Expression of Z39.50 Supported Search Capabilities by Applying Formal Descriptions

Michalis Sfakakis and Sarantos Kapidakis

Archive and Library Sciences Department / Ionian University ,
Plateia Eleftherias, Paleo Anaktoro GR-49100 Corfu, Greece
{sfakakis, sarantos}@ionio.gr

Abstract. The wide adoption of the Z39.50 protocol from the Libraries exposes their abilities to participate in a distributed environment. In spite of the protocol specification of a unified global access mechanism, query failures and/or inconsistent answers are the pending issues when searching many sources due to the variant or poor implementations. The elimination of these issues heavily depends on the ability of the client to make decisions prior to initiating search requests, utilizing the knowledge of the supported search capabilities of each source. To effectively reformulate such requests, we propose a Datalog based description for capturing the knowledge about the supported search capabilities of a Z39.50 source. We assume that the accessible sources can answer some but possibly not all queries over their data, and we describe a model for their supported search capabilities using a set of parameterized queries, according to the Relational Query Description Language (RQDL) specification.

1 Introduction

The Z39.50 client/server information retrieval protocol [1] is widely used in Libraries electronic communication for searching and retrieving information from a number of diverse, distributed, heterogeneous and autonomous sources. According to Z39.50 architecture, every client can communicate with multiple servers (in parallel or sequentially), and every server can publish many sources not necessarily with the same structure and search capabilities.

The protocol unifies the access to the sources by providing an abstract record-based view model, hiding the logical structure and the access methods of the underlying sources. The supported query mechanism, utilizes sets of predefined Access Points combined with specific attributes (i.e. Attribute Sets), in a number of different query language specifications (i.e. query types). The general conformance requirements of the protocol, for the accomplishment of the standard search primitives, specify that at least the Access Points defined in the attribute set Bib-1 and the query Type-1 for the query formulation has to be recognized (although not necessarily implemented).

The consequences of these general conformance requirements are the arbitrary support of different subsets of the attribute set Bib-1 and also the different capabilities of the Type-1 query language, in the working Z39.50 environments. When a Z39.50 server does not support a requested Access Point or its attribute type values, the response is either a message for unsupported search (query failure), or an arbitrary substitution of the unsupported attributes with others supported, giving unpredictable results. The client, can either restrict the available search characteristics to the set of the lowest common dominants, or reject all the attribute types' values for the query term and let each server apply any interpretation for them. Both approaches avoid query failures, but they either limit the querying facilities of the sources or produce inconsistent results.

When searching many sources, it is apparent that the elimination of the query failures and the improvement of the consistency for the answers depend on the client's ability: (i) to discover the supported search capabilities of every Z39.50 source and; (ii) based on this knowledge, to make decisions and probably to transform the query, prior to initiating search requests. For discovering the information about a Z39.50 source, the conformance to an implementation profile (e.g. Bath [16]) or the Explain facility of the protocol can be used. The ability of the client to decide and also to determine efficiently the appropriate query transformations heavily depends on the representation model used to capture the supported search capabilities of every source.

In the area of databases, a number of methods have been proposed for the representation and manipulation of the supported search capabilities from sources, based on formal descriptions [18]. Some of these describe the source's supported search capabilities by infinite families of queries, using a set of parameterized queries.

This work describes the supported search capabilities of a Z39.50 server at a higher level than the already existing mechanisms in the family of the Explain services, using a logic based language. The accessible sources are treated as sources which can answer some but not all possible queries over their data. Their supported search capabilities are described using a set of parameterized queries according to the Relational Query Description Language (RQDL) specification [12].

The rest of this work is organized as follows: section 2 presents the related work concerning the integrated access to multiple sources. Section 3 highlights the Z39.50 protocol, its access model, and describes the issues when searching many sources. Section 4, after a short introduction to the RQDL basics, presents the description of the supported search capabilities of a Z39.50 server. Finally, section 5 concludes and presents a number of interesting issues arrived from this work for further research.

2 Related Work

The problem of providing integrated access to multiple, distributed, heterogeneous and autonomous sources (databases, or other) has received considerable attention over a decade in the database research community, and is referred as constructing answers to queries using logical views. A common information integration architecture, shown in fig. 1, is based on the Mediators and Wrappers approach [20]. In this architecture, every source is wrapped by software (wrapper) which translates between the underlying source query language and data model to a common global language and data model. The Mediator receives queries from a client or a user, which are expressed in the global language and data model, and translates them into new queries, according to the wrapper capabilities description, which are sent to the wrappers. The translated queries are also expressed in the common language and model. Thus a mediator can be thought as a global view of the integrated system, the wrapper as a local view of the underlying source and the problem of the information integration as constructing answers to queries using views that represent the capabilities of the information sources.

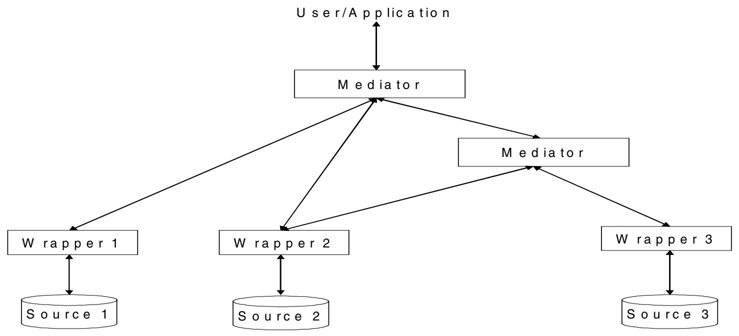

Fig. 1. Common Information Integration Architecture

Depending on the way that the global and the local views are constructed, there are two main approaches. The first one is the Local as View (LaV), where the global schema is defined independently of the local sources schemas. Each source is described in terms of the global schema, thus the sources are viewed as materialized views of the global schema. Using this approach is easy to add new sources in the system, but query transformation has exponential time complexity. The second approach is the Global as View (GaV), where the global schema is defined in terms of the local schemas. In this approach query transformation can be reduced to rule unfolding, but when a new source is added to the system, in most cases, the global schema has to be reconstructed. The Information Manifold [9] and the TSIMMIS [4] are two representative systems based on the Mediator/Wrapper architecture and implementing the two deferent view models respectively.

In the literature, a number of formal methods have been proposed [8] dealing with the problem of answering a query, posed over a global schema on behalf of representative source schemas. Most of theses methods are based on the assumption that there is unrestricted access to the participated sources and their data schema, which in many cases is not a realistic one. Later extensions of the query/view model describe the access to sources by infinite families of queries [11, 10]. These approaches view the sources as processors that can answer some but not all possible queries over their data and describe those using a set of parameterized queries.

3 Z39.50 Protocol and the Multiple Search Problem Description

The Z39.50 is a state-full protocol based on the client/server model [1, 6]. It defines a standard manner for the communication between the client and the server, giving them the ability to interoperate independently of the underlying source structure, search procedures and computer systems. The system level interoperability is approached by the definition of a set of specific services, which is accomplished by the exchange of specific messages, between the client and the server. For the content semantics of the published sources (databases), the protocol defines a standard model in a record-based abstract view, hiding the logical structure of the underlying source.

3.1 Access Model and the Explain Facility

For the implementation of the search primitives, the protocol utilizes the concept of the abstract Access Point, which the client can only use to query the sources. A server can supply access to many sources and for every source a different subset of the global set of Access Points could be supported.

In order to formulate a query, the protocol specifies many different query types (called Type-0, Type-1, etc.) mainly affecting the syntax of the query. For every search term, we have to define its characteristics by declaring the Attribute Set it belongs to. The Attribute Set defines the valid Access Points (i.e. what entities represent the search terms) from a specific set of attribute types, the way the system will match them against the underlying data, and the form in which the terms have been supplied. For the most commonly used Attribute Set Bib-1, the following attribute types exist: *Use* (e.g. Title, Author, etc.), *Relation* (e.g. Equal, less than, etc.), *Position* (e.g. First in field, any position in field, etc.), *Structure* (e.g. phrase, word, word list, etc.), *Truncation* (e.g. right, do not truncate, etc.) and *Completeness* (e.g. complete field, etc.).

According to the protocol, if a target does not support a given attribute list, the target should fail the search (i.e. query failure) and supply an appropriate diagnostic message, or the target will substitute it according to the 'Semantic Action' value. In most cases, the vast majority of the running Z39.50 servers ignores the 'Semantic Action' value and makes an arbitrary substitution of the unsupported attributes, without informing the client.

The Explain facility is the build-in mechanism in the protocol for a client to obtain the implementation details of a server. According to the service specification, among the information which a client can acquire from a server is the list of the supported Access Points with their Attribute Type combinations for every available source (database). The complexity of the implementation of the Explain facility, results to a small number of existing implementations. The latest approach to solve the problem of discovering information about a Z39.50 database is the ZeeRex [3], based on the experiences of the previous approaches. All the Explain approaches publish the supported access characteristics of a source by enumerating them in a list, without providing any information on the way they should be used.

3.2 Multiple Search Problem Description and Correlation to SRW

When searching multiple sources, the different implementations of the protocol result to query failures and/or inconsistent answers, despite of the unified access mechanism of the protocol. The different implementations mostly differ either to the subsets of the supported attribute types, or to the supported query language characteristics. The following examples illustrate some real world circumstances when a client tries to search many sources.

Example 1 (supported access point with different combinations of values for the other attribute types). Consider two sources, both of them answering queries using the Access Point (Use attribute) *Title*. Also both of them could combine this Access Point with the values *Phrase* or *Word* for the attribute type *Structure* and the last one supports additionally the value *Word List*. Finally the supported values for the attribute type *Truncation* are *Right* or *Do Not Truncate* on any *Structure* value. In summary the allowed searches for the Access Point *Title* by these two sources are:

```
(S1): Structure-(phrase, word),
                Truncation-(right, do not truncate)
(S2): Structure-(phrase, word, word list),
                Truncation-(right, do not truncate).
```

Obviously, Q1 (i.e. Search for the bibliographic records having the Title 'Data Structures in Pascal') is one supported query by both sources:

```
Q1: (Title, 'Data Structures in Pascal')
    (Structure, phrase) (Truncation, do not truncate).
```

The query Q2 is not supported by the source S1 due to the unsupported value *Word List* for the attribute type *Structure*.

```
Q2: (Title, 'Data Structures') (Structure, word list)
    (Truncation, do not truncate).
```

If a client knows that this query is not supported by the source S1, it could rewrite Q2 with the equivalent Q3, for the source S1, in a preprocessing step before sending it to the server, and will achieve the same recall and precision from the answer, as follows:

```
Q3: (Title, 'Data') (Structure, word)
(Truncation, do not truncate) AND (Title, 'Structures')
(Structure, word) (Truncation, do not truncate).
```

In order to simplify the description of the example, we made the assumption that both sources support the same value combinations for the remaining attribute types (i.e. *Relation*, *Position*, *Completeness*), for the used Access Point *Title*. In this example, the assumptions for the attribute types *Relation*, *Position*, *Completeness* were the values *Equal*, *First in Field*, *Complete Field*, respectively.

Example 2 (unsupported access point). Both sources support the Access Point *Author* with the following attribute types:

```
Access Point: Author
(S1): Structure-(phrase, word), Truncation-(right)
(S2): Structure-(phrase, word), Truncation-(right).
```

Also the S2 source additionally supports the Access Point *Author Personal Name*

```
Access Point: Author Personal Name
  (S2): Structure-(phrase, word, word list),
        Truncation-(right, do not truncate).
```

Q4 is an unsupported query from the source S1, due to the unsupported Access Point *Author Personal Name*:

```
Q4: (Author Personal Name, 'Ullman')
    (Structure, word) (Truncation, right).
```

A smart client must take into account the semantics of the Access Points (e.g. *Author Personal Name* is a subclass of *Author*) and transform the query for the S1 source, with less precision than the original one, as follows:

```
Q5: (Author, 'Ullman') (Structure, word)
    (Truncation, right).
```

Closing the description of the issues concerning the Z39.50 environment, one interesting point is to address how these issues might impact the deployment of the Search and Retrieve Web Service (SRW) protocol [14]. The SRW is building on the Z39.50 semantics and retains the main concept of the abstract Access Points used in the access model of the Z39.50 protocol [15]. Also, in spite of the differences in the used terminologies (e.g. Z39.50 uses Attribute Sets and Attribute Types, SRW uses Context Sets and Indexes), the CQL query language used in SRW attempts to combine the simplicity and intuitiveness of expression with the richness of the Z39.50's Type -1 query language [5].

Also, the Explain facility of SRW is mandatory and uses the ZeeRex description for publishing the supported search capabilities of a source. As we saw in section 3.1, ZeeRex simply lists the supported Access Points without providing a representation model for effective management and use of the supported search capabilities.

It is apparent that the same issues still exist when searching multiple sources in the SRW environment, and consistent searching requires the description of the supported search capabilities of the underling sources in a higher level than the one offered from the ZeeRex. Also, a higher-level description can be used as a bridge between the multiple sources when searching them either via the Z39.50 or the SRW protocol.

4 Description of a Z39.50 Server Supported Search Capabilities

In our approach we treat a Z39.50 server as a wrapper for the sources, able to answer some but not all possible queries over the data of every individual source. We recall that, all possible elements which can participate in a query are those defined as the Access Points in an Attribute Set and for every Access Point additional attributes could define its supplied form and the matching criteria against the data. Also, the queries are formulated according to a specific language (query type). Finally, an answer to a query is the set of all unique identifiers of the metadata records fulfilling the search criteria.

4.1 RQDL Basics

As the language for the description of the supported capabilities of the source, we use the Relational Query Description Language (RQDL). RQDL is a Datalog-based rule language, first proposed by Papakonstantinou et al. [12], for the description of a wrapper's supported queries. Its main advantages are the ability to describe infinite query sets and the support of schema-independent descriptions. The language focuses on conjunctive queries and is powerful enough to express the abilities of many sources. Also, its extended version [19] can describe the set of all conjunctive queries. Due to the Datalog-based nature of the RQDL, we express the queries using Datalog [17].

We informally introduce the basic characteristics of the description language. The complete language specification is in [12], and the formal specification for its extended version is in [19]. An RQDL description is a finite set of RQDL rules, each of which is a parameterized query (i.e. query template). A query template has a 'constant placeholder' instead of the constant value of an actual query, thus represents many queries of the same form. For the restrictions on the actual values, which will replace the constant placeholders, the description language provides metapredicates on them.

4.2 Access Point Templates

We consider that a source wrapped by a Z39.50 server exports a predicate *metarec(Id)* representing the set of the unique identifiers of its metadata records. Also the source exports the predicate of the general form:

```
property(Id, Pname, Pattribute1, …, Pattributen, Pval).
```

The relation expressing the meaning of the predicate *property* contains all the unique Ids from the metadata records having a property *Pname* with value that matches the *Pval* argument, according to the criteria specified from the additional *Pattribute*$_j$, j=1,…,n attributes. Thus a valid element making the predicate property successful is:

```
property(X, use_Title, rel_Equal, pos_FirstInField,
         str_Phrase, tru_DoNotTruncate,
         com_CompleteField, 'Data Structures')
```

stating that the metadata record X has a property *use_Title* (i.e. Title) with value that matches the last argument '*Data Structures*' according to the matching criteria defined from the third to seventh attributes (i.e. exact match).

From the predicate property we use the argument *Pname* to describe the supported Access Point from the source. Also, for the description of the other attribute types *Relation*, *Position*, *Structure*, *Truncation* and *Completeness*, defined in the Bib-1 attribute set, we use the other five arguments *Pattribute*$_j$. The values for the arguments *Pname* and *Pattribute*$_j$, in the predicate *property*, are the constants defined for the attribute types in the Bib-1 attribute set of the Z39.50 protocol. For readability purposes, we combine symbolic names and we do not use the actual numeric values as specified in the protocol. So, the symbolic name *use_Title* stands for the pair values (1, 4) representing the *Use* (i.e Access Point) attribute type with value 4 (*Title*).

According to the RQDL specification, in order to define a representation for the set of the same form queries, in our case the queries concerning an Access Points and its characteristics, we have to define a query template using 'constant placeholders'. These constant placeholders will be replaced in the actual queries with constant values. Thus, for the description of the family of the queries which use an Access Point with its attributes, we use the Access Point template:

```
property(Id, Pname, Pattribute1, …, Pattributen, $Pval).
```

The identifiers started with the '$' are the constant place holders (e.g. *$Pval*) following the syntax of the RQDL. As an example for an Access Point template which specifies that the source supports the Access Point Title combined with the values *Equal*, *First in Field*, *Phrase*, *Do not Truncate* and *Complete Field*, for the other attribute types *Relation*, *Position*, *Structure*, *Truncation* and *Completeness* respectively (i.e. an exact match for the Title), is the following:

```
property(Id, use_Title, rel_Equal, pos_FirstInField,
             str_Phrase,tru_DoNotTruncate,
             com_CompleteField, $Pval).
```

The matching process of an Access Point specification, used in a query, and an Access Point template is accomplished by replacing the constant placeholders (e.g. *$Pval*) with the corresponding actual constants and then by applying standard unification procedures

The number of the Access Point templates we have to write, in order to describe all possible combinations of the attribute types for a single Access Point, is the product of ($|Pa_j|$+1) for j= 1, …, 5, where Pa_j is the set of the constant values defined for the j^{th} attribute type (including the null value). Thus we have 23,936 possible templates for every Access Point from the Attribute Set Bib-1, according to the protocol specification [1]. As described in the semantics of the Attribute Set Bib-1 [2], there is a number of conflicting or meaningless combinations of Attribute Types which decrease the above number of possible templates. E.g., Position with value 'First in subfield', where 'subfield' has no meaning or, Position attribute 'any position in field' is compatible only with the 'incomplete subfield' Completeness attribute, etc. In practice, we expect the number of the required templates to be small, according to wide adopted implementation profiles like the Bath [16]. As an indication for the order of magnitude of the number, we refer that there are totally only five attribute type combinations for each Access Point Author, Title and Subject in Bath profile (Functional Area A, Level 1). Thus we need five Access Point Templates only for each of the three Bath's Access Points.

According to the protocol, the specification of a query term may omit values for some attribute types. This leads to unspecified arguments when we construct the predicate property for the used term in the query. In this case the underscore '_' symbol can be used, and in the unification process it matches with any value in the corresponding argument of the template. When using unspecified arguments, there is a possibility that the corresponding predicate to the requested query term, will match with more than one Access Point Templates. For the decision of the matching Access Point template and in case were only one source is in use, we can make an arbitrary selection reflecting the intentions of the user and without conflicting with the protocol. A more interesting approach, which we have to examine further, is to select the template by taking into account user preferences limiting the degree of the expansion or the restriction of the query results. When many sources are involved, the primary criterion could be the selection of a template supported from all sources, or a common one after taking into account user preferences for achieving the same semantic changes for all sources. Both approaches satisfy the consistency of the answers but they differ on the achieved recall and precision.

Using the Access Point Template we can enumerate the supported Access Points and their attributes by a source. Even if we know that a source supports all the Access Points used in the query we may not be able to decide for the support of the whole query, due to the possibility of unsupported combinations in the query expressions.

4.3 Query Templates

This section extends the description of the Z39.50 supported search capabilities so that, we will be able to decide if a template describes a specific class of queries. Combining the predicates *metarec* and *property*, we can write a query requesting all the metadata records from a source which supports exact *Author* search, as:

```
(Q1): answer(X):- metarec(X), property(X, use_Author,
        rel_Equal, pos_FirstInField, str_Phrase,
        tru_DoNotTruncate,com_CompleteField, 'Ullman').
```

A query template (D1), using the RQDL specification, which describes the capabilities of a source that supports only exact *Author* searches, is the following:

```
(D1): answer(Id):- metarec(Id), property(Id,
   use_Author, rel_Equal, pos_FirstInField, str_Phrase,
       tru_DoNotTruncate, com_CompleteField, $Pval).
```

A query is described by a template if each predicate in the query matches one predicate in the template and vice versa, and also, any metapredicate in the template evaluates to true when the constant value will replace the constant placeholder. The order of the predicates does not affect the matching process.

Query (Q1) matches the template of the description (D1), because the predicates used in the query match the predicates used in the template description and vice versa, with the following unification assignments: *X=Id*, *$Pval* = '*Ullman*'. Thus description (D1) describes the query (Q1). In case were other Access Points are supported from the source, in order the description (D1) to describe the whole capabilities (i.e. the set of all supported queries) we have to supplement D1 with a similar template for every other supported Access Point.

For the description of large or infinite sets of supported capabilities, we can use recursive rules. RQDL utilizes the concept of the *nonterminals* (as in context-free grammars), representing them by identifiers staring with an underscore (_) and a capital letter. A template containing *nonterminals* forms a *nonterminal template*. An expansion of a query template *qt* containing nonterminals is obtained by replacing each nonterminal of *qt* with one of the nonterminal templates that define it until there is no nonterminal in *qt*. Finally, a query template *qt* containing nonterminals describes a query *q* if there is an expansion of *qt* that describes *q*.

As an example, let's consider a source that supports the Access Points referred in the previous example, and also supports exact matches for the *Subject* and the *Title* Access Points plus any possible combination of them. A representative supported query by the server could be:

```
(Q2): answer(X):- metarec(X), property(X, use_Author,
      rel_Equal, pos_FirstInField, str_Phrase,
      tru_DoNotTruncate, com_CompleteField, 'Ullman'),
    property(X, use_Author, rel_Equal, pos_FirstInField,
        str_Phrase, tru_DoNotTruncate,
        com_CompleteField, 'Garcia-Molina'),
    property(X, use_Subject, rel_Equal, pos_FirstInField,
        str_Phrase, tru_DoNotTruncate,
        com_CompleteField, 'Datalog'),
    property(X, use_Title, rel_Equal, pos_FirstInField,
        str_Phrase, tru_DoNotTruncate,
        com_CompleteField, 'Database Systems').
```

Using the nonterminal templates *_Cond* and *_Cond1*, a description for the supported queries from the server could be:

```
(D2): answer(Id):- metarec(Id), _Cond(Id)
(NT2.1) _Cond(Id):- _Cond(Id), _Cond1(Id)
(NT2.2) _Cond(Id):- _Cond1(Id)
(NT2.3) _Cond1(Id):- property(Id, use_Title,
     rel_Equal, pos_FirstInField, str_Phrase,
     tru_DoNotTruncate, com_CompleteField, $Pvalue)
(NT2.4) _Cond1(Id):- property(Id, use_Subject,
     rel_Equal, pos_FirstInField, str_Phrase,
     tru_DoNotTruncate, com_CompleteField, $Pvalue)
(NT2.5) _Cond1(Id):- property(Id, use_Author,
     rel_Equal, pos_FirstInField, str_Phrase,
     tru_DoNotTruncate, com_CompleteField, $Pvalue).
```

Also, the (E1) is an expansion of the query template (D2):

```
(E1): answer(Id):- metarec(Id),
      property(Id, use_Title, rel_Equal,
        pos_FirstInField, str_Phrase,
        tru_DoNotTruncate,com_CompleteField, $Pv1),
      property(Id, use_Subject, rel_Equal,
        pos_FirstInField, str_Phrase,
        tru_DoNotTruncate, com_CompleteField, $Pv2),
      property(Id, use_Author, rel_Equal,
        pos_FirstInField, str_Phrase,
        tru_DoNotTruncate, com_CompleteField, $Pv3),
      property(Id, use_Author, rel_Equal,
        pos_FirstInField, str_Phrase,
        tru_DoNotTruncate, com_CompleteField, $Pv4).
```

We recall that the order of the predicates does not affect the matching process between the query and the query template. Also, before starting the expansion, all of the variables of the template are renamed to be unique. This expansion describes the query (Q2), because the predicates used in the query match the predicates used in the template (and vice versa) with the unification assignments *X=Id*, *$Pv1* = '*Database Systems*', *$Pv2* = '*Datalog*', *$Pv3* = '*Garcia-Molina*', *$Pv4* = '*Ullman*'.

4.4 Deciding for the Support of a Query

Having an RQDL description of the supported search capabilities of a Z39.50 source, the next step is to decide if the source is able to answer a given query. We recall that we express the queries (conjunctive) using the Datalog and also that an RQDL rule is a Datalog-based rule using constant placeholders in addition to variables and constants. The process of finding a supporting query in an RQDL description is reduced to the problem of determining whether a conjunctive query is contained in a Datalog program [11, 12, 19].

The Query Expressibility Decision (QED) [19] and the X-QinP [11] are two extensions of the classic algorithm for deciding query containment in a Datalog program [13, 17]. When the supported capabilities are described using recursive rules, the query template has an infinite number of expansions. Furthermore, we have to check the query for one or more matches within the infinite number of expansions in order to decide if a source is able to answer a query. In this case, a variant of 'magic set rewriting' [17] makes the process of deciding the support of a query more efficient [12].

Closing our approach for the description of a Z39.50 server supported search capabilities, we emphasize the importance of the applicability of the well-studied theory and algorithms from the area of the deductive databases.

5 Conclusions and Future Research

In this work we have addressed the need for the formal description of the supported search capabilities of Z39.50 sources, especially when multiple sources have to be searched. The proposed logic based description enables the client to make decisions prior to initiating the search requests. Also, the existing Explain family services can be used complementary to our description by providing input information, when they are implemented. The accessible sources are treated as sources which can answer some but not all possible queries over their data. We describe the search capabilities supported by a source using a set of parameterized queries, according to the Relational Query Description Language (RQDL) specifications.

From this work, a number of interesting points arrives for future development and research. Currently, our approach can help the client or a mediator to decide if a query is directly supported or not by a Z39.50 source (i.e. the server which publishes the source is able to answer the query as is, without any substitution of any attribute). In case where the query is not directly supported by a source, a powerful extension will be the transformation of the query to a different query or a set of queries, so that (preferably) identical or (otherwise) similar semantics are obeyed. Finding ways to extend the description templates using characteristics of the underlying data models and schemata of the sources will improve the overall process of deciding if a source supports a query, directly or indirectly. Also, the relations among the query language operations and the correlations between the Access Points hierarchies could really enforce the transformation procedures, especially when the query can be transformed only to a similar query.

References

1. ANSI/NISO: Z39.50 Information Retrieval: application service definition and protocol specification: approved May 10, 1995.
2. Attribute Set BIB-1 (Z39.50-1995): Semantics. ftp://ftp.loc.gov/pub/z3950/defs/bib1.txt.
3. An Overview of ZeeRex. 28th August 2002. http://explain.z3950.org/overview/index.html.
4. Chawathe, S., Garcia-Molina, H., Hammer, J., Irelandand, K., Papakonstantinou Y., Ullman, J. and Widom, J. The TSIMMIS Project: Integration of Heterogeneous Information Sources. IPSJ, Tokyo, Japan, October 1994.
5. CQL – Common Query Language, version 1.1, February 2004. Available from: http://www.loc.gov/z3950/agency/zing/cql.html
6. Finnigan, S., Ward, N. Z39.50 Made Simple. Available from: http://archive.dstc.edu.au/DDU/projects/Z3950/zsimple.html.
7. Gill, Tony and Miller Pall. Re-inventing the Wheel? Standards, Interoperability and Digital Cultural Content. D-Lib Magazine vol. 8:num. 1 (January 2002).
8. Halevy, A. Answering Queries using views: A Survey. The VLDB jour. 10: 270-294 (2001).
9. Kirk, T., Levy, A., Sagiv, Y. and Srivastava, D. The Information Manifold. AAAI Spring Symposium on Information Gathering, 1995.
10. Levy, A., Rajaraman, A., Ullman, J. Answering Queries Using Limited External Query Processors. PODS 96, Montreal Quebec Canada.
11. Papakonstantinou Y., Gupta, A. Garcia-Molina, H. Ullman, J. A Query Translation Scheme for Rapid Implementation of Wrappers. Proceedings of the Conference on Deductive and Object Oriented Databases, DOOD-95.
12. Papakonstantinou Y., Gupta A., Hass L. Capabilities-Based Query Rewriting in Mediator Systems. 4th International Conference on Parallel and Distributed Information Systems (PDIS-96), December 18-20, 1996.
13. Ramakrishnam, R., Sagiv, Y., Ullman, J. and Vardi, M. Proof Tree Transformation Theorems and their Applications. Proc. 8ht ACM Symposium on Principles of Database Systems, pp. 172-181, 1989.
14. Sanderson, R. A Gentle Introduction to SRW. Available from: http://www.loc.gov/z3950/agency/zing/srw/introduction.html
15. SRW – Search/Retrieve Web Service: SRW's Relationship to Z39.50. January 22, 2004. Available from: http://www.loc.gov/z3950/agency/zing/srw/z3950.html
16. The Bath Profile: An International Z39.50 Specification for Library Applications and Resource Discovery. Available from: http://www.ukoln.ac.uk/interop-focus/bath/current/
17. Ullman, J. Principles of Database and Knowledge-Based Systems, v. I, II. Computer Science Press, New York, 1988 & 1989.
18. Ullman, J. Information Integration Using Local Views. LNCS, Proceedings of the 6th International Conference on Database Theory, pages 19-40, 1997.
19. Vassalos, V., Papakonstantinou Y. Expressive Capabilities Description Languages and Query Rewriting Algorithms. Jour. of Logic Programming, vol. 43, number 1, 2000, 75-122.
20. Wiederhold, G. Mediators in the architecture of future information systems. IEEE Computer, 25: 25-49, 1992.