Pay-As-You-Go Mapping Selection in Dataspaces*

Cornelia Hedeler, Khalid Belhajjame, Norman W. Paton, Alvaro A.A. Fernandes,
Suzanne M. Embury, Lu Mao, Chenjuan Guo
School of Computer Science, The University of Manchester, Oxford Road, Manchester, UK
{chedeler, khalidb, norm, afernandes, sembury, maol, guoc}@cs.manchester.ac.uk

ABSTRACT

The vision of dataspaces proposes an alternative to classical data integration approaches with reduced up-front costs followed by incremental improvement on a pay-as-you-go basis. In this paper, we demonstrate DSToolkit, a system that allows users to provide feedback on results of queries posed over an integration schema. Such feedback is then used to annotate the mappings with their respective precision and recall. The system then allows a user to state the expected levels of precision (or recall) that the query results should exhibit and, in order to produce those results, the system selects those mappings that are predicted to meet the stated constraints.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems

General Terms

Algorithms, Design

Keywords

Dataspaces, User feedback

1. INTRODUCTION

Data integration is a challenging task. In time, the recognition has grown that specifying an integration schema over which users pose queries, as well as the mappings between that schema and the source schemas, is both hard and time-consuming. To reduce up-front costs, a new class of data integration platforms, called *dataspaces*, has been proposed [4] that uses a pay-as-go approach for continuous, gradual improvement (e.g., [3, 10, 7]).

We observe, that, although the idea of incremental improvement is an integral part of the dataspace vision, few proposals support it. Incremental improvement based on user feedback can take a variety of forms, e.g., through the manual provision of mappings (e.g., [10]), through the annotation of query results as to which items are spurious or which should be ranked higher (e.g., [9]), or through a process by which mappings are debugged (e.g., [8]). We observe that all these approaches require, to different degrees, an understanding of the syntax and semantics of mapping and schema languages on the part of the person providing the feedback. This has the drawback that only experts can provide feedback, shutting out non-expert users.

*The work reported in this paper was supported by a grant from the EPSRC. We are grateful for the support.

SIGMOD'11, June 12–16, 2011, Athens, Greece.
ACM 978-1-4503-0661-4/11/06.

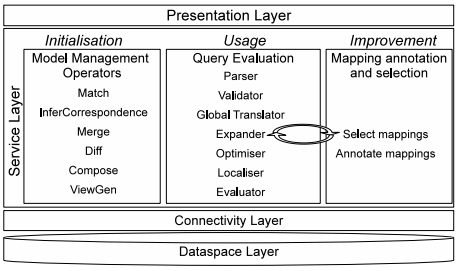

Figure 1: Layered architecture of DSToolkit

We have implemented and will demonstrate a dataspace management platform, DSToolkit, and in particular its techniques for incremental improvement. Rather than requiring users to have an understanding of schemas and mappings, we only ask the user to provide feedback on the results of queries over an integration schema. Such feedback is then used to annotate the mappings that produced the corresponding query results. On the basis of these annotations, DSToolkit is able to estimate the precision and recall that will be exhibited by subsequent queries depending on the mappings used. Thus, DSToolkit allows a user to state the expected levels of precision (or recall) that the query results returned should exhibit and, in order to produce those results, the system selects those mappings that are estimated to meet the stated constraints.

DSToolkit demonstrates the following features: (i) a dataspace architecture that supports a complete lifecycle including bootstrapping, improvement and maintenance; (ii) incremental annotation of schema mappings based on user feedback on query results; and (iii) mapping selection at query execution time based on user-specified QoS criteria. The techniques that underpin (ii) and (iii) are reported in [1]; this paper describes how the techniques can be incorporated into a dataspace architecture, and the demonstration shows how they can be exploited by users in practice.

2. SYSTEM OVERVIEW

In this section, we provide information on the main architectural aspects of DSToolkit, briefly describe how the stages of the dataspace lifecycle can be supported and focus on the functionality to be demonstrated, viz., feedback-based improvement via annotation and selection of mappings.

Architecture. Broadly speaking, DSToolkit uses a layered architecture summarised in Fig.1. The *dataspace* layer

persistently stores the model-independent representation of schemas, the elicited matches, the schematic correspondences supported by the matches, the mappings derived from the latter, the user feedback collected and the corresponding query results. Access to the persistently stored information is provided by the *connectivity* layer. This layer is used, in turn, by the *service* layer where the functionality resides, viz., the model management operators, the query processor and the improvement techniques. The *presentation* layer exposes a user interface through which the user can access the functionality provided. The service layer in Fig. 1 covers the three basic phases of the dataspace lifecycle [5], viz., initialisation, usage and improvement.

Initialisation. The *initialisation* phase, which aims to identify the data resources and integrate them, is also referred to as *bootstrapping*. It is supported predominantly by the standard model management operators [2], augmented with operations that infer high-level schematic correspondences from matches and that generate mappings from the correspondences. By providing implementations of these operators over a model-independent representation of constructs, DSToolkit supports the integration of heterogeneous data sources through manipulation of schemas and the correspondences that hold between them. As such, DSToolkit supports the coexistence of different integration schemas, e.g., one that unions the source schemas, one that merges them, and even one that selects one of the source schemas as the integration schema.

Usage. The integrated data sources are queried during the *usage* phase. The query engine in DSToolkit extends the OGSA-DQP distributed query engine [6] with an expander (expands the query using the mappings selected by the mapping selection operator) and a localiser (generates source-specific subqueries in the query language of the source).

As the demo focusses on the improvement phase, the mapping annotation and mapping selection operators are introduced in slightly more detail in what follows (see also [1]).

Improvement: Mapping Annotation. Given the result tuples, users are invited to state, for a subset of their own choosing, whether the tuple was expected to be present (i.e., true positive (TP)) or whether it was expected not to be present (i.e., false positive (FP)). Users can also input a tuple that was not returned but that they expected to see (i.e., a false negative (FN)). Fig. 3(b) and (e) show the relevant part of the user interface.

The user feedback provides partial information on the extent of a construct in an integration schema. Using the (partial) TP, FP, and FN annotations on the query result, we calculate the precision and recall of a mapping m (relative to the user feedback UF provided) that was used in producing those results, as follows:

$$Precision(m, UF) = \frac{|TP(m, UF)|}{|TP(m, UF)| + |FP(m, UF)|} \quad (1)$$

$$Recall(m, UF) = \frac{|TP(m, UF)|}{|TP(m, UF)| + |FN(m, UF)|} \quad (2)$$

where $|TP(m,UF)|$, $|FP(m,UF)$, $|FN(m,UF)|$ denote, resp., the number of TPs, FPs and FNs returned by mapping m according to user feedback UF on query results involving m.

Consider, as an example, the Mondial[1] database and an integration schema with the construct `city(name,province, country)`. Assume that `city` is populated by the mappings

```
m1: SELECT name, province, country FROM organization
m2: SELECT mountain as name, province, country
       FROM geo_mountain
m3: SELECT name, province, country FROM city
```

Figure 2: Mappings for populating city in the integration schema.

Table 1: Query result returning all cities.

Name	Province	Country	Expected	Not exp.	Mapping
UN	NY	USA		√	m1
Ben Nevis	Highland	GB		√	m2
Berlin	Berlin	D	√		m3

shown in Fig. 2 that have been derived automatically (hence the rather odd nature of `m1` and `m2`) during the initialisation phase. Also consider the query `SELECT * FROM city` posed over the integration schema. Table 1 shows a subset of the result with the per-tuple feedback and the mapping involved in producing the tuple. The feedback suggests that the user only expected to see the last tuple produced by (the correct) mapping `m3` and did not expect the first two tuples produced by mappings `m1` and `m2`. Based on this feedback, the three mappings `m1`, `m2` and `m3` will be annotated with precision and recall computed by Eq. 1 and 2, respectively.

Improvement: Mapping Selection. After annotating the mappings, this information can be used to select the mappings to be used to answer a query posed by a user. Not all users may require the same precision and recall with respect to the query results returned. As such, we allow users to choose the desired precision or recall target that the query results should meet (see Fig. 3(d)) and the selection operator chooses mappings so as to reach the specified target and maximise the unconstrained value.

Given a set of mappings M, selecting the mappings $sm \subseteq M$ to be used for answering a query given a target of λ (say, precision) that the result should at least achieve is formulated as a constrained optimisation problem that aims to maximise the unconstrained variable (say, recall) over the union of the results returned by the selected mappings sm. The problem is solved using tabu search.

3. DEMONSTRATION SCENARIOS

The demo will focus on mapping annotation and selection based on user feedback. This is motivated by the fact that feedback-based incremental improvement is central to the dataspace vision, as is the expectation that relatively small amounts of feedback can underpin significant improvements in outcomes. We assume that a dataspace consists of an integration schema, which will either have been provided manually or obtained automatically using the model management operators mentioned earlier. We also assume that a number of mappings have been provided manually or obtained using the model management operators, which can result in mappings of mixed quality.

The demo will show the following. Starting with an initial query that is unfolded using all applicable mappings and evaluated over the sources (see Fig. 3(a)), results are returned and the user is given the opportunity to provide feedback indicating TP and FP results as well as supplying FN results (see Fig. 3(b)). Assuming that no feedback had been provided yet, the mappings are not yet annotated with precision and recall. We will show how the feedback now provided is then used to annotate the mappings by re-running the same query (see Fig. 3(c)).

[1]http://www.dbis.informatik.uni-goettingen.de/Mondial/

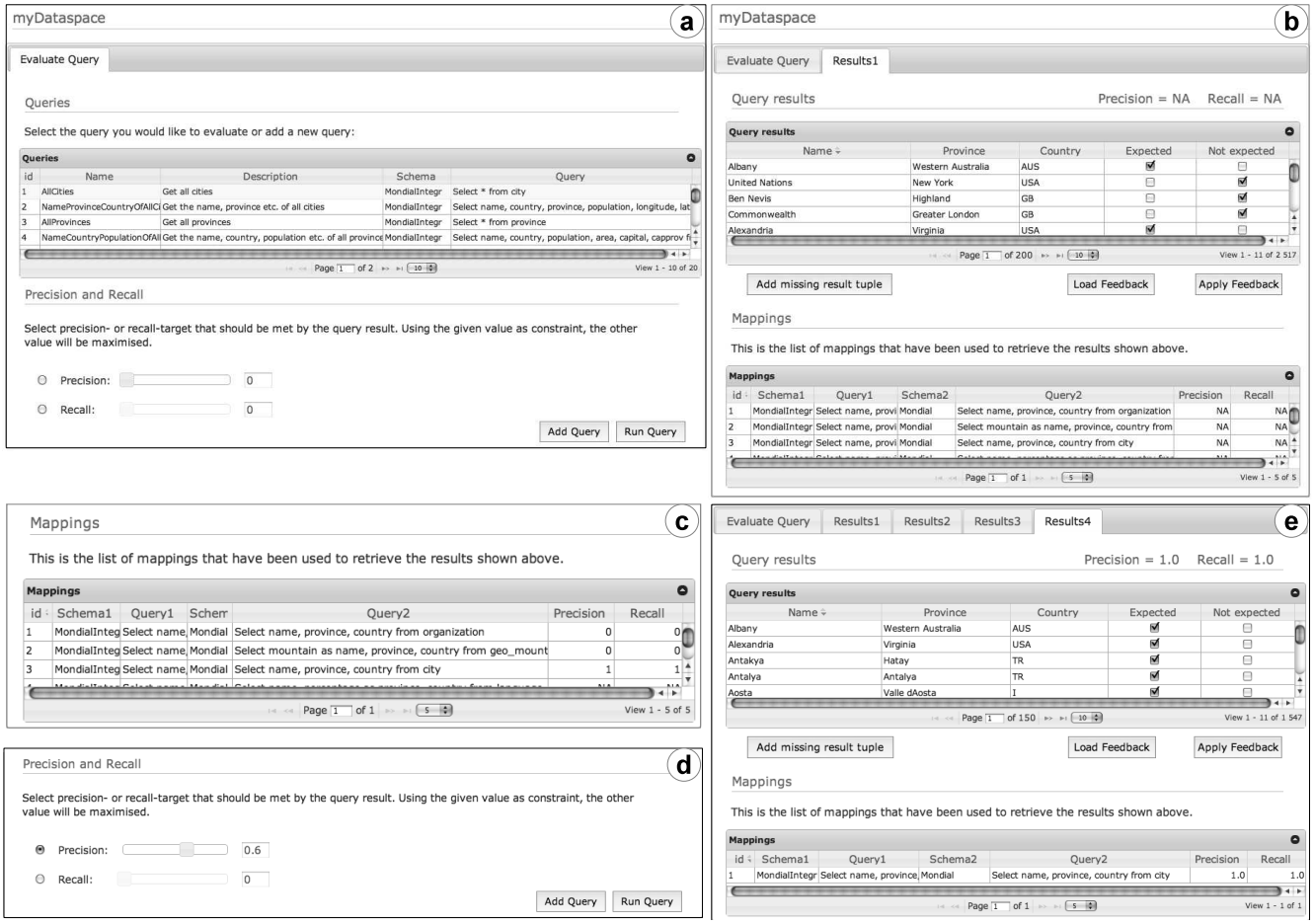

Figure 3: User Interface for demonstrating mapping annotation and selection for query evaluation.

To demonstrate the mapping selection for query evaluation, we will provide the user with the opportunity to re-run a number of queries with different precision or recall targets (see Fig. 3(d)). This will show how the choice by the user of one quality of service (either precision or recall) for maximisation, subject to a user-provided minimal threshold on the other quality being met, informs the search among alternative mappings for those that meet user requirements. We will show that this affects the selection of the mappings used for unfolding the query and, the query results returned.

4. CONCLUSIONS

DSToolkit is a flexible toolkit for initialising, using and improving dataspaces in a wider range of usage scenarios than have hitherto been implemented. In particular, the demo will show the improvement in query results that can be obtained by feedback instances that require no specific expertise from the user. The demo will show how a user can trade off precision and recall subject to a threshold on the non-selected quality; DSToolkit uses annotations derived from user feedback on past query results to guide the selection of mappings for answering the query that are estimated to meet the targets set by the user.

5. REFERENCES

[1] K. Belhajjame, N. W. Paton, S. M. Embury, A. A. A. Fernandes, and C. Hedeler. Feedback-based annotation, selection and refinement of schema mappings for dataspaces. In *EDBT*, pages 573–584, 2010.

[2] P. A. Bernstein and S. Melnik. Model management 2.0: manipulating richer mappings. In *SIGMOD Conference*, pages 1–12, 2007.

[3] A. Das Sarma, X. Dong, and A. Halevy. Bootstrapping pay-as-you-go data integration systems. In *SIGMOD*, pages 861–874, 2008.

[4] A. Halevy, M. Franklin, and D. Maier. Principles of dataspace systems. In *PODS '06*, pages 1–9, 2006.

[5] C. Hedeler, K. Belhajjame, A. A. A. Fernandes, S. M. Embury, and N. W. Paton. Dimensions of dataspaces. In *BNCOD*, pages 55–66, 2009.

[6] S. Lynden, A. Mukherjee, A. C. Hume, A. A. A. Fernandes, N. W. Paton, R. Sakellariou, and P. Watson. The design and implementation of OGSA-DQP: A service-based distributed query processor. *Future Generation Comp. Syst.*, 25(3):224–236, 2009.

[7] J. Madhavan, S. Cohen, X. L. Dong, A. Y. Halevy, S. R. Jeffery, D. Ko, and C. Yu. Web-scale data integration: You can afford to pay as you go. In *CIDR*, pages 342–350, 2007.

[8] G. Mecca, P. Papotti, S. Raunich, and M. Buoncristiano. Concise and expressive mappings with +spicy. *PVLDB*, 2(2):1582–1585, 2009.

[9] P. P. Talukdar, M. Jacob, M. S. Mehmood, K. Crammer, Z. G. Ives, F. Pereira, and S. Guha. Learning to create data-integrating queries. *PVLDB*, 1(1):785–796, 2008.

[10] M. A. Vaz Salles, J.-P. Dittrich, S. K. Karakashian, O. R. Girard, and L. Blunschi. itrails: Pay-as-you-go information integration in dataspaces. In *VLDB*, pages 663–674, 2007.