Data Integration Support for Mashups

Andreas Thor **David Aumueller** **Erhard Rahm**

University of Leipzig, Germany
{thor, david, rahm}@informatik.uni-leipzig.de

Abstract

Mashups are a new type of interactive web applications, combining content from multiple services or sources at runtime. While many such mashups are being developed most of them support rather simple data integration tasks. We therefore propose a framework for the development of more complex dynamic data integration mashups. The framework consists of components for query generation and online matching as well as for additional data transformation. Our architecture supports interactive and sequential result refinement to improve the quality of the presented result step-by-step by executing more elaborate queries when necessary. A script-based definition of mashups facilitates the development as well as the dynamic execution of mashups. We illustrate our approach by a powerful mashup implementation combining bibliographic data to dynamically calculate citation counts for venues and authors.

Introduction

Mashups are a new type of interactive web applications, combining content from multiple services or sources into a new service or data source. The emergence of such applications has been fueled by application development frameworks utilizing AJAX[1] technology and the increasing number of Web APIs for retrieving content from popular web sites such as Google Maps, Flickr or YouTube. ProgrammableWeb[2] currently (April 2007) lists more than 1700 mashups and about 3 new mashups are added every day. Many mashups annotate some information with geographical data to visualize the information on a map, e.g., the location of selected restaurants or real estate objects.

Content integration in mashups is typically dynamic, i.e., it occurs at runtime based on specific user input. In order to achieve short execution times most mashups only support rather simple kinds of data integration. For example, they often use standardized object identifiers (e.g., latitude/longitude for geographical positions or unique product numbers such as EAN or ISBN) for easy interrelation of different sources or services. Query access to data sources or search engines is typically based on keywords, tags or category names but without extensive post-processing to match results of different sources.

We argue that advanced mashups need improved support for dynamic data integration on heterogeneous data objects. As an example for such mashups we consider our Online Citation Service (OCS). For a list of publications, e.g., all papers of an author or all papers of a conference, it dynamically (online) determines the number of citations to the individual papers, e.g., to determine the top-cited papers. In our implementation we obtain the publication lists from the DBLP bibliography and the citation counts from Google Scholar (GS). Obviously, the data of GS cannot be downloaded a priori for arbitrary authors or conferences so that we need to query this search engine at runtime. Furthermore, GS has improvable data quality (e.g., duplicate publication entries) since it extracts bibliographic data automatically from millions of scientific publications (mostly available as PDF documents). The OCS example illustrates two key problems to be solved for complex data integration mashups, namely the adoption of effective query strategies (i.e., how to retrieve the relevant GS publications) as well as precise object matching (i.e., how to identify corresponding DBLP and GS publications). The main challenge is to solve these problems dynamically with sufficient accuracy – two opposing demands as we will see.

In this paper we propose a framework architecture for the development of dynamic data integration mashups. It consists of components for query generation and online matching as well as for additional data transformation. Our architecture supports interactive result refinement to improve the result quality by running refining queries when necessary. This is achieved by a script-based definition of mashups facilitating a fast development of mashups as well as their dynamic execution. We illustrate our data integration approach in detail for the mentioned OCS mashup.

The remainder of the paper is structured as follows. In the next section we present the framework architecture for data integration mashups. Afterwards we illustrate the Online Citation Service as a mashup implementation in the bibliographic domain. In particular we demonstrate the script-defined mashup definition and user-driven result refinement, and discuss implementation issues. Finally, we review related work and summarize our work.

[1] http://en.wikipedia.org/wiki/AJAX

[2] http://www.programmableweb.com

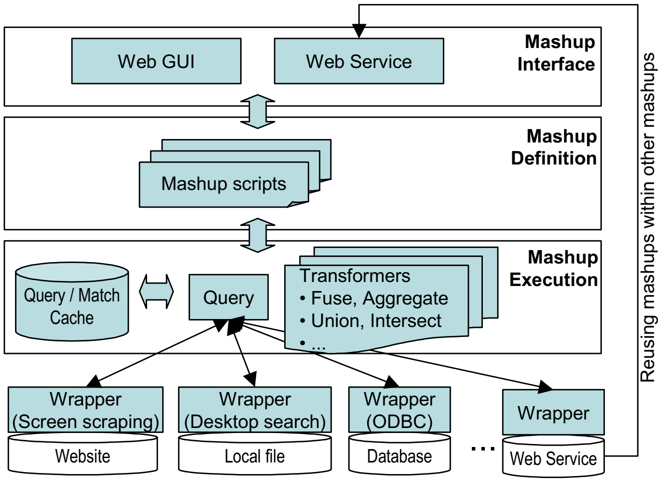


Fig. 1: Architecture of data integration framework for mashups

A Framework Architecture for Mashups

The initial design of the overall framework architecture is shown in Figure 1. The framework enables developers to specify their mashup algorithm within scripts and thus on a higher level than implementing it in a common programming language. The functionality of a mashup script is provided as a web service so that it can be invoked by a web interface or by another web service, e.g., within another mashup.

Mashups typically integrate data from heterogeneous data sources such as databases, search engines, or local files. Wrappers transform the data into a source-specific and self-describing XML structure for uniform data access within the framework. Typically, the resulting XML structure describes a set of objects, such as people, products, publications, or addresses (e.g., to be displayed using Google Maps). Objects are represented by a set of attributes each having a value or a set of values. Moreover, complex objects may contain component objects, e.g., a person may have one or multiple addresses.

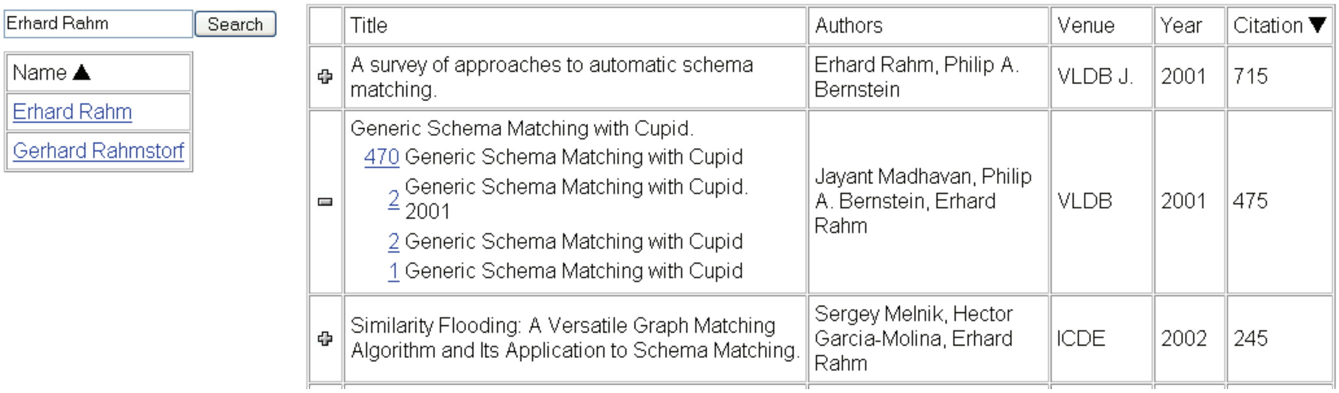
The proposed framework uses a high-level script language to define mashups similar to [14]. The language consists of powerful operators operating on XML data structures. Operators are generic and can thus be applied to different data sources and services. For example, a query operator takes as input the data source (e.g., the URL) and a query specification. All (intermediate) results can be stored in variables for use by other operators. There are several operators for data transformation (e.g., fuse, aggregate) and set operations (e.g., union, intersection, and difference). We now discuss some of these operators. Further details of the script language are outlined in the description of the OCS mashup.

Querying data sources or web services is an integral part of data integration mashups. Queries are directed towards a specific source and can be defined either explicitly or implicitly. Explicit queries are usually directly specified by the user via a web interface, e.g., for which author the publication list should be determined by OCS. Implicit queries are applied for an XML document (e.g., the result of a previous query) and define a query strategy on the input document. This can be used to generate a query for each input object. For example, given a list of publications (each having a <title> element), the query strategy "intitle:<title>" generates one search query for each publication where <title> is replaced by the actual publication title. An alternative strategy would be to use only one query (e.g., on author) to quickly obtain a first, approximate result.

For query sources such as search engines or hidden databases the definition of query strategies is a crucial task since the time for query execution and result transmission usually dominates the overall mashup runtime. Hence it is important to achieve a maximum of relevant results with a minimal number of queries. Reducing the number of queries is also necessary since some services limit the number of queries for a given period of time (e.g., the Google Web service API accepts only 1000 queries a day for a given client).

In contrast to classical data integration scenarios such as data warehouses, mashups heavily rely on user interaction. Users expect results in a few seconds and want to process the data in different ways such as using them as input for other mashups. Web technologies like AJAX only meet such user requirements from a technical point of view. We therefore identified the refinement of results as an important aspect in mashups for data processing. We use such refinement steps for result completion and the extension of selected result objects. We meet the interaction requirements by segmenting a mashup into several scripts, e.g., for execution in sequential order. When displaying the intermediate results of one script to the user, the framework can already execute the next script in the background to refine the query results. The execution of additional scripts may be automatic or triggered by the user (e.g., by clicking a button). User-driven refinement helps to invoke additional queries only when needed.

The framework provides several transformers that can be applied to input data or query results. An important transformer is fuse performing object matching and merging. Fuse takes as input two XML documents A and B of a simple object-attribute-value structure (see Figure 4 for an example) and identifies objects referring to the same real world entity. In the fuse result, the matching elements of B are appended as child elements of their matching counterparts in A. Hence the fuse result contains both the input objects as well the match result. It can be used as input for further querying or additional data transformations (by ignoring the child elements) or for a match refinement. Fuse determines the similarity of two objects (i.e., the likelihood that they are the same) from the similarity of their attribute values (e.g., by applying string similarity functions) and component objects. Within the mashup definition the developer may specify what elements should be



Erhard Rahm | Search

Name ▲
Erhard Rahm
Gerhard Rahmstorf

	Title	Authors	Venue	Year	Citation ▼
⊕	A survey of approaches to automatic schema matching.	Erhard Rahm, Philip A. Bernstein	VLDB J.	2001	715
⊟	Generic Schema Matching with Cupid. 470 Generic Schema Matching with Cupid 2 Generic Schema Matching with Cupid. 2001 2 Generic Schema Matching with Cupid 1 Generic Schema Matching with Cupid	Jayant Madhavan, Philip A. Bernstein, Erhard Rahm	VLDB	2001	475
⊕	Similarity Flooding: A Versatile Graph Matching Algorithm and Its Application to Schema Matching.	Sergey Melnik, Hector Garcia-Molina, Erhard Rahm	ICDE	2002	245

Figure 2: Screenshot of Online Citation Service for author "Erhard Rahm"

compared using which similarity function and parameters (e.g., minimal similarity threshold).

The transformer aggregate allows matching objects in a fuse result to be aggregated for a more concise result. It applies a user-specified resolution function on selected attributes of the matching objects, e.g., to determine the most common value or the sum of all values. The result is kept as an additional XML element (see OCS example in the next section).

We plan to implement the sketched framework based on our previous data integration platform iFuice [14] (Information Fusion utilizing instance correspondences and peer mappings). In contrast to many data integration platforms that utilize schema mappings, iFuice focuses on instance data of different sources and mappings between them. Such mappings are sets of correspondences between data sources, which may already exist, e.g., in the form of web links. iFuice has been successfully applied for an offline citation analysis [13], offline object matching [16] and instance-based ontology matching [15]. iFuice offers already a scripting language that can be used as a basis for our new framework.

However, iFuice has been tailored to offline processing so far and we therefore need to evolve it into a mashup platform supporting dynamic data integration. The major extensions will involve the discussed operators for dynamic queries and query refinement. Furthermore, we will use XML data structures as the internal data format and the operators have to be adopted appropriately. The new operator fuse combines object matching with result combination, i.e., matching objects are represented as nested objects in the XML data structure. Finally, the generic mashup engine should automatically offer iFuice scripts as web services so that developers can re-use them for new mashup scripts.

Strategy	Query	#Queries
Name	Author's / venue's name	1
Title pattern	Disjunction of title patterns	≈ #Pubs/10
Keywords	Title as keywords	#Pubs

Table 1: Query strategies of Online Citation Service

Online Citation Service

The Online Citation Service[1] (OCS) allows the generation of citation counts for publication lists of authors and venues (see Figure 2). In the current implementation, it obtains the publication lists from the DBLP bibliography[2] that is known for high data quality (e.g., complete publication lists for venues). The citation counts are obtained from Google Scholar[3]. For example, OCS allows a fast and precise ranking of all publications of a given venue (say, VLDB 1997) according to their citation counts. Such a ranking helps to determine influential papers and can be helpful to find candidates for a "10 year best paper" award.

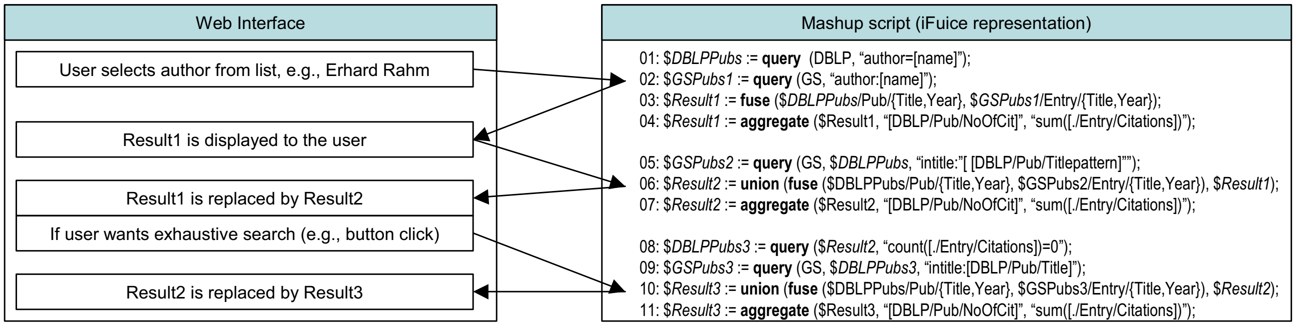
The user first performs a keyword search within the DBLP data source and selects one author or venue out of the search results. The left part of Figure 2 illustrates this step for an author-based search. OCS then determines the associated publications in DBLP and queries Google Scholar (GS) to achieve the corresponding GS entries. The retrieved GS publications are matched to the DBLP publications based on the similarity of the publication titles and years. An exact comparison for title equality fails in most cases due to spelling errors, special characters and title extensions (e.g., "(Demo)" for demonstration papers). Therefore OCS utilizes a string distance function with a given threshold. OCS additionally requires the years to be equal in case GS provides a year information.

The right part of Figure 2 illustrates the aggregated result shown to the end-user. It lists all DBLP publications of the specified author sorted by citation counts. The citation count per publication is determined by the sum of citations of all matching GS publications. Users may see the list of aggregated GS publications by clicking on the "+" icon. Moreover, users can retrieve the list of citing publications (as provided by GS) by following the hyperlink of the GS publication counts.

[1] http://labs.dbs.uni-leipzig.de/ocs

[2] http://www.informatik.uni-trier.de/~ley/db/

[3] http://scholar.google.com


Web Interface
User selects author from list, e.g., Erhard Rahm
Result1 is displayed to the user
Result1 is replaced by Result2
If user wants exhaustive search (e.g., button click)
Result2 is replaced by Result3
Mashup script (iFuice representation)

```
01: $DBLPPubs := query  (DBLP, “author=[name]”);
02: $GSPubs1 := query (GS, “author:[name]”);
03: $Result1 := fuse ($DBLPPubs/Pub/{Title,Year}, $GSPubs1/Entry/{Title,Year});
04: $Result1 := aggregate ($Result1, “[DBLP/Pub/NoOfCit]”, “sum([./Entry/Citations])”);

05: $GSPubs2 := query (GS, $DBLPPubs, “intitle:”[ [DBLP/Pub/Titlepattern]””);
06: $Result2 := union (fuse ($DBLPPubs/Pub/{Title,Year}, $GSPubs2/Entry/{Title,Year}), $Result1);
07: $Result2 := aggregate ($Result2, “[DBLP/Pub/NoOfCit]”, “sum([./Entry/Citations])”);

08: $DBLPPubs3 := query ($Result2, “count([./Entry/Citations])=0”);
09: $GSPubs3 := query (GS, $DBLPPubs3, “intitle:[DBLP/Pub/Title]”);
10: $Result3 := union (fuse ($DBLPPubs/Pub/{Title,Year}, $GSPubs3/Entry/{Title,Year}), $Result2);
11: $Result3 := aggregate ($Result3, “[DBLP/Pub/NoOfCit]”, “sum([./Entry/Citations])”);
```

Figure 3: Illustration of OCS mashup execution

Query strategies

After specifying a DBLP author or venue, OCS consecutively applies multiple query strategies to find matching GS publications and their citation counts. The strategies currently used are summarized in Table 1. The *name* strategy is very simple and fast since it only requires one GS query. It is especially effective for authors with uncommon names having a short publication list, but can lead to many irrelevant results for authors with very common names. Similarly, for smaller venues with unique names (e.g., IIWeb) the name strategy produces good results. On the other hand, a query for the term ‘VLDB’ often leads to irrelevant results (e.g., due to VLDB Journal or VLDB workshop publications) and misses relevant results that are only assigned to “Very Large Databases” instead of VLDB.

The second strategy, *title pattern*, determines the most important terms of the title based on the TF/IDF string metric. It determines the minimal list of terms that characterizes a publication title unambiguously within DBLP. (Different publications with the same title, e.g., a conference and journal version of a paper, result in the same term list but are differentiated in the matching step later on.) For querying GS, a title pattern is then constructed based on the title where all other (i.e., irrelevant) terms are blanked out by the GS wildcard character “*”. For example, publication title [12] contains many common words and its title pattern (“survey * approaches * * schema matching”) therefore contains four terms, whereas one single word is already sufficient for [14] (“iFuice”) due to the uncommon acronym iFuice[1]. Since Google allows up to 32 such search terms per query, the strategy on average combines 10 title patterns in a conjunctive query, i.e. “intitle:<Pattern1> OR intitle:<Pattern2> …”. This approach allows a precise and parallel search for a number of publications within one single query.

The *keyword* strategy is only applied for publications that do not have matching counterparts (see below) in one of the previous two strategies. For each publication the title is used as the query pattern (without quotes). This strategy focuses on finding a certain publication but accepts a significant number of irrelevant results.

After finishing the first query strategy (*name*), the author already receives the complete publication list with approximated citation counts from the first GS query. The end-user can start inspecting the results, e.g., re-order the publication list, get the list of citing publications etc. In the meantime OCS refines the result by applying the second strategy (*title pattern*) and – once finished – automatically updates the displayed result table using AJAX technology. The third strategy (*keyword*) is executed analogously on user’s demand, but only for DBLP publications that are still unmatched to GS.

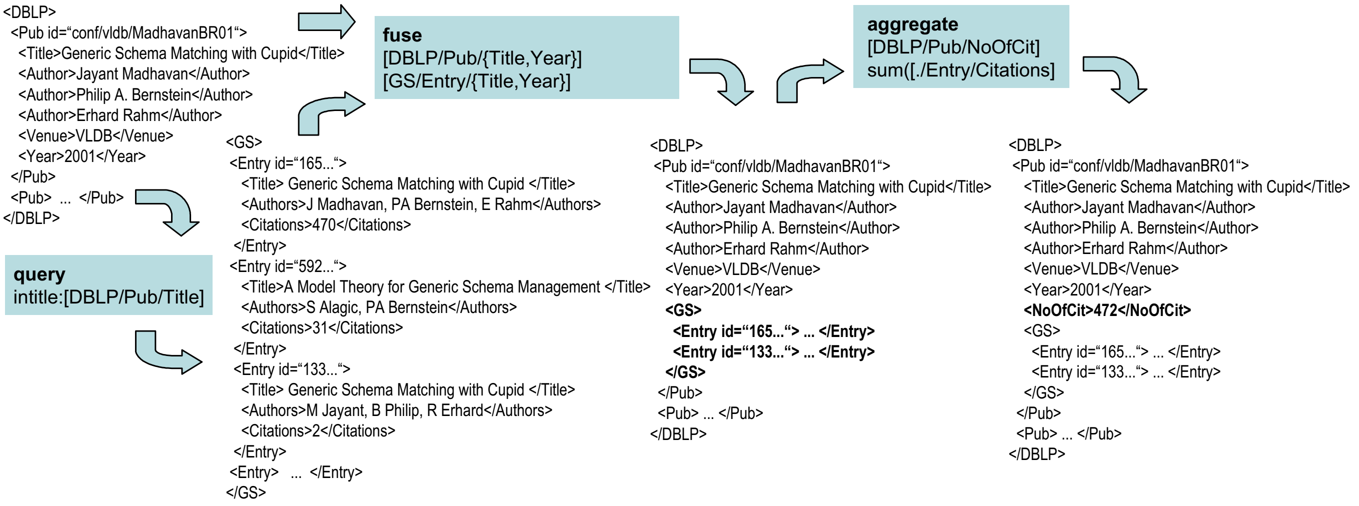
Mashup scripts

Figure 3 illustrates the OCS implementation based on the framework prototype. On the left hand side the Web GUI functionality is depicted that invokes three iFuice scripts (right, lines 1-4, 5-7, and 8-11), one for each query strategy. It illustrates the three-step refinement process where the result of each step is used in the following steps. The user decides “how far” the mashup is executed by invoking the particular scripts via the Web GUI.

Line 1 retrieves all DBLP publications for a given author (stored in variable *$DBLPAuthor*) and line 2 queries Google Scholar using the author name. The resulting publications (from DBLP and GS) are fused in line 3 so that *$Result1* contains the set of corresponding DBLP-GS publications. The sum of citations for each DBLP publication is computed by the aggregate step in line 4. The second query strategy refines the result by querying GS using title patterns (line 5). The retrieved GS publications are matched to the DBLP publications, too, and the result is unified with the result of the first query strategy (line 6). Line 7 performs the aggregation that updates the number of citations based on the new result. Finally, lines 8-11 implement the third strategy, but only for DBLP publications that do not have match counterparts, i.e., that do not have GS child elements (line 8). In contrast to the previous queries the query string only contains the title (line 9) and the results are fused, unified (line 10), and aggregated (line 11).

Figure 4 illustrates the query, fuse and aggregate operators for the third query strategy of OCS. The query operator is

[1] Leading and trailing “*” are removed from the title pattern.



```xml
<DBLP>
 <Pub id="conf/vldb/MadhavanBR01">
  <Title>Generic Schema Matching with Cupid</Title>
  <Author>Jayant Madhavan</Author>
  <Author>Philip A. Bernstein</Author>
  <Author>Erhard Rahm</Author>
  <Venue>VLDB</Venue>
  <Year>2001</Year>
 </Pub>
 <Pub> ... </Pub>
</DBLP>
```

```xml
<GS>
 <Entry id="165...">
  <Title> Generic Schema Matching with Cupid </Title>
  <Authors>J Madhavan, PA Bernstein, E Rahm</Authors>
  <Citations>470</Citations>
 </Entry>
 <Entry id="592...">
  <Title>A Model Theory for Generic Schema Management </Title>
  <Authors>S Alagic, PA Bernstein</Authors>
  <Citations>31</Citations>
 </Entry>
 <Entry id="133...">
  <Title> Generic Schema Matching with Cupid </Title>
  <Authors>M Jayant, B Philip, R Erhard</Authors>
  <Citations>2</Citations>
 </Entry>
 <Entry> ... </Entry>
</GS>
```

```xml
<DBLP>
 <Pub id="conf/vldb/MadhavanBR01">
  <Title>Generic Schema Matching with Cupid</Title>
  <Author>Jayant Madhavan</Author>
  <Author>Philip A. Bernstein</Author>
  <Author>Erhard Rahm</Author>
  <Venue>VLDB</Venue>
  <Year>2001</Year>
  <GS>
   <Entry id="165..."> ... </Entry>
   <Entry id="133..."> ... </Entry>
  </GS>
 </Pub>
 <Pub> ... </Pub>
</DBLP>
```

```xml
<DBLP>
 <Pub id="conf/vldb/MadhavanBR01">
  <Title>Generic Schema Matching with Cupid</Title>
  <Author>Jayant Madhavan</Author>
  <Author>Philip A. Bernstein</Author>
  <Author>Erhard Rahm</Author>
  <Venue>VLDB</Venue>
  <Year>2001</Year>
  <NoOfCit>472</NoOfCit>
  <GS>
   <Entry id="165..."> ... </Entry>
   <Entry id="133..."> ... </Entry>
  </GS>
 </Pub>
 <Pub> ... </Pub>
</DBLP>
```

Figure 4: Illustration of mashup workflow example of the OCS

invoked for the obtained DBLP publications (only one DBLP publication is illustrated in Figure 4 due to space constraints). The queries for Google Scholar use the "intitle:<title>" pattern where <title> is replaced by the actual publication title specified by the XPath expression "DBLP/Pub/Title". The resulting GS publications are then matched to the DBLP publications based on the title and year similarity. Matching GS publications are inserted into the XML representation of the corresponding DBLP publication by the fuse operation. In the aggregation step the numbers of citations are summed up and stored in a new XML element[1]. Therefore, the name of the new element ("NoOfCit") as well as the XPath function are specified.

The presented script is tailored to the publication list of a selected DBLP author. However, only the first two lines have to be changed for venues. This underlines the reusability of scripts within different mashups and shows the potential of our framework for the fast creation of mashups.

Related Work

By their nature data integration mashups touch many research areas such as data extraction, object search, hidden web crawling, and object matching. Data extraction techniques (see [8] for a survey) play an important role because not all data sources can be accessed via Web services. Such techniques are also needed in our framework and existing tools (e.g., using templates [1]) can be used to speed-up the development of wrappers.

Recent work deals with entity search engines (e.g., [4][10]) to shift search engine results from the document level to the object (entity) level where the results represent more specific objects such as products or publications. Such search engines would nicely fit into our framework as they support more complex queries involving context pattern and content restriction and improved result quality. Google Scholar can be seen as an example of an entity search engine used within OCS.

The querying of data sources using search interfaces is strongly related to the crawling of the hidden Web [2][11]. In contrast to the dynamic nature of mashups, hidden Web crawling usually is an offline process with the goal to download complete copies of the "hidden databases". Since the query execution and result transmission are crucial performance aspects, there exists work on minimizing the number of queries [17][3]. While our framework can adopt query strategies to completely download relevant parts of data sources, it is also able to dynamically query sources for fast results and ad-hoc exploration.

Object Matching aims at identifying object instances in (different) data sources that refer to the same real world entity (see [5] for a recent survey). It is a crucial task for data integration and data cleaning but it is usually performed as an offline process for relational data with a strong focus on data quality. Some of the proposed algorithms can be adopted for online matching within mashups and we will investigate this further in future work.

Several frameworks exist for facilitating the implementation of mashups and Web 2.0 applications, e.g. ASP.NET AJAX[2], Direct Web Remoting[3], Echo[4], and Google Web Toolkit[5] (the latter is used for OCS). Such frameworks focus on implementation issues, such as easy-to-use programming languages or libraries for frequently executed operations (e.g., Web service call) whereas our framework

[1] Only two GS publications are used in the example leading to a citation count of 472 in contrast to 475 of Figure 2.

[2] http://ajax.asp.net/

[3] https://dwr.dev.java.net/

[4] http://nextapp.com/platform/echo2/echo/

[5] http://code.google.com/webtoolkit/

focuses on a high level description of mashup algorithms or workflows.

A VLDB2006 keynote presented a mashup fabric [7] that covers the whole mashup process from the import of external data sources to the end-user presentation using different formats (e.g., HTML, RSS). Our work resides in between the ingestion and augmentation component, i.e., it addresses what data should be retrieved (queried), how the data should be combined, and how intermediate results can be refined.

The "mash-o-matic" tool [9] supports mashup developers in the preprocessing of data. Developers can select and clean data from different sources, transform it into a generic XML format and store all in a data container called sidepad. The mashup application makes use of this sidepad and the functionality provided by mash-o-matic. However, mash-o-matic focuses on the data infrastructure for mashups but can not be used for defining a dynamic mashup application.

The MashMaker [6] approach is a generic mashup application that allows users for interactive editing, querying, manipulating, and visualizing data. Based on an untyped tree data model users can operate with so-called widgets that may import data from external data sources or visualize data on a map. Moreover, users can create and share their own custom widgets by combining existing widgets. Thereby the MashMaker enables non-expert users to explore data sources by querying and browsing but it does not offer support for ad-hoc data integration. Similar to our framework, MashMaker tries to simplify the achievement and combination of data but is tailored to end-users whereas our framework supports mashup developers by building mashups and offering them as Web services.

Conclusions and Future Work

Mashups implement dynamic data integration by combining content from multiple sources at application runtime. At present, the data integration found in most mashups is fairly simple due to the lack of suitable frameworks for ad hoc data integration and the harsh response time requirements in web applications. We therefore proposed a framework architecture supporting the development of more complex mashups incorporating dynamic data integration. The framework supports a script-based definition of mashups and the use of multiple query strategies for accessing external data sources. Query results can be dynamically refined to reach a good trade-off between fast execution times and high result quality. Generic operators such as fuse, aggregate, union, and intersect help to perform dynamic object matching and data transformation within mashup scripts. As an example mashup implementation we presented the Online Citation Service for generating citation counts for publication lists of authors and venues.

In future work we are completing the design and implementation of the proposed framework. We further will evaluate different query strategies w.r.t. the achievable result improvement and execution time.

References

[1] Arasu, A., and Garcia-Molina, H.: Extracting structured data from Web pages. In *Proc. of SIGMOD*, 2003

[2] Barbosa, L., and Freire, J.: Siphoning Hidden-Web Data through Keyword-Based Interfaces. In *Proc. of SBBD*, 2004

[3] Byers, S., Freire, J., and Silva, C. T.: Efficient Acquisition of Web Data through Restricted Query Interfaces. In *Poster Proc. of WWW*, 2001

[4] Cheng, T., and Chang, K. C.-C.: Entity Search Engine: Towards Agile Best-Effort Information Integration over the Web. In *Proc. of CIDR,* 2007

[5] Elmagarmid, A.K., Ipeirotis, P.G., and Verykios, V.S.: Duplicate Record Detection: A Survey. In *IEEE Transactions on Knowledge and Data Engineering* 19(1), 2007

[6] Ennals, R., and Garofalakis, M.: MashMaker: Mashups for the Masses. In *Proc. of SIGMOD*, 2007

[7] Jhingran, A.: Enterprise information mashups: integrating information, simply. In *Proc. of VLDB*, 2006 (abstract only)

[8] Laender, A., and Ribeiro-Neto, B., da Silva, A., and Teixeira, J.: A brief survey of web data extraction tools. In *SIGMOD Record* 31(2), 2002

[9] Murthy, S., Maier, D. and Delcambre, L.: Mash-o-matic. In *Proc. of DocEng*, 2006

[10] Nie, Z., Wen, J.-R., and Ma, W.-Y.: Object-level Vertical Search. In *Proc. of CIDR*, 2007

[11] Raghavan, S., and, Garcia-Molina, H.: Crawling the Hidden Web. In *Proc. of VLDB*, 2001

[12] Rahm, E., and Bernstein, P.: A survey of approaches to automatic schema matching. The *VLDB Journal* 10(4), 2001

[13] Rahm, E., and Thor, A.: Citation analysis of database publications. In *SIGMOD Record* 34(4), 2005

[14] Rahm, E., Thor, A., Aumueller, D., Do, H.-H., Golovin, N., and Kirsten, T.: iFuice - Information Fusion utilizing Instance Correspondences and Peer Mappings. In *Proc. of WebDB*, 2005

[15] Thor, A., Kirsten, T., and Rahm, E.: Instance-based matching of hierarchical ontologies. In *Proc. of BTW*, 2007

[16] Thor, A., and Rahm, E.: MOMA - A Mapping-based Object Matching System. In *Proc. of CIDR*, 2007

[17] Wu, P., Wen, J.-R., Liu, H., and Ma, W-Y.: Query Selection Techniques for Efficient Crawling of Structured Web Sources. In *Proc. of ICDE*, 2006