Search Computing Systems

Stefano Ceri and Marco Brambilla

Politecnico di Milano, Dipartimento di Elettronica ed Informazione,
V. Ponzio 34/5, 20133 Milano, Italy
{ceri,mbrambil}@elet.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.

B. Pernici (Ed.): CAiSE 2010, LNCS 6051, pp. 1–6, 2010.

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.

2 The Search Computing Framework

A Search Computing application supports users in asking multi-domain queries; for instance, “Where can I attend a scientific conference close to a beautiful beach reachable with cheap flights?”. Search Computing proposes a configurable software architecture that supports any multi-domain query. The main idea is to delegate domain-neutral computation to a generic architecture, which is then configured with domain-dependent information to obtain the desired Search Computing application.

Search computing applications have several distinguished aspects:

- Data sources are usually heterogeneous and distributed; they must be made available to the search computing application as search services that respond to input queries by returning ranked results, possibly chunked into sub-lists.
- Multi-domain queries are answered by selecting a subset of the data sources, by individually extracting their responses, and then by joining the responses thereby building combinations that collectively constitute the results of the query; this is a special case of Web service selection and orchestration, where novel algorithms are needed for joining results sets. This implies that queries can be addressed only through complex orchestration algorithms, which must scale properly.

- Combined and ranked results must be presented to the user for inspection and query refinement, with an exploratory approach that allows augmenting complex results with new findings. This, together with the desired wide adoption of the applications by the user, requires scalability of the platform with respect to the number of users and of requests per user.
- Finally, search computing applications are especially effective for improving the visibility and economical value of so-called long-tail content, which can be used to produce highly customized search solutions. This entails the need of extreme scalability of the platform with respect to the number of registered services.

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. Data sources can be heterogeneous (examples are: a Web site wrapper, a Restful data source, a WSDL web service, a YQL data source, a materialized source, etc.); registration requires a standardization of their interface, so to comply with a service invocation protocol. A service mart is defined as an abstraction (e.g., Hotel) of one or more Web service implementations (e.g., Bookings and Expedia), each capable of accepting queries and of returning results, possibly ranked and chunked into pages. Registration metadata describes the service mart signature and connection information. A connection is defined as an input-output relationship between pairs of service marts that can be exploited for joining them (e.g., the declaration that the output city of the Hotel service mart can be used as an input destination to the Trip service mart). 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.

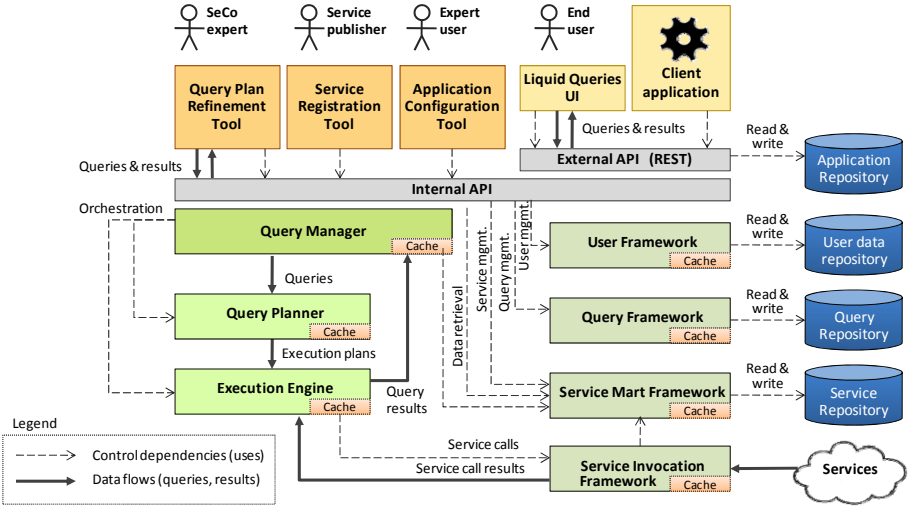

Fig. 1. Overview of the Search Computing framework

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 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.

3 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.

4 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.

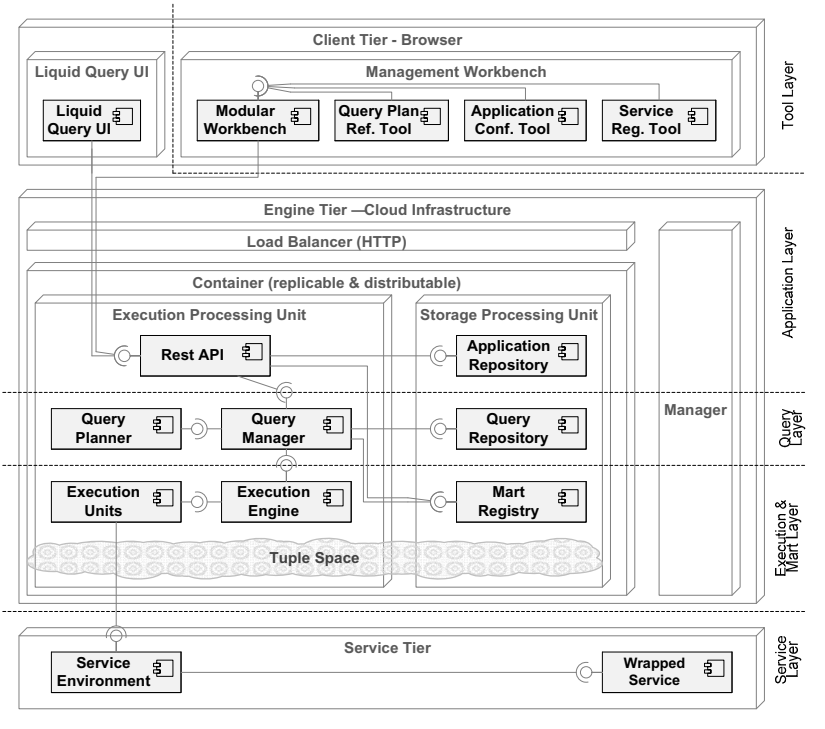

Fig. 2. Logical architecture and deployment diagram

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.

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.

References

[1] Al-Masri, E., Mahmoud, Q.H.: Investigating web services on the World Wide Web. In: WWW 2008, Beijing, April 2008, pp. 795–804. ACM, New York (2008)

[2] Baeza-Yates, R., Raghavan, P.: Next Generation Web Search. In: Ceri, S., Brambilla, M. (eds.) Search Computing. LNCS, vol. 5950, pp. 11–23. Springer, Heidelberg (2010)

[3] Bozzon, A., Brambilla, M., Ceri, S., Fraternali, P.: Liquid Query: Multi-Domain Exploratory Search on the Web. In: WWW 2010, Raleigh, USA, April 2010. ACM, New York (2010) (in print)

[4] Ceri, S., Brambilla, M. (eds.): Search Computing Challenges and Directions. LNCS, vol. 5950. Springer, Heidelberg (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. In: 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, `http://www.gigaspaces.com/WhitePapers`