Index-based Keyword Search in Mediator Systems

Ingolf Geist

Department of Technical and Business Information Systems,
School of Computer Science,
University of Magdeburg, Germany
geist@iti.cs.uni-magdeburg.de

Abstract. Many users and applications require the integration of semi-structured data from autonomous, heterogeneous Web sources. Over the last years mediator systems have emerged that use domain knowledge to overcome the problem of structural heterogeneity. However, many users of these systems do not have a thorough knowledge of the complex global schemas and of the comprehensive query languages. Consequently, easy-to-use query interfaces like keyword search and browsing have to be supported. The aim of the proposed PhD project is the index-based realization of keyword searches in concept-based mediator systems. In order to avoid unnecessary source queries an index structure is maintained on the global level and used during query planning and processing.

1 Research Problem

Nowadays, many search requests require the usage of information from different, distributed, and autonomous (Web) sources. Applications can be found in scientific as well as e-commerce environments. Prominent examples are the integration of biological data sources, digital libraries, but also the search for stolen cultural assets is distributed over many sources[1], which drives this work. Many of these sources provide structured query interfaces, but cannot be indexed by Web search engines. Furthermore, most of them are not cooperative and provide semantically similar data in different structural representation. Consequently, there is still a need for systems, that integrate these semi-structured data sources into a global, integrated, not materialized view.

In the last years mediator systems have been proposed that use domain knowledge as an integration anchor to overcome the problems of structural heterogeneity (e.g [20]). These systems allow querying the sources with complex query languages and use complex integration models. During using the concept-based mediator prototype YACOB [23], which integrates several sources about stolen cultural assets, the experiences showed, normal users have difficulties to handle the concept schema, even if a graphical query interface based on browsing the global concept schema is provided. In fact, a simple *keyword search* was required, that comprises the meta data, here called concept level, as well as the data, called instance level. This requirement led to the

[1] For instance, data about stolen cultural assets during World War II is distributed over at least 15 sites (`http://www.lostart.de/links`).

problem, how to map efficiently the keywords to structured source queries (in our case XPath). The proposed idea of this work is the usage of a global keyword index, which maps keywords to global query expressions that are decomposed by the mediator system to source queries, subsequently. Summarizing, the tasks of the PhD project are the identification of an effective keyword index structure, to provide an index-based query planning and processing, and to develop techniques to maintain the index structure in the context of autonomous, uncooperative Web sources.

2 State of the art

The mediator architecture was introduced by Wiederhold [26] and has been utilized for integrating semi-structured data sources over the last years. Most prominent examples are TSIMMIS [13] and Information Manifold [19]. The MIX mediator system [6] represents a successor of TSIMMIS, which integrates XML data sources. All mentioned mediator systems integrate the local sources on structural level.

Integration systems using domain knowledge are e.g. KIND [20], SIMS [4], Context mediator [14] and the XML mediator STYX [3]. The systems uses different models to represent domain knowledge, e.g. subset of F-Logic, LOOM or ontologies. The integration approaches range from modeling local and global data in semantic concepts to mapping local XML fragments to global concepts. All systems support only simple "`like`"-queries on attribute level. Another direction in querying and using semantic data is the *Semantic Web*. Several languages are provided in this context (an overview is given in [21]).

Abiteboul requests for semi-structured query languages IR-style keyword query operators comprising schema data as well as instance data [1]. Several approaches follow this requirement [11, 25]. Furthermore, relevant to our research are keyword queries over structured databases, like relational systems [2, 8, 16, 22]. However, all systems deal only with centralized databases systems.

Data has to be obtained from the sources in order to build and maintain indexes on the global level. The standard crawling architecture [5] has to be refined for reducing costs and supporting sources, that cannot be crawled. Index update costs can be reduced by concentrating only on interesting parts of index and data. That leads to focused crawling [10, 24]. Some sources do not allow the complete crawling, but the provide limited query interfaces for accessing the data. They build the so called "Hidden Web" or "Deep Web" [7]. In order to index these sources special protocols were developed (e.g. [15]) or techniques like query-based sampling [9, 17] are used.

3 Proposed approach

The PhD project is based on the YACOB mediator [23], a concept-based integration system. The system uses a two-level integration model: the *concept model* describes the domain knowledge using concept hierarchies and their associated properties. Special data values are expressed as category hierarchies.

Formally, the model is summarized as follows. A set of classes is defined as $\mathcal{T} = \mathit{URI} \times \mathit{Name}$ with *URI* the set of uniform resource identifier and *Name* the set of valid

names. The set of classes is distinguished into two disjoint subsets: concepts ($\mathcal{C} \subset \mathcal{T}$) describe kinds of instances, and categories ($\mathcal{V} \subset \mathcal{T}$ and $\mathcal{V} \cap \mathcal{C} = \emptyset$) describe sets of data values, so called synonyms.

Properties are assigned to categories. The set of properties is defined as $\mathcal{P} = \mathit{Name} \times \mathcal{C} \times (\mathcal{C} \cup \mathcal{V} \cup \{\mathcal{L}\})$ with one property consisting of a name, a concept it is assigned to, and either a concept, category, or the set of literals ($\mathcal{L}$) as instance domain. Moreover, concepts and categories are organized in disjoint specialization hierarchies. The *subClassOf* relationship is defined as $\textbf{is\_a} \subseteq \mathcal{T} \times \mathcal{T}$ with: if $t_2 \,\textbf{is\_a}\, t_1 : t_1 \in \mathcal{C} \wedge t_2 \in \mathcal{C} \vee t_1 \in \mathcal{V} \wedge t_2 \in \mathcal{V}$. Furthermore, properties are "inherited" by subconcepts, i.e. if $c_2 \,\textbf{is\_a}\, c_1 \wedge c_1, c_2 \in \mathcal{C} : \forall(p, c_1, x) \in \mathcal{P} : \exists(p, c_2, x) \in \mathcal{P}$). A concept schema $CS = (\mathbf{C}, \mathbf{P}, \mathbf{V})$ consists of a set of concepts $\mathbf{C}$, a set of properties $\mathbf{P}$, and a set of categories $\mathbf{V}$.

As the local sources export their data in XML, the *instance model* follows the semi-structured model OEM [13]. The extension of a concept c is denoted as $\textbf{ext}(c) = \{o = (\mathit{id}, \mathit{elem}, \mathit{val}) \mid c.\mathit{name} = \mathit{elem} \wedge \forall(p, c, x) \in \mathcal{P} : \exists i \in \mathit{val} : i.\mathit{elem} = p\}$. The local sources are integrated by specifying mappings, which describe how a local source supports a global concept. The mappings realize the GLaV approach [12] and are expressed in RDF, too. In detail, concept mappings, property, and category mappings define how one source supports the concept schema. Join mappings define intra-source relationships as joins over the global concept schema. Besides conventional join operations, similarity join operations can be specified, whose index support is discussed in the further work (Sec. 3.2).

The query language – CQuery – is concept-based and uses the known FLWR notation. CQuery comprises (*i*) selection of concepts using selection predicates, path expressions, and set operations between concept sets (**FOR** clause), *(ii)* obtaining and filtering instances (**LET** and **WHERE** clauses), and (*iii*) combining and projecting the results (**RETURN** clause). The following example returns information of all instances that are associated to the concept `painting` (or to one of the subconcepts) and contain "`van Gogh`" in the property `artist`.

```
FOR $c IN concept[name='painting']/*
LET $e := extension($c)
WHERE $e/artist = 'van Gogh'
RETURN
  <painting>$e</painting>
```

The complete description of the query planning/processing is not possible here because of given space limitation The interested reader is referred to [23]. The processing starts with translating the query into an algebraic expression. The necessary concepts are determined, and subsequently for each concept the instance expression (**WHERE** and **RETURN** clause) is evaluated and the results are combined. Using some heuristics the number of concepts is limited, and by applying the different mappings the global query is decomposed into source XPath queries. Finally, the results are combined using the outer union operation ($\uplus$).

3.1 Keyword search

A keyword query consists of a set of keywords KW, and returns a set of global, integrated instances, that contain each of the keywords at least once in an associated meta data object (concept name, property name) or in a property value (category name or literal). Formally, the keyword search can be defined as: Given a set of keywords $KW = \{kw_1, \ldots, kw_n\}$ and a concept schema $CS = \{\mathbf{C}, \mathbf{P}, \mathbf{V}\}$, the result set O_{KW} of global instances is defined as:

$$
\begin{aligned}
O_{KW} = \{o = (id, elem, x) | \exists c \in \mathbf{C} : o \in \mathbf{ext}(c) \wedge \\
\bigwedge\nolimits_{i=1}^{n} \big(&\mathit{contains}(c.\mathit{name}, kw_i) \vee \exists c' \in \Phi^{+}_{\mathbf{is\_a}}(c) : \mathit{contains}(c'.\mathit{name}, kw_i) \vee \\
&\bigvee\nolimits_{(p,c,v) \in \mathbf{P}} \mathit{contains}(p, kw_i) \vee \\
&\exists v \in \mathbf{V} : v.\mathit{name} = x \wedge \mathit{contains}(v, kw_i) \vee \exists v' \in \Phi^{+}_{\mathbf{is\_a}}(v) : \mathit{contains}(v', kw_i) \vee \\
&\mathit{ocontains}(o, kw_i) \ \big) \big\} .
\end{aligned}
$$

The predicate *contains*(*string*, kw) evaluates to *true*, if the keyword kw is contained in $string$. The predicate *ocontains*(*object*, kw) evaluates to *true*, if the keyword kw is contained in a value of the object or in the value of one of the subobjects, i.e. the keyword is contained as a value of a property. The expression $\Phi^{+}_{\mathbf{is\_a}}$ returns all super-concepts or super-categories, i.e. it computes the transitive closure over the **is_a** relationship.

Consider the example keyword query {*painting*, *gogh*, *flowers*}. Possible matching objects are a painting object with artist "Gogh" and a title containing "flower", a book object with a title "Van Gogh's work" written by an author "Flowers", or another painting object with artist "Gogh" and the motif category "Flowers". The keyword search as defined here assumes no additional structural information are given, which rises the following problems addressed in the PhD project:

Extension of the query language. The query language as well as the corresponding algebra was extended by two constructs: the $\sim=$ operator and a refinement of the `extension` function. The $\sim=$ operator evaluates to *true*, if a keyword is contained in the compared object. It can be used for selecting concepts, properties and categories on the concept level as well as for filtering instance values. The `extension` function is used to return instances for a given list of concepts. The general keyword search is realized by passing keywords to the `extension` function, which returns instances that satisfy the above given keyword condition. The following example searches instances that contain the keywords "`painting flowers gogh`".

```
FOR $c IN concept/*      # select all concepts
LET $e := extension($c,'painting flowers gogh')
RETURN
  <painting>$e</painting>
```

This query is translated into the expression

$$
\biguplus_{qe \in \mathbf{getQExpr}(\mathbf{C}, \{\mathit{panting}, \mathit{flowers}, \mathit{gogh}\})} \pi_{proj}(qe)
$$

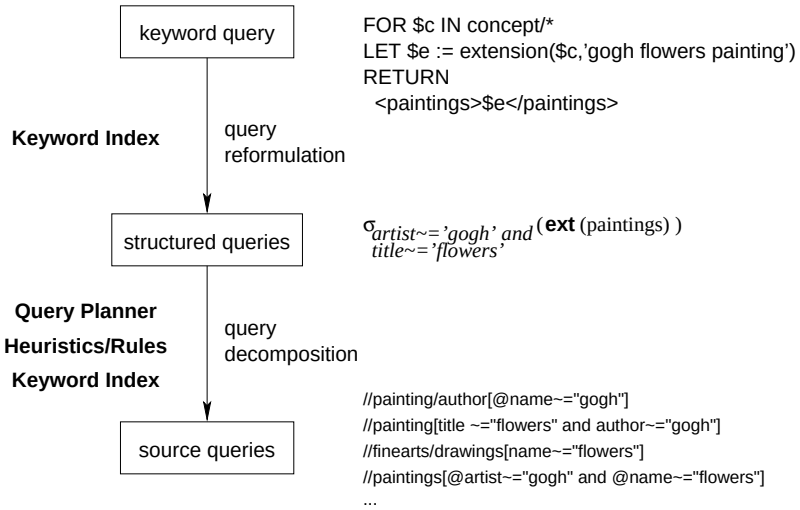


Fig. 1. Keyword query processing

with $\mathbf{C}$ all concepts existent in the schema and **getQExpr**(C, KW) a function returning a set of query expression of the form $\sigma_{cond}(\textbf{ext}(c))$, which are explained below. The operation $\biguplus$ is the outer union operation.

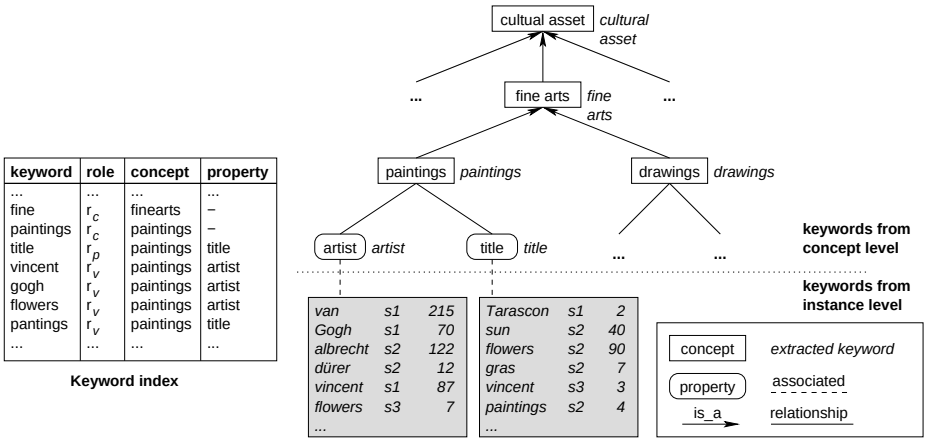
Index-based keyword query processing. A general keyword query without structural information is processed in two steps:

(i) reformulating the unstructured keyword query in structured global algebra expression, and
(ii) decomposing these expressions into source queries, which are sent to the sources.

Step (*i*) is executed in the function **getQExpr**. The returned set comprises expressions of the form $\sigma_{cond}(\textbf{ext}(c))$. A predicate is defined as

$$pred_{ij} = \begin{cases} c/\textit{property} = k & \text{if } k \in \mathbf{V} \wedge k.\textit{name} \sim= kw \\ c/\textit{property} \sim= kw & \text{otherwise} \end{cases}$$

and the conjunctions as $conj_i = pred_{i1} \wedge \ldots \wedge pred_{ik}$. Then $cond$ is defined as $cond = conj_1 \vee \ldots \vee conj_m$. Using, the provided mappings the result query expressions are decomposed into XPath expressions in step (*ii*), which are executed by the sources. Fig. 1 illustrates the evaluation showing an example query.
The naive implementation of step (*i*) results in a combinatorial explosion. Therefore, we need information about the existence of the keywords and their different contexts. A context of a keyword is identified by the associated concept, a property as well as a category or an instance value and the type of the keyword. The kind of the string, where the keyword was extracted from, specifies the type of the keyword. In detail, the type is either a concept keyword, category keyword, a property keyword, or an instance



keyword	role	concept	property
...	...	...	...
fine	r_c	finearts	–
paintings	r_c	paintings	–
title	r_p	paintings	title
vincent	r_v	paintings	artist
gogh	r_v	paintings	artist
flowers	r_v	paintings	artist
pantings	r_v	paintings	title
...	...	...	...

van	s1	215
Gogh	s1	70
albrecht	s2	122
dürer	s2	12
vincent	s1	87
flowers	s3	7
...		

Tarascon	s1	2
sun	s2	40
flowers	s2	90
gras	s2	7
vincent	s3	3
paintings	s2	4
...		

Fig. 2. Keyword index

value keyword. Fig. 2 shows extracted keywords from a small example. The keywords are stored in an inverted index which comprises the keywords and their corresponding contexts. For instance, the keyword "flowers" occurs in an instance value and its context is specified by (concept: `paintings`, property: `title`). Based on this information a global query expression can be deduced to extract all instances supporting the keyword. For the given example, the query is $\sigma_{\mathit{title}\sim=\mathit{flowers}}(\mathbf{ext}(\mathit{paintings}))$. That approach allows the easy implementation of the index within a relational database system.

The complete keyword query evaluation starts with the selection of the desired concepts. Then, the necessary query expressions are generated by function **getQExpr** for a set of keywords and the preselected concepts. For each query expression a possible further instance expression is added and the final expression is evaluated by the query processor of the mediator.

In detail, the function **getQExpr** works as illustrated in Fig. 3. Given are a set of concepts C and a set of keywords KW. Initially, all index entries ($\mathcal{E}_{\mathit{cand}}$) containing one of the keywords in KW are selected. Keywords found in meta objects (concept and category names) correspond also to sub-concepts and sub-categories, respectively. For example, the keyword "cultural" of the concept "cultural asset" applies to all direct and indirect subconcepts, e.g. "painting".

The next steps are executed separately for each found concept. If the current concept does not comprise entries for each keyword in KW it will be not processed, because no object can be found in the extension containing all keywords. Otherwise, we build all combinations of entries, i.e. of queries, that each given keyword occurs at least once. That is achieved by grouping the entries into sets E_{kw_i} for each keyword $kw_i \in KW$, $i = 1, \ldots, n$. Subsequently, the Cartesian product is computed. As each index entry corresponds to a predicate, one can interpret the result *Cond* as Disjunctive Normal Form. That means, each entry in *Cond* is a set of entries, which is translated to a conjunction of predicates, which are disjunctively connected in turn (function **translateToQExpr**).

Input:
C – set of concepts
$KW = \{kw_1, \ldots, kw_n\}$ – set of keywords
Output:
QE – query expression of the form $\sigma_{cond}(\textbf{ext}(c))$
Given:
$\mathcal{I}$ – set of index entries $e = \langle kw, \textit{role}, \textit{concept}, \textit{property}, \textit{category}\rangle$

function getQExpr(C,KW)
$QE := \emptyset$
/ including entries for sub-concepts and sub-categories */*
$\mathcal{E}_{cand}$:= **selectIndexEntries**(KW,$\mathcal{I}$)
/ Assume $C_{\mathcal{E}_{cand}}$ the set of concepts occurring in $\mathcal{E}_{cand}$ */*
for each $c \in C_{\mathcal{E}_{cand}}$ **do**
if $\mathcal{E}_{cand}$ comprises for concept c entries for each $kw_i \in KW$ **do**
Cond $:= \emptyset$
/ build sets of entries for each keyword*/*
$E_{kw_i} := \{e : e \in \mathcal{E}_{cand}, e.kw = kw_i \wedge e.\textit{concept} = c\}, i = 1, \ldots, n$
/ compute Cartesian product of entries */*
Cond $:= E_{kw_1} \times \ldots \times E_{kw_n}$
/ build query expressions */*
$QE := QE \cup$ **translateToQExpr**$(Cond)$
od
od

Fig. 3. getQExpr function evaluation.

As parts of the global instances can originate from different sources, the index stores also the source of a instance value keyword supporting the creation of source queries.

Evaluation. A first implementation within the YACOB prototype was used to evaluate the proposed approach. We evaluated the effectiveness of the index according to the reduction of source queries using real data sets. Using real[2] and generated query mixes showed a significant reduction of the number of source queries, e.g. the average number of source queries decreased from original 63 to 2.8 for queries comprising two keywords. That behavior is due to the distribution of the keywords over the different concepts. The distribution is normally not uniform but rather similar to a Zipf distribution [27], i.e. most keywords occur in one concept extension and only few occur in almost all concepts. As we cannot assume the index is complete in a real scenario, we tested partial indexes, which showed also a good performance according to the reduction of source queries as well as in order to provide a good recall.

[2] Extracted from an access-log file of the Web database `http://www.lostart.de`.

3.2 Further work

In the previous section we described the index-based keyword search. However, the present solution uses a static index, therefore the following issues are still in progress or planned for the future:

Maintenance. The local sources are assumed to be autonomous and uncooperative, i.e. neither do they allow a complete access to the data, nor do they provide the required information. Therefore, different maintenance methods are proposed: (*i*) a query-based sampling approach is used to (re-)build the indexes periodically. (*ii*) Extracting frequency information during query processing, that means, results of user queries are analyzed during runtime to improve the current index. The expected overhead can be reduced by using periodically the content of the semantic cache of the system [18] instead analyzing every query result. (*iii*) Using a focused crawling approach: that is, an overview about the frequency distribution of queries is maintained.

Based on this model, the index is periodically built and updated. All approaches showed in experiments advantages and disadvantages, therefore a hybrid approach is desired.

Ranked queries. The present approach supports boolean keyword search in combination of browsing the concept hierarchies. The combination allows a fast restriction of the result set sizes. Furthermore, the result set is subdivided into parts defined by the concept level.

However, there is still a need for ranking. The ranking should be based on the placement in the global concept hierarchy, i.e. an instance found in a sibling concept is less relevant, and the ranking of the instances returned by the sources. The approach requires the combination and the weighting of different ranks, but also offers new optimizations, like e.g. top-N operators.

Similarity queries. A second kind of text search includes *similarity queries*, i.e. to a given string all strings within a given distance are considered as similar. One popular measure is the edit or Levenshtein distance. Similarity queries can be used by the user directly or by integration operations which have to find duplicates. Unfortunately, most local sources do not support similarity queries directly, but substring or keyword queries. Hence, one has to map similarity queries to substring queries.

The filtering technique is one well-known approach to do this mapping. Assuming k errors are allowed between two strings, one can decompose the comparison string $k+1$ substrings. The union of the results of these $k+1$ substring queries contains all matching strings. To minimize the size of this preselection result, the most selective combination of substring queries has to be found. Therefore, one has to maintain frequency information on global level to make selectivity estimations.

Depending on the kind of substrings, possible structures are count-suffix trees, q-gram tables, or again keyword tables. In first tests, q-gram seemed to be most promising according to maintenance costs and query costs. Similar to the keyword search building and maintenance issues arise. Thus, the proposed methods of the keyword search can be generalized to this and possibly to other problems.

Further evaluation. The additional techniques and algorithms are to be evaluated using the prototype implementation. Especially, we want to test the adaptability to

changing sources and workloads. Furthermore, the maintenance overhead has to be evaluated for different strategies.

4 Conclusion

The proposed work investigates the problem of index-supported keyword search over virtually integrated data. The integration system is an ontology-based mediator, as described before in the literature. A keyword search strategy has been developed. This integration allows a more efficient usage of the integrated data than browsing and structured queries alone. Furthermore, it does not require to materialize the data of the different sources into one site. Tests with a prototype in a static scenario showed promising results. Ideas for maintaining the global index based on techniques known from search engines have to be adopted to the proposed approach.

In order to complement the keyword search we pointed out the support of similarity queries. Here, global string similarity queries can be supported even if the source does not support them. Frequencies of occurrences of substrings have to maintained on the global site, which rises similar problems as for the keyword search. Hence, we can generalize our approach to more problems.

References

1. S. Abiteboul. Querying Semi-Structured Data. In *ICDT 1997*, volume 1186 of *LNCS*, pages 1–18. Springer, 1997.
2. S. Agrawal, S. Chaudhuri, and G. Das. DBXplorer: A System for Keyword-Based Search over Relational Databases. In *ICDE 2002*, pages 5–16, 2002.
3. B. Amann, C. Beeri, I. Fundulaki, and M. Scholl. Ontology-Based Integration of XML Web Resources. In *ISWC'2002*, volume 2342 of *LNCS*, pages 117–131. Springer, 2002.
4. Y. Arens, C.Y. Chee, C.-N. Hsu, and C.A. Knoblock. Retrieving and Integrating Data from Multiple Information Sources. *International Journal on Intelligent and Cooperative Information Systems*, 2(2):127–158, 1993.
5. R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, 1999.
6. C.K. Baru, A. Gupta, B. Ludäscher, R. Marciano, Y. Papakonstantinou, P. Velikhov, and V. Chu. XML-Based Information Mediation with MIX. In *SIGMOD 1999*, pages 597–599, 1999.
7. M.K. Bergmann. The deep web: Surfacing hidden value, 2003. `http://www.brightplanet.com/deepcontent/tutorials/DeepWeb/`.
8. G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword Searching and Browsing in Databases using Banks. In *ICDE 2002*, pages 431 – 440, 2002.
9. J.P. Callan and M.E. Connell. Query-based sampling of text databases. *Information Systems*, 19(2):97–130, 2001.
10. S. Chakrabarti, M. van den Berg, and B. Dom. Focused Crawling: a new approach to topic specfi c Web resource discovery. *WWW8 / Computer Networks*, 31(11-16):1623–1640, 1999.
11. D. Florescu, D. Kossmann, and I. Manolescu. Integrating Keyword Search into XML Query Processing. *WWW9 / Computer Networks*, 33(1-6):119–135, 2000.
12. A. Friedmann, A. Levy, and T. Millstein. Navigational Plans for Data Integration. In *AAAI/IAAI 1999*, pages 67–73, 1999.

13. H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J.D. Ullman, V. Vassalos, and J. Widom. The TSIMMIS Approach to Mediation: Data Models and Languages. *Journal of Intelligent Information Systems*, 8(2):117–132, 1997.
14. C.H. Goh, S. Bressan, S.E. Madnick, and M.D. Siegel. Context Interchange: New Features and Formalisms for the Intelligent Integration of Information. *ACM Transactions on Information Systems*, 17(3):270–293, 1999.
15. N. Green, P.G. Ipeirotis, and L. Gravano. SDLIP + STARTS = SDARTS a protocol and toolkit for metasearching. In *ACM/IEEE Joint Conference on Digital Libraries*, pages 207–214, 2001.
16. V. Hristidis and Y. Papakonstantinou. Discover: Keyword Search in Relational Databases. In *VLDB 2002*, pages 670–681, 2002.
17. P.G. Ipeirotis and L. Gravano. Distributed Search over the Hidden Web: Hirarchical Database Sampling and Selection. In *VLDB 2002*, 2002.
18. M. Karnstedt, K.-U. Sattler, I. Geist, and H. Höpfner. Semantic Caching in Ontology-based Mediator Systems. In *Berliner XML Tage 2003, 3rd Int. Workshop "Web und Datenbanken"*, pages 155–169, October 2003.
19. A.Y. Levy, A. Rajaraman, and J.J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In *VLDB 1996*, pages 251–262, 1996.
20. B. Ludäscher, A. Gupta, and M.E. Martone. Model-based Mediation with Domain Maps. In *ICDE 2001*, pages 82–90, 2001.
21. A. Magkanaraki, G. Karvounarakis, Ta Tuan Anh, V. Christophides, and D. Plexousakis. Ontology Storage and Querying. Technical Report 308, Foundation for Research and Technology Hellas, Institute of Computer Science, April 2002.
22. U. Masermann and G. Vossen. Design and Implementation of a Novel Approach to Keyword Searching in Relational Databases. In *ADBIS-DASFAA 2000*, pages 171–184, 2000.
23. K.-U. Sattler, I. Geist, and E. Schallehn. Concept-based Querying in Mediator Systems. *The VLDB Journal*, 2004. To appear.
24. S. Sizov, M. Theobald, S. Siersdorfer, G. Weikum, J. Graupmann, M. Biwer, and P. Zimmer. The BINGO! System for Information Portal Generation and Expert Web Search. In *CIDR 2003*, 2003.
25. A. Theobald and G. Weikum. The Index-Based XXL Search Engine for Querying XML Data with Relevance Ranking. In *EDBT 2002*, pages 477–495, 2002.
26. G. Wiederhold. Mediators in the Architecture of Future Information Systems. *IEEE Computer*, 25(3):38–49, 1992.
27. G. K. Zipf. *Human Behavior and the Principle of Least Effort*. Addison-Wesley, 1949.