E – Government meets Search Computing

Alessandro Campi

*Politecnico di Milano, Dipartimento di Elettronica e Informazione
Via Ponzio 34/5, Milano, Italy*

Abstract

People shop online, compare online, book hotels and flights online. This happens because the data needed to complete these tasks are easily accessible and a lot of Web sites allow users to query the Web to obtain enough information to be confident. The aim of this work is to propose a framework tailored to extend the internet revolution to public administration exploiting the Search Computing paradigm. It is a new way for composing data. While state-of-art search systems answer generic or domain-specific queries, Search Computing enables answering questions via a constellation of cooperating data sources, called search services, which are correlated by means of join operations.

1. Introduction

One hundred years ago, sending 10 words cost 50 dollars. Today gigabytes of information can be sent for a fraction of that cost, and huge quantities of information can be sent and received without it costing anything. This has changed the way people shop, the way people travel, the way people do business. The information and internet revolution has actually gone all the way through societies in many different ways, but it hasn't yet touched the way states are governed [10]. People shop online, compare online, book hotels and flights online. This happens because the data needed to complete these tasks are easily accessible and a lot of Web sites allow users to query the Web to obtain enough information to be confident. The aim of this work is to propose a framework tailored to extend the internet revolution to public administration. This work is the first step towards an infrastructure allowing people to know in a very easy way the information they need. People could search what operations work out properly, what records doctors have, the cleanliness of hospitals, who does best at infection control. Moreover, our work is a step forward to the transparency of public administration and toward a real accountability of public services and real awareness of citizens about their governments politics. The Missouri Accountability Portal [27] is an example of a Web portal that made available online all the data of one state in America. Every single dollar spent by that government is searchable, is analyzable, is checkable. Any business

that wants to bid for a government contract can see what currently is being spent and possibly can offer to deliver it in a cheaper way. Finally, our work allows one to relate objective data available online or offline with news articles, blog posts or other comments available online enriching objective data with information about the mood of people. In today's politics the need of a fast reaction to discontent is paramount and the possibility to relate numbers, facts, and sentiment analysis is very important.

Our proposal exploits the Search Computing paradigm [11]. It is a new way for composing data. While state-of-art search systems answer generic or domain-specific queries, Search Computing enables answering questions via a constellation of cooperating data sources, called search services, which are correlated by means of join operations. Search Computing aims at responding to queries over multiple semantic fields of interest; thus, Search Computing fills the gap between generalized search systems, which are unable to find information spanning multiple topics, and domain-specific search systems, which cannot go beyond their domain limits. Paradigmatic examples of Search Computing queries are: "Where is the school closest to my home, offering a high teaching quality and a good food service?", "Who is the best doctor who can cure insomnia in a nearby public hospital?", "Which are the highest risk factors associated with the most prevalent diseases among the young population?". These queries cannot be answered without capturing some of their semantics, which at minimum consists in understanding their underlying domains, in routing appropriate query subsets to each domain-expert search engine, and in combining answers from each engine to build a complete answer that is meaningful for the user.

A prerequisite for setting such goal is the availability of a large number of valuable search services. We could just wait for SOA (Service Oriented Architecture) to become widespread. However, in the public administration scenario, very few data are offered by services designed to support search, and moreover a huge number of valuable data sources are not provided with a service interface. In this paper, we tackle the important issue of publishing service interfaces suitable for Search

Computing so as to facilitate the widespread use of data sources on the Web and to simplify their integration in Search Computing applications. This paper is organized as follows. A general overview of the state of the art is presented in Section 2. Section 3 describes how data are queried, Section 4 illustrates the Search Computing framework, and Section 5 illustrates mechanisms for service registration and adaptation. Finally, Section 6 provides some conclusions.

2. Related works

This work is the result of a long research stream. A state of the art is shown in [11]. In particular, the approach adopted in this paper is an application of the proposal described in [9] and in [5] enriched exploiting the results presented in [3][4][6].

The design of novel search systems and interfaces is backed by several studies aimed at understanding how users' search behaviour on the Web [20][23][24][29][31]. A review of approaches to search log data mining and Web search investigation is in [2] and [21]. A specific class of studies is devoted to exploratory search, where the user's intent is primarily to learn more on a topic of interest [25][32]. Such information seeking behaviour challenges the search engine interface, because it requires support to all the stages of information acquisition, from the initial formulation of the area of interest, to the discovery of the most relevant and authoritative sources, to the establishment of relationships among the relevant information elements [31]. An interesting distinction between complex and exploratory search is made in [1], where complex search is characterized by: multiple searches, possibly over multiple sessions and spanning multiple sources; a combination of exploration and more directed information finding activities; and the variation of the search goal during the search process. A number of techniques have been proposed to support exploratory search, and user studies have been conducted to understand the effectiveness of the various approaches (e.g., [22]).

Topic exploration is a case of complex and exploratory search, centred on the goal of collecting information on a subject matter of interest from multiple sources. The key challenge in topic exploration on the Web is the massive amount of disparate information available on each topic, which demands novel systems capable of constructing effective entry points for quickly grasping the essence of a topic and the possible directions for its exploration. Topic exploration has been traditionally served by vertical search engines which restrict the scope of the available topics to a specific domain. Horizontal, i.e., cross-domain, topic exploration is a recent development. Kosmix [28] is a general purpose topic discovery engine, which responds to

keyword search by means of a topic page that summarizes the most relevant information on the subject associated to the search. Powerset (now incorporated into Microsoft's Bing [26]) specializes in extracting and organizing information from Wikipedia. Hakia [18] is another search engine capable of producing resume pages for topics associated with user's queries. Hakia exploits natural language processing techniques, specifically Ontological Semantics, for building a large ontology of concepts and correlations. A hybrid approach between vertical search engines and topic exploration systems is taken also by the latest versions of the mainstream, general purpose search engines interfaces, which are enriching results lists with extra elements derived from vertical or topical searches.

Structured object search refers to the ability of processing queries and presenting results that address entities or real world objects described in Web pages. A number of concurring factors have renovated the interest in Web information extraction also for large scale, horizontal search systems: the availability of good quality, edited content, most notably Wikipedia; the popularity of social content tagging (e.g., Flickr); and the advances in deep Web crawling and distributed data processing. Web-extracted structure and knowledge can be exploited mainly in two ways: by increasing the expressive power of the query language beyond keyword search, allowing queries to refer to the properties of data items; by exploiting the structure and additional knowledge available to enrich the presentation of the results, e.g., by suggesting correlations with other potentially interesting data items outside the scope of the user's query. For example, the work on WebTables [8] reports on the massive extraction of structured data from the surface Web. Google Squared is an application from Google Labs demonstrating the interplay between structure and Web data [16]. The interaction can be started by an ordinary keyword search, but the results are collected in a table (called a square) featuring all the attributes relevant to the result items as columns headers. The initial square can be extended horizontally, by adding columns suggested by the system or by providing tentative column names. An existing square can be extended both vertically with new items and horizontally with new columns, for which the system tries and locates data matching the supposed semantics. Google

Fusion Table [15] allows one to upload a data table (e.g., a spreadsheet file) and join (or "fuse") the data in some column with other tables, either supplied by other uses or mined from the Web. Spreadsheet-like views of the base or joined tables can be defined, saved, shared with others, and commented collaboratively.

Alternative visualizations are possible, depending of the type of data contained in a table, e.g., timelines and maps.

Faceted search is a technique allowing users to explore large object sets by filtering items based on multiple properties. Each facet corresponds to the possible value of an object's property (e.g., color, age, and so on). Facets have been used to support the exploration of document collections or query results [12] [22]. Faceted classification allows the assignment of multiple tags to an object, and the set of facets can itself be explored and organized in multiple ways (e.g., taxonomically). Dynamic Faceted Taxonomies [30] are the state of the art in the field and demonstrated good acceptance by the users [13].

3. Service Marts

Service Marts are abstractions; publishing a Service Mart entails bridging an abstract description to several concrete implementations of services. Indeed, implementing a Service Mart may require the mapping to several data sources, each one configured either as Web services or as an API, or as a materialized data collection. Thus, the Service Mart concept offers an abstraction for giving a "regular" view of the world, together with a method and associated technology for building such a regular view out of concrete data sources. This section gives a top-down view of the definition of Service Marts, from the conceptual level through the logical to the physical level. It also describes composition patterns (at the conceptual and logical level) and introduce the service registration (at the physical level).

3.1. Conceptual level

A Service Mart is an abstraction describing a class of Web objects. Thus, every Service Mart definition includes a name and a set of exposed attributes. Service Marts have atomic attributes and repeating groups consisting of a non-empty set of sub-attributes that collectively define a property. Atomic attributes are single-valued, while repeating groups are multi-valued. For example, a "Movie" Service Mart has single-value attributes ("Title", "Director", "Score", "Year", "Language") and repeating groups ("Genres", "Openings", "Actor"), each with sub-attributes. The schema of a repeating group is introduced by one level of parentheses:

Movie(Title, Director, Score, Year, Genres(Genre), Language, Openings(Country, Date), Actors(Name))

Other Service Marts used in this paper describe cinemas and restaurants:

Cinema(Name, Address, City, Country, Phone, Movies(Title, StartTimes))

Restaurant(Name, Address, City, Country, Phone, Url, Rating, Category(Name)).

Attributes and sub-attributes are typed and semantically tagged when they are defined. Repeating groups model many-valued properties (such as the "actors") within the object instances of the Service Mart (the "movie"). In this way, besides adding expressive power to Service Mart properties, they also model 1:M or M:N relationships, i.e. conceptual elements whose purpose is bridging real world objects. Concepts such as "acts-in" between "actor" and "movie" are modeled by repeating groups, by placing actors as a repeating group of movie or movies as a repeating group of actor (or both). This goal is consistent with keeping the Search Computing infrastructure as simple as possible, and also with keeping the connection between the two Service Marts as simple as possible. Of course, such a pattern introduces a limitation upon the ways of describing reality, which seems rather coercive if one considers the richness of data modeling choices offered by top-down design. But in our framework we do not use a top-down process; rather, we model data sources bottom-up, and then we look for their integration; moreover, most data sources have a simple schema, which can be well represented by a one-level nesting. Therefore, the expressive power of Service Marts seems to be appropriate for its purpose.

3.2. Logical level

At the logical level each Service Mart is associated with one or more specific access patterns. An access pattern describes the way in which one can access the Service Mart. It is a specific signature of the Service Mart with the characterization of each attribute or sub-attribute as either input (I) or output (O), depending on the role that the attribute plays in the service call. In the context of logical databases, an assignment of labels I/O to the attributes of a predicate is called adornment, and thus access patterns can be considered as adorned Service Marts. Moreover, an output attribute is designed as ranked (R) if the service produces its results in an order which depends on the value of that attribute. To ease service composition, we assume that all ranked attributes return a normalized value within the interval [0..1] (in the case of service interfaces providing two or more ranking attributes the service definition includes an aggregation function which indicates how to obtain a score in the [0..1]). For example, for the Service Mart "Movie" we can have the following access patterns:

```

Movie1(TitleO, DirectorO, ScoreR, YearO,
        Genres.GenreI, LanguageO, Openings.CountryI,
        Openings.DateI, Actor.NameO)
Movie2(TitleI, DirectorO, ScoreR, YearO,
        Genres.GenreO, LanguageO, Openings.CountryI,
        Openings.DateI, Actor.NameO)

```

In all cases, “Score” is an output attribute (ranging in [0..1]) used for ranking movies, which are presented in descending order of their score, i.e. with highest score movies first. The openings “Country” and “Date” are input parameters, which are used to extract movies shown in a specific country after a specific opening date (enabling the extraction of recent movies for that country). In the first access pattern, movies are retrieved by providing as input also one of their genres (thus modeling the request “search recent movies by genre”). In the second access pattern, movies are retrieved by providing as input also the title (thus modeling request “search recent movies by title”). Other access patterns could be used for accessing movies by providing the director or one actor. The choice of access patterns is a limitation on the way in which one can obtain data, typically imposed by existing service interfaces. Therefore, defining access patterns requires both a top-down process (from query requirements) and a bottom-up process (from service implementations). In general, this tension between top-down and bottom-up processes is typical of service design. Sometimes access patterns have more attributes than the original Service Mart. Consider cinemas and restaurants: their address is a characteristic of the underlying object, but users searching for cinemas and addresses typically provide to the service and input address (e.g. their home or current location) and search by proximity. Thus, U versions of attributes “Address”, “City” and “Country” denote the user’s location, and T/R versions of the same attributes that represent the cinema/restaurant location;

```

Cinema1(NameO, UAddressI, UCityI, UCountryI,
        TAddressO, TCityO, TCountryO, TPhoneO, DistanceR,
        Movie.TitleO, Movie.StartTimesO)

```

3.3. Connection patterns

Connection patterns introduce a pair-wise coupling of Service Marts. Every pattern has a conceptual name and then a logical specification, consisting of a sequence of simple comparison predicates between pairs of attributes or sub-attributes of the two services, which are interpreted as a conjunctive Boolean expression, and therefore can be implemented by joining the results returned by calling service implementations. Connection patterns can be directed or undirected. For example, Movies and Cinemas are connected via

the undirected connection pattern “Shows”, which uses a join on titles:

```
Shows(Movie,Cinema): [(Title=Title)]
```

The interpretation of joins within connection patterns is existential: if the movie’s title is equal to the title of any movie shown in the cinema, then the predicate is satisfied, and the two instances of movie and cinema are composed to form an instance of the result; the two instances are composed without performing any selection on sub-attributes (in the example, if one title of cinema satisfies the join, then all movies shown at the cinema are selected). Using the existential interpretation of equality predicates in selection and joins involving sub-attributes as operands yields to a simple semantics and an efficient implementation of these operations. Consider next a directed connection between cinemas and restaurants; a directed pattern can be used “from” the first “to” the second (the query search first for cinemas and next for nearby restaurants). The connections is specified as a conjunction of predicate expressions, relating the cinema address to the input location of a restaurant service, so that after determining a cinema (close to the user’s address) the service will be invoked by using the cinema’s location as input for the restaurant search:

```

DinnerPlace(Cinema, Restaurant):
[(TAddress=UAddress), (TCity=UCity),
 (TCountry=UCountry)]

```

Logically, connection patterns are expressed among pairs of orderly type compatible attributes. A connection pattern must be supported by a pair of access patterns. All the attributes of both selected access patterns must have the same labels, either I or O, and they should not both be labeled I. If both the right and left operand have an O label, then the pattern is undirected, else if the left operand is labeled O and the right operand is labeled I then the pattern is directed from left to right. Visually, Service Marts and logical connection patterns can be presented as resource graphs, where nodes represent marts and arcs represent logical connection patterns; directed connections include an edge. Thus, the Search Computing model of the Web presents a simplification of reality, seen through potentially very large resource graphs. Such representation enables the selection of interconnected concepts that support the creation and dynamic extension of multi-domain queries.

3.4. Physical Level

At the physical level of Service Marts we model service interfaces, where each service interface is

mapped to a concrete data source. A service interface may not support some of the attributes of the Service Mart, e.g., because one source could miss a property; moreover, sources may be provided with type coercion facilities so as to adapt to a single type description. These provisions allow for a minimal amount of inconsistency between service interfaces and Service Marts.

A service interface is a unit of invocation and as such must be described not only by its conceptual schema or logical adornment, but also by its physical properties. There are a huge number of options for characterizing data-intensive services, both in terms of performance and quality. Service interfaces are described by four kinds of parameters:

- Ranking descriptors classify the service interface as a search service (i.e., one producing ranked result) or an exact service (i.e., services producing objects which are not ranked). Exact services are associated with a selectivity, which is a positive number expressing the average number of tuples produced by each call. When a search service is associated with an access pattern having one or more output attributes tagged as R, then the ranking is said explicit, else it is said opaque. Explicit ranking over a single attribute can be denoted as ascending or descending. Note that search services may not be present a result with ranked attributes; e.g., most commercial search engines can be characterized as Service Marts accepting input keywords and producing semi-structured output information which is mapped to a schematic representation, but they normally do not expose rankings.
- Chunk descriptors deal with output production by a service interface. The service is chunked when it can be repeatedly invoked and at each invocation a new set of objects is returned, typically in a fixed number, so as to enable the progressive retrieval of all the objects in the result; in such case, it exposes a chunk size (number of tuples in the chunk). Search Computing is focused on the efficient dataintensive computation and therefore most service interfaces are chunked. Of course, if the service is ranked, then the first chunk contains the objects with highest ranking, and subsequent chunks yields the next objects in the ranking; normally, with exact services a query should examine all chunks, while with search services a query can examine just the top chunks.
- Cache descriptors deal with repeated invocations of the service. A very efficient way to speed up service invocations consists in caching at the requester side the responses returned for given inputs, and then use such stored answers instead of invoking the service. But such policy is not acceptable with many services, e.g. those offering real-time answers. Hence, parameters indicate if a

service interface is cacheable and in such case what is the cache decay, i.e. the elapsed time between two calls at the source that make the use of stored answers tolerable.

- Cost descriptors deal with associating each service call with a cost characterization; this in turn can be expressed as the response time (time required in order to complete a request-response cycle), and/or as the monetary cost associated with making a specific query (for those systems who charge their answers).

Every access pattern may have several service interfaces. For instance, the first access pattern of the Service Mart “movie” requires a physical service capable of filtering movies by time (e.g., whose opening date in US is recent enough) and genre (e.g. action movies) and then extracting them ranked by their quality score. For this purpose we use the IMDB archive (<http://www.imdb.com>), which stores information about thousands of movies and enriches their description with a “score” attribute computed as the average of the scores assigned by worldwide user communities. We extract data by building an ad-hoc wrapper and using it to materialize all movie descriptions; this requires periodic downloads to maintain such data materialization up-to-date. Tools for data materialization and refreshments are described in Section 5. Similarly, for the Service Mart “cinema” we use “Movie Showtimes - Google Search” (<http://www.google.com/movies>), a service allowing the retrieval of all the cinemas nearby an input location that is expressed as a complete address (address, city, country) or as a city. The service returns results sorted by cinema distance from the input location, but it does not return the actual distance (therefore, ranking is opaque, and the implementation does not expose “Distance”). Finally, for the Service Mart “restaurant” we use the Yahoo Local source (<http://local.yahoo.com/>), a service that allow the users to find Businesses & Commercial Services (e.g. restaurants) that are in or nearby a requested address, city and state, or a specific zip code. These service interfaces support the connection patterns “shows” and “dinner place” described in the previous section.

4. The Framework

Search Computing applications are built by exploiting a configurable software framework approach, illustrated in Figure 1. The Service Mart Framework provides the scaffolding for wrapping and registering data sources; the Service Invocation Framework masks the technical issues involved in the interaction with the registered Service Mart, e.g., the Web service protocol and data caching issues. Their features are discussed in the next section.

The *User Framework* provides functionality and storage for registering users, with different roles and capabilities.

The *Query Framework* supports the management and storage of queries, which can be executed, saved, modified, and published for other users to see.

The *Query Processing Framework* is the central component of the architecture, which provides service for executing multi-domain queries.

The *Query Manager* takes care of splitting the query into sub-queries and binding them to the respective relevant data sources.

The *Query Planner* produces an optimized query execution plan, which dictates the sequence of steps for executing the query. A Search Computing query is a conjunctive query over services, which includes two main aspects: the logical query clauses over the relevant data sources and the result ranking criterion. Although queries may be initially expressed over the conceptual description of service marts, eventually all queries are translated into adorned queries over service interfaces. A query plan is a well-defined scheduling of service invocations, possibly parallelized, that complies with their service interface and exploits the ranking in which individual search services return results to rank the combined query results. A query plan is a DAG (direct acyclic graph) composed by nodes (i.e., invocations of services or other operators) and edges, which describe the execution flow and the data flow between nodes. Several types of nodes exist, including service invocators, sort, join, and chunk operators, clocks (defining the frequency of invocations), caches, and others. The plans are specified through Panta Rhei model [7]. Finally, the Execution Engine actually executes the query plan, by submitting the service calls to designated services through the Service Invocation Framework, building the query results by combining the outputs produced by service calls, computing the global ranking of query results, and producing the query result outputs in an order that reflects their global relevance. These components are not the target of this paper, but are investigated in the Search Computing project. The results of this research stream are described in [5,6,7].

To obtain a specific Search Computing application, the general-purpose architecture of Figure 1 is customized with the help of tools targeted to programmers, expert users, and end users.

- Service Publishers register Service Mart definitions within the service repository, and declare the connection patterns usable to join them. The registration process is realized through a Service Registration Tool that: 1) helps the publisher in the specification of the SM, AP and SI attributes and parameters respectively and 2) it hides to the user the Internal API, that allow the communication between the services and the engine levels. The service publishers are in charge of implementing mediators, wrappers, or data materialization components, so as to make data sources compatible with the Service Mart standard interface and expected behavior.
- Expert Users configure Search Computing applications, by selecting the Service Marts of interest, by choosing a data source supporting the Service Mart, and by connecting them through connection patterns. They also configure the complexity of the user interface, in terms of controls and configurability choices to be left to the end user.
- End Users use Search Computing applications configured by expert users. They interact by submitting queries, inspecting results, and refining/evolving their information need according to an exploratory information seeking approach, which we call Liquid Query [4].

Search Computing aims at building two new communities of users: Content providers, who want to organize their content (now in the format of data collections, databases, Web pages) in order to make it available for search access by third parties, and expert users, who want to offer new services built by composing domain-specific content in order to go "beyond" general-purpose search engines such as Google and the other main players. The service registration framework aims at facilitating content providers in their task of publishing data sources. This framework has been developed taking into account the trend towards empowerment of the user, as witnessed in the field of Web mash-ups [6]. Indeed, all the design activities from service registration on do not ask to perform low-level programming.

Figure 1. Overview of the Search Computing framework

Figure 2. Development process

5. Sources registration

The trend towards supporting users in publishing data sources on the Web is a general one. Google, Yahoo and Microsoft are building environments and tools (Fusion Tables [15], Yahoo! BOSS [29]) for helping Web users to publish their data, with the goal of capturing the so-called “long tail” of data sources. In Search Computing, data sources should produce ranked output and data extraction should be performed incrementally, by “chunks”; users can suspend a search and then resume it, possibly guiding the way in which data sources should be inspected. We are building tools and/or providing best practices, applicable to data sources of various kinds, for enabling data providers to build “search” service adapters. We distinguish three different scenarios:

- Data can be queried by means of a Web service or combining the results of different Web services.
- Data are available on the Web but must be extracted from Web sites through wrappers.
- Data are not directly accessible and must first be materialized.

Results returned by a call to a service interface expose an interchange format written in JSON (JavaScript Object Notation), a lightweight data interchange format easy to read and write by humans and easy to parse and generate by machines. The format descends directly from the conceptual description of the Service Mart, therefore all instances of a Service Mart share the same interchange format, regardless of the service interface which produces them. Below is a JSON “movie” instance:

```
{ "title": "Highlander",
  "director": "Russell Mulcahy",
  "score": "0.7",
  "year": "1986",
  "genres": [{"genre": "action"}],
  "openings": [{"country": "US", "date": "31-10-86"}],
  "actors": [{"name": "Christopher Lambert"},
             {"name": "Sean Connery"}]}
```

5.1. Web services

The typical service implementation is a real Web service registered in the platform. Web services return their output in arbitrary format, including but not limited to HTML, XML and JSON. Given that the Service Mart interchange format is a well-defined JSON structure, the service implementation developer must define a series of transformations on the results, and bundle them into a remote service implementation that hides the peculiar features of each remote source.

To tackle the need to combine the results of different Web services we built a Service Mart Framework containing some predefined software modules useful to manipulate data. The very first of them is the invocation module which invokes a service and returns a list of tuples; next, tuples are read by a tuple reader and possibly copied by a tuple cloner. Other modules perform projections, string replacements, computations of regular expressions, data conversions and splitting or concatenation of attributes. Once the services are transformed to return JSON, another step can be necessary in order to adapt the cardinality of the results returned in each service, which can be not appropriate (a search service could return too many results with each call, or even all the results together). In this case, a chunker module supports changing the chunk size: every call to the actual service is translated into the appropriate number of calls to the service implementation, which buffers results and produces chunks of the desired size.

5.2. Web pages

The second types of sources we want to use are HTML pages. The Web is rich of good quality information stored in HTML Web sites. Wrappers are particular programs that can make available data published in the Web. In the context of Service Marts, wrappers can be used to capture data which is published by Web servers in HTML format, because in such case a data conversion is needed in order to support data source integration – data must be rearranged according to the Service Mart normalized schema. Another typical use of wrappers in Search Computing occurs when services respond with HTML documents which must be translated in the normal schema and encoded in JSON. For building wrappers, several systems are available; in particular

we use Lixto [16]. By marking a region of an example Web document displayed on screen the user helps the tool to build a set of rules describing the structure of the pages of the Web site. These rules are used to generate a wrapper that "query" Web site in real time. Fig. 3 shows the relationships between data extracted on the Web and a tabular view on these data.

Figure 3. Data extraction from data published in HTML

5.3. Materialized data

Even if most service implementations require a call to a remote service, in some cases summarized and materialized data may need to be stored at the engine site. Data materialization is a general process, which can be applied to sources in order to transform their format, to eliminate redundancy, to improve their quality, and so on; data materialization moves data preparation from query execution to source registration time, together with a data materialization schedules setting the times when materialization should be repeated; therefore, data materialization is very useful for supporting efficient query execution. Intrinsic to the normalization process, however, is the capturing of a given snapshot of the data, which is not current; therefore the approach can be used only with data which rarely changes.

We developed a materializer specifically for use in Search Computing. The materializer is a software component whose objective is to read arbitrary data sources and organize data in a normalized format, suitable for data export according to a Service Mart definition. The materializer is organized with two logical layers: the data extraction layer operates directly upon the data sources, that can be of arbitrary formats (e.g., tables, XSL files, XML trees, and so on); its purpose is to transform the input data into relational tables of arbitrary format, called primary materialization; such tables are temporary, used only in the materializer, and invisible to the outside environment. A series of SQL procedures are applied to the primary materialization in order to produce a normalized schema. Such schema has maps every Service Mart to a primary table and every repeating group to an auxiliary table; the

primary table has a system-generated unique identifier, while the auxiliary table has a composite identifier built with the primary table's identifier and a progressive number.

A materializer uses the modules described in Section 5.1 to combine results returned by different Web services and contains some new units that operate together with the unit previously defined. For example, Tuple writer unit writes data items as rows in a database table. Figure 4 shows an example of materialization process. When data materializations are stored according to the normalized schema, the service implementation is automatically built by using SQL queries whose code depends only on the service interface description. Note that data providers need not use the materializer, as long as they build tables according to the normalized schema.

Figure 4. Process description within the Materializer

Specifically, queries over stored tables perform selection based upon input and ranking using the ORDER BY clause. While selection, ranking and nesting are supported by standard SQL, chunking requires returning at each call the "top k" tuples; unfortunately, "top k" queries are not supported in standard SQL, but all commercial DBMS support them in a specific SQL dialect; some of them offer as well "interval" queries, enabling the extraction of the "next k" (defined as the tuples within the interval $[k+1..2k+)$). MySQL offers "interval" queries through a LIMIT clause which returns at each query evaluation an ordered table with the tuples whose ranking falls between the first and second parameter. If we use such feature, a simple query pattern for extracting tuples from rank h to k (where k-h is the chunk size) is:

```
SELECT *
FROM Table
WHERE condition
ORDER BY rank DESC
LIMIT h,k
```


6. Conclusions

The work described in this paper is steps forward the possibility to exploit all the information we have about government and society for a better understanding of the actual situation. This work exploits the Search Computing paradigm in order to allow people to relate objective data available online or offline with news articles, blog posts or other comments available online enriching objective data with information about the mood of people.

7. References

- [1] Aula, A., and Russell, D.M. Complex and Exploratory Web Search. ISSS: Information Seeking Support Systems Workshop (Chapel Hill, NC, USA, June 2008), 23-24.
- [2] Baeza-Yates, R.A. Applications of Web Query Mining. ECIR: European Conference on Information Retrieval, 2005, Springer LNCS 3408, 7-22.
- [3] Barbosa, L., and Freire, J. Siphoning hidden-web data through keyword-based interfaces. SBBD, 19th Brazilian symposium on databases, 2004, 309-321.
- [4] A. Bozzon, M. Brambilla, S. Ceri, P. Fraternali. Liquid query: multi-domain exploratory search on the web. In Proc. of the 19th int. conf. on World wide web (WWW '10). ACM, New York, USA, 161-170.
- [5] Braga, D., Campi, A., Ceri, S., Raffio, A. Joining the results of heterogeneous search engines, Information Systems, Vol. 33, Issues 7-8, 2008, Pages 658-680.
- [6] Braga, D., Ceri, S., Daniel, F., Martinenghi, D. Mashing Up Search Services. IEEE Internet Comp. 12(5) (2008), 16-23.
- [7] D. Braga, F. Corcoglioniti, M. Grossniklaus, S. Vadacca: Panta Rhei: Optimized and Ranked Data Processing over Heterogeneous Sources. ICSOC 2010: 715-716.
- [8] Cafarella, M. J., Halevy, A., et al. WebTables: Exploring the Power of Tables on the Web. In VLDB 2008, 538-549.
- [9] A. Campi, S. Ceri, A. Maesani, S. Ronchi. Designing service marts for engineering search computing applications. In Proc. of the 10th int. conf. on Web engineering (ICWE'10). Springer, 50-65.
- [10] David Cameron. The next age of government. http://blog.ted.com/2010/02/16/the_next_age_of/ (accessed 1/2/2011).
- [11] Ceri, S., Brambilla, M. (eds.). Search Computing Challenges and Directions. Springer LNCS vol. 5950, March 2010.
- [12] D. Dash, J. Rao, et al. Dynamic faceted search for discovery-driven analysis. 17th ACM Conf. on information and Knowledge Management, CIKM 2008, 3-12.
- [13] DBLP Faceted Search. <http://dblp.l3s.de/>.
- [14] Google Base API. <http://code.google.com/apis/base/>.
- [15] Google Fusion Tables. <http://tables.googlelabs.com/>.
- [16] Google Squared. <http://www.google.com/squared>.
- [17] G. Gottlob, C. Koch, R. Baumgartner, M. Herzog, and S. Flesca. The Lixto data extraction project: back and forth between theory and practice. In PODS '04.
- [18] Hakia. <http://hakia.com/>.
- [19] Inselberg, A. The Plane with Parallel Coordinates. Visual Computer 1 (4). Springer: 69-91. (1985).
- [20] Jansen, B.J., Booth, D.L., and Spink, A. Determining the user intent of web search engine queries. WWW Conf. 2007: 1149-1150.
- [21] Jansen, B.J., Pooch, U.W. A review of Web searching studies and a framework for future research. JASIST 52(3): 235-246 (2001).
- [22] Kules, B., Capra, R., Banta, M., and Sierra, T. What do exploratory searchers look at in a faceted search interface? JCDL, Joint Conference on Digital Libraries(2009). 313-322.
- [23] R. Kumar, A. Tomkins. A Characterization of Online Search Behavior. Data Engineering Bulletin, June 2009, 32(2), 3-11.
- [24] Lee, U., Liu, Z., and Cho, J. Automatic identification of user goals in Web search. WWW 2005 (Chiba, Japan): 391-400.
- [25] Marchionini, G. Exploratory search: from finding to understanding. Communications ACM 49(4): 41-46 (2006).
- [26] Microsoft Bing. <http://www.bing.com/>. (accessed 1/2/2011).
- [27] Map your taxes. <http://mapyourtaxes.mo.gov/MAP/Portal/Default.aspx> (accessed 1/2/2011).
- [28] Rajaraman, A. Kosmix: High Performance Topic Exploration using the Deep Web, VLDB 2009, Proceedings of VLDB 2(2): 1524-1529.
- [29] Rose, D.E., and Levinson, D. Understanding user goals in Web search. 13th WWW Conf. (New York, 2004), 13-19.
- [30] Sacco, G. M., and Tzitzikas, Y. Dynamic Taxonomies and Faceted Search: Theory, Practice, and Experience. Series: The Information Retrieval Series, Vol. 25, Springer 2009.
- [31] White, R. W., and Drucker, S. M. Investigating behavioral variability in web search. 16th WWW Conf. (Banff, Canada, 2007), 21-30.
- [32] White, R.W., Roth, R.A. Exploratory Search. Beyond the Query-Response Paradigm. Synthesis Lectures on Information Concepts, Retrieval, and Services Series, Gary Marchionini (ed.), vol. 3. Morgan & Claypool, 2009.