CEUR-WS.org/Vol-1211
urn:nbn:de:0074-1211-3

Challenge+DC@RuleML 2014
Rule Challenge and Doctoral Consortium @ RuleML 2014

- **The 8th International Rule Challenge**
- **The 4th RuleML Doctoral Consortium**

Proceedings of the RuleML 2014 Challenge and the RuleML 2014 Doctoral Consortium

hosted by the 8th International Web Rule Symposium (RuleML 2014)

Prague, Czech Republic, August 18-20, 2014.

Edited by

Theodore Patkos *
Adam Wyner **
Adrian Giurca ***

* FORTH-ICS, Greece
** University of Aberdeen, UK
*** Brandenburg University of Technology Cottbus - Senftenberg, Germany

Table of Contents

20145-08-11: submitted by Adrian Giurca
2014-08-17: published on CEUR-WS.org |valid HTML5|

Extracting Data from the Deep Web with Global-as-View Mediators Using Rule-Enriched Semantic Annotations

Benjamin Dönz[1], Harold Boley[2]

[1]Vienna University of Technology, Institute of Computer Technology, Vienna
[2]University of New Brunswick, Faculty of Computer Science, Fredericton, NB, Canada
doenz[AT]ict.tuwien.ac.at, harold.boley[AT]unb.ca

Abstract. The Deep Web offers approximately 500 times more information than the Open Web, but is "hidden" behind search-forms intended for human users, and typically requires interaction, which makes it difficult to index by Web crawlers. We argue that traditional data extraction is therefore not suitable for the Deep Web and suffers from coverage problems similar to those search engines face when trying to index its content. Instead, it is proposed to transform and forward queries on demand using Global-as-View Mediators. To allow automated interaction with databases on the Deep Web, we use rules that exploit features (e.g. HTML attribute values) to identify elements on a Web page and infer semantic annotations that link these elements to known concepts (e.g. query parameters or result values). Using a prototypical implementation, Deep Web Mediator, the performance of this approach is demonstrated in a classified-advertising use case. Our system is able to answer complex queries by transforming and forwarding them to multiple sites as well as integrating the local results.

Keywords: Semantic Technologies, Mediated Data Access, Data Integration, Distributed Querying, Deep Web, Global-As-View Mapping, Rule-Based Mapping.

Introduction

The conventional approach for querying a set of independent databases is ETL (Extract – Transfer – Load): Here, the data from the local sources is transformed to fit a global schema and loaded into a common database, which is then used to answer queries. This transformation has to be done beforehand, for example every day at night. This means that the data available for queries is not up-to-date and only reflects the state at which it was last extracted. If this discrepancy or the effort and resources required for the ETL process are not acceptable, the alternative is to transform the query rather than the data, and forward subqueries to the local sources at query time. This is referred to as a mediator-based approach [1], since a third party, i.e. the mediator, accepts the initial query and then forwards it on behalf of the initiating party.

While querying multiple sources is the main task in data warehousing, the task of integrating information from disparate sources is not limited to businesses: anyone

who is looking for a new apartment on the Web must sift through offers on realtor websites, and anyone that is trying to find the best deal for a specific product or service is faced with a similar problem. Rather than being provided with a single interface, users must access several sites and manually compile the results of individual queries. The part of the World Wide Web that includes the database content accessible through Web shops, real estate offers and other database-interfaced portals is referred to as the "Deep Web", and is estimated to contain approximately 500 times more information than the common "Surface Web" [2]. Since these systems require filling out fields and calling functions, it is considered hidden from search engines, because Web crawlers generally only follow hyperlinks from document to document. Published results for existing approaches to bring this information to the "surface" using Web crawlers reveal that the coverage is as low as 10% - 30% [3].

In any case, the content that can be accessed is not used to directly answer queries, but only to direct the user to the site. For answering queries, an approach similar to ETL in data warehousing could be applied. However, in the context of the Deep Web, the actual data source is normally not directly accessible. This means that the data must be extracted using the available Web forms. An extraction tool must therefore submit suitable keywords and permutations of different parameters. Not only can this produce considerable traffic to the site, the extraction program cannot be sure if all of the content was actually accessed, resulting in a coverage problem similar to that search engines have. In addition, it is unknown when records change. Therefore, frequent periodic extraction runs are required to keep the database up to date, which may be not be possible due to limitations of the site.

So, while extracting data beforehand does not seem suitable for the Deep Web, a query forwarding approach may be a viable alternative: if the parameters of the initial query can be used to fill out the fields on the query form, submitting permutations of parameters will not be necessary reducing the traffic to the site and overcoming the coverage problem, while also returning up-to-date information without requiring frequent extraction runs. This paper therefore proposes a solution that allows access to data on the Deep Web by offering a SPARQL endpoint that transparently unfolds and forwards queries to multiple Web databases, returning the integrated results. To allow interaction with these sites, the elements of the pages are annotated by evaluating rules that use features they expose, e.g. the tag name or adjacency to certain labels, to identify and link them to known semantic concepts.

Related Work

Before describing the proposed system, the required background on mediators and existing solutions for data extraction and other approaches for semantic access to the Deep Web are discussed.

Accessing Data with Mediators

Given a number of independently developed systems, it is very likely that data is not structured in the same way and that different terms are used to describe equivalent

concepts even if all of the systems are in the same domain. To answer queries that span multiple databases, it is therefore necessary to align the data structure and vocabulary used in the query with that used in the individual databases. While this is done beforehand in an ETL approach by transforming and loading the data from all the individual systems into a common database, a mediated approach allows doing corresponding processing at query time. Here, a third party, i.e. the mediator, accepts the initial query and then transforms and forwards it on behalf of the initiating party [1]. To allow this, a mapping between the global schema and that of the local databases is required. This can be provided either by defining a mapping from the global ontology to the local schemas, e.g. in the form of a view that contains all the relevant sources (Global-as-View) [4], or by doing the opposite, i.e. creating a mapping for each source to the global schema (Local-as-view) [5]. In both cases, the resulting query consists of a set of subqueries that are forwarded to the individual sites to retrieve intermediate results, which are integrated and used to answer the initial query.

Besides the advantage of allowing integration at query time, this allows using a high level conceptual view of a domain by defining an ontology that is independent of that used by the actual data sources. Commercial systems such as Openlink Virtuoso's Sponger technique[1] or Ontobroker[2] use mapping rules to integrate various types of sources ranging from relational databases to file-based sources such as XML and also Web services, and offer the possibility to execute SPARQL queries that span multiple sources. Examples of academic projects in this field include Quest[6] and Mastro Studio [7]. Ontology-Based Data Access (OBDA) is also based on this approach and current efforts in this field, e.g. Optique [8], aim at offering a unified ontology based query interface that uses mediators to integrate not only conventional databases, but also other sources like data streams, XML or Excel documents.

Semantic Access to the Deep Web

As shown in **Fig. 1**, four different methods for semantic access to the data on the Deep Web can be identified and categorized along two dimensions – extraction on demand vs. extraction beforehand and direct access to the database vs. access via a Web form interface. The first of these approaches is source conversion: Here, data is converted from its original format to a semantic database, which is then used to answer queries, similar to the ETL approach. This conversion can be done either manually or by leveraging the database schema. In the case of relational databases, for example, a transformation is possible by defining a type for each table, a property for each column and individuals with the appropriate type and property values for each record [9]. On this basis, the W3C defined the direct mapping method [10] as a recommendation for a fully automated conversion to RDF. However, in this case merely the format is changed, but the original relational data structure remains, and also the names of the tables and columns are directly used as names for the corresponding types and properties. Since they are not linked to any existing vocabulary, interpreta-

[1] see virtuoso.openlinksw.com/dataspace/dav/wiki/Main/VirtSpongerWhitePaper
[2] see www.semafora-systems.com/en/products/ontobroker/

tion by a program is not possible and a human is generally still required to disambiguate terms or map the automatically generated vocabulary to a full ontology. Hence, it may become necessary to define the mapping manually, for example by using the second W3C recommendation for this task, i.e. R2RML [11], or other tools like Triplify [12]. If the underlying data changes frequently, it may be prudent to transform the query rather than the data itself, which corresponds to the data access manipulation approach. Here, as described in the previous section, a mediator is included as a third party, which transforms and forwards the original query to the individual sources.

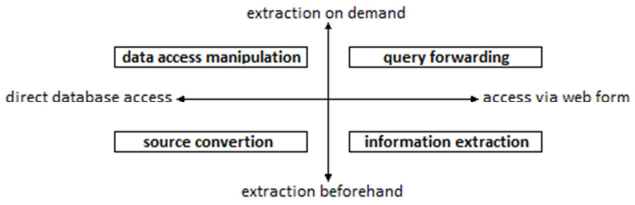


Fig. 1. Methods for semantic access to data on the Deep Web

Both of the approaches described above require access to the underlying data source. In cases where this is not possible, the available interfaces have to be used, e.g. Web forms intended for human users in the case of the Deep Web. Similar to the previously described approaches, data can be prepared beforehand or accessed on demand during query time. As discussed in the introduction, extracting data from Web databases beforehand suffers from two main problems: the first being that updates require extracting the full database every time, since there is no way to tell which records may have changed since the last extraction was performed if this is not explicitly published. While some databases, for example Wikipedia [13], have made such an update notification available, this cannot be expected for all Web databases. The second problem is coverage: since – without access to the source – it is difficult to determine how many records actually exist in order to determine if all of them have been extracted. In addition, some sites require that mandatory parameters are set on a query interface, and only values relevant for that database will produce any results at all. To overcome this, current projects first try to determine the domain, for example by query probing [14], and then submit relevant terms for that domain. However coverage may still be low as shown in [3].

The final approach, i.e. query forwarding, has the advantage that the parameters of the initial query can be used and forwarded to the site instead of “guessing” relevant parameters. Besides this, the data is always up-to-date, and no periodical updates have to be performed that may consume considerable resources by producing high traffic for the sites and store values that may never be required for a query. This approach has also been successfully applied in projects such as MetaQuerier [15] and WISE Integrator [16] to create combined search interfaces that forward queries to multiple sites and return combined results and is also the basis of the OBDA paradigm as mentioned in the previous section.

Web Data Extraction

To forward queries and extract results from a site on the Deep Web, a program has to interact with an interface originally intended for human users. While some sites consist of free text or documents collections, 77.3% of the data on the Deep Web is provided in structured form [3]. If it is known, for example, that a specific table contains one instance or record per row, and every column corresponds to a specific attribute, the content can be extracted from this table and the values can be used to populate a relational database [17] or an ontology [18]. If a set of results is presented in a more complex form than a simple table, the task of determining which attributes exist, and what values they have for each record becomes more demanding. Several information extraction approaches assume that Web databases return query results by filling out fixed templates with values from a database, e.g. [19], [20], [21]. To interact with such sites, information about the purpose of filter fields, or the location of specific result values is required. This set of information is referred to as a wrapper [22]. While several different approaches for defining Web wrappers can been identified, the basic task is to reference an element on the page and link it to a known concept. In order to define such a reference, features ranging from text-based patterns, e.g. phone numbers, to structural features of the HTML document [21], to style-based and visual features of the rendered page [23] can be used.

Regarding the generation of wrappers, projects like [24] or [25] propose fully automatic wrapper generation, but only cover specific domains and search engines. In addition, these projects make certain assumptions about how the site must work since the tools interact with the site in a predefined manner. These assumptions are, however, not clearly defined and the limitations of the approaches are not stated, which makes it difficult to compare solutions. On the other hand, semi-automatic wrapper generation tools are already available as commercial products for both laypersons, e.g. Dapper[3] or Mozenda[4], as well as professionals, e.g. Lixto [26]. While the latter allows many more degrees of freedom, but requires sophisticated configuration, Dapper and Mozenda can only be used within certain limitations that reflect the developers' assumptions about how Web databases work. Another commercial product in this field is Connotate (formally fetch.com)[5]. Here, both the navigational structure of the site as well as a wrapper for each of the involved pages can be defined by employing a semi-automatic wrapper generation process based on machine learning [27]. The authors acknowledge that the rules produced in this way are not comprehensible by the user and hence cannot be fine-tuned even by expert users.

Model for Web Databases

In contrast to existing solutions, the concept proposed in this paper is based on a clearly defined model for Web databases. While this model may not fit every database

[3] see open.dapper.net
[4] see www.mozenda.com
[5] see www.connotate.com

on the Deep Web, it unambiguously allows determining if a given site is compatible and can be included as a source. In addition, the model can be extended systematically to allow including more sites and hence improve the solution incrementally. This section gives an overview of this model originally published in Ref. [28].

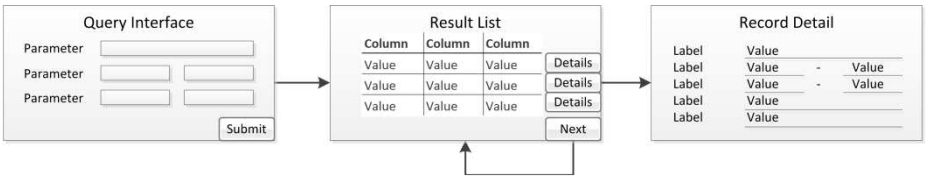


Fig. 2. Interfaces and Navigation

As shown in **Fig. 2**, the model assumes that sites consist of three types of pages: The first is the query interface, which can be reached by a fixed URL and offers fields for defining the parameters of the query. A single submit function allows submitting the query and returns a list of results. Each result may offer a link to a detail page, where further attribute/value pairs for a given record are presented. If the result list is limited to show only a maximum number of results, a "next page" link can be followed to access another result list that presents the next set of records. While the query interface is a fixed Web form, the result list and record detail pages are both template-based, i.e. the basic layout of the page remains the same independent of the records that are returned, merely the presented values change.

The data source is considered to be a single table. Since the real structure is unknown, the actual database does not need to be implemented in this way, but the results must allow assuming this structure[6]. This means that all records have to be of the same type, every record has to have the same attribute/value pairs, and there are no nested subtables. Transforming the content to RDF triples is done by defining a class for the table and a property for each column of the table. Each actual fact is then given similarly to [99] by defining a blank node with the table's type, and then assigning the values of the individual columns using the associated properties.

Based on this data structure and the interfaces described above, the query process itself is then defined to work as follows: each parameter on the query interface is associated with a single column of the table and a filter function, e.g. "equal to" or "less than". When submitting the query, each filter function is evaluated independently for each record, and only records where all filters are evaluated to be true are included in the result set. The model only allows conjunctive filters, but disjunctive queries can be answered by splitting the query into conjunctive subqueries, submitting them separately and creating the union of the results.

While the overall model is deliberately designed to be simple in order to form a base line from which it can be extended, evaluation in two domains, i.e. used car dealerships (111 sites evaluated) and realtor websites (167 sites evaluated), showed that, respectively, 58% and 50% are fully compliant while 79% and 63% of the sites

[6] If a site consists of multiple different interfaces with different parameters and sets of output values it can be treated as a set of independent databases and still meet this requirement.

are at least compatible to the model. In this context, “compliant” means that the sites follow the model to an extent that allows accessing all records and attribute/value pairs, while “compatible” sites allow accessing all records, but not all of their values, for example due to an incompatible detail page.

Our evaluation showed that the main reasons for sites failing to be compatible were mandatory fields in the query interface and additional navigational requirements such as having to select a category, e.g. choose between rental or sale offers for real estates, before the actual query interface can be accessed. Extending the model to include these two features would result in nearly full coverage for these two domains, with the exception of those sites that do not publish structured data. The majority of sites in the two evaluation domains is however already compatible to the basic model. It will therefore be retained in order to explore the limitations of the fundamental concept before adding features to increase coverage.

Query Process

Based on the model described in the previous section, a method for interacting with compatible sites can be derived. This allows developing a mediator that offers a SPARQL endpoint for submitting queries, which are then transparently forwarded to Deep Web sites to return the integrated and extracted results. An overview of this process is shown in **Fig. 3**.

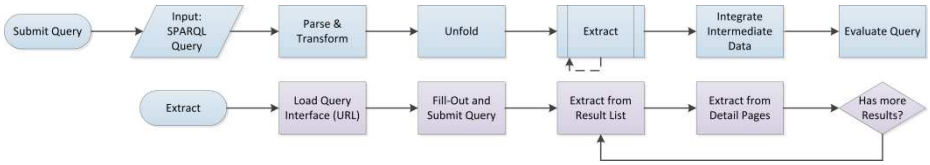


Fig. 3. Query Process Flowchart

In the first row of **Fig. 3**, the outer interface of the system is predefined to be a SPARQL endpoint, and hence the process is initiated by submitting a query. Since the model only supports conjunctions, the filter of the query has to be transformed into its disjunctive normal form and be split into a set of conjunctive subqueries. After determining relevant Web databases by comparing the properties and types used in the query to those published by the available sources, the query is unfolded to include the union of relevant sources in our Global-as-View approach. The resulting subqueries must then be forwarded to the local sources following the subprocess shown in the second row of **Fig. 3**. The information gathered in this way is then integrated as an intermediate data set before the actual query is executed and the results are returned.

The subprocess that forwards the query to the site and extracts the results can be derived directly from the model and requires carrying out the following tasks: the query interface has to be loaded, and filter values included in the subquery that are also available on the page have to be set in the corresponding fields. Following the model, after calling the submit function on the query page, a result list is returned that

contains the top results, which are extracted creating RDF triples as described in the previous section. If detail pages are available, they are accessed to also include the attribute/value pairs presented there. And finally, if a next link is available on the result page, it is followed to access further results and proceed in the same fashion.

In **Fig. 3** the individual steps are color-coded to show which parts of the process are generic (blue) and which parts depend on information about a specific Web database (purple). As can been seen, the controlling outer process is completely generic and only the part of the process that interacts with the site depends on site specific information. Since the elements required for the interaction are defined by the model, the algorithm itself is also generic, but what is needed is a link between each element of the model and the corresponding element on the actual website. To achieve this, it is proposed to use semantic annotations that create this relationship and then use a generic algorithm to perform the actual tasks, e.g. setting or getting values and triggering functions. Since embedded annotations, e.g. using RDFa, would require changing the source of the page, which can only be done by the publisher, it is suggested to infer the annotations using rules that exploit features of elements on the page to determine which annotations to apply. This set of rules can then be encapsulated as a site-specific wrapper, which contains all the non-generic information necessary to allow accessing and interacting with the site.

Rule-Based Annotation

The annotations must be sufficient to allow a generic algorithm to treat any compatible system in the same way. Following the process described in the previous Chapter, this includes setting parameters, submitting the query, extracting results from the list and detail page, and iterating through multiple result lists. To describe all the elements involved, a meta-vocabulary covering the basic terms required for this process is shown in **Table 1**. The elements defined here using RDF terminology form the basis for concrete implementations that include more detailed and specific concepts. The only class is `PageElement`, which represents an element of the Web page's DOM, i.e. an HTML tag. In order to state that a specific field (`_:field`) restricts a property such as `domain:price` to be less or equal than the value given in that field, a triple such as `_:field annot:restrictLessOrEqual domain:price` is used, where `annot:restrictLessOrEqual` is a subproperty of `restrictProperty` in a concrete implementation. Similar to this, other types of restrictions can be defined that allow expressing other filter operations.

`ExecuteFunctionWith` is the base property for all operations and is used to define subproperties for the operations in a concrete implementation. The model in the current state requires exactly three such operations: `executeSearchWith`, `executeNextListWith` and `executeGetDetailWith`. The triple `_:button annot:executeSearchWith "Click"`, for example, states that triggering the click event will execute the search.

Both the elements of the result list and those of the detail pages are linked to the properties of the values they contain using `containsProperty`. This property can

be used directly to define that the inner text of an element is the value for that property, but variations could also be defined to state that a certain attribute value contains the property, e.g. the src attribute of an image. In addition, subproperties of `hasValueFormat` can be used to define a certain data type or format, e.g. the actual value followed by a currency symbol or a specific date format. To group elements, i.e. attribute/value pairs in this case, that belong to the same record, properties based on `belongsToRecord` can be used, for example to state that all elements of the same row of a table belong to the same record.

Table 1. Annotation Meta-Vocabulary

Class/Property	Description
`PageElement`	Class representing an element on a Web page
`restrictProperty`	Base property for all query filters
`executeFunctionWith`	Base property for all operations
`containsProperty`	Base property for all result value associations
`hasValueFormat`	Base property for defining data types or representation of values
`belongsToRecord`	Base property for assigning a value that allows grouping attribute/value pairs of the same record
`hasFeature`	Base property for all features

Finally, the property `hasFeature` is the base property for features an element exposes. This can be a certain tag type, attribute value or also adjacency to other elements, e.g. a label. These features can be discovered by a generic algorithm that analyses a page's DOM.

With reference to the subprocess in the second row of **Fig. 3**, the non-generic information required for interacting with a concrete Web site is provided in the form of annotations using terms based on the meta-vocabulary described above. These annotations must therefore be created before the actual task (e.g. forward query parameters) can be performed. Hence, for each of the purple steps in **Fig. 3**, the page is first analyzed by a generic feature extractor to create a list of `PageElement` instances and assert the features they expose as triples of the form `_:element annot:hasFeature "feature value"`. The rules contained in the wrapper for the site are then used to identify elements on the basis of these features and infer additional annotations, which link them to the concepts of the model and allow the mediator to interpret the page. The rules used for this process are object-centered Datalog rules, whose conditions consist of conjunctions of required features and whose conclusions specify properties that are not feature-related. Since rules thus cannot infer facts that trigger other rules, a single efficient evaluation pass of all rules against a candidate page is sufficient for annotating the page.

To illustrate this approach, an example is shown in **Fig. 4**: The top left part of the figure shows an example query page with parameters for selecting a brand, model and price range. The HTML code of the page is passed to the feature extractor, which returns feature annotations such as those shown to the right, e.g. that the selection box

is adjacent to a label “Brand”, or that the button has the value “Search”. The rules shown in the middle of the Figure, which would be contained in the wrapper for this site, use combinations of these features as the condition to assert the additional annotations. The first rule, for example, states that any element adjacent to the label “Brand” restricts the property “`car:brand`” with an `equalTo` operation as shown in the bottom of the figure. It should be noted that, in practice, the feature extractor returns thousands of triples even for small web pages and that annotation rules typically contain conjunctions of 3 or 4 features per rule. After applying all these rules and inferring the annotations, the elements on the page are linked to the well-known concepts of the model Web database and hence allow a generic algorithm to interact with the site, i.e. set parameters, submit the query by triggering the click event of the button, and extract the results.

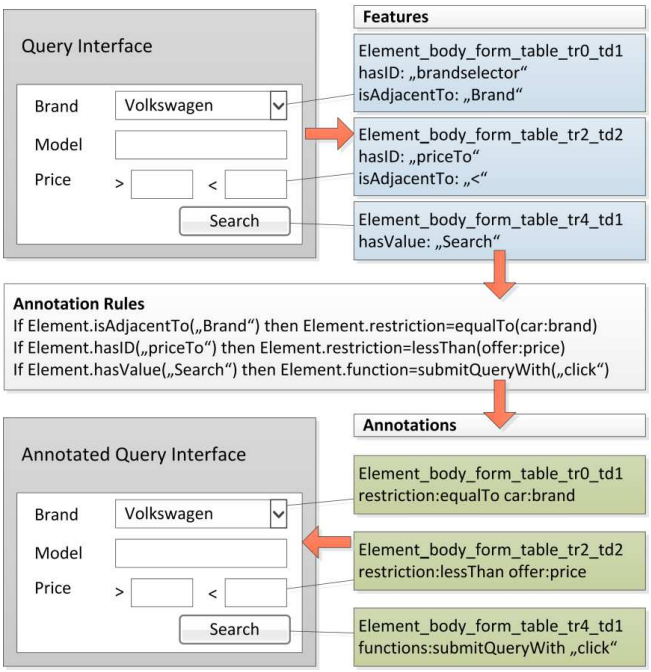


Fig. 4. Example of Rule-Based Annotations

Implementation

The evaluation of our approach is carried out in the domains that were also used to validate the database model, i.e. used cars and real estate. For both domains, a global vocabulary in the form of a small ontology was developed that contains the concepts that can be used for queries in these domains and are mapped to the individual sources. A concrete model taxonomy based on the meta-vocabulary described in the previous section was implemented which offers a set of 15 features that are used to

define annotation rules, e.g. hierarchical position, common attributes and adjacent texts[7]. The model-related properties of the meta-vocabulary were also extended: Regarding functions, subproperties of `executeFunctionWith` were defined for search submission, next link and detail link. For the query interface, concepts for `less`, `lessOrEqual`, `greater`, `greaterOrEqual`, `equal`, `unequal` and `containsString` are defined as subproperties of `restrictProperty`. And finally the `hasValueFormat` property was extended to allow defining the data type and match or replace patterns from the text using regular expressions.

A prototype, the Deep Web Mediator, was also implemented by the first author and offers a standard SPARQL query endpoint that accepts queries passed as the q-parameter of a GET http-command, and additional Web services to upload and download wrappers, as well as a session-aware variant of the SPARQL endpoint that allows accessing intermediate status notifications and additional information, such as the transformed query and included sites for evaluation purposes. The mediator includes a custom parser that allows transforming a query into the style of conjunctive subqueries required by the model and injecting relevant sources from the available wrappers in our Global-as-View approach. Asynchronous control flow allows forwarding multiple subqueries to the local sources in parallel, collecting intermediate results in an RDF store (OpenLink Virtuoso[8]) before finally executing the query. The query process and the interaction with the sites follow a generic algorithm, but use the annotations created by the rules contained in the wrapper to identify the model elements on the actual Web page. The evaluation of these rules is done using a proprietary implementation rather than an existing rule engine so there is no interface related overhead. Our Deep Web Mediator is also available online[9] and offers a standard SPARQL endpoint, a wizard-based GUI, and a set of running examples. Experimental Validation

To evaluate our approach we designed a set of use cases that interact with sources that vary from plain lists to sites incorporating the full model and that cover different types of queries and integration problems. For the domain of used cars, 3 different used-car dealership websites and a site containing test results are used, and for the domain of real estate, 3 different realtor websites and a site containing average lot prices are used[10]. **Table 2** shows the complete list of use cases, which are also available as running examples on the project's website.

The first three use cases cover the different page types of the model: use case #1 uses a site that consists only of the result list, which is split into several pages that can be accessed following a "next" link. Use case #2 accesses a source that requires submitting a query to access the result list, and use case #3 accesses a source that uses all three types of pages, i.e. query interface, result list and record detail. All three use cases are covered and accurately return the records and values relevant for the given

[7] The list of features is available via semann.bdoenz.com/downloads/annotVoc.ttl
[8] The homepage for OpenLink Virtuoso can be accessed via virtuoso.openlinksw.com
[9] Deep Web Mediator project homepage can be accessed via semann.bdoenz.com
[10] The list of sites is given on the project homepage semann.bdoenz.com/default.aspx

filter, which was evaluated by manually submitting the queries and comparing the results.

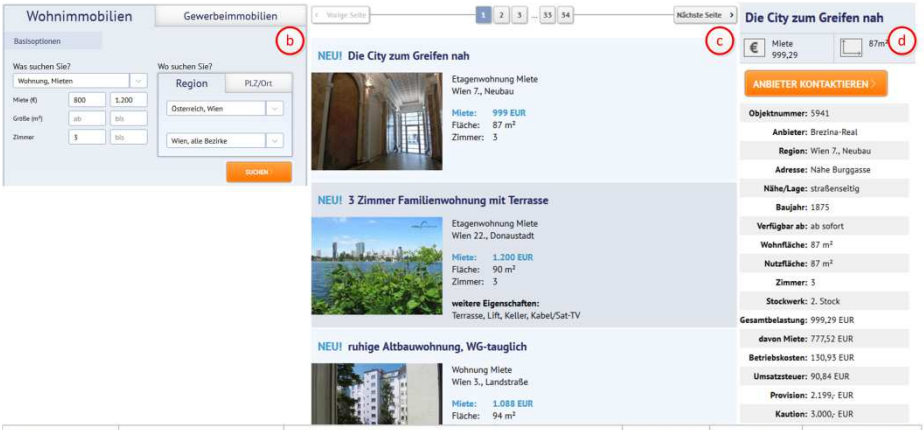
Table 2. Overview of use cases

#	Name	Description
1	Plain list	Extract the average rent per town from a single site.
2	Search and result list	Extract test results on cars of the brand Audi from a single site and return brand, model and the test conclusion.
3	Search, list and detail page	Extract real estate offers from a single site and return details for offers with 3 or more rooms and a rent of 800€ to 1200€.
4	Disjunctive query	Extract used car offers from a single site and return details of all offers for cars of the brand "Audi" that are priced under 12.500€ if the construction year is after 2011 or under 15.000€ if the construction year is after 2012.
5	Union	Extract used car offers from all available sites and return details of offers for cars of the brand "Audi" that are priced under 12.500€ and have a construction year after 2011.
6	Disjunctive union	Extract used car offers from all available sites and return details of all offers for cars of the brand "Audi" that are priced under 12.500€ if the construction year is after 2011 or under 15.000€ if the construction year is after 2012.
7	Relations between sources	Extract average rents and real estate offers from all available sites and return those that are located in a specific town and have a lower rent/m² than the average for that town.
8	Deep Web and local databases	Extract real estate offers from all available sites and add the type of town and population from a local dataset.
9	Deep Web and external databases	Extract real estate offers from all available sites and add a description of the town and the population from an external SPARQL endpoint (dbPedia).

As an example, **Fig. 5** shows use case #3: The SPARQL query is given at the top (a) of the Figure. It asks for town name, offer name, description, rent, rooms and floor space for offers from a specific site and filters these by restricting the rent to the range of 800€ to 1200€ and the number of rooms to a value greater than 3. Underneath (b-d), the three types of pages of the source site are shown. The first is the query interface. After navigating to this page, our mediator uses the annotations created by the feature extractor and annotation rules to determine the relevant fields, set the parameters of the initial query and submit them. The second screenshot shows the result list for that query, which also offers a link to a detail page for each record that is shown to the right. Again, both the list and detail pages are annotated to allow extracting the values and navigating. The values collected in this manner are then used to answer the query and return the results shown at the bottom of the Figure (e).

```
SELECT ?realestatetownname ?realestateoffername ?realestatedescription ?realestaterent ?realestaterooms ?realestatefloorSpace
FROM <http://derstandard.at/anzeiger/immoweb/Immobilien-suche.aspx>
WHERE {
   ?object rdf:type <http://semannot.bdoenz.com/realestate#RealEstateOffer>;
   <http://semannot.bdoenz.com/realestate#townname> ?realestatetownname;
   <http://semannot.bdoenz.com/realestate#offername> ?realestateoffername;
   <http://semannot.bdoenz.com/realestate#description> ?realestatedescription;
   <http://semannot.bdoenz.com/realestate#rent> ?realestaterent;
   <http://semannot.bdoenz.com/realestate#rooms> ?realestaterooms;
   <http://semannot.bdoenz.com/realestate#floorSpace> ?realestatefloorSpace}.
   FILTER (?realestaterent>=800 && ?realestaterent<=1200 && ?realestaterooms>=3)
}
```

a



e

realestate#townname	realestate#offername	realestate#description	realestate#rent	realestate#rooms	realestate#floorSpace
Naehe Burggasse	Die City zum Greifen nah	Die City zum Greifen nah Das Museumsquartier ums Eck, der 1.Bez...	999,29	3	87
Godlewskigasse	3 Zimmer Familienwohnung mit...	3 Zimmer Familienwohnung mit Terrasse Diese durchdachte Wohn...	1199,75	3	91
	ruhige Altbauwohnung, WG-ta...	ruhige Altbauwohnung, WG-tauglich Sehr schoene Altbauwohnung...	1088	3	95
	ERSTBEZUG: Schicke Neubauw...	ERSTBEZUG: Schicke Neubauwohnung mit Garten Zur Vermietung ...	1150	3	84
Lugeck	Einzigartige Designerwohnung ...	Einzigartige Designerwohnung mit Autoabstellplatz! Einzigartige De...	1184,24	3	107
Wertheimstein-Park	Absolute TOP-Lage - hauseige...	Absolute TOP-Lage - hauseigene Parkanlage! Sanierte 3-Zimmer- ...	990	3	69
Murlingengasse	Exklusiver Erstbezug im Herzen...	Exklusiver Erstbezug im Herzen des 12. Bezirks Exklusiver Erstbezu...	984,16	3	77
Murlingengasse	Exklusives Wohnen im Herzen ...	Exklusives Wohnen im Herzen des 12. Bezirks Exklusives Wohnen i...	945	3	66

Fig. 5. Use Case #3: a) Input query, b) Source's query interface, c) Source's result list, d) Source' record detail, e) Deep Web Mediator Result

The next three use cases listed in **Table 2** cover additional types of queries. While the first three queries only used conjunctions, use case #4 includes a disjunction. This query is transformed and split into two conjunctive subqueries by our Mediator, which are executed independently and returned as a concatenated list. While the previous use cases only accessed a single site, use case #5 accesses all available sites of the domain and returns the union of the collected records. Use case #6 then extends this to also include a disjunction, resulting in 6 sub queries (2 queries for 3 sites).

The final three use cases cover integration from different sources: use case #7 extracts information from a Deep Web site that publishes average rents per square meter and then uses the value to restrict the offers extracted from realtor websites so only those that are below this average are returned. And finally, use cases #8 and #9 combine real estate offers taken from Deep Web Sites with information taken from a local database (#8) and dbPedia (#9).

All nine use cases could be implemented successfully and are available as running examples on the project's homepage.

Conclusion and Outlook

Our approach allows accessing and integrating sources on the Deep Web with Global-as-View mediators. The application of a model as the basis of the extraction process has proven to be valuable in two respects: it allows employing a generic extraction process that can be tested, improved and enhanced independently of the actual sources. Secondly, in contrast to other projects, where the success of the extraction approach is shown by stating the percentage of correctly extracted facts, but the reasons for the success or failure with individual sources cannot be clearly explained, the presented approach involves no such uncertainties: If the site is compatible to the model, which can be determined by comparing a set of constraints derived from the model, it can be reliably included as a source, and if it is not, the reason, i.e. the criteria that could not be met by the site, is clearly defined. Furthermore, the approach is extensible in the sense that the model and the set of features used by the annotation rules can be extended to allow including further sites.

Our next steps will be to extend the model to include additional sites and to use reasoning as a means to integrate different vocabularies that might be used in the query or the sources, but are linked by relations such as `owl:sameAs`. In addition, we are planning experiments to apply machine learning to generate the wrappers. While fully automated, domain-independent wrapper generation may be out of reach, creating rule sets for a site of a specific domain using a manually created training set for the same domain seems feasible and should at least allow us to create an automated suggestion, or possibly even fully applicable rule sets for new sites on demand.

References

1. Wiederhold, G.: Mediators in the architecture of future information systems. Computer 25, 38-49 (1992)
2. Bergman, M.K.: The Deep Web: Surfacing Hidden Value. vol. 7. The Journal of Electronic Publishing (2001)
3. He, B., Patel, M., Zhang, Z., Chang, K.C.-C.: Accessing the Deep Web: A Survey. Communications of the ACM 50, 94-101 (2007)
4. Chawathe, S., Garcia-Molina, H., Hammer, J., Ireland, K., Papakonstantinou, Y., Ullman, J., Widom, J.: The TSIMMIS Project: Integration of Heterogeneous Information Sources. Proceedings of the 10th Meeting of the Information Processing Society of Japan 7-18 (1994)
5. Y.Levy, A., Rajaraman, A., Ordille, J.J.: Querying Heterogeneous Informatin Sources Using Source Descriptions. Proc. of the 22nd Int. Conference on Very Large Databases 251-262 (1996)
6. Rodriguez-Muro, M., Calvanese, D.: Quest, an OWL 2 QL Reasoner for Ontology-Based Data Access. Semantic Web 2, 43-53 (2011)
7. Civili, C., Console, M., Giacomo, G.D., Lembo, D., Lenzerini, M., Lepore, L., Mancini, R., Poggi, A., Rosati, R., Ruzzi, M., Santarelli, V., Savo, D.F.: Mastro studio: managing ontology-based data access applications. In: Proceedings of the VLDB Endowment VLDB Endowment, pp. 1314-1317. (2013)
8. Kharlamov, E., Jiménez-Ruiz, E., Zheleznyakov, D., Bilidas, D., Giese, M., Haase, P., Horrocks, I., Kllapi, H., Koubarakis, M., Özçep, Ö., Rodríguez-Muro, M., Rosati, R.,

Schmidt, M., Schlatte, R., Soylu, A., Waaler, A.: Optique: Towards OBDA Systems for Industry. Lecture Notes in Computer Science 7955, 125-140 (2013)
9. Berners-Lee, T.: Relational Databases on the Semantic Web (1998)
10. Arenas, M., Bertails, A., Prud'hommeaux, E., Sequeda, J.: A Direct Mapping of Relational Data to RDF. World Wide Web Consortium - W3C (2012)
11. Das, S., Sundara, S., Cyganiak, R.: R2RML: RDB to RDF Mapping Language. (2012)
12. Auer, S., Dietzold, S., Lehmann, J., Hellmann, S., Aumueller, D.: Triplify – Light-Weight Linked Data Publication from Relational Databases. In: Proceedings of the 18th international conference on World wide web (WWW'09), pp. 621-630. (2009)
13. Hellmann, S., Stadler, C., Lehmann, J., Auer, S.: DBpedia Live Extraction. Lecture Notes in Computer Science 5871, 1209-1223 (2009)
14. Wang, J., Wen, J.-R., Lochovsky, F., Ma, W.-Y.: Instance-based Schema Matching for Web Databases by Domain-specific Query Probing. In: 30th VLDB Conference, pp. 408-419. (2004)
15. Chang, K.C.-C., He, B., Zhang, Z.: Toward Large Scale Integration: Building a MetaQuerier over Databases on the Web. In: 2nd Biennial Conference on Innovative Data Systems Research (CIDIR), pp. 44-55. (2005)
16. He, H., Meng, W., Yu, C., Wu, Z.: Automatic integration of Web search interfaces with WISE-Integrator. The VLDB Journal 13, 256-273 (2004)
17. Cafarella, M.J., Halevy, A., Wang, D.Z., Wu, E., Zhang, Y.: WebTables: Exploring the Power of Tables on the Web. In: Proc. of the VLDB Endowment, pp. 538-549. (2008)
18. Jannach, D., Shchekotykhin, K., Friedrich, G.: Automated Ontology Instantiation from Tabular Web Sources - The AllRight System. Web Semantics: Science, Services and Agents on the World Wide Web 7, 136-153 (2009)
19. Liu, B., Grossman, R., Zhai, Y.: Mining data records in Web pages. In: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 601-606. (2003)
20. Furche, T., Gottlob, G., Grasso, G., Orsi, G., Schallhart, C., Wang, C.: AMBER: Automatic Supervision for Multi-Attribute Extraction. arXiv (e-Print), (2012)
21. Kahan, J., Koivunen, M.-R., Prud'Hommeaux, E., Swick, R.R.: Annotea: an open RDF infrastructure for shared Web annotations. Computer Networks 39, 589-608 (2002)
22. Laender, A.H.F., Ribeiro-Neto, B.A., Silva, A.S.d., Teixeira, J.S.: A Brief Survey of Web Data Extraction Toolst. SIGMOD Record 31, 84-93 (2002)
23. Liu, W., Meng, X., Meng, W.: ViDE: A Vision-Based Approach for Deep Web Data Extraction. IEEE Transactions on Knowledge and Data Engineering 22, 447-460 (2010)
24. Furche, T., Gottlob, G., Guo, X., Schallhart, C., Sellers, A., Wang, C.: How the Minotaur turned into Ariadne: Ontologies in Web Data Extraction. Lecture Notes in Computer Science 6757, 13-27 (2011)
25. Zhao, H., Meng, W., Wu, Z., Raghavan, V., Yu, C.: Fully Automatic Wrapper Generation for Search Engines. In: World Wide Web Conference (WWW 2005), pp. 66-75. (2005)
26. Gottlob, G., Koch, C., Baumgartner, R., Herzog, M., Flesca, S.: The Lixto data extraction project: back and forth between theory and practice. In: Proc. of the 23rd ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, pp. 1-12. (2004)
27. Minton, S.N., Ticrea, S.I., Beach, J.: Trainability: Developing a responsive learning system. In: Proc. of the 2003 Workshop on Information Integration on the Web, pp. 27-32. (2003)
28. Dönz, B., Bruckner, D.: Extracting and Integrating Structured Information from Web Databases Using Rule-Based Semantic Annotations. In: 39th Annual Conference of the IEEE Industrial Electronics Society. (2013)