Just-in-time Data Integration in Action

Martin Hentschel
ETH Zurich
hemartin@inf.ethz.ch

Laura Haas
IBM Almaden Research
laura@almaden.ibm.com

Renée J. Miller
University of Toronto
miller@cs.toronto.edu

ABSTRACT

Today's data integration systems must be flexible enough to support the typical iterative and incremental process of integration, and may need to scale to hundreds of data sources. In this work we present a novel data integration system that offers great flexibility and scalability. Our approach to data integration is unique in that it executes mapping rules at query runtime using annotations. On top, we have built the *People People People* application. It allows users to search for people, display information about people, and browse through a network of related people, where the data is integrated from local and remote data sources. The demo presents all features of our underlying data integration engine through a set of motivating scenarios.

1. INTRODUCTION

Setting up a data integration system typically involves an iterative process of exploration and integration. In the beginning the user explores the data to get familiar with its content and structure. This enables him to integrate a first chunk of data, and proceed to further exploration of the same or additional data sources, eventually expanding and refining the integration. Data integration systems should be flexible enough to support such an iterative process. Furthermore, today's businesses deal with hundreds of data sources containing similar data. For example, Novartis, a global pharmaceutical company, has over 170 sources of person-related data. Thus, data integration systems should be scalable in the number of data sources.

In this work we present a novel data integration engine that offers great flexibility and scalability. Our approach is flexible in that it executes schema-level and instance-level integration rules (mapping rules) together in a single framework. As we will show, this allows an integrated view of diverse data to be built in an incremental and iterative fashion.

In our approach the integration of data takes effect at the runtime of a query while the data that needs to be integrated is being processed. We call this *just-in-time data integration*. During runtime the data is annotated with additional information that is introduced by processing the mapping rules. We are able to execute schema and instance mapping rules using these annotations. Only mapping rules that match the data being processed are executed. We call our technique *Mapping Data to Queries (MDQ)* [9, 10]. Our approach scales well in the number of source schemas and ultimately outperforms state-of-the-art approaches by orders of magnitude [10].

In order to present this new approach to data integration, we have created an application that highlights the features of our underlying data integration engine. We call the application *People People People*. It allows users to search for people, display information about people, and browse through a network of related people. All information is aggregated from the DBLP data set [14] and Wikipedia using our novel integration technique.

Section 2 provides an overview of our data integration engine. Section 3 reviews related work. Section 4 describes our demo, including the system set-up (Section 4.1) and several usage scenarios (Section 4.2).

2. MAPPING DATA TO QUERIES

The MDQ engine can integrate locally stored data with data fetched dynamically from remote sources, resolving both schema-level and instance-level conflicts. It works with data represented as XML, and supports XQuery. Figure 1 shows sample documents from Wikipedia and DBLP. All documents contain namespaces (i.e., prefixes) to associate data items with their respective sources. Figure 2 shows the schemas of the sample documents (source schemas) as well as the schema of the People People People application (target schema). Mapping rules in Figure 3 map the source schemas to the target schema. Note that we show only a subset of input data, schemas, and mapping rules, just to explain our data integration approach. We will present the complete set of data, schemas, and mapping rules to the demo audience.

The basic idea of Mapping Data to Queries is that when a query is executed, each data item touched is annotated using any mapping rules that apply. The annotations provide new paths through the XML data. Consider the tree on the left side of Figure 4. It shows (in black) the Wikipedia document of Figure 1. Take schema mapping rule $R1$ in Figure 3, *w:table* → *p:person* (*w:table* "is a" *p:person*), and a query such as *//p:person* (i.e., find all persons). When query processing reaches this document, the rule will fire,

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Articles from this volume were presented at The 36th International Conference on Very Large Data Bases, September 13-17, 2010, Singapore.
Proceedings of the VLDB Endowment, Vol. 3, No. 2
Copyright 2010 VLDB Endowment 2150-8097/10/09... $ 10.00.

Wikipedia

```
<w:table xmlns:w="...">
  <w:tr>
    <w:th id="#2">
      Hector Garcia-Molina
    </w:th>
    <w:td>...</w:td>
  </w:tr>
</w:table>
```

DBLP

```
<d:article xmlns:d="...">
  <d:author id="#8">
    Hector Garcia-Molina
  </d:author>
  <d:title>Simrank++</d:title>
  <d:year>2008</d:year>
</d:article>
```

Figure 1: Data Snippets

Wikipedia	DBLP	People
w:table	d:article	p:person
w:tr	d:author	p:name
w:th	d:title	
w:td	d:year	

Figure 2: Source and Target Schemas

R1 w:table → p:person

R2 w:tr/w:th → p:name

R3 d:author as $a →
<p:person><p:name>
{$a/text()}
</p:name></p:person>

R4 #2 ← #8

Figure 3: Mapping Rules

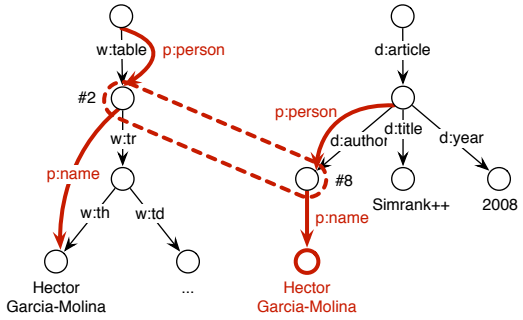


Figure 4: Data Snippets from Wikipedia and DBLP annotated to match the People People People Schema

and the bold red edge labeled "p:person" will be added. Similarly, the other red lines represent the firing of other schema mapping rules mapping the Wikipedia and DBLP schemas to our application schema.

These new edges provide additional access paths to data nodes. The query processor traverses these edges in the same way as it traverses the original (black) edges. In Figure 4 for example, to find all names of persons, the engine traverses the red person and name edges to return the correct results. Mapping rule $R2$ specifies that the nested structure w:tr/w:th should be unnested (flattened) into p:name. In Figure 4 this rule introduces an annotation (labelled p:name) that points from w:table directly to w:th. Schema mapping rules may also create new nodes if needed for the target schema. For example, the author entries in DBLP do not have a child element "name". Schema mapping rule $R3$ causes the introduction of a new child. In particular, rule $R3$ constructs a "p:name" child containing the text value of the original author (through the variable binding $\$a$). In general, our MDQ approach handles all schema mapping scenarios in the STBenchmark [2].

To merge two data instances, a new type of mapping rule is added. These rules are of the form *nodeId1* ← *nodeId2*, specifying that the node with id nodeId2 should be merged into (be seen as part of) the node identified by nodeId1. For example in Figure 3, mapping rule $R4$ merges the node with id #8 into node #2. Whenever either of these nodes is accessed for the first time, the data is annotated with a new composite node that contains the original nodes. In Figure 4 the composite node is represented by the bold red dashed line. This composite node will be handled by the query processor in the same way as any original node. The new composite node keeps all of the inbound and outbound arcs. That means that the composite node in Figure 4 will be reached twice by the query looking for all persons. Duplicate elimination is thus needed. After duplicate elimination only one (merged) person record will be output.

Mapping Data to Queries is a fundamentally new approach to data integration. The difference between MDQ and any other data integration system is that the processing of mapping rules forms a virtual data layer (the annotations). We are able to produce (portions of) this layer just-in-time. Of course, we only generate the relevant portions taking into account both the data and the queries. Because we index the mapping rules, this just-in-time generation is very fast. Just-in-time data integration is particularly effective if the workload or data characteristics are unpredictable. Examples are scenarios in which many data sources need to be integrated, continuous query processing on data streams, or Web mash-ups in which the characteristics of the data sources frequently change in an unpredictable way. In these scenarios MDQ outperforms state-of-the-art approaches in throughput by orders of magnitude [10].

This paper claims no novelty with regard to the integration rules. There are many integration frameworks and ways to define data integration logic (e.g., Clio [11] and others).

3. RELATED WORK

The state-of-the-art approaches to data integration are federation and transformation [16]. Federation engines (*aka* mediators [17]) provide a virtual view of underlying data, as if it were already integrated [8, 13, 4]. With the help of mapping rules, federation engines rewrite a query (written against the virtual view) to match the different data sources. The rewrite happens at compile-time and without knowledge of the data. In environments where data characteristics are unpredictable (e.g., schema-less input data or schema mixes), the rewrite has to provide for all possible input data, leading to query complexity and potentially exponential runtime inefficiency [10]. In Figure 5 for example, the query *p:person[1]/p:name* (i.e., get the name of the first person) must be rewritten to match data from Wikipedia, DBLP, transformed data, and mixes thereof.

The transformation approach (*aka* extract-transform-load, ETL) transforms all data to match the local schema, here the People schema. All major database vendors provide tools to support ETL. The research community has mostly focussed on the problem of maintaining materialized views [7, 18, 1]. The transformation approach integrates data in advance of query processing. Because the query is not known, *all* data has to be transformed regardless of whether it is needed to produce the query result. In Figure 5, the complete document is transformed (highlighted in red) even though the query only asks for the name of the first person.

In contrast to these two approaches, just-in-time data integration takes effect at runtime of a query. It is able to adapt the integration to the query *and* the data. Our technique for just-in-time data integration, MDQ, does not

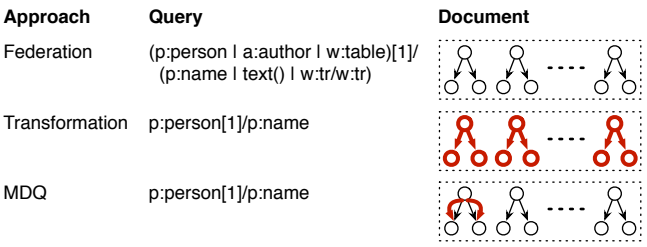


Figure 5: Different approaches to execute query *p:person[1]/ p:name* (get the name of the first person)

rewrite the query (as the federation approach does) nor does it transform all data. MDQ ignores mapping rules that are not applicable to the input data and annotates only those parts of the data that are touched by the query. In Figure 5, only the first part of the document is annotated and the query is able to return the correct result.

There has been work on adapting query execution to the actual input data [3, 12]. These techniques gather statistics (e.g., selectivity estimates) at runtime of a query to dynamically re-plan the query execution. In particular, Tukwila [12] employs a federated approach to integrate different data sources. In this work queries are still rewritten at compile-time but later optimized during query runtime. Potter's Wheel [15] allowed incremental cleansing of data; MDQ allows incremental refinement of the overall integration (Scenario 3).

People People People is inspired by the community portal DBLife [6]. DBLife is an application of Cimple [5]. Cimple focuses on automatically generating mapping rules and employs an ETL approach to integrate data. Our application is only meant to demonstrate the functionality of the underlying MDQ engine. We do not automatically generate mapping rules. In our work we focus on the execution of (previously generated) mapping rules.

4. PEOPLE PEOPLE PEOPLE

To demonstrate our MDQ engine, we have built an application that lets users integrate data about computer science researchers as they explore: People People People. In our demo there are three different data schemas: the DBLP schema, the Wikipedia schema, and the schema of our application (Figure 2). Rules to map the DBLP and Wikipedia schemas to the application's schema are created before the start of the demo. A (semi-)automatic schema mapping tool could be used to create this initial set of mapping rules, though ours are hand-written. On the instance level, there are scientists that are present in DBLP as well as in Wikipedia. For most scientists there are multiple data entries in DBLP. These data entries may be merged using instance mapping rules. Since DBLP uniquely identifies authors [14], we were able to create an initial set of instance mapping rules. More mapping rules are generated whenever the user merges results (see Scenario 2 below). All mapping rules in the demo, schema and instance, may be freely edited.

4.1 System Setup

The demo system consists of a client and a server. The client runs People People People as a web application in a standard browser. The application connects to the server through remote procedure calls. It sends XQuery queries to the server and the server answers with XML. The application presents the results as figures and text. Clicking on a figure or on the text generates a new query, which is in turn sent to the server and answered quickly to provide an interactive user interface.

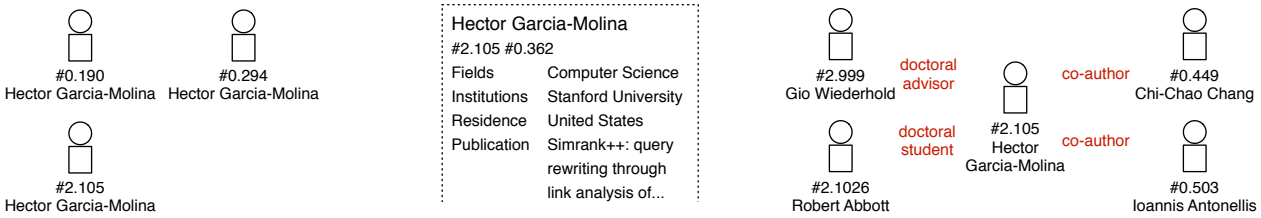
The server runs our data integration engine, an XQuery processor enhanced to integrate data using MDQ. The engine is loaded with the pre-defined mapping rules. Two types of data access are supported, standard file input/ output to local storage, and connections to web servers on the internet over standard HTTP. Local storage consists of a directory structure containing data files, indexes, and metadata, and is loaded with parts of the DBLP data set, available in XML. The DBLP data set is stored on the server without any modifications to the data. In particular, it has not been transformed to any other schema. In the demo, the engine will retrieve Wikipedia web pages in an on-demand fashion. The web pages are returned by Wikipedia in the XHTML format – a variant of XML. Therefore it is possible to process these web pages using standard XQuery processing techniques.

4.2 Demonstration Scenarios

The scenarios below deal with three methods of interaction with the system, ranging from simple to complex. They demonstrate finding information about people and displaying their relationships, manually merging duplicate data records, and integrating a third new data source by adding mapping rules and modifying existing queries.

Scenario 1: Searching and Browsing Information A fresh PhD student wants to know more about famous scientists in his area of research. He logs on to the People People People web application and "searches" for a researcher he has heard of: Hector Garcia-Molina. (Each search translates into an XQuery query under the covers). The system returns three results, displayed as small figures tagged with name and id (Figure 6a). The student clicks on the last result. The system provides the student with additional, structured information about this person (Figure 6b). The student further wants to know what people Hector Garcia-Molina is related to. He double-clicks on the last search result. The system displays the data record of Hector and four persons Hector is related to. The related persons are again shown as small figures tagged with name and id. Furthermore, there are now red labels in between Hector and each of the related persons stating the type of relationship (Figure 6c). For example, Gio Wiederhold was Hector's doctoral advisor. Clicking on any of the related persons allows the student to display information about them. Double-clicking on one of the persons will result in displaying their relationships. In this way, the student is able to browse through a network of related scientists.

Scenario 2: Merging Duplicate Records Judging from the additional information in two of the data records found for Hector Garcia-Molina, the PhD student decides that these two data records refer to the same real world person with name Hector Garcia-Molina. The student wants to add this knowledge to the system. He therefore selects both data records by clicking on them while pressing the shift key. The system provides a button that lets the user merge data records. The student clicks this "merge" button. Immediately, the system updates the view to display only



(a) Search Results (b) Aggregated Information (c) Relationships

Figure 6: Screenshots of Integrated Data from DBLP (stored locally) and Wikipedia (fetched on demand)

one data record instead of the previous two records. Internally, a new instance mapping rule has been added to the system. The student may continue to use the merged data record as described in Scenario 1.

Note that Figures 6b and 6c already display information from such a merged data record. In Figure 6b the merged data record's identifier shows the identifiers of the individual records #2.105 and #0.362. The data items: fields, institutions, and residence stem from Wikipedia while the publication data stems from the DBLP data set. Similarly, in Figure 6c, the relationships doctoral advisor and doctoral student are retrieved from Wikipedia while the co-author relationships stem from DBLP.

Scenario 3: Adding a New Data Source

The third scenario highlights the internals of the system. The People People People web application offers two additional tabs *Rules* and *Query*. The Rules tab lets users view and edit all mapping rules stored in the system. This allows the user to refine current mapping rules, to delete mapping rules in case of mistakes (e.g., false merges), or to incorporate additional data sources. The Query tab lets users post individual queries as well as modify existing queries used by the web application.

After a while the PhD student thinks that there is important information missing. He wants to incorporate an additional data source. He therefore opens the Query tab to modify the existing "search" query. He adds a look up of Amazon webpages to the search query, so that data from Amazon will be included in the results. The student then queries for Hector again. No additional data appears in the result view. The student recognizes that he must map the Amazon schema to the People People People schema. Therefore he opens the Rules tab and adds the schema level rule *a:h1* → *p:person* (let *a:* be the namespace of Amazon). This rule views the webpage (the headline of the webpage) as a person. As the student queries for Hector again, he will notice an additional data record. Yet, this additional record is empty. The student goes back to add another mapping rule to include all books of people as additional information. Revisiting the last search result of Hector, the student finds that all of Hector's books show up in the information box. Now, he merges this record with the existing records for Hector. The aggregated data record now contains information gathered from DBLP, Wikipedia, and Amazon.

Incrementally, the PhD student was able to include a new data source. Iteratively, he mapped information from that source to the People People People web application schema. Finally, he merged data from the new source with data that already existed in the database. Integrated data!

5. REFERENCES

[1] S. Abiteboul, J. McHugh, M. Rys, V. Vassalos, and J. L. Wiener. Incremental Maintenance for Materialized Views over Semistructured Data. In *VLDB Conf.*, pages 38–49, 1998.

[2] B. Alexe, W. C. Tan, and Y. Velegrakis. STBenchmark: towards a benchmark for mapping systems. *PVLDB*, 1(1):230–244, 2008.

[3] R. Avnur and J. M. Hellerstein. Eddies: Continuously Adaptive Query Processing. In *ACM SIGMOD Conf.*, pages 261–272, 2000.

[4] S. Chawathe et al. The TSIMMIS Project: Integration of Heterogeneous Information Sources. In *Proc. of the 100th Anniversary Meeting of the Information Processing Society of Japan(IPSJ)*, pages 7–18, Tokyo, Japan, Oct. 1994.

[5] P. DeRose, W. Shen, F. Chen, A. Doan, and R. Ramakrishnan. Building Structured Web Community Portals: A Top-Down, Compositional, and Incremental Approach. In *VLDB*, pages 399–410, 2007.

[6] P. DeRose, W. Shen, F. Chen, Y. Lee, D. Burdick, A. Doan, and R. Ramakrishnan. DBLife: A Community Information Management Platform for the Database Research Community. In *CIDR*, pages 169–172, 2007.

[7] A. Gupta and I. S. Mumick. Maintenance of Materialized Views: Problems, Techniques, and Applications. *IEEE Data Eng. Bull.*, 18(2):3–18, 1995.

[8] L. M. Haas, D. Kossmann, E. L. Wimmers, and J. Yang. Optimizing Queries across Diverse Data Sources. In *VLDB Conf.*, pages 276–285, 1997.

[9] L. M. Haas, R. J. Miller, M. Hentschel, and D. Kossmann. A First Step Towards Integration Independence. In *NTII*, 2010.

[10] M. Hentschel et al. Scalable Data Integration by Mapping Data to Queries. Technical Report 633, ETH Zurich, Systems Group, Dept. of Computer Science, 2009.

[11] M. A. Hernández, R. J. Miller, and L. M. Haas. Clio: A Semi-Automatic Tool For Schema Mapping. In *ACM SIGMOD Conf.*, page 607, 2001.

[12] Z. G. Ives, D. Florescu, M. Friedman, A. Levy, and D. S. Weld. An adaptive query execution system for data integration. In *ACM SIGMOD Conf.*, pages 299–310, 1999.

[13] A. Y. Levy, A. Rajaraman, and J. J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In *VLDB Conf.*, pages 251–262, 1996.

[14] M. Ley. DBLP - Some Lessons Learned. *PVLDB*, 2(2):1493–1500, 2009.

[15] V. Raman and J. M. Hellerstein. Potter's Wheel: An Interactive Data Cleaning System. In *VLDB Conf.*, pages 381–390, 2001.

[16] J. Widom. Integrating Heterogeneous Databases: Lazy or Eager? *ACM Computing Surv.*, 28(4es):article 91, 1996.

[17] G. Wiederhold. Mediators in the Architecture of Future Information Systems. *IEEE Computer*, 25(3):38–49, 1992.

[18] Y. Zhuge, H. Garcia-Molina, J. Hammer, and J. Widom. View Maintenance in a Warehousing Environment. In *ACM SIGMOD Conf.*, pages 316–327, 1995.