Designing Service Marts for Engineering Search Computing Applications

Alessandro Campi, Stefano Ceri, Andrea Maesani, and Stefania Ronchi

Politecnico di Milano, Italy
{campi,ceri,maesani,ronchi}@elet.polimi.it

Abstract. The use of patterns in data management is not new: in data warehousing, data marts are simple conceptual schemas with exactly one core entity, describing facts, surrounded by multiple entities, describing data analysis dimensions; data marts support special analysis operations, such as roll up, drill down, and cube. Similarly, Service Marts are simple schemas which match "Web objects" by hiding the underlying data source structures and presenting a simple interface, consisting of input, output, and rank attributes; attributes may have multiple values and be clustered within repeating groups. Service Marts support Search Computing operations, such as ranked access and service compositions. When objects are accessed through Service Marts, responses are ranked lists of objects, which are presented subdivided in chunks, so as to avoid receiving too many irrelevant objects – cutting results and showing only the best ones is typical of search services. This paper gives a formal definition of Service Marts and shows how Service Marts can be implemented and used for building Search Computing applications.

1 Introduction

Search Computing is a new paradigm for composing search services [7]. While state-of-art search systems answer generic or domain-specific queries, Search Computing enables answering questions via a constellation of cooperating 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 can I attend an interesting scientific conference in my field and at the same time relax on a beautiful beach nearby?", "Where is the cinema closest to my hotel, offering a high rank action movie and a near-by pizzeria?", "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.

B. Benatallah et al. (Eds.): ICWE 2010, LNCS 6189, pp. 50–65, 2010.

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, few software services are currently designed to support search, and moreover a huge number of valuable data sources (the so called "long tail" of Web information) are not provided with a service interface. In this paper, we therefore focus on 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. At the basis of our work, we observe the pervasive role of software services and SOA[1]. While the SOA principles are becoming widespread, however, we observe distinct standards, languages, and programming styles. Thus, the SOA vision is developed in a variety of directions, and we emphasize the relevance of supporting ordered queries upon data sources, by means of a new pattern.

In the framework of Search Computing, we define a **Service Mart** as the data abstraction for data source publication and composition. The goal of a Service Mart is to ease the publication of a special class of software services, called search services, whose responses are ranked lists of objects. Every Service Mart is mapped to one "Web object" available on Internet; therefore, we may have Service Marts for "hotels", "flights", "doctors", and so on. Thus, Service Marts are consistent with a view of the "internet of objects" which is gaining popularity as a new way to reinterpret concept organization in the Web and go beyond the unstructured organization of Web pages. Moreover, pairs of Service Marts are augmented with "connections", so as to support their linking; Service Marts and their connections constitute a resource network that can be used as a high-level interface for expressing search queries.

This paper is organized as follows. A general overview of the state of the art is presented in Section 2. Section 3 introduces Service Marts by illustrating their three levels of description (conceptual, logical, and physical). Section 4 illustrates the Search Computing framework so as to position Service Marts within the global reference architecture, and Section 5 illustrates mechanisms for service registration and adaptation. Finally, an example of usage of Service Marts is briefly described in Section 6, and Section 7 provides some conclusions.

2 Related Work

Current service description languages and protocols, such as WSDL and SOAP, provides limited support in mechanizing service recognition, combination, and automated negotiation. Several proposals, such as WSFL [19] or DAML-S [1], extend these languages and protocol in a semantic direction. OWL-S [20] is an attempt at formalizing the semantics of Web services using ontology technology. COSMO [22] provides concepts for reasoning about services, and for supporting operations, such as composition and discovery, which are performed on them at design and run-time. The framework facilitates the use of different service description

[1] Consider the prominent role given to Software and Services in Call 5 of the EU-funded Seventh Framework Programme (FP7), whose goal is to achieve an "Internet of the Future, where organizations and individuals can find software as services on the Internet, combine them, and easily adapt them to their specific context [10]".

languages tailored to different service aspects. Web Service Modeling Framework (WSMF) [13,14] is a fully-fledged modeling framework for describing the various aspects related to Web services using four main elements: ontologies providing the terminology, Web services providing access to resources, goals representing user desires, and mediators dealing with interoperability problems. Other ongoing efforts aim at creating a conceptual framework for service modeling; a prominent example is the W3C's Web Services Architecture [27]. In [9] authors propose a conceptual model that describes actors, activities and entities involved in a service-oriented scenario and the relationships between them. This work extends the W3C's Web Services Architecture, but the authors do not specify the semantics of the concepts described; their conceptual model is a glossary of terms.

Previous approaches to Web service combination is described by the SOA scientific community with two different flavors: orchestration and choreography. The distributed approach of choreographed services (e.g., using WS-CDL [25] or WSCI [24]) deals with service compatibility and compliance to predefined behavior descriptions but has limited relevance in the Search Computing context. The centralized approach of orchestrated services (e.g., using BPEL [21]) suits better the research problem addressed in this paper, as orchestrations are executable service compositions and services need not be aware of being the object of query execution. The research on composition and integration of data sources is very rich, some interesting results are in [2,11,12,26,28].

The work described in this paper is the result of a research stream starting with [23], in which the authors propose a Web service management system that enables querying multiple Web services in a transparent and integrated fashion and propose an algorithm for arranging a query's Web service calls into a pipelined execution plan that exploits parallelism among Web services. Subsequent work [5] established the theoretical framework for stating the problem of joining heterogeneous search engines; while [6] presented an overall framework for multi-domain queries on the Web. These works are the predecessors of the current research project named "Search Computing", whose preliminary results are described in [7].

Service Marts are specific "data patterns"; their regular organization helps structuring Search Computing applications. The search of patterns for easing data modeling for specific contexts is not new; the most well known data modeling pattern is the so called "data mart", used in data warehousing as conceptual schemas for driving data analysis. Data marts [3] are simple schemas having one core entity, describing facts, surrounded by multiple entities, describing the dimensions of data analysis. Such subschema allows a number of interesting operations for data selection and aggregation (e.g. data cubes, rollup, drilldown) whose semantic definition is much simplified by data characterization as either fact or dimension and by the regular structure of the schema. Analogously, a "Web mart" [8] is a pattern introduced in the Web design community to characterize the role played by data items in data-intensive Web applications. Web marts have a central entity, the core concept, which describes a collection of core objects, surrounded by other entities which are classified as "access entities", enabling selection of core objects through navigation, and "detail entities", describing core objects in greater detail. Thus it is possible to drive a design process that produces first-cut standard Web applications (e.g. sales, inventories, travels, and so on).

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** describe 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][2]. For example, for the Service Mart "Movie" we can have the following access patterns:

$Movie_1$(Title^O, Director^O, Score^R, Year^O, Genres.Genre^I, Language^O, Openings.Country^I, Openings.Date^I, Actor.Name^O)
$Movie_2$(Title^I, Director^O, Score^R, Year^O, Genres.Genre^O, Language^O, Openings.Country^I, Openings.Date^I, Actor.Name^O)

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;

$Cinema_1$(Name^O, UAddress^I, UCity^I, UCountry^I, TAddress^O, TCity^O,TCountry^O, TPhone^O, Distance^R, Movie.Title^O, Movie.StartTimes^O)

[2] We also consider the possibility of service interfaces providing two or more ranking attributes, in such case the service definition includes an aggregation function which indicates how to obtain a score in the [0..1] interval as a function of the ranking attributes.

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 represents 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 data-intensive 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 Search Computing 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, as discussed in Section 4.1. 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. 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 very first 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.

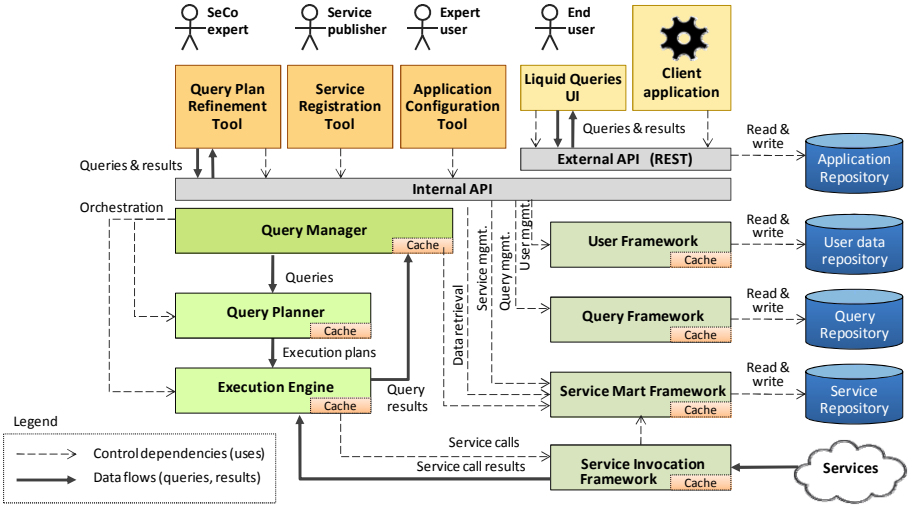

Fig. 1. Overview of the Search Computing framework

- **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.

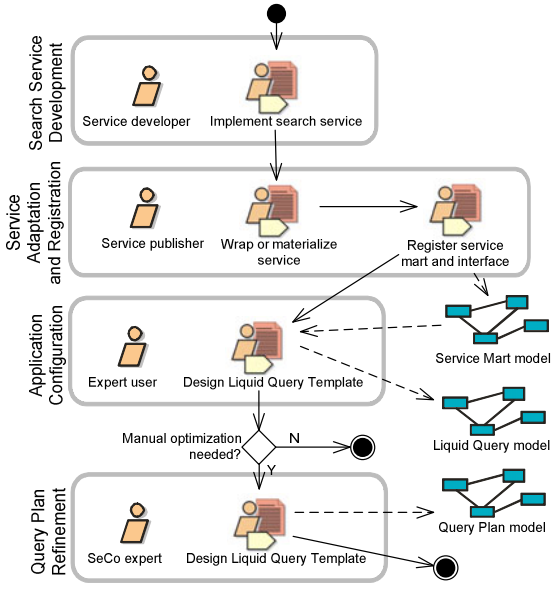

Fig. 2. Development process for SeCo applications (SPEM notation)

5 Web Service Registration and Adaptation

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) [17], 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": "31-10-1986"}],
 "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.

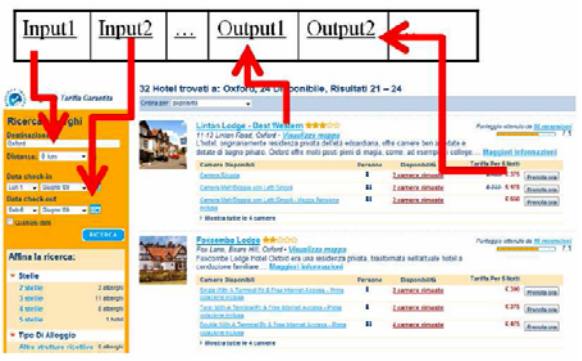

Fig. 3. Data extraction from query results published in HTML

5.3 Materialized Databases

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.

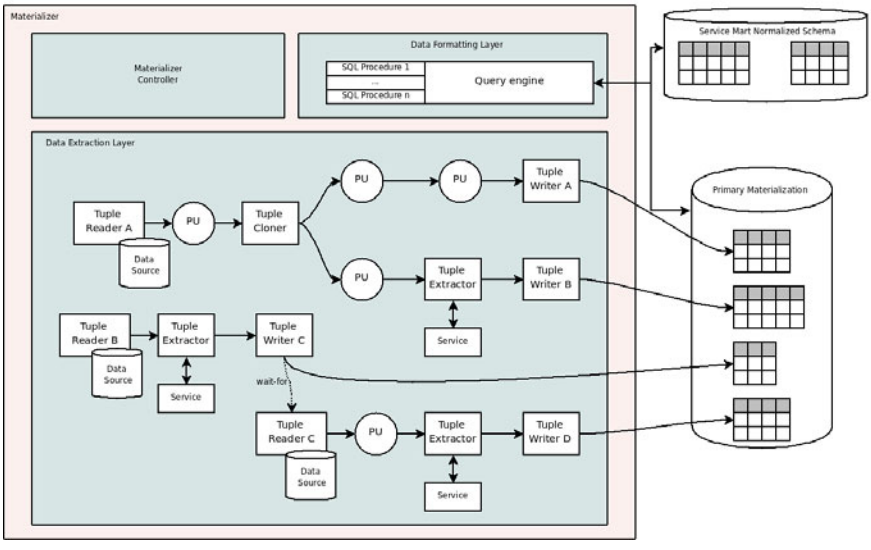

Fig. 4. Process description within the Materializer

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 Applications and Use Cases

This section describes typical user interaction scenarios based on the running example presented in the previous sections. We describe a search interaction concerning close-by movies, cinemas, and restaurants, and the refinement and exploration of results through application of additional local filters. Suppose the user submit a query asking for a cinema with a high rated movie of a given kind next to a good vegetarian restaurant. Once the user submits the search parameters, the query is performed and results are calculated and displayed in the result table, as illustrated in Fig. 5. The result page is enriched with interaction options that the user can choose.

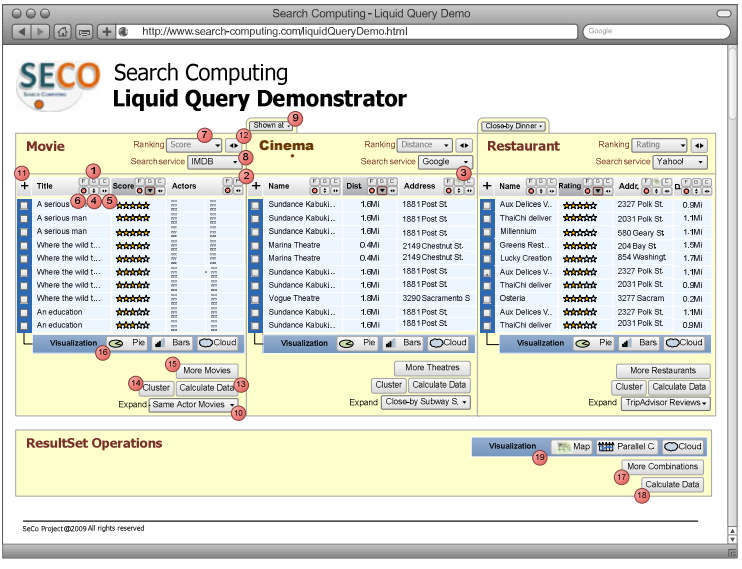

Fig. 5. Result interface

Once the results are shown, the user can interact with them through the available commands. Some operations (i.e., visualization options and expansion to new services) require the user to select a subset of result instances; selection is performed by means of checkboxes. When needed, a popup window asks for additional parameters or details on the operation to be performed.

Local filters on column values can be applied by clicking the "F" button on the column header of interest (e.g. the user may want to select only the restaurants having rating higher than three stars). Additional search dimensions (e.g. public transportation schedules, or other shows nearby for after dinner) can be added in order to augment a given solution set. Data summarization and visualization methods can also be used. The user interface and HCI aspects of Search Computing are further discussed in [4].

7 Conclusions

This paper has provided the definition of Service Marts as an interoperability concept for building Search Computing applications, with associated technologies for registering and adapting Web services. The Web world is described as a resource graph with Service Marts linked by and connection patterns, and then Service Marts are associated with service interfaces and implementations. Tool associated with Service Marts help the publication of arbitrary content (e.g., extracted from data sources or Web pages) in a standard format. Formats and extraction rules enable the execution of queries and the composition of query results.

Future work in this direction of research within the Search Computing project will address extending connection patterns so as to express semantically rich concepts in the context of search ("nearness" interpreted in the context of terminological, spatial, and temporal distance), so as to support richer corms of service compositions.

References

[1] Ankolenkar, A., et al.: DAML-S, http://www.daml.org/services/daml-s/2001/10/daml-s.html

[2] Berners-Lee, T., Handler, J., Lassila, O.: The Semantic Web. Scientific American (May 2001)

[3] Bonifati, A., Cattaneo, F., Ceri, S., Fuggetta, A., Paraboschi, S.: Designing data marts for data warehouses. ACM Trans. Software Engineering Methodology 10(4), 452–483 (2001)

[4] Bozzon, A., Brambilla, M., Ceri, S., Fraternali, P.: Liquid query: multi-domain exploratory search on the Web. In: Proc. WWW-2010 Conference (2010, to appear)

[5] Braga, D., Campi, A., Ceri, S., Raffio, A.: Joining the results of heterogeneous search engines. Information Systems 33(7-8), 658–680 (2008)

[6] Braga, D., Ceri, S., Daniel, F., Martinenghi, D.: Optimization of multi-domain queries on the Web. In: Proc. VLDB, vol. 1(1), pp. 562–573 (August 2008)

[7] Ceri, S., Brambilla, M. (eds.): Search Computing - Challenges and Directions. LNCS, vol. 5950. Springer, Heidelberg (to appear, March 2010)

[8] Ceri, S., Matera, M., Rizzo, F., Demaldè, V.: Designing data-intensive Web applications for content accessibility using Web marts. Commun. ACM 50(4), 55–61 (2007)

[9] Colombo, M., Di Nitto, E., Di Penta, M., Distante, D., Zuccalà, M.: Speaking a common language: A conceptual model for describing service-oriented systems. In: Benatallah, B., Casati, F., Traverso, P. (eds.) ICSOC 2005. LNCS, vol. 3826, pp. 48–60. Springer, Heidelberg (2005)

[10] CORDIS (September 17, 2009), http://cordis.europa.eu/fp7/ict/ssai/

[11] De Witt, D.J., Ghandeharizadeh, S., Schneider, D.A., Bricker, A., Hsiao, H.-I., Rasmussen, R.: The Gamma Database Machine Project. IEEE TKDE 2(1), 44–62 (1990)

[12] Doan, A., Domingos, P., Halevy, A.Y.: Reconciling schemas of disparate data sources: A machine-learning approach. In: SIGMOD (2001)

[13] Fensel, D., Bussler, C.: The Web Service Modeling Framework WSMF. Electronic Commerce Research and Applications 1(2), 113–137

[14] Fensel, D., Musen, M.: Special Issue on Semantic Web Technology. IEEE Intelligent Systems (IEEE IS) 16 (2)

[15] Fusion Tables, http://tables.googlelabs.com/

[16] Gottlob, G., Koch, C., Baumgartner, R., Herzog, M., Flesca, S.: The Lixto data extraction project: back and forth between theory and practice. In: PODS '04 (2004)

[17] JSON, http://JSON.org/

[18] Levy, A.Y., Rajaraman, A., Ordille, J.J.: Querying Heterogeneous Information Sources Using Source Descriptions. In: VLDB (1996)

[19] Leymann, F.: WSFL, http://www-4.ibm.com/software/solutions/Webservices/pdf/WSFL.pdf

[20] Martin, D., Burstein, et al.: Bringing Semantics to Web Services with OWL-S. In: World Wide Web, vol. 10(3), pp. 243–277 (2007)

[21] OASIS. Web Services Business Process Execution Language. Technical report (2007), http://www.oasis-open.org/committees/wsbpel/

[22] Quartel, D.S., Steen, M.W., Pokraev, S., Sinderen, M.J.: COSMO: A conceptual framework for service modeling and refinement. Information Systems Frontiers 9(2-3), 225–244 (2007)

[23] Srivastava, U., Munagala, K., Widom, J., Motwani, R.: Query optimization over Web services. In: VLDB '06, VLDB Endowment, pp. 355–366 (2006)
[24] W3C. Web Service Choreography Interface (WSCI) 1.0. W3C Note (August 2002)
[25] W3C. Web Services Choreography Description Language Version 1.0 (December 2004)
[26] Wang, J., Wen, J.R., Lochovsky, F., Ma, W.Y.: Instance-based schema matching for Web databases by domainspecific query probing. In: VLDB (2004)
[27] Web Services Architecture (2004), http://www.w3.org/TR/ws-arch/
[28] Wu, W., Yu, C., Doan, A., Meng, W.: An interactive clustering-based approach to integrating source query interfaces on the Deep Web. In: SIGMOD (2004)
[29] Yahoo! Search Boss, http://developer.yahoo.com/search/boss/