Integrating a Query Language for Structured and Semi-Structured Data and IR Techniques

Andreas Heuer Denny Priebe
Database Research Group
Computer Science Department
University of Rostock
D–18051 Rostock
Germany
{heuer, priebe}@informatik.uni-rostock.de

Abstract

In this paper we describe the basic ideas and concepts behind the Information Retrieval Query Language (IRQL) that is used as one of the back-ends in the GETESS project. The front-end provides a user interface which is embedded in a dialogue system. This dialogue system allows queries to be formulated in a user friendly (i.e. exploiting a limited range of natural language) and interactive way. Access to the analyzed data is provided by IRQL. The principal focus of IRQL development is the integration of concepts of information retrieval, database query languages, and query languages for semi-structured data. Therefore, we will be able to exploit the structure of documents, if known, and can additionally use information retrieval techniques regardless of whether the structure is known or not. Our approach develops a query language that is compatible with the recently adopted SQL99 standard and information retrieval clauses (e.g. boolean retrieval). It then integrates features of database query languages such as (1) exploiting the document's structure and (2) restructuring (including linking of multiple documents); information retrieval techniques such as (I) content-based retrieval, (II) ranking, and (III) relevance feedback; and features to also query semi-structured data. Our data model extends the object-relational model and additionally supports an abstraction of attributes. That is, we can use attribute-independent queries as well as attribute-dependent ones as in RDBMSs. We evaluate IRQL queries by mapping them to queries supported by existing systems such as object-relational DBMSs, full-text DBMSs, or conventional search engines, and post processing the results supplied by these systems, if necessary.

1 Introduction

During the last years the WWW became generally accepted as a medium to publish various kinds of information (documents). In general, this information can be categorized as structured and semi-structured/unstructured. Although storing and querying of structured data (e.g. using relational DBMSs) are well understood, there is still no agreement in managing semi-structured data (e.g. data kept in files; possibly using XML). Keeping this potential heterogeneity in mind, it is quite difficult to search for particular information. On the one hand, there are many search engines (e.g. Altavista or Infoseek) that permit the search for particular documents as it relates to their content, but these search engines are often not capable of exploiting the structure of documents in order to support advanced queries. Additionally, often data stored in DBMSs are not taken into account, although these search engines could benefit from the features of database query languages. On the other hand, pure database query languages are also inappropriate for querying heterogeneous semi-structured data [1, 8] as it relates to documents. These query languages certainly support operations on structured parts of documents, but the ability to query semi-structured data is rather limited and often realized by vendor-specific extensions to the DBMS.

In the GETESS[1] [20, 21] project we are developing a search system that is not only capable of using syntactic methods to extract information from WWW data, but also uses the semantics of the data if inferable. This is realized by building *abstracts* for each document using a parser that partially, but robustly understands natural language and an ontology that represents knowledge specific to the restricted domain "tourism". These abstracts are stored in data bases and queried with a specialized query language. The user in-

[1]GErman Text Exploitation and Search System

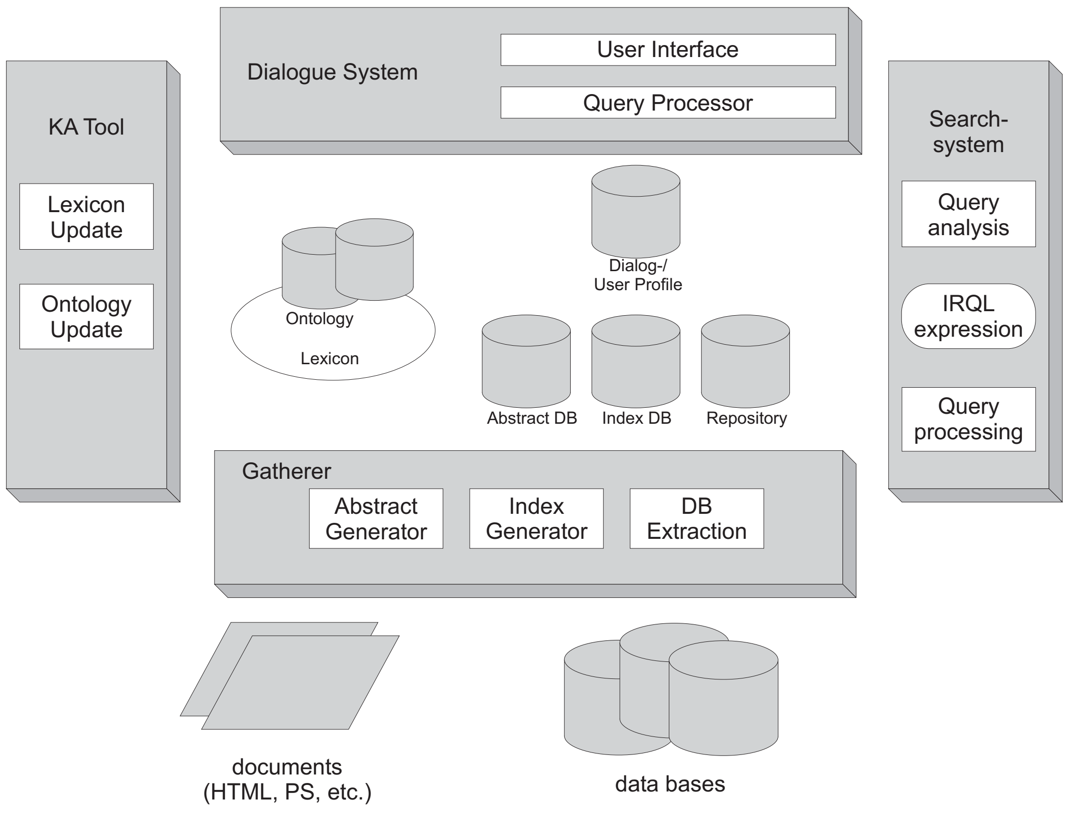


Figure 1: Architecture of GETESS

terface is embedded in a dialogue system that allows queries to be formulated in a user friendly (i.e. exploiting a limited range of natural language) way. By means of the architecture in Figure 1 we show the interaction of the different components. The front-end of GETESS provides a user interface that is embedded in a dialogue system. With this interface, users can interactively formulate queries in a limited range of natural language that is parsed by a partially, but robustly natural language processing component and translated into an IRQL expression by the query processor. This expression is handed to the search system. Its task is to evaluate the expression using the available abstracts and indices and return the results to the dialogue system. The data itself are stored in different data bases (abstract DB, index DB, DB repository) and filled by the gatherer at regular intervals. The gatherer is able to create abstracts from HTML documents and data bases using the natural language processing component mentioned earlier and to create the indices required for the information retrieval process. All subsystems are influenced by the ontology and lexicon which provide the required metaknowledge. This knowledge is acquired with the help of knowledge acquisition tools (KA tools).

In the following sections we describe the basic concepts of the Information Retrieval Query Language (IRQL). The principal focus of IRQL development is to integrate the features of database query languages such as (a) access to the data's structure, (b) use of type specific information, (c) restructuring, and (d) linking of data, features of query languages for semi-structured data, and information retrieval techniques such as (i) content-based retrieval, (ii) vague queries, (iii) ranking, and (iv) relevance feedback into a single query language. Thus, IRQL allows us to query both structured and heterogeneous semi-structured data related to documents.

Our approach is to store our data in existing systems such as object-relational DBMSs, relational DBMSs, or full-text DBMSs. Therefore, we implement IRQL on top of these systems. In principle, we evaluate IRQL queries by mapping them to the query languages supported by the corresponding platform as illustrated in Figure 2. Obviously, we have to post process the results delivered by these platforms as none of the systems support all of the IRQL features. This post-processing is either done by wrappers or compensators. Essentially, the difference between wrappers and compensators is that compensators are "big" wrappers, i.e.

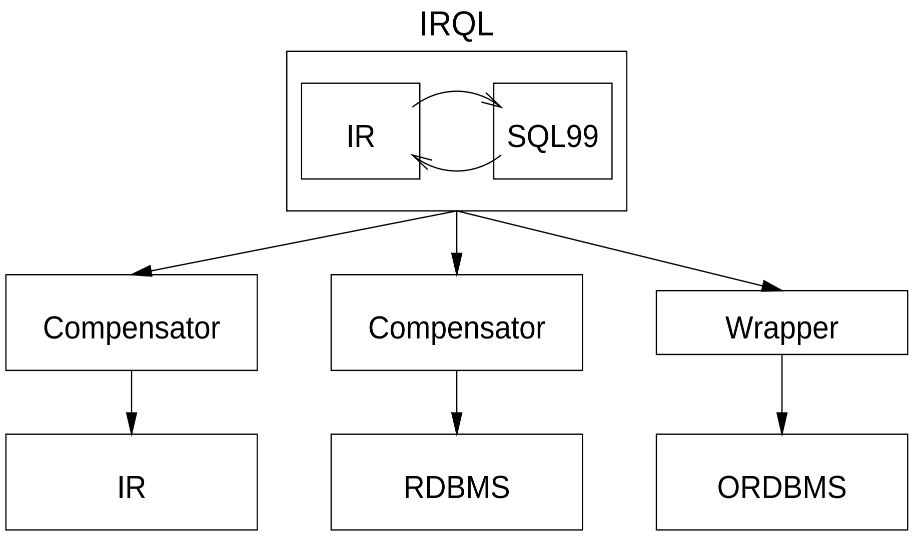


Figure 2: Implementation of IRQL

compensators encapsulate systems that support only a very limited set of the features of IRQL. To be more concrete, assume that we want to evaluate an IRQL query on top of a relational database system and the query includes selection, projection, proximity search, and ranking functions. First, we map the query's relational parts (e.g. selection and projection) of this query to the SQL supported by the RDBMS and evaluate the query. As a result, we obtain a superset of the "real" result. Now we ask the compensator to compute the answer to the IRQL query using the result from the RDBMS and the parts of the IRQL query that could not be mapped to the RDBMS query language (e.g. proximity search and ranking).

The rest of this paper is organized as follows: We describe our data model in Section 2. Section 3 presents the syntax of IRQL using some examples. In Section 4, we discuss some related work and compare other approaches with IRQL. We conclude with a summary in Section 5 and mention some future works.

2 Data Model

The principal focus of IRQL development is to implement a query language that allows to query both structured data (this implies that IRQL adopts features from database query languages) and semi-structured data. Additionally, IRQL also includes information retrieval techniques. Apart from the separate use of these query types, IRQL integrates the different possibilities in an orthogonal way.

Our data model extends the object-relational data model. There are atomic types such as[2] `integer`, `float`, `boolean`, and `string`. Composite types include the collections `set`, `bag`, `list`, and `array` as well as the `struct` constructor. Furthermore, there is a `named` type constructor that we use to model data like XML (for example) and apply type specific operations to instances of this type. In order to model heterogeneous semi-structured data, we introduce a composite type `doc` similar to the struct constructor, but querying data of this type does not produce any type checking errors. Within a doc type, we also allow for referencing non-existent labels. We illustrate the doc type

name	*place*	*rooms*
Neptun	Warnemünde	$\perp$
Hübner	Warnemünde	95
Mecklenburger Hof	Rostock	21
Atrium Hotel Krüger	Sievershagen	59

Figure 3: Structured instance R1

by an example. Figure 3 shows a structured instance of type

```
set⟨struct(name:string, place:string,
          rooms:integer)⟩.
```

Figure 4 shows a semi-structured instance containing two tuples. The first tuple type could be

```
struct(name:string,
       equipment:set⟨string⟩,
       drinks:set⟨string⟩,
       price:struct(single:integer,
                    double:integer))
```

while the second tuple could be of type

[2]We plan to support a large subset of the atomic types mentioned in [3].

name	*equipment*	*drinks*	*price*	
			single	*double*
Hübner	bathroom shower WC ...	beer wine	195	235

name	*equipment*	*price*			*cards*
		single	*double*	*twin*	
Krüger	bathroom shower WC phone ...	95	150	185	Visa Diners Club

Figure 4: Heterogeneous instance R2

```
struct(name:string, equipment:set⟨string⟩,
       price:struct(single:integer,
                    double:integer,
                    twin:integer)
       cards:set⟨string⟩).
```

In our approach, heterogeneous data is modelled as doc type. Therefore, the tuples from Figure 4 are typed as

```
doc(name:string,
    equipment:set⟨string⟩,
    drinks:set⟨string⟩,
    price:doc(single:integer,
              double:integer))
```

and

```
doc(name:string, equipment:set⟨string⟩,
    price:doc(single:integer,
              double:integer,
              twin:integer)
    cards:set⟨string⟩).
```

As we cannot unify two semi-structured types in general, but need some kind of notation for semi-structured types, we omit the attribute-value pairs from the doc type and instead use `doc` for both types. Therefore, the instance from Figure 4 could be typed as `set⟨doc⟩`. Instances of the doc type are only subject to limited type checks so that, for example, $\pi_{cards}(R2)$ only delivers the credit cards accepted by the Krüger hotel.

While the modelling of semi-structured heterogeneous data as previously discussed is not new (see e.g. WebOQL [5] and its "web" data type), our main contribution concerning the data model is to allow a set of attributes to be abstractly referenced by single attribute names as illustrated in Figure 5.

For example, we introduce two default attributes if the corresponding data originated in web documents: *source* indicates the document's URL and *complete_content* indicates the full text of the original page. As Figure 5 shows, *complete_content* is an abstraction of a set of different attributes, e.g. *metadata* such as *authors* and *text*. *Text*, in turn, is another abstraction of further attributes such as the abstract or the references of the modelled article.

In contrast to object-oriented or object-relational database models, the attributes *complete_content* and *text* are no tuple-valued attributes. For example, *complete_content* would consist of two different components *metadata* and *text* in the object models. Here, *complete_content* is considered as one text value again. The advantage of this kind of abstraction operator is the usability for information retrieval operations. If useful, the *complete_content* value can be seen as one atomic value. Another advantage of the abstraction of attributes is the possibility to easier refine queries if queries against a specific attribute level yield too few or too many objects in the result. In the case of too few results, we can automatically use a higher level of abstraction for the same query. Using tuple constructors instead, would leed to a very complicated reformulation of the query.

The problem that object-oriented and object-relational models (that are used as implementation models) do not support this kind of abstraction is hidden from the user: Our abstraction operator is implemented on top of existing object-oriented and object-relational concepts.

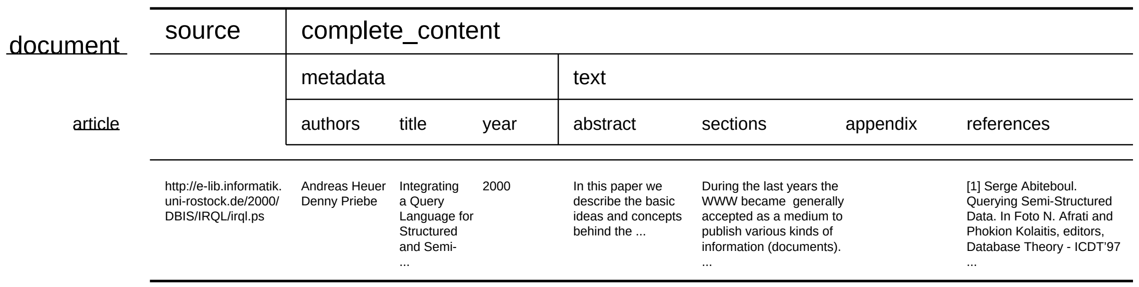
3 Language

The aim of IRQL development is to integrate database query languages, query languages for semi-structured data, and information retrieval techniques. Similar to Lorel, our approach is to realize a query language in the style of SQL, but we additionally support information retrieval techniques by adding new clauses. Like some of the query languages mentioned in Section 4, we also change the type checking rules of SQL to also support querying semi-structured heterogeneous data.

Because of these demands, we take the recently adopted SQL99 standard [3, 4] as a starting point. We plan to implement a large subset of the proposed syntax in order to be able to answer queries conforming to this standard. Using examples, we subsequently show possibilities for querying structured and semi-structured data and integrating information retrieval techniques into IRQL.

3.1 Structured and Semi-Structured Data

The data model described in Section 2 supports querying structured and semi-structured data. Structured composite data are modelled as elements of the struct data type and are therefore subject to the strong type checking as found in e.g. SQL99. Semi-structured data are modelled as elements



document	source	complete_content						
		metadata			text			
article		authors	title	year	abstract	sections	appendix	references
	http://e-lib.informatik.uni-rostock.de/2000/DBIS/IRQL/irql.ps	Andreas Heuer Denny Priebe	Integrating a Query Language for Structured and Semi- ...	2000	In this paper we describe the basic ideas and concepts behind the ...	During the last years the WWW became generally accepted as a medium to publish various kinds of information (documents). ...		[1] Serge Abiteboul. Querying Semi-Structured Data. In Foto N. Afrati and Phokion Kolaitis, editors, Database Theory - ICDT'97 ...

Figure 5: Abstract attributes

of a special data type (doc). We modify the type checking rules for instances of this data type[3] so that meaningful queries are possible, even if the schema is not known or only partially known. These modifications include: (1) incompatible data types are casted to compatible types, if necessary and (2) in heterogeneous data, non-existent attributes may be referenced. The concrete semantics are dependent on the type operation used. For example, a non-existent attribute referenced in a projection is ignored for each tuple it does not appear in. Therefore, the operation's result is again heterogeneous. Selection predicates referencing such attributes are evaluated to false. (3) Partially known schemata can be queried using path expressions and path variables. For example, the query

```
select r.name, ##z, r.cards
from   R2 r, r.price.{.*}z
where  ##z < 200
```

name	*single*	*double*
Hübner	195	235

name	*single*	*double*	*twin*	*cards*
Krüger	95	150	185	Visa Diners Club

Figure 6: Heterogeneous result

results in the heterogeneous instance shown in Figure 6. R2 denotes the instance from Figure 4. The elements of this set are typed as doc and are therefore subject to the type checking rules mentioned earlier. In the `from` clause, regular path expressions are used and the path variable z is declared. The regular path expression {.*} expands to all existing attributes below *price*. The appended (optional) label z denotes the corresponding variable name. The expression ##z dereferences the path variable z and is substituted by the complete paths (in this example R2.price.single, R2.price.double for the first tuple, as well as R2.price.single, R2.price.double, R2.price.twin for the second tuple). The attribute *cards* referenced in the `select` clause is ignored while processing the first tuple as there is no such attribute there. If one or more prices are string types, these prices would have to be converted to numeric values to evaluate the predicate ##z $<$ 200. If such a conversion is not possible, the predicate is evaluated to false.

name	*price*	
	single	*double*
Hübner	195	235
Krüger	95	150

Figure 7: Instance R3

The integration of structured and semi-structured data is realized by modelling these data as instances of different data types. For example, let the instance shown in Figure 7 be of type

```
set⟨doc(name:string,
        price:struct(single:integer,
                     double:integer))⟩.
```

The query

```
select r.price.single, r.address
from   R3 r
```

delivers the prices of all single rooms because the non-existent attribute *address* is directly ignored within a doc type. But the query

```
select r.price.twin, r.address
from   R3 r
```

leads to a runtime error because the price is modelled as a struct type where non-existent attributes may not be referenced.

[3]Essentially, we adopt the techniques (primarily Lorel's) used in existing query languages for querying semi-structured data.

The previously illustrated differences between operations on structured and semi-structured data can be adapted in order to be valid for other operations, too.

3.2 Extensions

In IRQL, there is one basic extension describing all known documents. We call this extension *d_world*. In order to avoid considering all these documents in every query, we introduce some possibilities to create further (e.g. smaller) extensions. Useful criteria include (1) information about how or whether a document can be reached via a particular path, (2) the language of documents, (3) the possibly named document types, and (4) the document's domain. We express each of these as an extension to the `from` clause. The extension (*coll*) of documents that are reachable starting from a given URL is determined by

⟨coll⟩ `REACHABLE FROM` ⟨URL⟩
[`DEPTH` ⟨value⟩] [`LOCAL`].

As options, a maximum path length (DEPTH parameter) can be specified or only local documents can be chosen. All documents in a given language can be determined by

⟨coll⟩ `IN LANGUAGE` ⟨lang⟩.

The clause

⟨coll⟩ `OF [NAMED] TYPE` ⟨tycon⟩

creates an extension of documents that meet the specified type condition. If the keyword `NAMED` is omitted, *tycon* stands for a type constructor like the `struct` or `doc` constructor, otherwise it is a label like postscript (PS) or XML that is modelled using the named type constructor mentioned in Section 2. Finally,

⟨coll⟩ `OF DOMAIN` ⟨domain⟩

determines the extension of all documents of a given domain (e.g. tourism).

3.3 Information Retrieval

In the following, we describe a further extension to SQL99; namely predicates that implement information retrieval techniques (e.g. content-based retrieval, soundex and proximity search, term weighting and ranking of query results). As our data model integrates structured and semi-structured data, these possibilities are applicable to both structured and semi-structured data.

3.3.1 Content-based Retrieval

We denote content-based retrieval by the clause

⟨attribute⟩ `CONTAINS` ⟨text⟩
[`ATLEAST` ⟨value⟩] [`ATMOST` ⟨value⟩]
[`WITH WEIGHT` ⟨value⟩]
[`CASE SENSITIVE`] [`SUBSTRING`]
[(⟨value⟩ | `NO`) `ERRORS`].

The following optional parameters exist: (1) ATLEAST, ATMOST specifies how often *text* must occur in *attribute*. If the number of occurrences of *text* is not within the specified bounds, the predicate is evaluated to false. If one or both of the parameters are omitted, no limit is assumed. (2) WITH WEIGHT specifies the weight of the query term. The default value is a weight of one. (3) By default, the search is case insensitive. This can be changed by specifying the CASE SENSITIVE parameter. (4) SUBSTRING specifies not only matching word bounds (e.g. spaces) but also searching for any occurrence of the given substring. (5) There is also a possibility for considering typing errors (see glimpse [12] how this can be realized) by specifying a value for the ERRORS parameter. By default, no typing errors are considered. The values of *text* can be either keywords or phrases. Furthermore, we support regular expressions (wildcards) here.

3.3.2 Soundex

The soundex algorithm allows the search for phonetically similar keywords or phrases. We denote the soundex search by

⟨attribute⟩ `SOUNDEX` ⟨text⟩
[`ATLEAST` ⟨value⟩] [`ATMOST` ⟨value ⟩].

The meaning of ATLEAST and ATMOST can be taken from Section 3.3.1. Further parameters mentioned there are not meaningful within the context of a soundex search.

3.3.3 Proximity

The next supported concept of content-based retrieval is the proximity search. Using a proximity search, it is possible to specify the distance between two keywords or phrases. The denotation is as follows:

⟨attribute⟩ `CONTAINS`
⟨text⟩ [`WITH WEIGHT` ⟨value⟩]
[⟨value⟩ ⟨unit⟩] `BEFORE` | `AFTER` ⟨text⟩ [`WITH WEIGHT` ⟨value⟩]
[`ATLEAST` ⟨value⟩] [`ATMOST` ⟨value⟩]
[`CASE SENSITIVE`] [`SUBSTRING`]
[(⟨value⟩ | `NO`) `ERRORS`].

Here, we only describe the new parameters. The others can be found in Section 3.3.1. The new parameter *unit* can be substituted by a type-dependent unit. For example, if d is a LaTeX document and a method exists to split this document into sections, then

```
d CONTAINS ``related work''
2 SECTIONS BEFORE ``conclusion''
```

is a valid predicate that checks whether d contains the phrase "related word" not more than 2 sections before the keyword "conclusion".

3.3.4 Ranking

We support ranking results by user-defined criteria. Syntactically, this is denoted by

RANK BY $f_0, \ldots, f_n$
[LIMIT TO $\langle$value$\rangle$]

The f_i denote user-defined functions that define the calculation of the retrieval status value (RSV). The RSV is an attribute that is introduced by the `rank by` clause and, after calculating this value, the result is sorted by RSV. Although we next plan to support the vector space model, we don't need to change our syntax if we implement a probabilistic model, as the following example demonstrates:

```
SELECT RSV, name
FROM hotels
RANK BY stars=5, beachdist=0
```

In this query we define a ranking using boolean predicates. These predicates are not evaluated to true or false, but define the "best" hotel. Thus, the retrieval status value of one is assigned to a five-stars-hotel directly situated at the beach. Using probabilistic methods, the other hotels are ranked accordingly.

The optional part of the `rank by` clause is used to limit the number of returned elements to *value*. By default, the number of elements is unlimited.

3.3.5 Compatibility with DBQLs and Information Retrieval

On the one hand, compatibility with SQL is achieved if there are no semi-structured data, and therefore, no `doc` type data in any of the extensions queried. In this case, any query that is a valid query within the supported subset of SQL99 is also a valid IRQL query and delivers the same result. On the other hand, compatibility with information retrieval expressions is achieved by transparently mapping these expressions to IRQL queries, as the following example demonstrates: Assume we are interested in a hotel near the beach. Using one of the search engines, we would probably enter

hotel `and` beach.

This expression is also accepted by IRQL and transparently mapped to[4]

```
select ⟨default_projection⟩
from   ⟨default_extension⟩
where  ⟨default_attribute⟩ CONTAINS "hotel" AND
       ⟨default_attribute⟩ CONTAINS "beach".
```

The default values can be adjusted within the IRQL shell. A good choice would be to use source,title as default_projection, d_world as default_extension, and complete_content as default_attribute. In this simple example the resulting query

```
select source,title
from   d_world
where  complete_content CONTAINS "hotel" AND
       complete_content CONTAINS "beach"
```

delivers the expected information.

4 Related Work

IRQL integrates concepts from database query languages, query languages for semi-structured data, and information retrieval. In this section, we discuss some query language proposals that are related to these areas (i.e. we focus primarily on query languages for semi-structured and web data) and compare them with our approach. We do not discuss XML query languages here because these query languages don't consider the integration of information retrieval techniques and, in principle, XML data can also be queried using some of the following query languages.

Information Retrieval

Most of the existing search engines (e.g. Altavista or Infoseek) use information retrieval techniques to search for particular (web) documents. Users describe their search criteria by entering keywords, phrases, or combinations using boolean operators. However, these search engines don't normally take the document's structure into account, and only selections are primarily supported. Features that are typical for DBMSs (and also for IRQL) like restructuring (e.g. projection) or joins are still missing.

Access to the document structure is supported by freeWAIS-sf [18, 17]. Documents can be partitioned into a set of attribute-value pairs. The set of possible attributes is defined by the document type. One of the pre-defined types is HTML and further types can be defined by the user.

[4]For simplicity, we ignore the ranking.

Queries are also limited to selections, but parts of the document can be queried via attribute names. Users can describe their search criteria using free text, phrases, wildcards, soundex and proximity expressions, as well as combinations of these using boolean operators. Comparisons of numeric values are supported, too. Besides the access to the document's structure in IRQL, we also allow for the restructuring of data.

Query languages for semi-structured data

In the past, the disadvantages of exclusively using information retrieval techniques to query semi-structured data has been pointed out by several authors. As a result, there are numerous proposals and implementations that also integrate concepts of database query languages. A survey can be found in [10].

Lorel [2] is the query language of the Lore system [16]. Syntactically, Lorel is based on OQL. Semi-structured data are supported by using OEM graphs as the data model and by extending OQL with appropriate features. These extensions include (a) implicit type casts (type coercion) and (b) regular path expressions. Path expressions and path variables support queries on unknown or partially known schemas and on the schema itself. Implicit type casts, called type coercion, address the heterogeneity of semi-structured data. Some variants of content-based retrieval (e.g. soundex search) are provided by corresponding predicates. The drawback of this data model is the missing support of ordered collections[5]. As a consequence, for example, no ranking criteria can be specified at the language level.

WebSQL [6] is based on the relational model and supports an SQL-like query language. Additional features of WebSQL include dynamic creation of extensions based on content and link structure of web documents, and path expressions. HTML tags are treated as attribute names in order to access parts of web data. The restructuring of HTML pages is not supported.

Compared with OEM graphs, WebOQL [5] uses an improved data model. Hypertrees facilitate the modelling of nested structures and further support ordered collections. Using the "web" as a data type is the key to providing a number of operations for restructuring data. The query language is based on OQL and provides some further possibilities, e.g. the creation of query results. Content-based retrieval is supported by a grep operator. In our approach, we support further means of information retrieval, such as term weighting and ranking.

UnQL [7] uses a graph-based data model. The query language supports selection, projection, join, and grouping, as well as path expressions. Both modelling of ordered data and content-based retrieval are not supported.

W3QL [13] is the SQL-like query language of W3QS. The focus of this query language's development is the reuse of available tools. For example, predicates that realize content-based retrieval are implemented using external tools. Both nesting of queries and restructuring of data are not supported.

The aim of the development of the Strudel query language StruQL [9] is to provide means for restructuring existing data. In StruQL, semi-structured data are modelled as an OEM graph. The supported query operations include navigation using path expressions, projection, and selection as well as operations for restructuring of existing graphs and for creation of new graphs. In principle, user-defined predicates could be used to implement content-based retrieval.

WebLog [14] is based on SchemaLog and supports access to the structure of documents and content-based retrieval by using built-in or user-defined predicates. The restructuring of data is supported, too. As in IRQL, it is possible to express recursive queries.

In WQL [15], both the web and the structure of the individual documents are modelled. The query language implements projection, selection, sorting, and grouping. Content-based retrieval and querying the structure of web documents are supported. From our point of view, missing features are dynamically creating extensions, nesting, and restructuring.

Data base query languages

Apart from the query languages for semi-structured data mentioned in this section, there are also proposals to extend DBMSs. For example, the SQL/MM proposal [19] defines a data type *full text* whose operations support content-based retrieval for those data that are stored using this data type. Different implementations are made available by commercial companies in the form of text extenders, data blades, and so on.

Figure 8 summarizes the features of the query languages mentioned in this section. We use the following notation to assess the features: $+$ supported, $\pm$ supported with limitations, $\mp$ supported to some extent, and $-$ not supported.

5 Conclusion and Future Work

In this paper we present the basic ideas and concepts behind the Information Retrieval Query Language (IRQL) that is used in the GETESS project by a dialogue system with a natural-language-like user interface to query the summaries of linguistically analysed web documents. Our data model distinguishes structured and semi-structured heterogeneous data based on type information and supports an abstraction of attribute names. IRQL integrates concepts of database

[5]Recently, Lorel's data model has been extended to support XML data [11]. Therefore, ordered subelements can now be modelled. To the best of our knowledge, it is not possible to express user-defined rankings in Lorel like it is in IRQL.

	Lorel	WebOQL	WebSQL	UnQL	W3QL
data model	OEM graph	hyper tree	relational	labeled graph	labeled graph
language style	OQL	OQL	SQL	structural recursion	SQL
path expressions	+	+	+	+	+
type coercion	+	−	−	−	−
updates	+	−	−	+	−
strong typing	−	−	−	−	−
recursion	−	−	−	+	−
content bases retr.	∓	∓	∓	−	∓
ranking	−	−	−	−	−
term weighting	−	−	−	−	−
ordered collections	−	+	−	−	−
linking different docs.	+	+	+	+	+
querying the schema	+	−	−	+	−

	StruQL	WebDB	WebLog	freeWAIS-sf	IRQL
data model	labeled graph	object-relational	relational	relational	object-relational
language style	Datalog	SQL	Datalog	boolean retrieval	SQL
path expressions	+	−	−	−	+
type coercion	−	−	−	∓	+
updates	−	−	−	−	−
strong typing	−	−	−	−	+
recursion	+	−	+	−	+
content based retr.	−	±	∓	+	+
ranking	−	−	−	+	+
term weighting	−	−	−	−	+
ordered collections	−	+	−	+	+
linking different docs.	+	+	+	−	+
querying the schema	−	−	−	−	+

Figure 8: features of some query languages

query languages, query languages for semi-structured data, and information retrieval techniques. The starting point of IRQL development is SQL99, which we extend with new clauses to integrate information retrieval techniques. Furthermore, we modify the type system to support semi-structured heterogeneous data. IRQL is built on top of existing systems such as object-relational DBMSs, relational DBMSs, or full-text DBMSs. The current prototype implementation has been built on top of DB2 and its text extender.

To the best of our knowledge, there is no similar proposal that attempts to integrate features of these three areas.

Future works include the complete formalization of the query language and the development of an algebra.

Acknowledgements

The GETESS project is funded by the German Ministry of Education and Research (Bundesministerium für Bildung und Forschung; BMBF) under grant number 01IN802.

References

[1] S. Abiteboul. Querying Semi-Structured Data. In F. N. Afrati and P. Kolaitis, editors, *Database Theory - ICDT '97, 6th International Conference*, volume 1186 of *Lecture Notes in Computer Science*, pages 1–18, Delphi, Greece, Jan. 1997. Springer Verlag.

[2] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. L. Wiener. The Lorel Query Language for Semistructured Data. *International Journal on Digital Libraries*, 1(1):68–88, 1997.

[3] *ANSI/ISO/IEC International Standard (IS) Database Language SQL – Part 1: SQL/Framework, ISO/IEC 9075-1:1999 (E)*, Sept. 1999.

[4] *ANSI/ISO/IEC International Standard (IS) Database Language SQL – Part 2: Foundation (SQL/Foundation), ISO/IEC 9075-2:1999 (E)*, Sept. 1999.

[5] G. O. Arocena and A. O. Mendelzon. WebOQL: Restructuring Documents, Databases, and Webs. In *Proceedings of the Fourteenth International Conference on Data Engineering*, pages 24–33, Orlando, Florida, USA, Feb. 1998. IEEE Computer Society Press.

[6] G. O. Arocena, A. O. Mendelzon, and G. A. Mihaila. Applications of a Web Query Language. In *Proceedings of the 6th International WWW Conference*, Santa Clara, California, 1997.

[7] P. Buneman, S. B. Davidson, G. G. Hillebrand, and D. Suciu. A Query Language and Optimization Techniques for Unstructured Data. In H. V. Jagadish and I. S. Mumick, editors, *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, volume 25(2) of *SIG-*

MOD Record, pages 505–516, Montreal, Quebec, Canada, June 1996.

[8] S. Cluet. Modeling and Querying Semi-Structured Data. In M. T. Pazienza, editor, *Information Extraction: A Multidisciplinary Approach to an Emerging Information Technology, International Summer School, SCIE-97*, volume 1299 of *Lecture Notes in Computer Science*, pages 192–213, Frascati, Italy, 1997. Springer Verlag.

[9] M. F. Fernandez, D. Floresu, A. Y. Levy, and D. Suciu. A Query Language for a Web-Site Management System. In *SIGMOD Record*, volume 26(3), pages 4–11, 1997.

[10] D. Florescu, A. Y. Levy, and A. O. Mendelzon. Database Techniques for the World-Wide Web: A Survey. In *SIGMOD Record*, volume 27(3), pages 59–74, 1998.

[11] R. Goldman, J. McHugh, and J. Widom. From Semistructured Data to XML: Migrating the Lore Data Model and Query Language. In S. Cluet and T. Milo, editors, *ACM SIGMOD Workshop on The Web and Databases (WebDB'99)*, pages 25–30, Philadelphia, Pennsylvania, USA, June 1999. INRIA. Informal Proceedings.

[12] Harvest. http://harvest.transarc.com.

[13] D. Konopnicki and O. Shmueli. W3QS: A Query System for the World-Wide Web. In U. Dayal, P. M. D. Gray, and S. Nishio, editors, *VLDB'95, Proceedings of 21th International Conference on Very Large Data Bases*, pages 54–65, Zurich, Switzerland, Sept. 1995. Morgan Kaufmann Publishers.

[14] L. V. S. Lakshmanan, F. Sadri, and I. N. Subramanian. A Declarative Language for Querying and Restructuring the WEB. In *Proceedings: Sixth International Workshop on Research Issues in Data Engineering — Interoperability of Nontraditional Database Systems*, IEEE-CS 1996, pages 12–21, New Orleans, Louisiana, USA, Feb. 1996.

[15] W.-S. Li, J. Shim, K. S. Candan, and Y. Hara. WebDB: A Web Query System and its Modeling, Language, and Implementation. In *Proceedings of the IEEE Forum on Research and Technology Advances in Digital Libraries, IEEE ADL'98*, pages 216–227, Santa Barbara, CA, USA, Apr. 1998.

[16] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. Lore: A Database Management System for Semistructured Data. In *SIGMOD-Record*, volume 26(3), pages 54–66, Sept. 1997.

[17] U. Pfeifer. *freeWAIS-sf*. Universität Dortmund, Oct. 1995. Manual of the enhanced freeWAIS distribution.

[18] U. Pfeifer, N. Fuhr, and T. Huynh. Searching Structured Documents with the Enhanced Retrieval Functionality of freeWAIS-sf and SFgate. In *Proceedings of The Third International World-Wide Web Conference*, Darmstadt, Germany, Apr. 1995.

[19] *ISO Working Draft, SQL Multimedia and Application Packages (SQL/MM), Part 2: Full-Text*, Sept. 1995.

[20] S. Staab, C. Braun, I. Bruder, A. Düsterhöft, A. Heuer, M. Klettke, G. Neumann, B. Prager, J. Pretzel, H.-P. Schnurr, R. Studer, H. Uszkoreit, and B. Wrenger. A System for Facilitating and Enhancing Web Search. In *IWANN '99 — Proceedings of International Working Conference on Artificial and Natural Neural Networks*, Alicante, ES, 1999.

[21] S. Staab, C. Braun, I. Bruder, A. Düsterhöft, A. Heuer, M. Klettke, G. Neumann, B. Prager, J. Pretzel, H.-P. Schnurr, R. Studer, H. Uszkoreit, and B. Wrenger. GETESS — Searching the Web Exploiting German Texts. In M. Klusch, O. Shehory, and G. Weiss, editors, *Cooperative Information Agents III, Proceedings 3rd International Workshop CIA-99*, volume 1652. Springer Verlag, July 1999.