A Hybrid Approach to Schema and Data Integration for Meta-search Engines

Tabbasum Naz[1], Jürgen Dorn[1] and Alexandra Poulovassilis[2],

[1] Vienna University of Technology, Institute for Information Systems,
Favoritenstrasse 9-11, A-1040, Vienna, Austria
[2] University of London, London Knowledge Lab, 23-29 Emerald Street,
London WC1N 3QS, United Kingdom
{naz, juergen.dorn }@dbai.tuwien.ac.at, ap@dcs.bbk.ac.uk

Abstract. In this paper, we describe an approach to schema and data integration for meta-search engines. The integration of heterogeneous, distributed information from the Web is a complicated task, especially the task of schema/data matching and integration. During the matching and integration process, we need to handle syntactic, semantic and structural heterogeneity between multiple information sources. In this paper, our main objective is to resolve semantic conflicts. The data, ontology and information integration communities face similar types of problems, and we leverage techniques developed by these communities. Our approach is a hybrid one, in that we use multiple matching criteria and multiple matchers. We employ several element-level, structure-level and ontology-based techniques during the integration process. A domain ontology serves as a global ontology and allows us to resolve semantic heterogeneity. Our matching process handles different mapping cardinalities (1:1, 1:n, n:1, m:n). The mappings derived are used to generate an integrated meta-search query interface, to support query processing in the meta-search engine, and to resolve semantic conflicts arising during result extraction from the source search engines. Experiments conducted in the job search domain show that the cumulative use of element-level, structure-level and ontology-based techniques increases the correctness of matching during the automatic integration of source search interfaces.

Keywords: Schema matching, schema integration, data integration, meta-search engine, schema and data integration.

1 Introduction

The Web has drastically changed the online availability of data and the amount of electronically exchanged information. However, the volume and heterogeneity of the information that is available online via Websites or databases makes it difficult for a user to visit each and every Website that is relevant to their information needs. Primary tools for searching the information on the Web are search engines, subject directories and social network search engines. Traditional search engines are based on keyword or phrase search, without taking into account the semantics of the word or

phrase, and hence may not provide the desired results to the user. Other traditional search tools suffer from low recall and precision. These tools do not provide comprehensive coverage of the Web [1][2][3]. To overcome these problems, meta-search engines aim to offer topic-specific search using multiple heterogeneous search engines. One of the major steps in accessing heterogeneous and distributed data via a meta-search engine is the task of schema and data matching and integration.

Much research has focused on developing techniques for schema matching and mapping as it is required in many areas e.g. heterogeneous database integration, e-commerce, data warehousing, semantic query processing, B2B applications, P2P databases, agent communication, Web Service integration. The schema matching process identifies correspondences between elements from different schemas. The schema mapping process defines these correspondences i.e. provides view definitions that link the two schemas [4]. Schema matching and mapping may generally be undertaken manually, semi-automatically or automatically. However, the volumes and heterogeneity of Web data, in particular, mandate the development of automatic schema matching and mapping techniques.

Different types of heterogeneity may arise when schemas are matched e.g. syntactic, semantic, structural. Different types of semantic conflicts may arise e.g. confounding, scaling, naming [5]; and there may be different mapping cardinalities between elements from different schemas, 1:1, 1:n, n:1 or n:m [6]. One of the important challenges in automatic schema matching is to resolve such semantic heterogeneities.

Example 1: In the job search domain, one search engine schema may use "career type(s)" (see Fig. 1) and another may use "categories" (see Fig. 2) to represent the same category of information. This is an example of a 1:1 mapping. Other search engines may use "type of job", "job category", "employment type" etc to represent the same category of information.

Example 2: One schema may use "salary" and another a combination of "minimum salary" and "maximum salary" to represent salary. This is an example of a 1:n mapping.

The ontology integration, data integration and information integration research communities are addressing similar types of problems in matching and integrating heterogeneous schemas and ontologies. Our research starts from the premise that the techniques developed by these communities are of relevance to schema/data matching in meta-search engines, and that a combination of approaches is required c.f. [7][8]. In our setting, we are concerned with automatic schema and data integration of information arising from different Web portals. Consider, for example, the three search interfaces shown in Fig. 1, 2 and 3. There is semantic heterogeneity both at the schema level and the data level. At the schema level, we see that three different concepts, "career type", "categories" and "select a category", are used to represent the same category of information. At the data level, we see that canjobs.com uses "Administrative support", careerbuilder.com uses "Admin – Clerical", and jobs.net uses "Admin & Clerical" to represent the same item of information. For business-related jobs, careerbuilder.com uses "Business Development" and "Business Opportunity" while jobs.net uses "Business Development" and "General Business".

Our focus in this paper is on automatic schema matching and integration techniques aiming to resolve semantic conflicts between different search engines, in order to support the construction of meta-search engines. Semantic heterogeneity both at the schema and the data level needs to be resolved. We need to discover meanings encoded in schemas from multiple search engines and create an integrated schema. The matching process must handle mappings of different cardinalities (1:1, 1:n, n:1, m:n). Moreover, appropriate integrated terms both at the schema and the data level must be selected for the meta-search interface. The mappings between the source and integrated schemas need to be used by the meta-search engine for query processing. Semantic conflicts arising during result extraction from multiple search engines need to be resolved too. We have developed a configurable approach to meta-search engine construction that aims to meet these requirements.

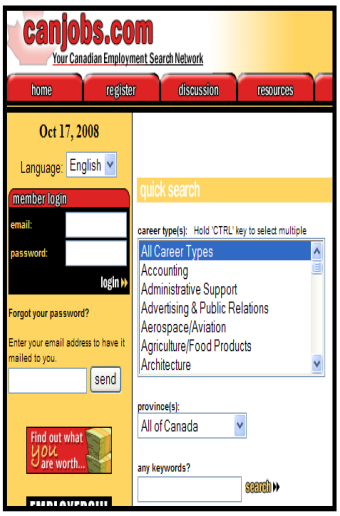

Fig. 1. canjobs.com

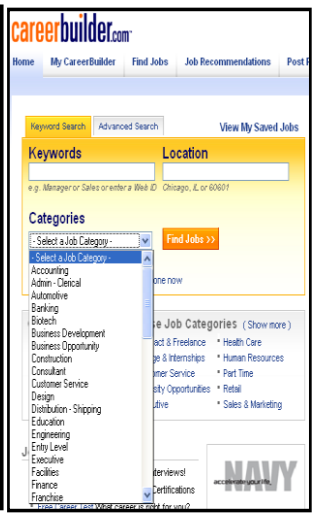

Fig. 2. careerbuilder.com

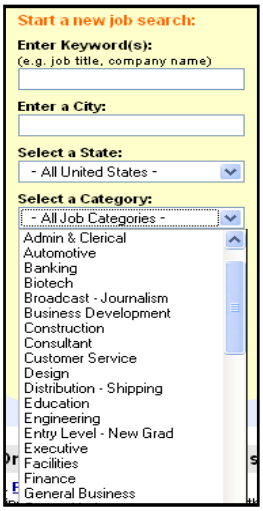

Fig. 3. jobs.net

The rest of the paper is organized as follows. Section 2 briefly reviews related work in schema and ontology matching/mapping, meta-search engines, Web data integration systems, and wrapper generation. Section 3 presents our overall architecture for meta-search engine construction. Section 4 describes our query interface generation process along with the techniques used in the integration process for the meta-search engine. Section 5 describes a case study in the area of job search and some experimental results from this case study. Finally, Section 6 summarizes our contributions, and gives directions for future work.

2 Related Work

There has been much work on schema/data matching and mapping. [4] and [9] present reviews and classifications of the main schema matching approaches. Cupid

[7] uses multiple matching approaches, and also a thesaurus to find acronyms, short forms and synonyms of words. COMA++ [10] supports multiple matchers and also uses a taxonomy that acts as an intermediate ontology for schema or ontology matching. [11] merges different ontologies into a single global schema, using similarity relations. [6] uses a combination of structural similarity between two schemas and a also a domain-specific ontology to discover mappings. String distances can also be utilized for schema matching, e.g. for matching entity names [12]. Techniques developed for schema matching can also be employed for ontology merging: [13] distinguishes between using a single ontology describing all the sources, and using multiple ontologies – one for each data source – which are then merged to form a single shared ontology. There has also been much research into ontology matching and mapping e.g. [5][8][14].

In the area of meta-search, the meta-search engine presented in [15] consists of the "WISE: iExtractor" for interface extraction and "WISE-Integrator" for automatic integration of schema, attribute values, format and layout. [15] uses traditional dictionaries along with multiple matching techniques, to find semantic similarity between schema elements and values. The Lixto Suite [16] provides a platform to access Web sites and extract information from them. Lixto meta-search uses the Lixto visual wrapper for extraction of relevant information and then integrates the different Web sites into a meta-search application. [17] introduces a special-purpose meta-search named "Snorri" which extends the Lixto meta-search by eliminating limitations such as synchronous provision of results. Snorri also introduces caching, scalability and load balancing. The MetaQuerier project aims to explore and integrate databases that are not visible to traditional crawlers. Its semantic matching sub-system stores 1:1 or m:n matchings for the construction of a unified query interface and query translation purposes. MetaQuerier uses a statistical/probabilistic approach for schema matching [18].

In the information extraction area there has been much research in wrapper induction techniques. For example, [19] utilizes visual content features and the tag structure of HTML result pages for the automatic wrapper generation of any given search engine. [20] extends this work by introducing new techniques for the extraction of search result records from result pages. [21] describes the Lixto visual wrapper generator, which provides a visual, interactive user interface supporting semi-automatic wrapper generation. It extracts the relevant information from HTML documents and translates it into XML, which can be queried and further processed.

Compared to WISE and MetaQuerier, our approach to schema matching uses also a domain ontology to resolve semantic conflicts. Compared to Lixto, we aim to provide an automatic and simple construction process for information extraction. Compared to vertical search engines (such as http://www.kayak.com and http://www.skyscanner.net), which provide hard-wired solutions, we are aiming for a configurable and extensible approach.

In our previous papers [22] [23] we described a domain-specific scenario of job portal integration and we developed an appropriate ontology for this domain. Our work here generalizes that work for application in arbitrary domains. [24] discusses our work on design patterns for the construction of meta-search engines. Our work here focuses on the schema/data matching and integration process for query interface generation of meta-search engines. We utilize several existing techniques from the

database and ontology communities for schema/data matching, and we introduce some innovations e.g. a default value matcher and an ID matcher (see Section 4). We use similar techniques for result extraction from the result pages returned by the source search engines.

Finally, we distinguish our work from Web-scale architectures such as PAYGO [25], in that we are aiming to develop techniques to support the construction of domain-specific meta-search engines, rather than Web-scale search of the deep Web. Our aim is to combine the respective benefits of vertical search engines, meta-search engines and semantic search engines within a domain-specific context, in which there is a well-understood domain ontology.

3 Meta-search Architecture

In this paper, we are concerned with techniques to support two key aspects of meta-search engines: i) meta-search engine creation by meta-search engine providers, and ii) meta-search engine usage by information seekers. Fig. 4 and 5 show the components of our meta-search architecture that support these two processes. In Section 4 we will focus in more detail on one component, the *Query Interface Generator*. Here, we first describe in overview how the various components support processes i) and ii).

The meta-search engine creation process (see Fig. 4) is as follows. First, the meta-search provider submits its preferences via the *Preferences Collector*. Currently, preferences may be for which countries or geographical areas meta-search is required. In the longer term, we plan to extend this to capture also information about preferred units for data types (e.g. preferred currency and preferred periodicity for salaries e.g. yearly salary, monthly salary, weekly salary etc.). The *Search Engine Selector* is then activated, and search engines that meet the preferences of meta-search provider are selected from an already known set of URLs of candidate search engines. This is done by analyzing the URL, IP address and country of the search engine.

Next, the *Interface Extractor* derives attributes from those search engines' interfaces. The process of interface extraction has two phases: i) attribute extraction, and ii) attribute analysis. During attribute analysis, we identify the relationships between the extracted attributes: a set of attributes may be inferred as forming a group, or an attribute may be inferred to be 'part-of' another attribute. We undertake this identification by analyzing the HTML control elements in the search interface; the order of labels and control elements; and keywords or string patterns appearing in the attribute labels. Next, we similarly derive metadata about these attributes e.g. how many values can be associated with an attribute, default value, value type. Currently, boolean, string, integer, date, time and currency types are supported. The range of values that an attribute can take may be finite (e.g. selected from an enumerated list), infinite (entered as free text by the user), or comprise a range of lower and upper values.

The *XML Schema Generator* then creates an XML schema corresponding to each search interface. We assume that a domain ontology is available for the target domain of the meta-search engine (e.g. jobs, flights, hotels etc.). This ontology is used by the

Query Interface Generator to create mappings between the XML schemas and the ontology, and hence indirectly mappings between the different XML schemas via the ontology. The *Query Interface Generator* undertakes three main steps: i) schema matching and integration, ii) data integration, and iii) generation of a single query interface for the meta-search engine (we give more details of this in Section 4).

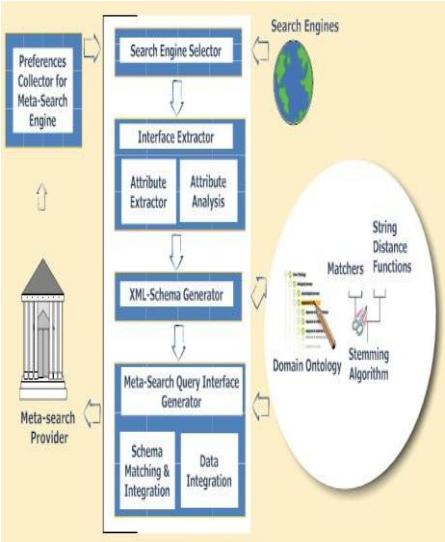

Fig. 4. Meta-search engine creation process.

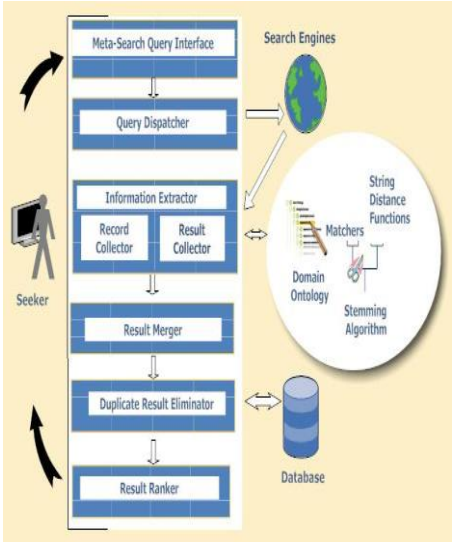

Fig. 5. Meta-search engine usage process.

The meta-search engine usage process (see Fig. 5) is as follows. An information seeker can access and use the meta-search query interface generated by the meta-search creation process. Queries submitted to the query interface are rewritten by the *Query Dispatcher* in order to target the individual source search engines, using the mapping rules. The Query dispatcher submits the rewritten queries to the individual search engines (see Section 4 for more details of this). The result pages from various search engines are passed to the *Information Extractor (IE)* component. Automatic wrapper generation techniques are used for the implementation of this component. In particular, the IE consists of *Record Collector* and *Result Collector* sub-components. The *Record Collector* is responsible for identification of the record section from each result page i.e. a list or table containing results. It also identifies the required URL and title of each identified record. The *Result Collector* visits the identified URL and is responsible for extracting the record description, e.g. the job, flight or hotel description, and the record fields, e.g. job salary, hotel location, flight timings, from the result page. Since different search engines use different concepts and data structures for results in their result pages, the Result Collector utilizes again the domain ontology and a variety of matching techniques (see Section 4.3) in order to conform the different concepts and data structures of result descriptions and result attributes, and to convert them to a single common format for presentation to the information seeker. The conformed results are merged by the *Result Merger* component. Duplicate results are removed by *Duplicate Result Eliminator* and stored in a database for further use. Finally, the results are ranked by the *Result Ranker* according to the preferences of the information seeker and displayed to the seeker.

4 Meta-search Query Interface Generator

Once schemas from the source search engines have been generated by the *XML-Schema Generator,* the next step is schema/data matching and integration. Our key requirement is to provide automatic techniques for undertaking this, utilizing the cumulative techniques from the database and ontology communities. We use multiple matchers, some of which use multiple string distance algorithms. Stemming and removal of stop words are also used. We adopt a single-ontology approach and utilize the domain ontology to find matchings between attributes of different search engine interfaces; a synonym matcher is also used during this process. The mappings generated are stored in XML format for use by the query interface generator and query dispatcher components. After schema/data matching and integration, a query form for the meta-search engine is generated, using XForms [26].

Since our meta-search engine generation and creation processes use multiple matching criteria and multiple techniques/matchers, we term this a 'hybrid' approach. We use a combination of element-level techniques, structure-level techniques and ontology-based techniques (see [9] for a general review of the main techniques used in schema matching). The techniques that we use are described briefly in Section 4.1. Section 4.2 then discusses our schema and data integration algorithm, and 4.3 the use of our hybrid approach in the extraction of search results too.

4.1 Matching Techniques

The **element-level techniques** we use include a string-based matcher, language-based matcher, data values cardinality matcher, ID matcher, default value matcher and alignment reuse matcher. We use element-level techniques to find similarity between schema elements, and between data values.

Our string-based matcher uses a stemming algorithm and different string distance functions to find a similarity between strings. In particular, the porter stemming algorithm removes the prefix and suffix of a string, handles singular and plural of concepts, and then finds the similarity between strings. The following are examples resolved with the porter stemming algorithm [27].

Keywords→keyword, Provinces→province, Industry→industries, States→state, Posted Date→Post Date, Job Types→Job Type, Starting date→Start Date

We utilize three different string distance algorithms, Levenshtein distance, Cosine similarity and Euclidean distance, and an aggregate of their similarity scores is used. The following are examples of strings matched using these string distance functions:

business operations→business, intern→internship, engineering software→ software engineering, contractor→contract,

Our language-based matcher is based on natural language processing techniques, including tokenization and elimination. Tokenization involves the removal of punctuation, blank spaces, and adjustment of cases. Elimination involves the removal

of stop words (a list of stop words for the given domain needs to be provided to the system). Stop words includes articles, prepositions and conjunctions etc. The following are examples of strings transformed using tokenization and elimination:

"Enter a Keyword"→ keyword, "Career type(s)"→career types, "Select a State:"→ state, "Full-time"→ full time

Our data value cardinality matcher uses the cardinality of attributes to find a match. For example, suppose an attribute "Job Type" that contains 7 data values may match an attribute "Type of Hour" that contains 8 data values or an attribute "Job Category" that contains 44 data values. In this situation, the number of data values can be compared, from which it can be inferred that attribute "Job Type" is more similar to "Type of Hour".

If element name matching fails, then our ID matcher may help to find a match. Some examples of IDs from job search engines for the element "keyword" are

qskwd, keywords, jobsearch, keywordtitle, kwd

Suppose a search engine contains an element with name "Type of Skills" and ID="kwd". Suppose that the element name fails to match with any element in ontology. In this situation, the ID matcher will be utilized and it will compare "kwd" to elements of the ontology e.g. keyword, type of hour, job category etc. With the help of the string distance functions above, the ID matcher will find a similarity between "kwd" and "keyword".

Sometimes, search engine interfaces provide default values with attributes, so that if the user does not select any value, the default value is used. If a default value is available, our default value matcher can be helpful in increasing the matching results. For example, suppose there is ambiguity between the "Job Type" attribute of one schema and the "Type of Hour" or "Job Category" concepts of the domain ontology. The default value matcher can find that the default value "intern" of the "Job Type" attribute is matched to data value "internship" of the "Type of Hour" concept.

As already noted, our schema/data matching process is based on a domain ontology. This ontology is incrementally extended with synonyms and hypernyms of attributes from previously matched schemas. As soon as new matching is found, we store it in the domain ontology. When matching fragments of schemas, we employ an alignment reuse matcher to reuse these previously stored match results: if there already exists a matching for an attribute, then there is no need to attempt to match the attribute again.

Structure-level matchers [4] consider a combination of elements that appear near to each other within a schema in order to identify a match. Two elements are considered similar if the elements in their local vicinity are similar. In particular, bottom-up structure-level matchers compare all elements in their sub-trees before two elements are compared i.e. data values are considered first. Top-down matchers compare first parent elements and, if they show some similarity, their children are then compared. This is a cheaper approach, and we utilize this, although it may miss some matches that a bottom-up matcher would detect.

For example, suppose there is a choice in matching an attribute "Job Type" of a schema with either attribute "Type of Hour" or attribute "Job Category" of the ontology. Our top-down matcher will match the children of "Job Type" with the

children of “Type of Hour” (e.g. full time, part time, contract etc.) and with children of “Job Category” (e.g. computer science, business, engineering etc.). It will select whichever of these two attributes has the set of children having the closest combined match to the children of “Job Type”.

Finally, with respect to **ontology-based techniques,** we use a single ontology approach, and the domain ontology acts as a global ontology. After completion of the schema integration process, the meta-search query interface generated contains concepts from this domain ontology. We recall that an XML schema is generated for every search engine to be included in the meta-search. A synonym matcher is used to find similarities between such a source schema S_1 and the global ontology O_G, using synonyms associated with concepts in O_G. For example, in the job domain, synonyms for “job category” might be “industry”, “occupation”, “career type”, “function”. We note that a domain-specific ontology is likely to perform much better than traditional dictionaries or thesauri in finding semantic similarity between source terms.

4.2 Schema/Data Integration Algorithm

The XML schemas of the source search engines are given as input to our schema and data integration algorithm, and an integrated XML schema for the meta-search engine is generated as an output. All the discovered mappings are stored within this schema. Our algorithm works as follows:

First, the set of attributes (i.e. schema elements) from every source XML schema is extracted, and the *schema matching and integration process* starts. For every attribute, the algorithm attempts to find an equivalent attribute in O_G by applying multiple matchers in the following order: a) directly within O_G, possibly using also the synonym matcher, b) using the string-based matcher or language-based matcher, c) using the data-value cardinality matcher or top-down matcher, d) using the ID or default value matcher. If an equivalent attribute is detected at any step, the matching process stops and the discovered mapping is stored in the integrated XML schema. Our rationale for applying the various matching techniques in the sequence a)-d) above is as follows. The domain ontology is examined first, together with the use of a synonym matcher, as the ontology will be a source of high-quality, broad-coverage information about the domain. If a match fails to be found for an attribute, we then use the techniques in b) because they are cheaper than the techniques in c) (in terms of execution time), as observed from our experiments with several search engine interfaces. Finally, regarding d), we apply the ID and default value matchers last because they are of low precision: in many cases the ID is not meaningful (Web developers may use arbitrary IDs for HTML control elements) or a default value is not specified.

When the matching process for all the attributes is completed, the *data matching and integration process* starts. The children of each XML schema attribute are matched against O_G. Children attributes are only matched against attributes in O_G if there has already been found to exist some similarity between the parent attribute in the XML schema and the parent attribute in O_G. The same matchers as in a) and b) above are applied in sequence and the mappings discovered are stored in the integrated XML schema.

Our algorithm can generate 1:1 mappings at the schema level, and 1:1, 1:n, n:1 and m:n mappings at the data level. The integrated XML schema generated, incorporating the discovered mappings, is then used for generating the integrated meta-search interface and for subsequent processing of queries. An information seeker can pose a query from the integrated meta-search interface. This query is rewritten by the meta-search engine to target every source search engine that was incorporated in the generation of the meta-search engine.

4.3 Result Collection

Since different search engines use different concepts and data structures for results in their result pages, our *Result Collector* component, too, utilizes the domain ontology, multiple matchers, a stemming algorithm and multiple string distance functions in order to conform the different concepts and data structures of result descriptions and result attributes arising from different search engines (again by generating appropriate mappings), and to convert these into a single common format for presentation to the information seeking user. In this way, our hybrid approach is used in the extraction of search results too.

5 Case Study and Evaluation

Examples 5.1 and 5.2 are related to two search engines in the jobs domain (http://www.jobs.net. and http://www.mymatrixjobs.com) and this domain is used to show how our schema/data matching and integration process operates for meta-search query interface generation and query processing. The URLs of possible job search engines are given as input to our system, and the GUI of the job meta-search engine is automatically generated. All schema and data mappings are detected with the use of the techniques described in the previous section.

We have developed an HR domain ontology to support our job meta-search case study, in order to evaluate our approach. We collected job attributes from different job search engines and their corresponding attributes from HR-XML (www.hr-xml.org) into an occupations sub-ontology. We integrated the computing and business-related occupations from widely used standards – Standard Occupation Classification (SOC) and International Co-operation Europe Ltd, into one format and added this job category information to our ontology. Our domain ontology also contains data values for attributes in the job domain [24]. For example, the data values for the attribute "Type_of_Hour" are {Contract, Full_Time, Internship, Part_Time, Permanent, Student, Temporary, Voluntary}. The attribute "Occupation" has multiple sub-classes, and the "Computer_Science" sub-class has data values {Software_Engineer, Administrtor, Multimedia_Designer, System_Specilist etc.}.

5.1 Query Interface Generation for the Job Meta-search Engine

Each source job search engine has a different interface and job search criteria. For simplicity, we describe here just a fragment of our case study, and consider just two

simple schemas from the full set of job search engines used in the case study. We also consider only a subset of the attributes and data values from these schemas.

S_1 is the schema for search engine http://www.jobs.net, and contains attributes "Enter Keywords(s)", "Enter a City", "Select a State", "Select a Category", "Employment Type". The "Select a Category attribute" has data values {Business Development, General Business, Information Technology, Science, Telecommunications, Design}. The "Employment Type" attribute has data values {Full-Time, Part-Time, Contractor, Intern}.

S_2 is the schema for search engine https://www.mymatrixjobs.com and contains attributes "Keywords", "City or Zip", "States", and "Job Type" which has data values {Contract or Permanent, Contract, Permanent}.

Our job domain ontology O_G contains a class "Job attributes" with sub-classes "Competency", "City", "State", "Job Category", and "Type of Hour". "Job Category" has multiple synonyms, and data values {Computer science, Business, Engineering, Telecommunication, Web Design etc.}, along with synonyms for each once of these. "Type of Hour" has synonyms "Employment Type" and "Job Type", and data values {Contract, Full-time, Internship, Part-time, Permanent, Student, Temporary, Voluntary}, along with their individual synonyms. In O_G there is a class-subclass relationship between each job attribute and its set of data values.

When the schema/data matching process starts, first S_1 is matched with O_G. By applying a combination of matchers as described in the previous section, the following schema-level mappings are discovered:

S_1.*Enter Keyword(s)* → O_G.*Competency*
S_1.*Enter a City* → O_G.*City*
S_1.*Enter a State* → O_G.*State*
S_1.*Select a Category* → O_G.*Job Category*
S_1.*Employment Type* → O_G.*Type of Hour*

As we use a top-down structural matching approach, when schema-level concepts are successfully matched, then data-level matching starts. At the data level, the following mappings are discovered:

S_1 .*Business Development* → O_G.*Business*
S_1 .*General Business* → O_G.*Business*
S_1.*Information Technology* → O_G.*Computer Science*
S_1.*Science* → O_G.*Computer Science*
S_1.*Telecommunications* → O_G.*Telecommunication*
S_1.*Design* → O_G.*Web Design*
S_1.*Full-Time* → O_G.*Full-time*
S_1.*Part-Time* → O_G.*Part-Time*
S_1.*Contractor* → O_G.*Contract*
S_1.*Intern* → O_G.*Internship*

Next, S_2 is matched with O_G and the following matchings are discovered at the schema level:

S_2.*Keywords* → O_G.*Competency*
S_2.*City or Zip* → O_G.*City*
S_2.*States* → O_G.*State*

$S_2.Job\ Type \rightarrow O_G.Type\ of\ Hour$

and at the data level:

$S_2.Contract\ or\ Permanent \rightarrow O_G.Contract$
$S_2.Contract\ or\ Permanent \rightarrow O_G.Permanent$
$S_2.Contract \rightarrow O_G.Contract$
$S_2.Permanent \rightarrow O_G.Permanent$

From these mappings, schema attributes and data values, an integrated job search schema, $S_{MSE,}$ for the meta-search query interface is then generated. This consists of attributes *Competency, City, State, Job Category* and *Type of Hour.* Attribute *Job Category* has data values {*Business, Computer science, Telecommunication, Web Design*} and *Type of Hour* has data values {*Full-time, Part-Time, Contract, Internship, Permanent*}. Finally, a GUI is generated from S_{MSE} for the job meta-search engine.

5.2 Query Processing by the Job Meta-search Engine

A job seeker can pose a query from the integrated meta-search interface GUI, in terms of the integrated schema S_{MSE}. This query is rewritten by the meta-search engine, to target every search engine involved in the meta-search engine generation process. For example, suppose a job seeker poses a query Q_{MSE} requesting all Contract jobs with keyword "java" in the computer science field:

Q_{MSE} : *Jobs (Competency=Java, Job Category=Computer science, Type of Hour= Contract)*

Q_{MSE} query is transformed using the earlier discovered mappings both at the schema and the data level, to target each individual search engine. So we have queries Q_{11} and Q_{12} below targeted at http://www.jobs.net and queries Q_{21} and Q_{22} targeted at https://www.mymatrixjobs.com:

Q_{11} : *Jobs (Enter Keyword(s)= java, Select a Category=Information technology, Employment Type = Contractor)*
Q_{12} : *Jobs (Enter Keyword(s)= java, Select a Category=Science, Employment Type=Contractor*
Q_{21} : *Jobs (Keywords=Java, Job Type = Contract)*
Q_{22}: *Jobs (Keywords=Java, Job Type = Contract or Permanent)*

$Q_{11,}$ $Q_{12,}$ $Q_{21,}$ Q_{22} are then submitted to the two search engines, the results are extracted by the Information Extractor component of our meta-search engine architecture, and are then merged, ranked (according to the preferences of the information seeker) and returned to the user, as described in Sections 3 and 4.3 earlier.

5.3 Evaluation

We have evaluated our techniques in the job domain, using the following job search engines:

- http://www.careerbuilder.com
- http://www.learn4good.com/jobs/
- https://www.mymatrixjobs.com/candidate/Login.action
- http://www.jobs.net/
- http://jobsearch.monster.com/
- http://www.canjobs.com/index.cfm
- http://www.brightspyre.com/opening/index.php?
- http://www.top-consultant.com/UK/career/appointments.asp

Fig. 6 shows the contributions of element-level, structure-level and ontology-based techniques in the matching process for each job search engine. We see that for the careerbuilder search engine, for example, our hybrid approach identifies a total of 6 job-related attributes, with element-level techniques identifying 3 attributes, structure-

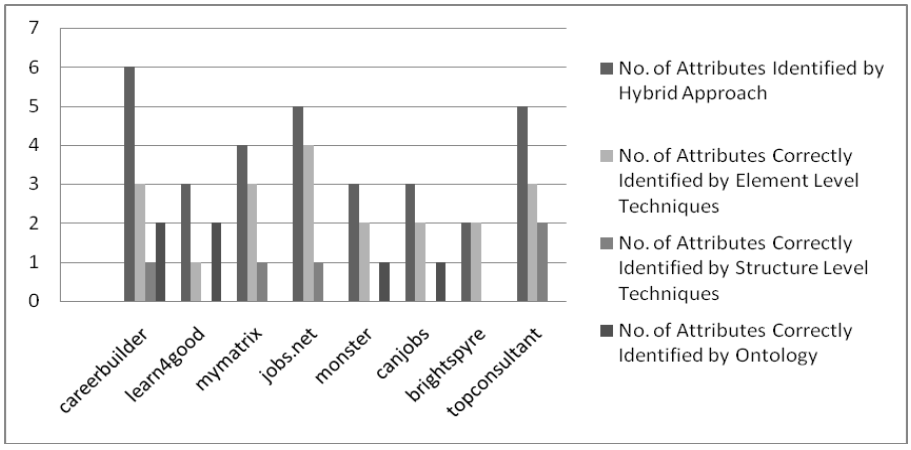

Fig. 6. Evaluation of hybrid approach in schema matching.

level techniques 1 attribute and ontology-based techniques 2 attributes. The results for the other search engines are shown in a similar way, and we can see the benefits of adopting our hybrid approach.

Combining the above results, we calculate an overall contribution to the identification of job-related attributes within all the search engine interfaces of 60.60% for element-level techniques, 15.15% for structure-level techniques, and 18.18% for ontology-based techniques. When we combine all the techniques, our hybrid approach for meta-search engine generation achieves overall correctness of 60.60% + 15.15% + 18.18% = 93.93%, where we define correctness as:

$$\frac{\textit{number of attributes correctly identified over the set of search engine interfaces}}{\textit{total number of attributes in the set of search engine interfaces}}$$

The precision achieved in this experiment was 100% (all the attributes identified were correct) and the recall was 93.93%. This experiment took 1 minute and 44 seconds for the job meta-search query interface generation process, for the eight job search engines above, on a machine with 1.60 GHz processor, 512 RAM and running Microsoft Windows XP. For this particular experiment, if the ordering of the groups a)-d) described in Section 4.2 is altered, then the same overall set of matchings would be discovered, but this may not be the case in general i.e. different orderings of application of a)-d) may yield different sets of matchings.

6 Conclusions and Future Work

We have developed a prototype system that uses a hybrid approach for schema and data matching and integration in meta-search engines. Although evaluated so far in the context of the job domain, our techniques are general and can be used in the development of meta-search engines in any domain provided there is an appropriate ontology describing that domain. Our work can also be viewed as providing a generic approach for semi-automatically creating a "vertical" search engine for a given domain, which combines multiple domain-specific search engines.

In this paper, our main focus has been on the schema/data matching and integration aspects of meta-search engine generation and usage. We have introduced a hybrid approach that leverages the cumulative work of the large research community in order to resolve schema/data matching and mapping problems between heterogeneous search interfaces. Our experiments in the job domain show that the combined use of element-level, structure-level and ontology-based techniques increases the correctness of matching during the automatic integration of the source search engine interfaces.

Our techniques and results provide a contribution in the area of generating more comprehensive and more concise meta-search query interfaces, more accurate meta-search query processing, and more comprehensive and concise presentation of search results to users. Using a domain ontology is advantageous if the data values of the ontology attributes are also modelled within the ontology. Data level matching can be undertaken with greater precision and, in our context of search engine interface integration, we have the added advantage of a typically a limited set of data values for each attribute, as compared with the data values typically arising in a database integration setting, for example. In our setting, we will generally have fewer values to compare, and the matching will take less time and computational effort.

For future work, we will report on the query processing performance of meta-search engines that are generated using our techniques. We plan to investigate introducing further matchers and techniques, and to capture and use also preferences about units for numeric data types. Also, rather than requiring the URLs of candidate source search engines to be made known to our system, in the future our plan is to identify and choose search engines on the fly from the Web.

References

1. Aharoni, Y., Frank, A.J., Shoham, S.: Finding Information on the Free World Wide Web: A Specialty Meta-search Engine for the Academic Community. In: First Monday, vol. 10 no. 12 (2005)
2. Ding, C.H., Buyya, B.: Guided Google: A Meta Search Engine and its Implementation using the Google Distributed WebServices. Technical Report, GRIDS Laboratory, The University of Melbourne, Australia (2003)
3. Rao, B.S., Rao, S.V., Sajith, G.: A User-Profile Assisted Meta Search Engine. In: TENCON, Conference on Convergent Technologies for Asia-Pacific Region, vol. 2, pp. 713-717 (2003)
4. Rahm, E., Bernstein, P.A.: A Survey of approaches to Automatic Schema Matching. VLDB Journal. vol. 10, no. 4, pp. 334-350, ISSN: 1066-8888 (2001)
5. Wache, H., Vögele, T., Visser, U., Stuckenschmidt, H., Schuster, G., Neumaan, H., Hübner, S.: Ontology-Based Integration of Information - A Survey of Existing Approaches. In: IJCAI-01 Workshop, pp. 108-117 (2001)
6. Embley, W.D., Xu, L., Ding, Y.: Automatic Direct and Indirect Schema Mapping: Experiences and Lessons Learned. In: ACM SIGMOD Record, pp. 14-19 (2004)
7. Madhavan, J., Bernstein, P.A., Rahm, E.: Generic Schema Matching with Cupid. In: Proceedings of the 27th VLDB Conference, pp. 49-58. Roma, Italy (2001)
8. Noy, N.F.: 2004. Semantic Integration: A Survey of Ontology-Based Approaches. Special section on Semantic Integration. In: Column ACM SIGMOD Record, vol. 33, issue 4: 0163-5808, pp. 65-70 (2001)
9. Shvaiko, P., Euzenat, J.: A Survey of Schema-based Matching Approaches. Technical Report, DIT-04-087, Informatica e Telecomunicazioni, University of Trento (2005)
10. Aumueller, D., Do, H.H., Massmann, S., Rahm, E.: Schema and Ontology Matching with COMA++. In: Proceedings of the 2005 ACM SIGMOD International Conference on Management Data, pp. 906-908. Maryland, USA (2005)
11. Hakimpour, F., Geppert, A.: Global Schema Generation Using Formal Ontologies. In: Proceedings of the 21st International Conference on Conceptual Modeling, pp. 307-321 (2002)
12. Cohen, W.W., Ravikumar, P., Fienberg, S.E.: A Comparison of String Distance Metrics for Name-Matching Tasks. In: Proceedings of IJCAI-03 workshop on information integration on the Web, pp. 73–78. Acapulco (2003)
13. Linková, Z.: Schema Matching In the Semantic Web Environment. In: PhD Conference, pp. 36-42. Matfyzpress (2007)
14. Linková, Z.: Ontology Based Schema Integration. In: Proceedings of SOFSEM, pp. 71-80. Prague (2007)
15. He, H., Meng, W., Yu, C., Wu, Z.: Automatic Integration of Web Search Interfaces with WISE-Integrator. VLDB Journal, vol. 13, no. 3, pp. 256-273 (2004)
16. Lixto, The Web Intelligence Company, http://www.lixto.com
17. Ondrej, J.: A Scalable Special-Purpose Meta-Search Engine. PhD Thesis, Institute for Information Systems, Faculty for Informatics, Vienna University of Technology, Vienna, Austria (2006)
18. Chang, K.C., He. B., Zhang, Z.: Toward Large Scale Integration: Building a MetaQuerier over Databases on the Web. In: Proceedings of the Second Conference on Innovative Data Systems Research, pp. 44-55, Asilomar, California (2005)
19. Zhao, H., Meng, W., Wu, Z., Raghavan, V., Yu, C.: Fully Automatic Wrapper Generation for Search Engines. In: International World Wide Web Conference, (2005)
20. Zhao, H., Meng, W., Yu, C.: Automatic Extraction of Dynamic Record Sections From Search Engine Result Pages, In: Proceedings of 32 International Conference on VLDB, pp. 989-1000. Seoul, Korea (2006)

21. Baumgartner, R., Flesca, S., Gottlob, G.: Visual Web Information Extraction with Lixto. In: Proceedings of the 27th VLDB Conference, Rome, Italy (2001)
22. Dorn, J., Naz, T.: Integration of Job portals by Meta-search. In: Proceedings of 3rd International Conference on Interoperability for Enterprise Software and Applications, pp. 401-412. Funchal, Portugal (2007)
23. Dorn, J., Naz, T., Pichlmair, M.: Ontology Development for Human Resource Management. In: 4th International Conference on Knowledge Management, pp. 109-120. Vienna, Austria (2007)
24. Dorn, J., Naz, T.: Structuring Meta-search Research by Design Patterns. In: Proceedings of International Computer Science and Technology Conference, San Diego, California, USA (2008)
25. Madhavan, J., Jeffery, R.S., Cohen, S., Dong, X.L., Ko, D., Yu, C., Halevy, A.: Web-scale Data Integration: You can only afford to Pay As You Go. In: Third Biennial Conference on Innovative Data Systems Research, pp. 342-350. CA, USA (2007)
26. Rainer, A., Dorn, J., Hrastnik, P.: Strategies for Virtual Enterprises using XForms and the Semantic Web. In: Proceedings of International Workshop on Semantic Web Technologies in Electronic Business, pp. 166-172. Berlin (2004)
27. Porter, M.: The Porter Stemming Algorithm, http://tartarus.org/~martin/PorterStemmer (2006)