Erschienen in: SEBD 2010 : Proceedings of the Eighteenth Italian Symposium on Advanced Database Systems ; Rimini, Italy, June 20 - 23, 2010 / Bergamachi, Sonia et al. (Hrsg.). - Bologna : Esculapio, 2010. - S. 446-453. - ISBN 978-88-7488-369-1

Search Computing Systems

(Extended abstract)

Stefano Ceri, Adnan Abid, Mamoun Abu Helou, Alessandro Bozzon, Daniele Braga, Marco Brambilla, Alessandro Campi, Francesco Corcoglioniti, Emanuele Della Valle, Davide Eynard, Piero Fraternali, Michael Grossniklaus, Davide Martinenghi, Stefania Ronchi, Marco Tagliasacchi, Salvatore Vadacca

Politecnico di Milano, Dipartimento di Elettronica ed Informazione,
V. Ponzio 34/5, 20133 Milano, Italy
first.last@polimi.it

Abstract. Search Computing defines a new class of applications, which enable *end users* to perform exploratory search processes over multi-domain data sources available on the Web. These applications exploit suitable software frameworks and models that make it possible for *expert users* to configure the data sources to be searched and the interfaces for query submission and result visualization. We describe some usage scenarios and the reference architecture for Search Computing systems.

Keywords: Search Computing, software engineering, search engine, software architectures, Web information systems.

1 Introduction

Throughout the last decade, search has become the most adopted way to access information over the Internet. Users are asking for queries that are more and more engaging for search engines, in several senses: the user search activity assumes the form of an interaction process instead of a single query; the single query itself becomes more complex (in terms of amount and extension of information the user asks for); and the information to be retrieved is often hidden in the so called "deep Web", which contains information perhaps more valuable than what can be crawled on the surface Web.

Different classes of solutions have emerged to cover this information need: general-purpose search engines; meta-search engines that query several engines and build up a unified result set; vertical search engines that aggregate domain-specific information from a fixed set of relevant sources and let users pose more sophisticated queries (e.g., finding proper combinations of flights, hotels, and car rental offers).

However, an entire class of information needs remains to be supported: the case in which a user performs a complex *search process* [2], addressing different domains, possibly covered by distinct vertical search engines. For example, a user may wish to find a place where an interesting event occurs, that has good weather in a given period, with travel and accommodation options that match given budget and quality standards, maybe with close-by trekking opportunities. This kind of search can be performed by separately looking for each individual piece of information and then mentally collating partial results, to get a combination of objects that satisfies the needs, but such procedure is cumbersome and error prone.

We define *search computing applications* [4], as the new class of applications aimed at responding to multi-domain queries, i.e., queries over multiple semantic fields of interest, by helping users (or by substituting to them) in their ability to decompose queries and manually assemble complete results from partial answers. The contribution of this paper is to provide an overview of search computing system, comprising a description of the user roles and of the system architecture and deployment strategy.

Our work is coherent with the current trend in service-oriented architectures (SOA). Proposals for service specification include WSDL, UML models, and ontologies (WSMO and OWL-S). Service registration is based on UDDI and its variants, portals like XMethods.com and RemoteMethods.com, systems like Woogle, Wsoogle.com, Web Service Crawler Engine [1], and others. A large set of languages and tools address service orchestration too (e.g., BPMN, WS-BPEL, XPDL). The SeCo Project is focused on the integration of search services, i.e. services producing ranked and chunked output, allowing consumers to halt the, when enough results are available. Research projects with comparable goals include: Microsoft Symphony, which enables non-developers to build and deploy search-driven applications that combine their data and domain expertise with content from search engines[7]; Google Squared (www.google.com/squared) and Fusion Tables (tables.googlelabs.com), which produce tabular results of searches over web and proprietary information respectively; Kosmix (www.kosmix.com), a general-purpose topic discovery engine, which responds to keyword search by means of a topic page. All these proposals miss several significant features with respect to SeCo, including: join of results, cost awareness, definition and optimization of query plans, and others.

The paper is organized as follows. Section 2 introduces the core concepts of search computing, required for registering services, formulating queries, executing them, and then interacting with the system for progressively exploring and augmenting query results. Section 3 describes the components of the search computing framework. Section 4 describes the various kinds of users that interact with the system at service registration, query configuration and query execution time. Section 5 describes the software architecture of the system.

2 Core Concepts

In this section we describe the main concepts involved in search computing applications.

2.1 Service Marts

Service marts are abstractions of several Web services dealing with the same underlying Web objects. They are modeled at three levels of abstraction that lead from the conceptual representation of web objects down to the implementation of search services. At the conceptual level, we represent the properties (attributes and data types) of such Web objects in a standardized way. Every service mart definition includes a name and a signature (a collection of strongly typed attributes, which can be atomic, composed, or multi-valued). The association between service marts is expressed by *connection patterns*, defined by a *conceptual name* and a *logical specification*, consisting of simple comparison predicates between pairs of attributes of the two services, which are interpreted as a conjunctive Boolean expression, and therefore can be implemented by joining the objects returned by calling Web service implementations.

Each service mart is then associated with one or more access patterns. An *access pattern* is a specific signature of the service mart in which each attribute is denoted as either input (I) or output (O), depending on the role that it plays in the service call. Moreover, an output attribute can be designated as ranked (R), if the service results are ordered based on the value of that attribute (e.g., the number of stars for hotels).

Finally, *service interfaces* describe the physical implementations of services. Two categories of service interfaces are possible: *search service* (i.e., one producing ranked sets of objects) or an *exact service* (i.e., services producing unranked objects). Service interfaces are characterized by the properties of *chunking* (a service is *chunked* when it returns objects one subset at a time, with specified chunk size, so as to enable the progressive retrieval of all the objects in the result set), *caching* (a service is *cacheable* when its results after repeated invocations can be cached, and in such case it is associated with a cache *decay*), and *cost* (a cost characterization expressed as the *response time* and/or as the *monetary cost of invocation*).

2.2 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. The current version of our system supports conjunctive queries which explicitly refer to registered services; end users select given queries from an existing repository and then provide the query input through a form, thereby activating a give query execution. We envision that in the future we will support a simpler query language (e.g. keyword-based) and the system will include a Query Analysis component for selecting the relevant services and mapping them subsets of the query keywords.

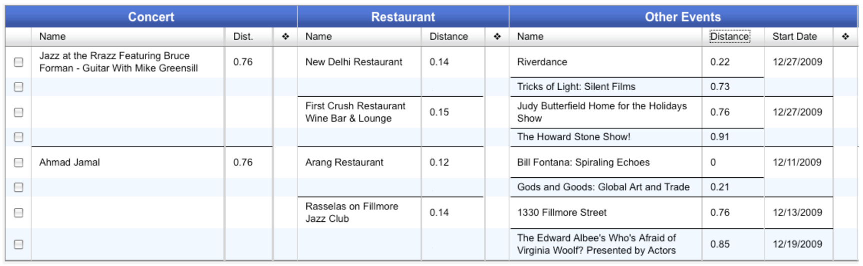
2.3 Query Plan

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.

2.4 Liquid Interactions

Liquid Query is a new paradigm allowing users to formulate, and get responses to, multi-domain queries through an exploratory information seeking approach, based upon structured information sources exposed as software services. The Liquid Query interface presents the users composite answers, obtained by aggregating search results from various domains; result composition is achieved through *join* – a classic operation of data management, by exploiting the structural information afforded by the search service interfaces that wrap the domain-specific data sources.

Liquid Query consists of a set of abstractions supporting exploratory search over the results of search computing systems. The Liquid Query interface visibly highlights the contribution of each search service to the composite result set, and supports the users in fine tuning their requests to the underlying search services in several ways, e.g. by expanding the query with an extra search service, by asking for more results from a specific service, by aggregating results, by reordering them, by adding details (drill-down) or removing them (roll-up), and so on. Fig. 1 shows a clustered and expanded result, combining information about concerts, restaurants, and other co-located events.



	Concert			Restaurant			Other Events			
	Name	Dist.	❖	Name	Distance	❖	Name	Distance	Start Date	❖
☐	Jazz at the Rrazz Featuring Bruce Forman - Guitar With Mike Greensill	0.76		New Delhi Restaurant	0.14		Riverdance	0.22	12/27/2009	
☐							Tricks of Light: Silent Films	0.73		
☐				First Crush Restaurant Wine Bar & Lounge	0.15		Judy Butterfield Home for the Holidays Show	0.76	12/27/2009	
☐							The Howard Stone Show!	0.91		
☐	Ahmad Jamal	0.76		Arang Restaurant	0.12		Bill Fontana: Spiraling Echoes	0	12/11/2009	
☐							Gods and Goods: Global Art and Trade	0.21		
☐				Rasselas on Fillmore Jazz Club	0.14		1330 Fillmore Street	0.76	12/13/2009	
☐							The Edward Albee's Who's Afraid of Virginia Woolf? Presented by Actors	0.85	12/19/2009	

Fig. 1. Clustered liquid results, with Concerts, Restaurants, and Other Events.

3 The Search Computing Framework

Figure 1 shows an overview of the Search Computing framework, constituted by several sub-frameworks. The *service mart framework* provides the scaffolding for wrapping and registering data sources in service marts. 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 as first class citizens: a query can be executed, saved, modified, and published for other users to see. The *service invocation framework* masks the technical issues involved in the interaction with the service mart, e.g., the Web service protocol and data caching issues.

The core of the framework aims at executing multi-domain queries. The *query manager* takes care of splitting the query into sub-queries (e.g., "Where can I attend a

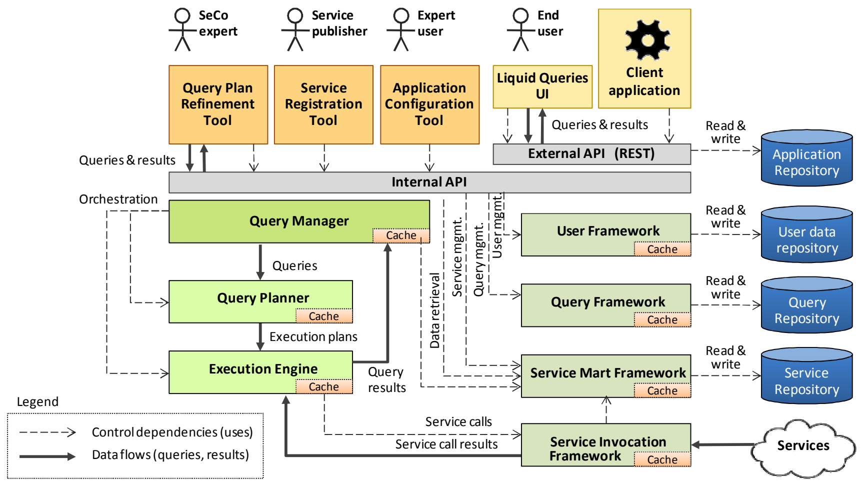


Fig. 2. Overview of the Search Computing framework

scientific conference?"; "Which place is close to a beautiful beach?"; "Which place is reachable from my home location with cheap flights?") and bounding them to the respective relevant data sources registered in the service mart repository (in this case, conferences could be retrieved using Eventful.com, places close to a beach using Yelp.com, flights information using Expedia); starting from this mapping, 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.

To obtain a specific Search Computing application, the general-purpose architecture of Figure 1 is customized by users, supported by appropriate design tools.

4 Search Computing Users

The development of Search Computing applications involves users with different roles and expertise:

- **Service Publishers:** they 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; they register service mart definitions within the service repository, and declare the connection patterns usable to join them.
- **Expert Users:** they 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 user interface, in terms of controls and configurability choices for the end user.
- **End Users:** they 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* [3].

The development process steps lead to the final application accessed by the end user. The Liquid Query interface, instantiated during the application configuration phase, supports the "search as a process" paradigm, based on the continuous evolution, manipulation, and extension of queries and results; the query lifecycle consists of iterations of the steps of **query submission,** when the end user submits an initial liquid query; **query execution,** producing a result set that is displayed in the user interface; and **result browsing**, when the result can be inspected and manipulated through appropriate interaction primitives, which update either the result set (e.g., re-ranking or clustering the results) or the query (e.g., by expanding it with additional service marts or requesting for more results). This approach to development takes into account the trend towards empowerment of the user, as witnessed in the field of Web mash-ups [5]. Indeed, all the design activities from service registration on do not ask to perform low-level programming.

5 Search Computing Architecture

Given the aim of providing users with an efficient Search Computing framework and the expected amount of services, users and methods to solve multi-domain search queries, the adoption of a distributed paradigm is a natural consequence. This entails dealing with the common challenges of a distributed system, e.g. heterogeneity, scalability, concurrency, failure handling and transparency.

With this objective in mind, we have designed the architecture in Figure 2 in terms of logical layers:

- The *service layer* offers facilities to wrap and expose existing services.
- The *execution & mart layer* physically orchestrates the query execution plan by means of an engine component and provides the service mart abstraction through a repository and an invocation environment.

- The *query layer* translates the high-level query description over service marts into an optimized execution plan over service implementations. Queries and optimized plans are stored in a repository for subsequent reuse.
- The *application layer* provides a REST API to submit queries, an application repository to store application-specific data (such as UIs' definitions) and liquid query support. The latter enables a fluid user interaction thanks to client-side data management and to asynchronous communications with the SeCo back-end.
- The *tools layer* transversely provides design, management and interaction facilities to the levels of the proposed architecture.

The architecture is deployed in accordance with the scalability and performance requirements. We use a service-oriented architecture built on shared spaces within a grid computing framework (Space-Based Architecture). The power of spaces comes from not having to separate various parts of an application into discrete physical runtimes [8]. We organize the deployment in three separate tiers (Fig. 2):

- The *service tier* consists of the processing nodes providing access to registered services.
- The *client tier* consists of client machines locally running the liquid query UI, which is offered as a JavaScript component running inside Web browsers.
- The *engine tier* represents the query engine, which is invoked by clients and executes Search Computing queries over registered services.

The deployment of the *engine tier* exploits a high-end application server. In order to achieve scalability and high-performance, a load balancer distributes the execution load over a set of available processing units. A grid manager is in charge of the monitoring of the infrastructure, providing the instantiation and the recovery of grid containers according to a predefined Service Level Agreement (SLA). Finally, grid containers host processing and storage units and communicate by means of a tuple space. We adopt Gigaspaces XAP [6], a scalable, high-performance, reliable data grid implementation, supporting multiple clustering topologies to provide a clustered in-memory data repository (cache and application data), a fast message bus and a distributed platform for running the application's code.

6 Conclusions

This paper presents an overview of the search computing system that has been consolidated after the first 18 months of the project. Some initial ambitions have been reconsidered, thereby allowing us to concentrate our efforts upon a cohesive set of concepts and concrete software architecture. Future plans include improving and extending the components of the execution engine, and completing its deployment onto a parallel architecture. At the same time, we are working on interfaces and tools, so as to support easier user interactions, and we are planning semantic extensions at service registration time, thereby progressively supporting keyword-based or natural language queries in the system.

Acknowledgements. This research is part of the Search Computing (SeCo) project, funded by the European Research Council (ERC), under the 2008 Call for "IDEAS

Advanced Grants", a program dedicated to the support of frontier research. We thank all the project contributors and the advisory board members for their work.

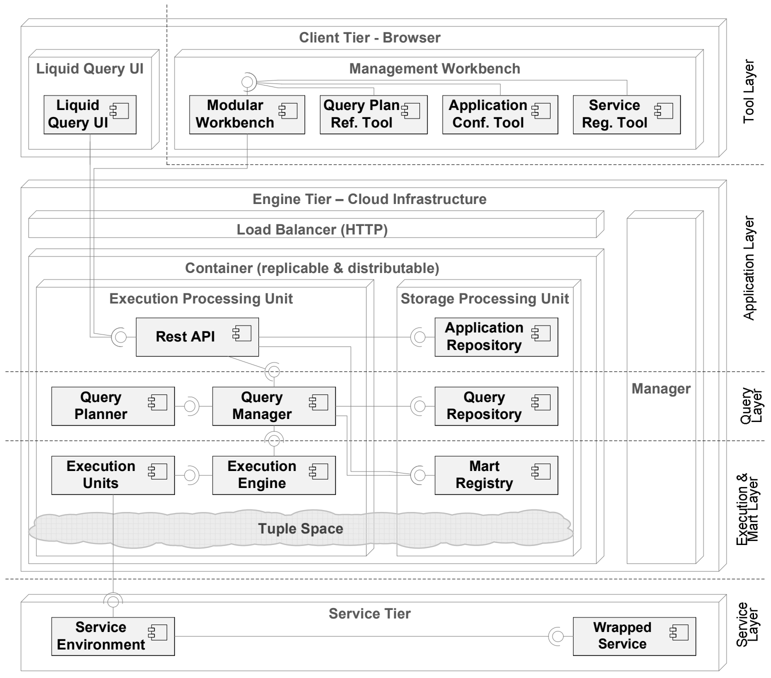


Fig. 3. Logical architecture and deployment diagram

References

[1] Al-Masri E., Mahmoud Q. H. Investigating web services on the World Wide Web. WWW 2008 (Beijing, April 2008). ACM, New York, NY, 795-804.
[2] Baeza-Yates, R., Raghavan, P. Next Generation Web Search. In: Search Computing Challenges and Directions. Ceri, Brambilla (eds.), Springer LNCS vol. 5950 (2010).
[3] Bozzon, A., Brambilla, M., Ceri, S., Fraternali, P. Liquid Query: Multi-Domain Exploratory Search on the Web. WWW 2010, Raleigh, USA, ACM, April 2010.
[4] Ceri, S., Brambilla, M. (Eds.). Search Computing Challenges and Directions. Springer LNCS, Vol. 5950 (2010).
[5] Daniel, F., Yu, J., Benatallah, B., Casati, F., Matera, M., Saint-Paul, R. Understanding UI Integration: A Survey of Problems, Technologies, and Opportunities. IEEE Internet Computing 11(3), 59-66 (2007).
[6] Gigaspaces XAP – http://www.gigaspaces.com
[7] Shafer, J.C., Agrawal, R., Lauw, H.W. Symphony: Enabling Search-Driven Applications, USETIM (Using Search Engine Tech. for Inf. Mgmt.) Workshop, VLDB Lyon (2009).
[8] Shalom, N. Space-Based Architecture and The End of Tier-based Computing. Available at http://www.gigaspaces.com/WhitePapers