The Light-Weight Semantic Web: Integrating Information Extraction and Information Retrieval for Heterogeneous Environments

Jens Graupmann
Max-Planck-Institut für Informatik
Saarbrücken, Germany
graupman@mpi-inf.mpg.de

Ralf Schenkel
Max-Planck-Institut für Informatik
Saarbrücken, Germany
schenkel@mpi-inf.mpg.de

ABSTRACT

Today's Web, large intranets and even the documents collected by a single user are enormous sources of distributed, heterogeneous information that cannot be easily mastered. Syntactical and semantical differences as well as missing semantic annotations make effective query evaluation on such corpora a hard task. The Semantic Web aims at providing a standard for semantic annotations, but has not yet made large progress in the real world.
This paper presents a light-weight version of the Semantic Web. We advocate the use of Information Extraction tools to automatically detect and annotate important classes of information that are frequently used in queries, like locations and dates. We propose a query language that can exploit the extra annotations and allows novel range and join conditions.

1. INTRODUCTION

1.1 Motivation

The World Wide Web of today is a steadily growing collection of information. Unlike the early days when HTML pages dominated the Web, information is available in a plethora of different formats, with PDF and Word documents, the diversity of XML documents, form interfaces to structured databases, images and even multimedia data as prominent, but by far not exhaustive examples. A similar situation arises in intranets of large companies with information stored in different formats and in different places, and even a user's personal data on her own computer forms a heterogeneous collection of documents in diverse formats.

The heterogeneous and distributed nature of information opens two difficult research problems that must be answered in order to effectively answer queries in such a setting:

- Information is stored in semantically different forms and most often without any semantic annotation at all. How can a search engine map a query (e.g., asking for cheap books about XML) to documents where information is available, but not explicitly annotated?
- The result of a query may be distributed across several documents that may or may not be explicitly connected with a hyperlink. How can a search engine identify such a distributed result?

The Semantic Web [4, 14] aims at solving the first problem in the Web environment with semantical annotations that facilitate intelligent query processing using inference mechanisms and large handcrafted ontologies. However, while the Semantic Web standards are available today, hardly any page in the Web provides explicit annotations, and it is not foreseeable that annotations will become more popular in the near future. It should be questioned anyway whether an approach based on manual annotations is feasible at such large scale as the Web. The second problem has just recently got the attention of the research community with the upcoming graph-based query models [5, 19], but it remains unclear if and how these techniques can be applied when explicit links are not available (e.g., when two pages deal with the same topic, but are not linked).

To overcome the problems imposed by the heterogeneity of data, we propose a light-weight version of the Semantic Web. Instead of advocating rich, manual annotations like the real Semantic Web, we use automatic Information Extraction tools to detect and annotate important classes of information that are frequently used in queries, like locations, dates, and persons. Documents that are found on the Web are automatically tagged with such meta information. Additionally, we propose a query language that can exploit the extra annotations for more precise queries with novel range and join conditions.

1.2 How IE Helps - an Application Scenario

We show the benefit from integrating Information Extraction techniques and IR using an application scenario familiar to most researchers. Let's assume that a researcher wants to plan her trip to Salvador, Brazil in order to attend SIGIR 2005. Some queries (or, to be more precise, *information needs*) that may arise in this context are the following:

(I1) What are cheap, but not shabby hotels (i.e., with prices between $40 and $80 per night) within 5km distance of the Pestana Bahia Hotel in Salvador (where the conference takes place)?

(I2) Which rock concerts take place within a distance of 50km of Salvador in the week after SIGIR (from August 20 to August 26)?

(I3) Which researchers that do research about XML have a paper at SIGIR 2005, so it's likely that I'll meet them?

Such queries should be evaluated on the Web, possibly an intranet (e.g., containing travel regulations), and local documents (like mails or downloaded documents) on the user's computer. The documents involved can be rather structured (like flight or train schedules), unstructured (like information about hotel descriptions and cultural events in plain HTML files), or semi-structured (like travel guides in XML or PDF documents).

Keyword-based queries, the smallest common denominator used by all of today's Web search engines, are by no means sufficiently expressive. As an example, consider information need (I1): It is not enough to simply find a hotel (which could be done with a keyword-based query), but its distance to the conference hotel has to be computed and its price has to be in a certain range. For (I2), the date of an event has to be detected and compared to a given range. At first sight it seems that such complex information needs can be satisfied only by complex and highly specialized question answering systems that incorporate Natural Language Processing (NLP) techniques.

Luckily, large classes of information needs can be answered with much less effort. The key issue for our examples is finding information of certain classes (like persons, locations, prices, and dates) in documents and using them when answering the queries. We propose to use well-established Information Extraction tools for this task, together with additional external knowledge about certain classes of information (like coordinates of locations, exchange rates for currencies etc.). Using type-specific wrappers, documents encountered by the search engine are first fed into an information extraction engine that automatically annotates a predefined set of information classes before the extended document is added to the index. Figure 1 shows an overview of our system's architecture. Note that the system can be easily extended with new wrappers and new information classes.

Once we have additional (light-weight) annotations, we can exploit them to identify relevant answers to queries; details of our query language are presented in Section 4. For our example (I1), once a document describing a hotel is found, the corresponding price is extracted (using the annotation for prices) and compared to the price in the query (possibly after converting one of them). Additionally, its location is extracted and compared to the location in the query by converting both locations to their corresponding coordinates using external information and comparing the coordinates).

If the answer to a query is distributed over several documents, annotations alone are not enough. However, they can help identifying pages that are "semantically" connected, but don't have an explicit link. As an example, consider (I3): We can get all people with a paper at SIGIR 2005 from the conference's homepage, but that does not include links to the researchers' homepages where we would expect information about their research areas. If we annotate persons (which basically corresponds to names) and do some data cleaning, we can detect that another page contains the same person as the paper list and therefore add a 'virtual' link between the two pages, enabling us to answer the query.

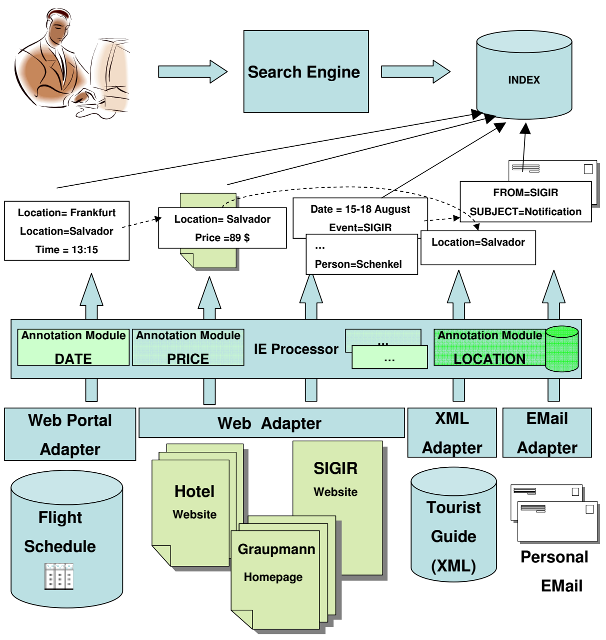

Figure 1: Architecture of an integrated IE-IR system

Note that any automatic annotation must include an inherent risk of missing or erroneous annotations. Even though this risk is noticeably small for state-of-the-art annotation engines, we take it into account by maintaining a confidence level for each annotation that corresponds to the estimated probability that the annotation is correct.

1.3 Related Work

Information Extraction has been an active research area in Natural Language Processing for quite some time ([8, 18]). However, despite all this work, the integration of IE with classical information retrieval is still not very deep ([2]). Additionally, data integration based on NLP techniques has not attracted much attention; only recently some heavy-weight approaches have been proposed [23, 27].

Information extraction from text and HTML data is an area with intensive work. The approaches mostly follow a rule-based paradigm [3, 10, 16, 25], or employ learning techniques and/or linguistic methods [2, 9, 11, 13]. But actually using automatically converted and "semantically" enhanced Web data in a search engine has not been pursued in the literature on information extraction.

Ranked retrieval over graph-structured data has been pursued in [5, 19]. The graphs studied there are derived from foreign-key relationships in relational databases and are quite different from the settings of the current paper. Queries in this prior work were limited to keyword search on attributes scattered across different tables. Chakrabarti [7] names vague search over graph structures as one of the major open challenges in areas where database and information retrieval technologies meet.

Integrating information from heterogeneous and usually distributed sources is the primary task of federated information systems (see, e.g., [26]) that apply system-specific wrappers and an integrated global schema. Unlike these systems, we propose a ad-hoc, light-weight integration of heterogeneous data sources that focuses on selected information types. Furthermore, we do not need strict correctness guarantees like distributed transactions, as we only provide unified read-only access and our snapshot of the Web data is not time-synchronous anyway.

1.4 Outline of the Paper

This paper shows the benefits of integrating information extraction techniques with information retrieval as a form of light-weight data integration. Furthermore we show how these technique facilitate the integration of distributed heterogeneous data sources and information system not only by providing a unified query interface but also by the integration of and connection by "virtual links".

We start with a short summary on the background of information extraction techniques in Section 2. In Section 3 we introduce a data model for documents that captures annotations. Section 4 presents our query language that can exploit the extra annotations and provides novel range and join conditions. Section 5 gives details on the necessary preprocessing and data cleaning, and Section 6 shows how new annotation types can be integrated.

The ideas and concepts presented in this paper have been implemented in and evaluated with the SphereSearch Engine [17], but can be easily adapted to other engines. The original SphereSearch paper [17] presented the big picture of the engine and its evaluation; this paper complements it with a thorough discussion of the use of information extraction techniques.

2. INFORMATION EXTRACTION BASICS

Information Extraction (IE) is a subfield of Natural Language Processing (NLP), which is itself a subfield of computer science and computer linguistics that deals with the problems inherent in the processing and manipulation of natural language. Information Extraction consists (according to the leading forum for this research, the Message Understanding Conferences [12, 22]) of five main types (also called *IE tasks*):

- *Named Entity recognition (NE)* finds and classifies names, places, etc.
- *Coreference resolution (CO)* identifies identity relations between entities in texts.
- *Template Element construction (TE)* adds descriptive information to NE results (using CO).
- *Template Relation construction (TR)* finds relations between TE entities.
- *Scenario Template production (ST)* fits TE and TR results into specified event scenarios.

According to Gaizauskas et al. [15], Information Extraction (IE) techniques "process a document to identify prespecified entities and the relationships between them and then fill in a [...] 'template' with the identified information", while Information Retrieval (IR) aims at "identifying documents from a larger collections which are (hopefully) relevant with respect to some query". They state that the combination of IE and IR "has the potential to create a powerful tool in text processing".

```
The <company>Pelican Hotel</company> in
<location>Salvador</location>, operated by
<person>Roberto Cardoso</person>, offers
comfortable rooms starting at
<price>$100</price> a night, including
breakfast. Please check in before <time>7pm</time>.
```

Figure 2: Result of annotating a hotel description

In this work we concentrate on using Named Entity Recognition to identify predefined categories such as the names of persons, organizations, locations, expressions of times, quantities, monetary values, percentages, etc. in texts. Figure 2 shows an example where the annotations created by Named Entity Recognition are integrated as XML-like tags into the existing description of a hotel.

One of the most popular tools for Natural Language Processing is *GATE* [12, 11] (General Architecture for Text Engineering) and its integrated Information Extraction system ANNIE. ANNIE (a Nearly-New Information Extraction System) consists of different modules that form a pipeline. After some basic steps like part-of-speech tagging (e.g., for noun phrases, temporal adverbial phrases etc) and lookups in lists by the Gazetteer module (e.g. for locations or currency symbols) the Sematic tagger is executed. The semantic tagger uses rules which exploit annotations assigned in earlier steps to produce annotated entities. It is based on the JAPE Transducer, that provides finite state transduction over annotations based on regular expressions. ANNIE can be easily extended by providing additional dictionaries and rules.

Although tools like GATE have been available for some years they have only recently been integrated in the area of information retrieval which is still dominated by statistical and probabilistical measures.

Another important class of algorithms for information extraction uses probabilistic and statistical methods, most notably Hidden Markov Fields (e.g., [24]). Such algorithms could be easily integrated into our approach, but we have concentrated on rule-based systems like GATE so far.

3. COMBINING INFORMATION EXTRACTION AND RETRIEVAL

In this section we propose a foundation for a search engine that integrates IR and IE techniques. We introduce versatile data model, a query language that can exploit annotations, and a scoring model.

3.1 Data Model

Since a classical bag-of-words representation ignores the structure of a document, but annotations and links are bound to specific parts, a search engine that combines IE and IR has to use a more appropriate data model. We propose a graph-based document model as it preserves the original document structure and can easily represent annotations as well as links between documents.

We consider the whole collection of documents known to the search engine as a single, huge graph. Nodes represent substructures of documents (like elements in XML documents), each node stores its textual content and has assigned at least one type (e.g., denoting if its an XML element or a textual node). Edges connect two nodes and are either *intra-document*, corresponding to the internal document structure, or *inter-document*, representing links between documents. The upper part of Figure 3 shows an example for a collection of two documents. In this model, a potential result of a query is a subgraph of the whole graph. Such a potential result can consist of fragments belonging to different documents.

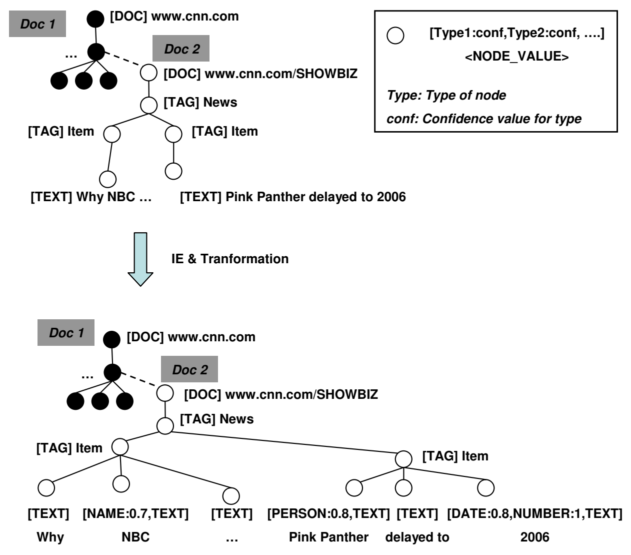

Figure 3: Document representations before (top) and after (bottom) annotation

IE and other techniques can be seamlessly integrated as the results of annotation steps (e.g., Named Entity Recognition) are simply added to the internal graph-based representation as types. If an annotation applies only to a part of a node's value, the node is split up into sibling nodes. Figure 3 shows the graph-based representation of two linked documents before and after annotation. The leaf text nodes are split up into sibling nodes with different annotated types, but the original information is still retained.

As most annotation techniques are to a certain extent error-prone, information added by annotation steps includes a certain amount of vagueness. We reflect this in our model by attaching a confidence weight to each annotation that reflects the probability that this annotation is correct. In Figure 3 this confidence weight is shown after each type that has a confidence value lower than one.

Formally, for a collection $\mathcal{X} = (\mathcal{D}, \mathcal{L})$ of documents $\mathcal{D}$ together with a set $\mathcal{L}$ of (href, Xpointer, XLink, virtual, ...) links between their elements, we maintain the element-level graph $G_E(\mathcal{X}) = (V_E(\mathcal{X}), E_E(\mathcal{X}))$ that has the union of the elements of all documents as nodes and undirected edges that correspond to parent-child edges and links. Note that we intentionally dropped the direction of edges from the model as it is easier to phrase queries if the user does not have to think about the direction of links.

For each element $x \in V_E(\mathcal{X})$, val(x) denotes the node's value, type(x) denotes the set of the node's types and conf(x, t) denotes the confidence value (a nonnegative weight between 0 and 1) for type t of node x.

For an XML or HTML document the type of a node may be *Tag*, *Processing Instruction* or *Text*; for an email it may be *Sender*, *Subject*, *Message* or *Text*. The confidence value for an element's base type provided by the original document is always 1, for an annotation it reflects the expected correctness of this annotation.

A node can have multiple types for different reasons. First, different annotation steps could add different annotations to the same node, e.g., *person* and *researcher*. Second, types can be derived from other types, e.g., the base type of the type *person* is always the type *name*; the base type of *name* is *string*.

Two nodes with the same type t are compared with a type-specific comparator function c_t. As an example, to compare two nodes of type *location*, a location comparator is used that compares the geographical position of both locations (that is available from external data sources). If two nodes with differing types are compared, the comparator of the least common basetype of both types is applied. As an example, a *location* and a *person* node are compared to each other using the comparator for *string*, which could apply string edit distance to compare the values of the nodes.

The choice for the weight conf(x, t) for a type assignment that reflects the probability of its correctness can be determined in different ways. If an annotation technique, like algorithms based on Hidden Markov Models [24], provides a confidence value by itself, this should be used. Unfortunately, many techniques do not provide a confidence, like most rule-based approaches, e.g. the Jape-Transducer used in GATE/ANNIE. In this case, a useful value can be the statistical correctness of the technique based on a fixed traininig set. For ANNIE experiments have been published showing that 89% of all person annotations are correct [21]. Thus a value for 0.89 of conf$(x, person)$ will be assigned to any person node added by ANNIE's annotation component. If the same annotation is generated by different tools, the confidence for this type can be computed by building the average or the maximum of the different confidence values.

Each edge e is assigned a nonnegative weight weight(e), which is 1 for parent-child edges contained in the original document and has a fixed value λ for standard links. For virtual links (that are introduced in Section 4) weight(e) reflects the strength of the connection. The distance function $\delta_{\mathcal{X}}(x, y)$ takes two elements as input and computes the weight of a shortest path (i.e., a path from x to y where the sum of edge weights is minimal) in $G_E(\mathcal{X})$ between them.

3.2 Queries

In this section we propose a query language that can take advantage of the link structure of the document collection and utilize additional annotations. A key concept of the query language is to group query conditions that refer to the same entity, i.e., an (intuitively defined) object of the real world.

As a simple example, consider the query

```
(hotel, location=Salvador)
```

that asks for hotels in Salvador. This query consists of a single group and exploits the annotation of locations by using an exact-match condition on the location. If two or more groups are used in a query, they are labeled like in the following example:

```
A(hotel, location=Salvador)
B("air condition" shower telephone)
```

Here, the second group states properties of rooms. Each group is evaluated independently. The result for the whole query are subresults that are compact, i.e., whose distance in the graph is small, so good results would be hotels that have rooms with the requested properties. The rationale behind this is that elements that are close to each other are, with a high probability, related to each other as well. While this works only for elements that are linked in the original documents, specifying an additional join condition allows relating elements from unconnected documents. As an example, the following query phrases the information need (I3) from the introduction:

```
A(researcher, XML)
B(SIGIR 2005 paper)
A.person=B.person
```

Formally, a query $S = (Q, J)$ consists of a set $Q = \{Q_1, \ldots, Q_g\}$ of one or more nonempty *query groups* and a (possibly empty) set $J = \{J_1, \ldots J_m\}$ of join conditions. Each query group Q_i consists of a (possibly empty) set of keyword conditions $t_1^i \ldots t_{k_i}^i$ and a (possibly empty) set of annotation-based conditions that are detailed in the following section.

Additionally, exact-match or similarity joins can be specified between query groups. A join has the form $Q_i.v = Q_j.w$ for exact-match joins and $Q_i.v \sim Q_j.w$ for similarity joins, where Q_i,Q_j are query groups and v,w are annotation types like date, location etc.

Please notice that this internal representation of the query should never be shown to the user. Instead, a query interface for end users could be fully graphical or provide another kind of query language that is transformed into this representation later on.

3.3 Scoring

As a result of a query is a subgraph, different results not only differ in their scores for matching nodes but also in the size of the subgraph. More compact subgraphs are preferable as the contained information is more strongly connected and therefore, with a high probability, more highly related.

A score s of a potential answer N to a query Q is a weighted combination of the scores of the nodes n_i that match conditions of Q and the overall compactness C of the potential answer:

$$s(N, Q) = \beta C(N) + (1 - \beta) \sum_{i=1}^{g} s(n_i)$$

Here, the weight β adjusts the ratio of compactness to scores, and the compactness C is the inverse of the sum of weights of edges in N. The score for a node n_i is defined as the sum of the scores for all matching subconditions c of Q

$$s(n_i) = \sum_{c \in S} s_c(n_i)$$

The score for a subcondition depends on the condition type. For a simple keyword query it is a standard IR-type tf/idf based weight. For other subqueries the score computation is described in the next section.

3.4 Data model, queries, scoring – summarized

The following facts summarize our data model:

- The whole data collection is internally represented as one huge graph structure
- Every element (text, annotation) is a node with (possibly multiple) types and corresponding weights measuring the confidence for this type assignment.
- The results of a query is a subgraph, possibly consisting of connected nodes of different documents.
- Queries consist of groups that are independently evaluated.
- The score for a potential result is based on the scores of subresults and the compactness of the result.

4. ANNOTATION-AWARE QUERIES

In this section we show how to efficiently use annotations for information retrieval. We assume that all annotated information data has been converted to a standard representation and is therefore readily available for evaluation; we show details of the necessary preprocessing in the following section. We introduce different kinds of annotation-aware queries, sketch their evaluation and describe their scoring.

4.1 Exact-Match Annotation-Based Queries

The simplest form of an annotation-aware query is an exact-match annotation-based query. It can help disambiguating query terms that are ambigous. As an example, consider the query for the politician Condoleezza Rice. A simple keyword query `(american, politician, rice)` leads to poor precision as many documents about rice farms in the USA are found. Disambiguating the term `rice` by denoting that it should describe a person, i.e., rephrasing the same query as `(person=rice, politician, american)` drastically increases precision. Further examples are `date=1980` or `money_amount=10999`.

Formally, an exact-match annotation-aware query condition (of a query group) has the form $c_i = v_i$ with an annotation type c_i and a required value v_i. A node n_i matches the condition $c_i = v_i$ if value$(n_i) = v_i$ and $c_i \in$ type(n_i). As this is an exact-match condition, the score for the match is defined as the confidence of the annotation:

$$s_{cv}(n_i) = \text{conf}(n_i, c_i)$$

4.2 Range Queries

For any numeric annotation type, range queries are obviously applicable and useful. A query to find cheap, but not shabby hotels, i.e., hotels with a price between \$50 and \$80, in Salvador would read

```
(hotel location=Salvador $50<=price<=$80)
```

To find rock concerts between August 20 and August 26, a query like `(rock concert 08-20-05<=date<=08-26-05)` could be used. Ranges can also be open on one side, e.g., `price<$80`. The implementation for such numerical range conditions is straightforward. If the system was based on a relational database, a range condition could be directly transformed to an SQL-query with a range condition on a numerical column containing all money amounts or dates.

Even for non-numerical annotations like locations range queries are useful as long as the annotated values can be mapped to a numerical domain. As an example, locations can be mapped to their geographical coordinates using additional external information, we show details in the following section. Once we have this mapping, we can answer range queries on locations, like the following for information need (I1) from the introduction:

```
(hotel location-Salvador<=5km $50<=price<=$80)
```

This location-based range query implies that Salvador is a location and asks for locations within a distance of 5km of Salvador. A query can also specify a range that is limited by two fixed locations, e.g.:

```
 (rock concert Salvador<location<Rio)
```

Figure 4 visualizes the query by showing the geographical rectangle spanned by the coordinates of Salvador and Rio; to qualify as a result, a location must be within this rectangle.

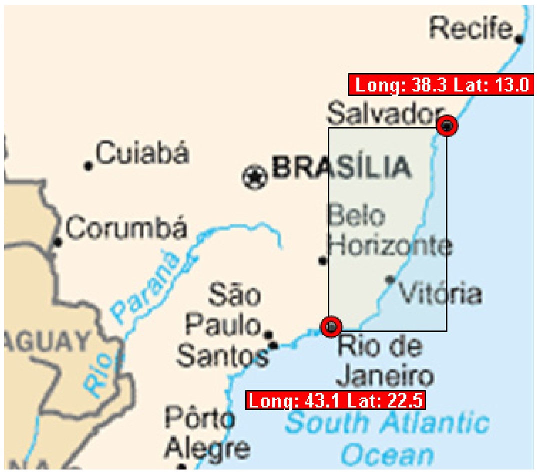

Figure 4: A range query with two fixed locations

In an implementation, this query could be evaluated as follows:

1. The coordinates for Salvador (38.3 longitude, 13.0 latitude) and Rio (43.1 longitude, 22.5 latitude) are looked up.

2. Locations in the geo-database (see Section 5.3) are looked up that fulfill the conditions 38.3<longitude<43.1 and 13.0<latitude<22.5.

3. Pages in the index are looked up containing 'rock concert' and one of the locations determined in the previous step.

Formally, range queries exist in three variations:

- *open range queries* where only one boundary is fixed, like in `date=>1980`, `money_amount=>10000`

- *closed range queries* where both boundaries are fixed, like in `1980<date<=1985`

- *center queries* where the range is an equi-sized interval around a center, like in `date-1980<=5`

A node n_i matches a condition $c_i = v_i$ if value$(n_i) \in$ Set(v_i) and $c_i \in$ type(n_i) =. Here, Set(v_i) is the (possibly infinite) set containing all values contained in the range v_i of type c_i. The score is then computed in the same manner as for exact-match queries:

$$s_{cv}(n_i) = conf(n_i, c_i)$$

4.3 Similarity queries

Similarity queries of the form `type=`$\sim$ `value` can be seen as implicit center queries `type-value<=`ϵ with a query-specific center value and a predefined range ϵ. In contrast to ordinary' center queries, matches with a value 'nearer' to the specified center are preferable and should therefore be ranked higher.

The main problem is to determine an appropriate value for ϵ and a damping function that provides a score that decreases with increasing distance to the center value. Both may be application specific as the following examples demonstrate: For the query `(gotic church date`$\sim$`1400)`, the range 1300 <date< 1400 may be appropriate, but for a query `(rock concert date`$\sim$`06-01-2005)`, a range of 100 years is obviously inadequate. As it is almost impossible to determine the appropriate similarity range for each query only based on the query itself, an adjustment based on user feedback seems to be best suited. As values for each named entity can be shown in the result list, the user can provide feedback without inspecting the documents itself. Such a feedback can be type-specific, e.g., too late/too early for dates, too far for locations, etc.

Formally, for each datatype dt a similarity function

$$\text{sim}_{dt}(\text{value}, \text{center}, \epsilon)$$

computes the degree of matching of a node with a certain value to a center query with the given center and ϵ. A node n_i matches a condition $c_i =\sim v_i$ if value$(n_i) \in$ Set$(\sim v_i)$ and $c_i \in$ type(n_i). Here, Set$(\sim v_i)$ is the (possibly infinite) set containing all values contained in the range $[v_i - \epsilon, v_i + \epsilon]$ of the domain of c_i. The score for a similarity condition sc is computed as follows:

$$s_{sc}(n_i) = \text{conf}(n_i, c_i) * \text{sim}_{c_i}(\text{value}(n_i), v_i, \epsilon)$$

A simple implementation of the similarity function returning a score reciprocal to the distance to the optimal value is the following:

$$sim(\text{value}, \text{center}, \epsilon) = \frac{1}{|\text{center} - \text{value}| + 1}$$

4.4 Link/Join queries

The most powerful variant of annotation-aware queries are join queries. As an example, consider again information need (I3) from the introduction. Based on the annotation of persons, this query can be formulated similar to an SQL-query in a relational database setting using the annotation type as join attribute:

```
A(researcher, XML)
B(SIGIR 2005 paper)
A.person=B.person
```

This query tries to match the terms `researcher` and `XML` on page A, the terms `SIGIR`, `2005` and `paper` on page B and returns a pair of pages containing the same person (which is likely to be the author of one of the SIGIR papers).

Besides such exact-match join queries, it is possible to specify similarity joins. As an example, consider a query asking for a German and a French composer that were born on almost the same date:

```
A(composer French)
B(composer German)
A.date∼B.date
```

A join condition is differently evaluated compared to the conditions shown before, because it does not change the score for a node. As a result of a join condition a virtual link is generated whose weight reflects the similarity of its endpoints, so a join condition connects matching nodes.

Analogously to a similarity query, an implicit range δ and a damping function have to be specified. A new edge – a virtual link – between two nodes n_i and n_j is generated if $|ni - nj| < \delta$. The weight of the new edge $e = (n_i, n_j)$ is computed as follows:

$$\begin{aligned} w(e) \; = 1/(\; & \mathrm{conf}(n_i, c_i) \\ & \cdot \mathrm{conf}(n_j, c_j) \\ & \cdot \mathrm{sim}_{jc}(\mathrm{value}(n_i), \mathrm{value}(n_i), \delta)) \end{aligned}$$

If such a join on the same annotation is regularly evaluated it could be advisable to precompute this join on the whole data collection (if possible) once and store the generated virtual links in the engine's index.

5. PREPROCESSING

Before extracted information can be used for query evaluation, data cleaning and integration tasks have to be applied. These precomputations and normalizations are necessary to map different representations of information into a canonical format.

We exemplarily show these precomputations for some kinds of named entities (dates, money amounts and locations). These normalizations should facilitate two aspects: First, identical entities should be identified as identical, e.g., the entity "NY state" should be identical to "New York state". Second, entities of the same kind should be comparable: 17 March '04 *is smaller than* 15.04.2003.

5.1 Dates

The canonical format of a date is a simple numerical representation that is computed with the following formula: `date_value = year*10000+month*100+day` (for appropriate values of year, month, and date). It would be easy to integrate time values in this representation if they are needed. The correct recognition and segmentation of a date into year, month and day is the task of the IE component.

5.2 Money Amounts

Money amounts are more than just simple numbers as they have a currency assigned that has to be taken into account when normalizing them. As exchange rates between most currencies are varying over time, it is impossible to use a single currency as basis and transform each money amount into this currency when it is annotated. However, using a single currency for all amounts is a good idea as it facilitates an easy comparison. We propose to externally maintain a current list of exchange rates and transform all money amounts in the index to the reference currency on a regular basis, e.g., weekly.

5.3 Locations

Compared to dates and money amounts precomputations for locations are much more laborious. To efficiently use locations during query evaluation we externally maintain a location database to look up detailed properties for each location. For this purpose we imported some gazetteer lists comprising locations, corresponding abbreviations and their type (city, country, mountain etc). Like dates and money amounts, which are inherently of a numerical nature, we want to compare locations in a similar way. A natural way of comparing locations is their distance and their relative geographical position. To connect locations to numercial values we imported data containing geographical coordinates (longitude and latitude) [1], yielding a list comprising names of locations, abbreviations and their corresponding coordinates. Table1 shows an excerpt of this list. During query evaluation, locations are mapped onto corresponding locations in our geographical database.

Full Name	Abbreviation	Type	Long	Lat
...	...	...	...	...
New York	NY	3	-73.96	40.6983
Los Angeles	LA	3	-118.6306	33.7433
Saarbrücken	SB	3	7.0	49.2333
...	...	...	...	...

Table 1: Location list

6. EXTENSIBILITY

As not every application domain benefits from location and date annotations, but has its own highly specialized annotation types, a system implementing such techniques should be designed in a modular way that allows an easy extension with additional annotation modules. Using the GATE/ANNIE framework, the implementation of extensions for the annotation component is fairly simple based on its open modular architecture.

An annotation module (comprising all components for the support of a new annotation type) for a type c_k (roughly) consists of the following parts:

- The *annotator* takes as input a graph structure and adds annotations of type c_k.
- A *domain function* $\mathrm{dom}_{c_k}(n_i)$ returns a boolean value indicating whether n_i is an instance value of the domain of c_k.
- A *similarity function* $\mathrm{sim}_{c_k}(\mathrm{value1}, \mathrm{value2}, \mathrm{range\_size})$ returns a number in the interval $[0, 1]$ measuring the similarity of the instance values value1 and value2 in the domain of c_k.
- For some annotations, like locations, additional external data about the domain is needed.

Examples for additional annotations are specific invoice or identification numbers.

7. CONCLUSION

This paper presented a systematic integration of information extraction techniques into information retrieval, based on a formal data model. It covered the whole lifetime of data from annotating them over preprocessing and using them in queries. The paper introduced a query language that allows to exploit notations for novel range and join queries that were not available in IR systems before.

8. REFERENCES

[1] D. Ancona, J. Frew, G. Jane, D. Valentine: Accessing the Alexandria Digital Library from Geographic Information Systems. ACM/IEEE Joint Conference on Digital Libraries, (JCDL 2004), Tuscon, USA

[2] M. Abolhassani, N. Fuhr, and N. Govert. Information extraction and automatic markup for XML documents. In Blanken et al. [6], pages 159-174.

[3] A. Arasu, H. Garcia-Molina: Extracting structured data from Web pages. ACM SIGMOD International Conference on Management of Data (SIGMOD 2003), San Diego, USA, 2003.

[4] T. Berners-Lee and E. Miller: The Semantic Web lifts off. ERCIM News 51, Oct. 2002.

[5] G. Bhalotia et al.: Keyword searching and browsing in databases using BANKS. 18th International Conference on Data Engineering (ICDE 2002),San Jose, USA, 2002.

[6] H. Blanken, T. Grabs, H.-J. Schek, R. Schenkel, and G. Weikum, editors. Intelligent Search on XML Data, volume 2818 of LNCS. Springer, Sept. 2003.

[7] S. Chakrabarti: Breaking through the syntax barrier: Searching with entities and relations. 15th European Conference on Machine Learning (ECML 2004), Pisa, Italy, 2004.

[8] F. Ciravegna, Y. Wilks: Designing Adaptive Information Extraction for the Semantic Web in Amilcare, in S. Handschuh and S. Staab (eds), Annotation for the Semantic Web, in the Series Frontiers in Artificial Intelligence and Applications, IOS Press, Amsterdam, 2003.

[9] W. W. Cohen, S. Sarawagi: Exploiting dictionaries in named entity extraction: combining semi-markov extraction processes and data integration methods. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2004) , Seattle, USA , 2004.

[10] V. Crescenzi, G. Mecca, P. Merialdo: RoadRunner: Automatic data extraction from data-intensive Web sites. ACM SIGMOD International Conference on Management of Data (SIGMOD 2002), Madison, USA, 2002.

[11] H. Cunningham: GATE, a general architecture for text engineering. Computers and the Humanities, 36, 223-254, 2002.

[12] H. Cunningham, D. Maynard, K. Bontcheva, V. Tablan. GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications. 40th Anniversary Meeting of the Association for Computational Linguistics (ACL'02). Philadelphia, July 2002.

[13] O. Etzioni et al.: Web-scale information extraction in KnowItAll (preliminary results). 3th international conference on World Wide Web (WWW 2004), New York, USA, 2004.

[14] D. Fensel, W. Wahlster, H. Lieberman, J. Hendler: Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential. MIT Press, 2003.

[15] R. Gaizauskas, A. Robertson: Coupling information retrieval and information extraction: a new text technology for gathering information from the Web, Computer-Assisted Information Searching on Internet Conference(RIAO 1997), Montreal, Canada, 1997.

[16] G. Gottlob et al.: The Lixto data extraction project - back and forth between theory and practice. ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 2004), Paris, France, 2004.

[17] J. Graupmann, R. Schenkel, G. Weikum: The SphereSearch Engine for Unified Ranked Retrieval of Heterogeneous XML and Web Documents, 31st International Conference on Very Large Databases (VLDB 2005), Trondheim, 2005

[18] J. Hobbs et al. FASTUS: a cascaded finite-state transducer for extracting information from natural-language text. In E. Roche and Y. Schabes, editors, Finite State Devices for Natural Language Processing. Cambridge MA: MIT Press, 1996.

[19] V. Hristidis, L. Gravano, Y. Papakonstantinou: Efficient IR-style keyword search over relational databases. 29th International Conference on Very Large Data Bases (VLDB 2003), Berlin, Germany, 2003.

[20] W. Kent. Limitations of Record-Based Information Models, in ACM Transactions of Database Systems, 4, 1979.

[21] D. Maynard, K. Bontcheva, H. Cunningham. Towards a semantic extraction of named entities. Recent Advances in Natural Language Processing (RANLP 2003), Bulgaria, 2003

[22] Proceedings of the 7th Message Unterstanding Conference (MUC7). Fairfax, USA, 1998

[23] Pathak et al.: INDUS: A System for Information Integration and Knowledge Acquisition from Autonomous, Distributed, and Semantically Heterogeneous Data Sources, Intelligent Systems for Molecular Biology (ISMB 2005), Michigan, USA, 2005

[24] S. Ray and M. Craven: Representing sentence structure in hidden Markov models for information extraction. 17th International Joint Conference on Artificial Intelligence (IJCAI 2001), Seattel, USA, 2001.

[25] A. Sahuguet, F. Azavant: Building light-weight wrappers for legacy Web data-sources using W4F. 25th International Conference on Very Large Data Bases (VLDB 1999), Edinburgh, UK, 1999.

[26] A.P. Sheth, J.A. Larson: Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. ACM Computing Surveys 22(3), pages 183-236, 1990.

[27] D. Williams, A. Poulovassilis: Combining Data Integration with Natural Language Technology for the Semantic Web, Workshop on Human Language Technology for the Semantic Web and Web Services at ISWC'03, Sanibel Island, USA 2003