MASHING UP THE DEEP WEB
Research in Progress

Thomas Hornung, Kai Simon and Georg Lausen
Institut für Informatik, Universität Freiburg, Georges-Köhler-Allee, Gebäude 51, 79110 Freiburg i.Br., Germany

Keywords: Assisted Mashup Generation, Deep Web, Information Integration.

Abstract: Deep Web (DW) sources offer a wealth of structured, high-quality data, which is hidden behind human-centric user interfaces. Mashups, the combination of data from different Web services with formally defined query interfaces (QIs), are very popular today. If it would be possible to use DW sources as QIs, a whole new set of data services would be feasible.
We present in this paper a framework that enables non-expert users to convert DW sources into machine-processable QIs. In the next step these QIs can be used to build a mashup graph, where each vertex represents a QI and edges organize the data flow between the QIs. To reduce the modeling time and increase the likelihood of meaningful combinations, the user is assisted by a recommendation function during mashup modeling time. Finally, an execution strategy is proposed that queries the most likely value combinations for each QI in parallel.

1 INTRODUCTION

Aggregating information about a specific topic on the Web can easily result in a frustrating and time-consuming task, when the relevant data is spread over multiple Web sites. Even in a simple scenario, if we try to answer the query *"What are the best-rated movies which are currently in theaters nearby?"* it might be necessary to first query one Web site which returns a list of all movies and then iteratively access another site to find the ratings for each movie. One characteristic that underlies many of such ad hoc queries is the dynamic nature of their results, e.g. information about movies that were shown last week is of no avail. Additionally it is often necessary to fill out HTML forms to navigate to the result pages, which contain the desired information.
The part of the Web that exhibits the abovementioned properties is often referred to as the Deep or Hidden Web (DW) (Raghavan and Garcia-Molina, 2001). Due to its exponential growth and great subject diversity as recently reported in (He et al., 2007) it is an excellent source of high-quality information. This sparked several proposals for DW query engines, which either support a (vertical) search confined to a specific area with a unified query interface, e.g. (He et al., 2005) and (Chang et al., 2005), or offer ad hoc query capabilities over multiple DW sources such as (Davulcu et al., 1999).
Although these proposals are quite elaborate and powerful, they either constrain the possible combinations of DW sources or are based on a complex architecture which needs to be administrated. For instance domain experts might be necessary to define the salient concepts for a specific area or to provide wrappers for data extraction from result pages. Hence, a framework which allows non-expert users to adminstrate the system while retaining the possibility to pose complex queries and solve real-life information aggregation problems in an intuitive fashion would be beneficial. A promising approach for this could be mashup tools such as Yahoo Pipes[1], which offer a graphical interface to arrange different Web query interfaces (QIs) in a graph, where nodes represent QIs and arcs model the data flow. This enables users with little or no programming skills to intuitively aggregate information in an ad hoc fashion.
Unfortunately mashups are nowadays based on a fix set of pre-defined Web QIs that return structured data with a formal, known semantics. Therefore users have to rely on content providers to supply these programmable interfaces, which unnecessarily limits the

[1] `http://pipes.yahoo.com/pipes/`

Hornung T., Simon K. and Lausen G. (2008).
MASHING UP THE DEEP WEB - Research in Progress.
In *Proceedings of the Fourth International Conference on Web Information Systems and Technologies*, pages 58-66
DOI: 10.5220/0001523900580066

number of combinable data sources.
In this paper we propose a framework which enables non-expert users to generate mashups based on DW sources. All aspects of the mashup lifecycle are supported: starting with the acquisition of new sources via the assisted generation of mashup graphs until the final execution.

1.1 Challenges

As our approach is focused on DW sources that are designed for human visitors, neither the fully automated access to sources nor their combination is trivial. Particularly, we have to cope with the following issues:

- *Form interaction:* In order to fill out the respective form fields, the user input has to be matched to a legal combination of input element assignments. We assume in this context a single stage interaction, i.e. the form field is filled with meaningful combinations and submitted, which directly leads to the result page.
- *Data record extraction and labeling:* Each result page usually contains multiple data records which cluster related information, similar to a row in a labeled table. These data records need to be identified, extracted and labeled correctly. For this purpose we use the fully automated Web data extraction tool ViPER (Simon and Lausen, 2005). The interested reader is referred to (Laender et al., 2002) for a brief survey of other existing tools and their capabilities.
- *Fuzzy result lists:* A formally defined QI always returns exact results, i.e. if more than one result is returned, all results are known to be equally relevant. However Deep Web sources normally return a list of results that match the input criteria to some extent, often ranked by relevance.
- *Data cleaning:* For each column of a data record the correct data type has to be determined and the data has to be transformed into a canonical representation, e.g. to bridge different representations of numbers. Additionally it might be necessary to convert the data to another reference system, for instance another currency.
- *Assisted mashup generation:* Manually combining different QIs to a mashup graph is only feasible in a small world-scenario, because the user easily looses track of possible and meaningful combinations. Therefore it is of paramount importance to assist the user during the generation phase.
- *Combinatorial explosion:* As mentioned above DW sources normally return result lists. This can lead to a combinatorial explosion in possible value combinations. E.g. let source Q_3 be the sink in a mashup graph with two incoming data edges from sources Q_1 and Q_2. For an exhaustive search we would have to query Q_3 with $|Q_1| * |Q_2|$ value combinations[2], or in the general case for a data mashup graph with one sink and n incoming data edges $\Pi_{i=1}^{n}|Q_i|$ value combinations need to be considered. Moreover, a typical data mashup graph would more likely have multiple levels as the one shown in Figure 1, which means that the combinations additionally multiply along each path as well.

1.2 Contributions

We propose a DW-based mashup framework called FireSearch, which:

- Supports the semi-automatic acquisition of new DW sources (Section 2),
- Assists the user in combining these sources to a mashup graph based on a recommendation function (Section 3),
- And finally executes mashup graphs with special consideration for the limiting factors of an online Web scenario (Section 4).

The main components of the framework have been implemented as an extension for the Firefox[3] browser and tested on Linux and Windows platforms.

1.3 Running Example

When planning to buy an electronic device, it is often desirable to aggregate information from several trustworthy sources considering different aspects. It might for instance not be advisable to buy the device at the store that offers the cheapest price, if the customer service is disappointing. Therefore the user wants to define a mashup graph that assembles the necessary information based on the following DW sources:

- Source Q_1 is the official Web site of a trustworthy magazine, which regularly performs extensive tests of new electronic devices and publishes the results on the Web site,
- Source Q_2 is a searchable, community-driven Web portal with user experience-based ratings and reviews of Web retailers,

[2]Here $|Q_i|$ denotes the number of results of source Q_i.
[3]http://www.mozilla.com/en-US/firefox/

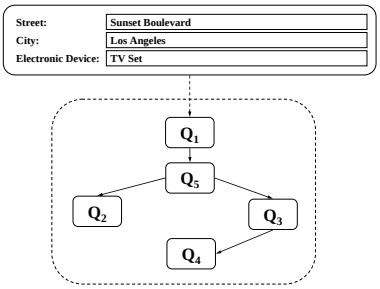

Figure 1: Data mashup graph that collects data about TV sets.

- Source Q_3 is an online database of Web retailers and their addresses, searchable by the name of the store,
- Source Q_4 is a route and map service, and
- Source Q_5 is a price comparison service which searches multiple stores for the cheapest price of electronic devices.

Figure 1 shows a mashup graph that yields the desired results. The user is presented with an interface where she can enter the necessary data that is needed to initialize the mashup. First Q_1 is accessed to find the best-rated TV sets and then for each TV set the cheapest stores are determined by Q_5. Afterwards Q_2 and Q_3 are queried in parallel to obtain reviews and addresses for each store and finally Q_4 computes the distance between the initial entered address and each store. As a result the user is presented with a tabular view of the aggregated data and can now decide which TV set meets her desired criteria.

2 ACQUISITION OF NEW DEEP WEB SOURCES

Because our framework is implemented as an extension to the Firefox browser, new DW sources can be integrated while surfing the Web similar to PiggyBank (Huynh et al., 2007). But unlike PiggyBank we consider dynamic Web pages that can only be accessed by filling out HTML forms. To transform such a DW source into a machine-processable QI, we have to identify the implicitly associated signature, i.e. the labels and datatypes of the input arguments of the form and the labels of datatypes of the data records that are buried in the result page. Additionally each signature has to be associated with a configuration that assures that the correct form fields are filled in and the data records in the result page are extracted and labelled correctly. Because our framework is geared towards a non-expert user, the technical details of the configuration generation are transparent. It is sufficient to label the relevant input form field elements and the column headers of a tabular representation of the result page.

2.1 User Vocabulary

The labeling of DW sources is inspired by the idea of social bookmarking: each user has a personal, evolving vocabulary of tags. Here a tag is the combination of a string label with an XML datatype (Biron and Malhotra, 2004). The user can generate new tags any time during the labeling process and can thereby successively build up her own terminology. The associated datatype is used for data cleaning purposes and to check that only meaningful comparison operators are used in the final mashup. As a means to organize the vocabulary the user can specify relationships between different tags, if she wants. For this purpose we identified three types of meaningful relationships thus far, called tag rules:

1. **Equality:** Two tags refer to the same concept (indicated by $\equiv$). If the tags refer to similar concepts but are given with respect to different reference systems, e.g. different currencies, then the user can register conversion functions to assure compatibility,
2. **Compound tag:** Two or more tags that are concatenated (indicated by $+$) are equal to another concept,
3. **Subtag:** One tag is more specific than another tag (indicated by $\sqsubseteq$).

Table 1 illustrates the abovementioned tag rules with an example. The system assures that the user can only specify rules that are datatype compatible to encourage meaningful rules.

2.2 Deep Web Query Interfaces

To enable easy and accurate labeling of input arguments the relevant tags can be dragged to the elements of the selected form. When the user submits the form, her label assignments and form submission related information, i.e. the POST or GET string is captured. As a result the system can now interact with the Web source and delegate values for the input arguments appropriately. In the next step, the result page is analyzed and converted into a tabular representation.
The analysis of the result page is done by the fully automatic data extraction system ViPER (Simon and Lausen, 2005). ViPER suggests identified structured data regions with decreasing importance to the user

Table 1: Supported tag rules.

#	Relationship	Example
1	Equality	film $\equiv$ movie
2	Equality and Function	price-eur $\equiv$ convert(price-usd, price-eur)
3	Compound tag	firstName + familyName $\equiv$ name
4	Subtag	posterPicture $\sqsubseteq$ picture

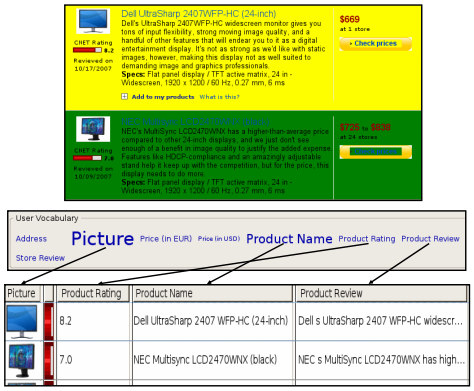

Figure 2: User-defined vocabulary, depicted as tag cloud.

based on visual information. In general the foremost recommendation meets the content of interest and thus is suggested by default. On the other hand it is also possible to opt for a different structured region if desired. Regardless of the selection the extraction system always tries to convey the structured data into a tabular representation. This rearrangement enables to automatically clean the data and serves as a comfortable representation for labeling. Again, the user can label the data by dragging tags of her personal vocabulary as depicted in Figure 2[4]; this time to the columns of the resulting table consisting of extracted instances. The picture at the top shows an excerpt of the original Web page and the tabular representation at the bottom shows the relevant part of the ViPER rendition of this page. The arrows indicate the tags the user has dragged to the column headers. Having in mind that the data originates from a backend database the extraction and cleaning process can be seen as reverse engineering on the basis of materialized database views published in HTML. For the data cleaning process the rule-based approach presented in (Simon et al., 2006), which allows to fine-tune the extraction and cleaning of data from structured Web pages, is used. For the basic datatypes, such as integer and double a heuristics-based standard parser is provided. The user has the chance to additionally implement and register her own parsers, e.g. a parser specifically for addresses, which can be arbitrarily sophisticated. The system then decides at runtime which parser is used based on the annotated data type.

The assigned labels constitute the functionality characteristics of a QI. We additionally consider two non-functional properties for each QI: the average response time, denoted t_{avg}, and the maximum number of possible parallel invocations of the QI, denoted k. We can determine t_{avg} by measuring the response time of a QI over the last N uses and then taking the arithmetic mean, i.e. $t_{avg} := \frac{1}{N} * \Sigma_{i=1}^{N} t_i$. Here, t_1 is the response time of the last invocation, t_2 the response time of the invocation before that and so on, and N denotes the size of the considered execution history, e.g. ten. We initially set $t_{avg} := \infty$.

Since each QI is essentially a relational DB with a form-based query interface, which is designed to handle requests in parallel, we can invoke a QI multiple times with different input parameters in parallel. Obviously, the number of parallel executions needs to be limited by an upper bound, because otherwise we would perform a Denial of Service attack on the Web source. To assure this, we set $k := 3$ for all QIs for now, but we are working on a heuristic to update k to reflect the capabilities of the underlying QI more faithfully. In the remainder of the paper the following more formal definition of a QI is used:

Definition 1 (Query Interface (QI)). A query interface Q_i is the quintuple (I, O, k, t_{avg}, d), where I is the set of tags that have been dragged to input arguments (the input signature) and O is the set of tags that have been dragged to columns in the tabular representation of the result page (the output signature), k is the maximum number of parallel invocations, t_{avg} is the average response time, and d is a short description of the QI.

The provided input arguments of a QI are used to fill out the form, which is then submitted. The data records contained in the result page are then extracted, cleaned and returned as RDF triples (Manola and Miller, 2004). RDF is used as data format because data can be represented in an object-centered fashion and missing or multiple attributes are supported. The integration of data from different sources, explained in Section 4, is another place where the properties of

[4]The vocabulary is depicted as tag cloud (Hassan-Montero and Herrero-Solana, 2006) based on the frequency of the used tags.

RDF are convenient. Generally, an RDF graph is defined as a set of triples $\text{oid}_i := (s_i, p_i, o_i)$, where the subject s_i stands in relation p_i with object o_i. In the following we use $\text{ID}(s_i) = \{\text{oid}_1, \ldots, \text{oid}_n\}$ to denote the set of triples where the subject is identical to s_i, akin to the definition of an individual proposed in (Wang et al., 2005). Similarily, $\pi_2(\text{ID}(s_i)) := \{p_i \mid (s_i, p_i, o_i) \in \text{ID}(s_i)\}$ is the set of all RDF properties of an individual. In the remainder of the paper, we use the notation $\pi_i(\mathfrak{R})$ to project to the ith position of an n-ary relation $\mathfrak{R}$, where $1 \leq i \leq n$, and the terms tag and property will be used interchangeably either to identify an item in the user's vocabulary or to denote its counterpart in the RDF result graph.

3 ASSISTED MASHUP GENERATION

The previous section has illustrated how new DW sources can be integrated and how QIs are formally defined. We can use this information to assist users in generating data mashup graphs without writing a single line of code. The basic idea is that a subset T of the set of all used tags $\mathfrak{T}$ can be interpreted as properties of $\pi_2(\text{ID}(s_i))$ analogously to the Universal Relation paradigm presented in (Maier et al., 1984). In fact, (Davulcu et al., 1999) used the same approach to describe a user query, with the difference that the attributes were defined by the system and in our scenario each user can utilize her own vocabulary. This alleviates the unique role assumption[5] in our case to some extent because the user has a clear conception of the meaning of the tag. Additionally since we assist the user in incrementally building a mashup graph she can further decide on the intended meaning of the query by selecting appropriate sources.

Figure 3 illustrates the mashup generation process. The user first drags the desired tags into the goal tags pane (shown at the bottom). Here she can additionally specify the desired constraints, e.g. that the price should be smaller than 450 EUR. Based on these goal tags, the system assists her in choosing QIs that can contribute to the desired results. By clicking on each start tag the system recommends relevant input QIs. She can select the ones she likes and then iteratively refine the mashup by invoking the recommendation function for each input QI again. When she is satisfied with the generated data mashup graph, the system collects all missing input arguments. These constitute the global input for the data mashup and are shown

[5]The name of a tag unambigiously determines a concept in the real world.

Figure 3: Excerpt of a result specification for an electronic device.

Recommendations for $\mathbf{Q_5}$:

QI	Description	Rank
Q_1	Product reviews and ratings for electronic devices… (more)	0.42
Q_8	Product reviews and ratings for books… (more)	0.42
Q_9	Movie information… (more)	0.16
…	…	…

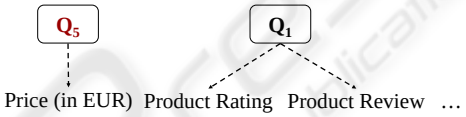

Figure 4: Ranked list of proposed QIs, if the recommendation system is invoked for $\mathbf{Q_5}$.

in the box above the mashup graph in Figure 1. After finishing the mashup generation, the user can store the generated mashup graph for future use as well.

Each time the user invokes the recommendation system, a ranked list of the best matching QIs is computed and shown based on the coverage of input arguments and the utilization frequency in the past (cf. Figure 4). More formally, let $Q_{selected}$ denote the selected QI. Thereby, the rank of each QI with respect to $Q_{selected}$ can be computed as follows:

Definition 2 (QI Rank). $R(Q_i) := w_1 * \text{coverage}(Q_i) + w_2 * \text{utilization}(Q_i)$, where:

- $\text{coverage}(Q_i) := \frac{|\text{ext}(\pi_2(Q_i)) \cap (\pi_1(Q_{selected})|}{|(\pi_1(Q_{selected})|}$; $\text{ext}(T)$ expands the original set of tags T with all tags that are either equal or more specific with respect to the user's tag rules,
- $\text{utilization}(Q_i) := \frac{f(Q_i)}{f_{total}}$; $f(Q_i)$ is the amount of times Q_i has been used in data mashups before, and f_{total} is the total amount of used QIs. If $f_{total} = 0$, i.e. the first mashup graph is generated, $\frac{f(Q_i)}{f_{total}} := 1$,
- $\Sigma_{i=1}^{n} w_i = 1$, where n is the number of available QIs. Intuitively, the weights w_i balance the influence of coverage vs. utilization frequency of a QI,
- Since $\text{coverage}(Q_i) \in [0,1]$ and $\text{utilization}(Q_i) \in [0,1]$, given the constraints on the weights above, we can assure that $R(Q_i) \in [0,1]$, where 1.0 indicates a perfect match and 0.0 a total mismatch.

We additionally distinguish between two special cases: If the rank function is computed for a start tag T_i, we define $\pi_1(Q_{selected}) := \{T_i\}$ and if two QIs have the same rank, they are re-ordered with respect to their average response time t_{avg}.

The coverage criterion favors QIs that can provide a large portion of the needed input arguments. The utilization criterion captures the likelihood of a QI to be relevant without regard for the signature. Therefore it is intended to provide an additional hint when choosing between different QIs that match the input signature to some extent. Thus, w_1 should be chosen, such that $w_1 \gg w_2$, e.g $w_1 = 0.8$ and $w_2 = 0.2$.
Figure 4 shows a use of the recomendation function. The user has already identified two QIs she wants to use and now wants a recommendation for an input to Q_5. The system then presents her with a ranked list of matching QIs and connects the QIs appropriately after she made her choice. To assure a meaningful data flow between the QIs, we require that the final mashup graph is weakly connected, i.e. replacing all directed edges with undirected edges results in a connected (undirected) graph.

4 MASHUP EXECUTION

The mashup graph determines the order, in which the QIs are accessed. All QIs whose input vertices have either already been accessed or that do not have an input vertice, can be queried in parallel. The access to a QI consists of two phases: first the valid input combinations need to be determined based on the input vertices and the output values of the QI have to be integrated afterwards.

4.1 Computing Query Combinations

As mentioned above it is not advisable to query all possible value combinations. Therefore we need a way to identify the top m combinations, which are most likely to yield relevant results. (Raghavan and Garcia-Molina, 2001) presented three different ranking value functions for improving submission efficiency in a Deep Web crawler scenario. They incrementally collected possible value assignments for form elements and assigned weights to them. A weight is increased each time a value assignment is used successfully in a form submission and decreased otherwise.
DW sources usually return a ranked list of results,

Input QIs. Results are identified via their rankings:

- $Q_1 = \{1,2,3,4\}$
- $Q_2 = \{1,2,3\}$
- $Q_3 = \{1,2\}$

Resulting combinations:

#	$R_{overall}$	R_1	R_2	R_3
1	3	1	1	1
2	4	1	1	2
3	4	1	2	1
4	4	2	1	1

Figure 5: Query combination computation for a QI with three different input QIs where $m = 4$.

which we interprete as an assigned weight[6]. Therefore we can similarily compute a rank for each value combination based on those weights. Thus $R_{overall} := \Sigma_{i=1}^{n} R_i$ is the overall ranking, where R_i is the ranking of a value from the ith DW source. Since the ranks of the input QIs usually depend on their predecessors as well, the notion of accumulated ranks is introduced in the next section, which considers the access history of each QI. Figure 5 illustrates the computation for three QIs which each return a different number of results[7]. In this example m is four and therefore we would query the first three combinations in parallel, since we decided to have $k = 3$ in Section 2.2, and afterwards the fourth combination.

4.2 Data Integration

The final goal of the integration process is to align the triples that were generated during query time in such a way, that each set $\pi_2(\mathrm{ID}(s_i))$ contains all required goal tags that were initially specified by the user. Hence, if multiple Web QIs contribute properties to the overall result as in this scenario, different sets $\mathrm{ID}(s_i)$ need to be merged, split or aggregated in the data reconciliation process.
Tables 2 and 3 are used to further illustrate the data reconciliation process. Table 2 shows an excerpt of the state of the global triple store after the first source of the mashup (Q_1) depicted in Figure 1 has been accessed and all results have been inserted. Since this was the first QI, which has been accessed, no aggregation, splitting or merging operations are necessary.
Table 3 shows the global store after the next QI (Q_5) has been accessed. The original triple set $\mathrm{ID}(s_1)$

[6]If no ranking information is provided, we assign to each value the rank 1.

[7]Note that in a real-life scenario the accumulated ranks are usually not integers.

Table 2: Triple Store *before* accessing Q_5.

#	S	P	O
1	s_1	product-name	”Sharp Aquos”
2	s_1	product-rating	80.0
3	s_1	source	Q_1
4	s_1	rank	1
...	...	...	...

Table 3: Triple Store *after* accessing Q_5.

#	S	P	O
1	s_1	product-name	Sharp Aquos
2	s_1	product-rating	80.0
3	s_1	source	Q_1
4	s_1	rank	1
5	s_1	store-name	Amazon
6	s_1	price-eur	1120.00
7	s_1	rank	1
8	s_1	source	Q_5
9	s_3	product-name	Sharp Aquos
10	s_3	product-rating	80.0
11	s_3	source	Q_1
12	s_3	rank	1
13	s_3	store-name	Dell
14	s_3	price-eur	1420.00
15	s_3	source	Q_5
16	s_3	rank	2
...	...	...	...

has been split in two branches: one with the information about the price for the device at Dell (the triple set ID(s_3)) and the other with information about Amazon (the triple set ID(s_1)) and aggregated with the related pricing data. This split is important, since if all data had been aggregated around s_1 the connection between store and price would have been lost. Another aspect which is shown in Table 3 is that we accumulate the source and ranking information. The source information is used to fill the according upcoming QIs with the correct data and the ranking information is used to estimate the accumulated ranking of a subject, which is used as input for the query combination computation described in Section 4.1. The accumulated rank is the arithmetic mean over all ranks; $R_{accumulate} := \frac{1}{n}\Sigma_{i=1}^{n} R_i$, where n is the number of rankings.

After all branches have finished executing, all output values occuring from different QIs in a branch are automatically associated with each other depending on the query history, i.e. the respective input values. Hence in our running example the branches (Q_1, Q_5, Q_2) and (Q_1, Q_5, Q_3, Q_4) are already integrated. To correlate the information between this branches we merge the respective sets ID(s_i) based on their common properties. Note that this is always possible, because as described in Section 3 we only allow weakly connected, directed mashup graphs and therefore each set ID(s_i) in a different branch shares at least on property.

Finally the results are selected with an automatically generated SPARQL (Prud'hommeaux and Seaborne, 2007) query, where all goal tags occur in the `SELECT` clause and the user-specified constraints are checked in the `WHERE` clause. The integrated results are then presented in a tabular representation, as if the user had just queried a given RDF store, which already contained all relevant information.

5 RELATED WORK

Even nowadays many frameworks for querying heterogeneous sources present the end user with a predefined query interface or a fix query language she has to learn in order to pose more complex queries to the system. One of the main drivers behind the idea of Web 2.0 is that users can control their own data. That might be one reason why data-centric mashup tools are such a strong trend at the moment. Here the user is a first-class citizen and can arrange her sources in an intuitive ad hoc fashion. Representative frameworks to build such mashups based on distributed services or structured data are for instance YahooPipes[8], IBM's QEDwiki[9] and MashMaker (Ennals and Garofalakis, 2007). YahooPipes provides an interactive feed aggregator and manipulator to quickly generate its own mashups based on available XML related services. Additionally, YahooPipes realizes data aggregation and manipulation by assembling data pipes from structured XML feeds by hand. QEDwiki is similar in spirit and is implemented as a Wiki framework, which offers a rich set of commands with a stronger focus on business users. MashMaker supports the notion of finding information by exploring, rather than by writing queries. It allows to enrich the content of the currently visited page with data from other Web sites. Our system differs in the way that we focus on unstructured resources and aim to provide a user with strong automatic mechanisms for data aggregation. As a consequence we concentrate on the problems yielding from Web resource integration, such like form field interaction, automatic content extraction, and data cleaning. Furthermore we assist the user during the assembly of the mashup with our recommendation service, whereas in the above

[8] `http://pipes.yahoo.com/pipes/`

[9] `http://services.alphaworks.ibm.com/qedwiki/`

examples the sources have to be manually wired together. In contrast to MashMaker our idea is not to enrich the content of one Web page but to provide an integrated view of multiple sources to aggregate information about a specific topic.

Since our approach is focused on DW sources, we need to solve the problem of extracting and labeling Web data in a robust manner. (Laender et al., 2002) have shown in a survey the most prominent approaches to Web data extraction. More recent frameworks for this task are Dapper[10], which is focused on extraction of content from Web sites to generate structured feeds. It provides a fully visual and interactive web-based wrapper generation framework which works best on collections of pages. Thresher (Hogue and Karger, 2005), the extraction component of the standalone information management tool Haystack (Karger et al., 2005), facilitates as well a simple but still semi-supervised and labor-intensive wrapper[11] generation component. Another more complex supervised wrapper generation tool, where the generation of the wrapper is also done visually and interactively by the user, is Lixto (Baumgartner et al., 2001). Our proposal does not rely on an interactive wrapper generation process which is feasible at a small scale where sources can be manually wrapped. Instead a user only has to label form elements and automatically extracted content with concepts of her own vocabulary. Thus, the integration of new Web resources is totally automated except of the annotation process. This meets the problem of large scale information integration where many tasks have to be automated and wrapper maintenance is an issue.

Another related field are Web search engines, which usually offer a keyword-based or canned interface to DW sources. Recently proposed complete frameworks for querying the Web, especially the deep Web are (He et al., 2005) and (Chang et al., 2005) for instance. They cover tasks like Deep Web resource discovery, schema matching, record linkage and semantic integration tasks. Although there are many similarities between these approaches and our proposed system, they are operating in a bottom-up approach, starting with the Web sources, whereas we model the world from a user-oriented perspective. Additionally we support an ad hoc combination of heteregenous data sources, which is not supported by these frameworks.

Because our underlying data model is an adapted version of the Universal Relation assumption we share some elements with the architecture proposed in (Davulcu et al., 1999). Especially the idea to bootstrap the mashup generation process with a set of goal tags is similar to the way queries can be specified in their framework. However, in our approach the user can organize his vocabulary without any dependency on domain experts, and the way queries are build on top of the initial specification is different as well. Finally, the Semantic Web application Piggy Bank (Huynh et al., 2007) is also focused on the conversion of Web information into semantically enriched content, but requires as well skilled users to write screen scraper programs as information extraction components. The main idea is that users can share their collected semantically enriched information. We share with Piggy Bank the idea to semantically enrich the Web and the idea of an omnipresent browser extension accompanying a user during his daily surf experience. In contrast we do not store the information, instead we are interested in querying fresh information on-the-fly with fast and robust extraction and annotation components.

6 FUTURE WORK

To integrate more sophisticated DW sources which make extensive use of JavaScript, we are currently investigating the use of navigation maps as proposed in (Davulcu et al., 1999). Moreover this would allow us to incorporate sources with multi-form interactions, where the final result pages are only shown after several interactive user feedback loops.

As a side-effect or our implementation efforts, we were surprised that there seems to be no support for shared property values between different subjects in existing RDF stores. As this can happen easily in our approach when a split occurs, as described in Section 4.2, we are implementing an RDF store, which is optimized for the needs of our framework, at the moment.

Finally, the user-centric nature of our approach makes it an excellent candidate for integrating collaborative intelligence or human computation (von Ahn and Dabbish, 2004), i.e. to ease the large-scale annotation of DW sources, which is in a way similar to the way tagging is done in social networks, such as del.icio.us[12]. Currently we are researching the use of machine learning approaches to recommend tags for new DW sources based on the user's labeling behavior in the past.

[10] http://www.dapper.net/

[11] A wrapper in this context is only concerned with the extraction of data from Web pages and does not provide any transformation of query capabilities.

[12] http://del.icio.us/

7 CONCLUSIONS

Many real-life queries can only be answered by combining data from different Web sources. Especially Deep Web (DW) sources offer a vast amount of high-quality and focused information and are therefore a good candidate for automatic processing. In this paper we presented a framework which allows to convert these sources into machine-accessible query interfaces (QIs) by tagging the relevant input arguments and output values. Since the framework is geared towards non-expert users the whole acquisition can be done without writing a single line of code.
In the next step users can iteratively build a mashup graph in a bottom up fashion, starting with the desired goal tags. Each vertex in the resulting mashup graph represents a QI, and edges organize the data flow between the QIs. To reduce the modeling time and increase the likelihood of meaningful combinations, she can invoke a recommender for each vertex, which returns a ranked list of possible new input QIs.
The execution strategy for the thus generated mashup graphs is based on the idea to process the most likely value combinations for each QI in parallel while avoiding to access a particular DW source too often in a given time frame. Although a modular architecture based on an analysis of the main challenges has been proposed, more experimental work needs to be done to evaluate the practicability of our framework and to fine-tune the QI recommender.

REFERENCES

Baumgartner, R., Flesca, S., and Gottlob, G. (2001). Visual Web Information Extraction with Lixto. In *VLDB*, pages 119–128.

Biron, P. V. and Malhotra, A. (2004). XML Schema Part 2: Datatypes Second Edition. http://www.w3.org/TR/xmlschema2/.

Chang, K. C.-C., He, B., and Zhang, Z. (2005). Toward Large Scale Integration: Building a MetaQuerier over Databases on the Web. In *CIDR*, pages 44–55.

Davulcu, H., Freire, J., Kifer, M., and Ramakrishnan, I. V. (1999). A Layered Architecture for Querying Dynamic Web Content. In *SIGMOD Conference*, pages 491–502.

Ennals, R. and Garofalakis, M. N. (2007). MashMaker: Mashups For the Masses. In *SIGMOD Conference*, pages 1116–1118.

Hassan-Montero, Y. and Herrero-Solana, V. (2006). Improving Tag-Clouds as Visual Information Retrieval Interfaces. In *InScit2006*.

He, B., Patel, M., Zhang, Z., and Chang, K. C.-C. (2007). Accessing the Deep Web. *Commun. ACM*, 50(5):94–101.

He, H., Meng, W., Yu, C. T., and Wu, Z. (2005). WISE-Integrator: A System for Extracting and Integrating Complex Web Search Interfaces of the Deep Web. In *VLDB*, pages 1314–1317.

Hogue, A. and Karger, D. R. (2005). Thresher: Automating the Unwrapping of Semantic Content from the World Wide Web. In *WWW*, pages 86–95.

Huynh, D., Mazzocchi, S., and Karger, D. R. (2007). Piggy Bank: Experience the Semantic Web Inside Your Web Browser. *J. Web Sem.*, 5(1):16–27.

Karger, D. R., Bakshi, K., Huynh, D., Quan, D., and Sinha, V. (2005). Haystack: A General-Purpose Information Management Tool for End Users Based on Semistructured Data. In *CIDR*, pages 13–26.

Laender, A. H. F., Ribeiro-Neto, B. A., da Silva, A. S., and Teixeira, J. S. (2002). A Brief Survey of Web Data Extraction Tools. *SIGMOD Record*, 31(2):84–93.

Maier, D., Ullman, J. D., and Vardi, M. Y. (1984). On the Foundations of the Universal Relation Model. *ACM Trans. Database Syst.*, 9(2):283–308.

Manola, F. and Miller, E. (2004). RDF Primer. http://www.w3.org/TR/rdf-primer.

Prud'hommeaux, E. and Seaborne, A. (2007). SPARQL Query Language for RDF. http://www.w3.org/TR/rdf-sparqlquery/.

Raghavan, S. and Garcia-Molina, H. (2001). Crawling the Hidden Web. In *VLDB*, pages 129–138.

Simon, K., Hornung, T., and Lausen, G. (2006). Learning Rules to Pre-process Web Data for Automatic Integration. In *RuleML*, pages 107–116.

Simon, K. and Lausen, G. (2005). ViPER: Augmenting Automatic Information Extraction with Visual Perceptions. In *CIKM*, pages 381–388.

von Ahn, L. and Dabbish, L. (2004). Labeling Images With a Computer Game. In *CHI*, pages 319–326.

Wang, S.-Y., Guo, Y., Qasem, A., and Heflin, J. (2005). Rapid Benchmarking for Semantic Web Knowledge Base Systems. In *ISWC*, pages 758–772.