tools, such as IBM Clio [6, 4], MS BizTalk Server [1] and Stylus Studio [3]. Similarly to them, RIDE enables source owners to register their sources by creating mappings between their source schema and the target schema solely through a set of visual actions listed below (Figure 4 depicts such a mapping of the B&N source created through RIDE):

Drawing *projection arrows* from a source attribute to a target attribute, to specify where the latter gets its value from. For example, in Figure 4, the price in the B&N database is exported as the discounted price in Bibliophilia's database.

Entering *(source / target) selections* next to attributes. A source selection restricts the exported source data. For example, the selection "class = gold members" in Figure 4 limits the exported prices to only those for gold members. A target selection on the other hand allows the source owner to enter information in the target database that is not stored explicitly in the source database. For instance, B&N's owner specifies in Figure 4 through target selections that her books are hardcovers and that her bookstore's name is "B&N".

Drawing *(source / target) join lines* between pairs of source / target attributes. Join lines have similar semantics to selection conditions with the only difference that they represent equalities between two attributes instead of equalities between an attribute and a constant. For instance, B&N's source owner in Figure 4 employs a source join line between the ISBNs to export only pairs of book and price tuples that join on the ISBN. Additionally, she uses a target join line to declare that B&N sells books to non-members at the suggested retail price, regardless of what this price may be.

Note that this visual representation has a solid underpinning: As explained in [5] each visual mapping corresponds to a Global-Local-As-View (GLAV) constraint. Moreover, as is customary in GLAV, queries are interpreted under certain answer semantics.

Suggestions. However in contrast to other mapping tools, RIDE also contains a suggestion component explaining to the owner how she can reach the desired self-reliance level. The interaction proceeds as follows: RIDE shows in bold all target attributes of interest to the query (i.e. selected, projected or joined) together with all different subsets of them that can be provided to reach the desired self-reliance. For instance, in Snapshot 1.1 of Figure 5 the source can become Self Sufficient only by providing *all* required attributes (shown as a single set on the right pane). The user chooses the set of missing attributes that she wants to provide and then selects each of them to see RIDE's suggestions on how to provide them. The suggestions are shown by shading the corresponding components on the interface and the attribute for which the suggestions are shown is marked with a flag. For instance, in Snapshot 1.1 the shaded projection arrow box and selection box next to Book.ISBN mean that to reach Self Sufficiency this attribute can be provided either through a projection arrow or a target selection. We describe the suggestions generated by the tool in detail in the following section.

4. DEMONSTRATED INTERACTION

The goal of our demonstration is to showcase how RIDE assists a source owner in the registration of her source. To this end, we will start with a member-less integration system and allow people from the audience to pose as source owners who add their sources into the system one by one. Since RIDE's suggestions for a source depend on the registrations of the already registered sources (which will have been added by other audience members), in each such interaction with the audience RIDE will give different suggestions.

In the following we describe a sample demonstration session, which also illustrates the type of suggestions that RIDE can generate. For our sample interaction we will employ the help of two audience members. The first, acting as B&N's source owner, will be asked to use RIDE to register her source in such a way that it is Self-Sufficient w.r.t. the query of Figure 3. The second will impersonate the owner of PH; her goal will be to create a Complementary registration w.r.t. the same query and the B&N registration created by the first member. Figures 5 and 6 depict the snapshots from the respective sample interaction sessions with RIDE.

4.1 B&N Registration for Self-Sufficiency

Using RIDE B&N's owner will explore the following suggestions, corresponding to different ways of contributing to the query:

Directly providing attributes. B&N's owner starts by trying to provide target attribute Book.ISBN. RIDE explains to her in Snapshot 1.1 that she can provide it in 2 ways; either by mapping to it values from a source attribute (through a projection arrow) or by assigning to it some constant value (i.e. entering a target selection).

Trading off cleaning cost savings vs generality. Directly providing an attribute through an arrow does however not always suffice to acquire the desired degree of self-reliance to a query Q: the source may only contribute to Q if it contains tuples with specific values asked by the query (e.g. books by Ullman in our example). B&N's owner will come across such a case, when among other actions she draws a projection arrow into Book.author. As shown in Snapshot 1.2, RIDE explains to her the two available options:

Source selection. If the owner wishes to minimize the cleanup cost, she can restrict the exported tuples to only those with the particular value asked by Q. RIDE suggests the corresponding source selection option (in our case Hardcovers.author = 'Ullman').

Intra-source assertion. However, if for the sake of contributing to several queries the owner prefers to export more tuples than those relevant to Q, she can choose to not include the selection in her mapping. In this case RIDE asks her if she believes that the exported tuples will always include at least one tuple relevant to Q. If she answers positively, RIDE takes this answer into account when generating subsequent suggestions. These questions, called *assertions*, are boolean queries and are drawn using the classical visual paradigm designed for Query-By-Example interfaces such as the query builders of MS Access and MS SQL Server [2]. In our case the gray box in Snapshot 1.2 contains an assertion asking the owner whether her source contains at least one Ullman hardcover.

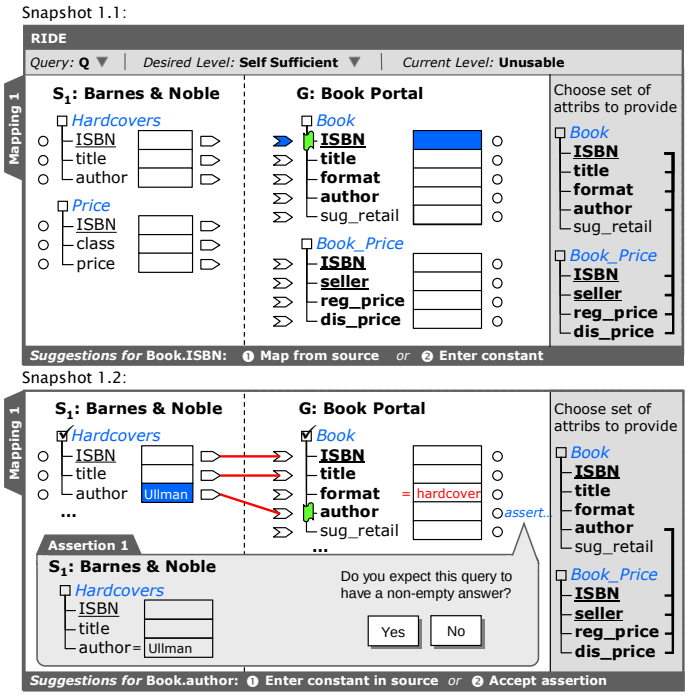

Figure 5: Sample interaction for the B & N registration

Assume that B&N's owner accepts the assertion and through more actions ends up in the registration of Figure 4. Since her source does not contain the regular price asked by the query, she decides to stop there even though her registration is not Self-Sufficient.

4.2 PH Registration for Complementarity

Given B&N's registration from Figure 4 we will subsequently ask another audience member to register PH using RIDE. Her goal is to create a complementary registration w.r.t. the same query and B&N's registration. Figure 6 shows snapshots of this interaction session. Through the interaction the source owner will understand the following ways of contributing to a query:

Indirectly providing attributes. So far we have seen cases where the source directly provides a required target attribute through a projection arrow or target selection. However, a source owner may be able to provide an attribute value *indirectly* by operating on a *different* target attribute. One such case is shown in Snapshot 2.1. Assume that PH's owner chooses to become complementary by providing Book_Price.reg_price (since her entire interaction with RIDE is done under this assumption we omit the right pane in Figure 6). RIDE explains in Snapshot 2.1 that she can provide this attribute either directly (through a projection arrow or target selection) or by instead providing attribute Book.sug_retail. The reason is simple: B&N's registration in Figure 4 specifies that its regular price equals the suggested retail price (without providing its exact value). In that case PH should also map the corresponding primary key (i.e. Book.ISBN) so that if PH and B&N provide a common book their book info is merged into a single Book tuple.

Inter-source assertions: supporting data merging. Assume that PH's owner followed the tool's suggestions and provided Book.sug_retail and the primary key. This is not yet sufficient for PH to become complementary w.r.t. the query. For this to happen, PH must provide at least one Book *in common* with B&N.

To guarantee that, RIDE asks the source owner an assertion involving both her own source and the other sources in the system with which she has to merge to achieve complementarity. For instance, the dialog box in Snapshot 2.2 shows an assertion, asking whether PH has at least one book also sold by B&N. Note that the assertion does not talk in terms of B&N's schema but in terms of its contribution to the target database. This was deliberately done to have assertions that refer only to concepts that the source owner knows (i.e. her own source's schema and the target schema). Upon accepting the assertion PH's registration will turn complementary since together with B&N it will contribute reg_price to the query.

During the interaction the user will also see warnings about inconsistencies that might arise. Recall that these fall in 2 categories:

Potential Inconsistency. If PH's owner subsequently extends the mapping to also export book titles, she creates a potential inconsistency, since B&N and PH could provide *different* titles for the same book. The box in Snapshot 2.3 explains this conflict.

Definite Inconsistency. While potential inconsistency is quite common, definite inconsistency usually results from human error and should be corrected. For instance, assume that in the future PH decides to store only paperbacks in its relation Book_Info. She informs the system about her decision through a target selection on format, resulting in the registration of Snapshot 2.4. This leads to a definite inconsistency, since PH *always* provides a book in common with B&N (due to the accepted assertion of Snapshot 2.2) and therefore this book has to be both paperback (since PH sells only paperbacks) and hardcover (since B&N through its registration in Figure 4 provides only hardcovers). RIDE notifies the user and explains the inconsistency as shown at the bottom of Snapshot 2.4.

5. RIDE PROPERTIES

By providing an interaction with RIDE our demonstration will corroborate the following properties of RIDE also explained in [5]:

Soundness of suggestions: RIDE only makes suggestions that are *guaranteed* to lead to registrations of the desired self-reliance.

Suggestions relevant to owner's focus: RIDE only makes suggestions relevant to the owner by allowing her to guide the search in several ways: First, she chooses a subset of attributes to provide and thus avoids seeing suggestions on attributes that she cannot or is unwilling to provide. Second, she accepts assertions (proposed pro-actively by RIDE) about her data. These restrict the structure of the source database and are exploited by RIDE to skip suggestions and warnings if they do not apply to data satisfying the restrictions.

Adaptive response to user's actions: RIDE does not pre-compute all suggestions beforehand but instead recomputes them adaptively after each user action. It does so even when the user ignores its suggestions and carries out a non-suggested action instead.

6. REFERENCES

[1] Microsoft Biztalk Server. http://www.biztalk.org/.

[2] Microsoft SQL Server. http://www.microsoft.com/sql/.

[3] Stylus Studio. http://www.stylusstudio.com/.

[4] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data Exchange: Semantics and Query Answering. *TCS*, 336(1):89–124, 2005.

[5] Y. Katsis, A. Deutsch, and Y. Papakonstantinou. Interactive Source Registration in Community-oriented Information Integration. In *VLDB*, 2008.

[6] C. Yu and L. Popa. Constraint-based XML query rewriting for data integration. In *SIGMOD*, 2004.

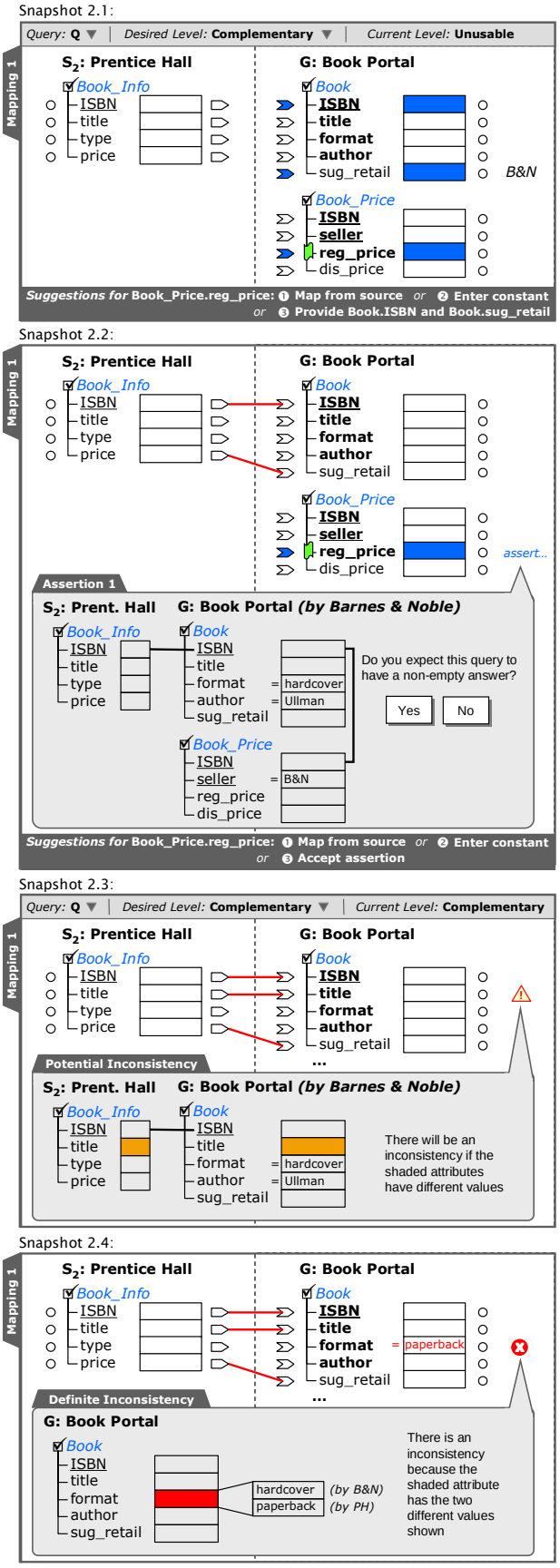

Figure 6: Sample interaction for the Prentice Hall registration