The Anatomy of a Multi-domain Search Infrastructure

Stefano Ceri, Alessandro Bozzon, and Marco Brambilla

Dipartimento di Elettronica e Informazione,
Politecnico di Milano, P.zza Leonardo Da Vinci 32,
20133 Milan, Italy
{name.surname}@polimi.it

Abstract. Current search engines do not support queries that require a complex combination of information. Problems such as "Which theatre offers an at least-three-stars action movie in London close to a good Italian restaurant" can only be solved by asking multiple queries, possibly to different search engines, and then manually combining results, thereby performing "data integration in the brain." While searching the Web is the preferred method for accessing information in everyday's practice, users expect that search systems will soon be capable of mastering complex queries. However, combining information requires a drastic change of perspective: a new generation of search computing systems is needed, capable of going beyond the capabilities of current search engines. In this paper we show how search computing should open to modular composition, as many other kinds of software computations. We first motivate our work by describing our vision, and then describe how the challenges of multi-domain search are addressed by a prototype framework, whose internal "anatomy" is disclosed.

Keywords: Web information retrieval, multi-domain query, search computing, software architecture, modular decomposition.

1 Introduction

Search is the preferred method to access information in today's computing systems. The Web, accessed through search engines, is universally recognized as the source for answering users' information needs. However, offering a link to a Web page does not cover all information needs. Even simple problems, such as "Which theatre offers an at least-three-stars action movie in London close to a good Italian restaurant", can only be solved by searching the Web multiple times, e.g. by extracting a list of the recent action movies filtered by ranking, then looking for movie theatres, then looking for Italian restaurants close to them. While search engines hint to useful information, the user's brain is the fundamental platform for information integration.

An important trend is the availability of new, specialized data sources – the so-called "long tail" of the hidden Web of data. Such carefully collected and curated data sources can be much more valuable than information currently available in Web pages; however, many sources remain hidden or insulated, in the lack of software technologies for bringing them to surface and making them usable in the search context. We believe that in the future a new class of search computing systems will support the publishing and integration of data sources; the user will be able to select

S. Auer, O. Diaz, and G.A. Papadopoulos (Eds.): ICWE 2011, LNCS 6757, pp. 1–12, 2011.

sources based on individual or collective trust, and systems will be able to route queries to such sources and to provide easy-to-use interfaces for combining them within search strategies, at the same time rewarding the data source owners for each contribution to effective search. Efforts such as Google's Fusion Tables show that the technology for bringing hidden data sources to surface is feasible. We argue then that a new economical model should promote data sharing, giving incentives to owners - for data publishing - and to brokers - for building new search applications by composing them.

The Search Computing project (SeCo) [11][12] aims at building concepts, algorithms, tools and technologies to support complex Web queries, through a new paradigm based on combining data extraction from distinct sources and data integration by means of specialized integration engines. The project has the ambitious goal of lowering the technological barrier required for building complex search applications, thereby enabling the development of many new applications.

In essence, the new requirements imposed to search call for a component-oriented view of search computations. While the current landscape in Web search is dominated by giant companies and monolithic systems, we believe that a new generation of search computing systems needs to be developed, with a much more composite software organization, addressing the needs of a fragmented market. Generic search systems are already dominated by domain-specific vertical search systems, e.g. with travels and electronic bookstores. When the threshold complexity of building such verticals will be lowered, a variety of new market sectors will become more profitable. In several scenarios, search-enabled Web access will grow in interest and value when SMEs or local businesses will see the opportunity of building search applications tailored to their sale region, which integrate "local" and "global" information.

Therefore, building a search computing systems requires coping with several issues. On one side, new methods and algorithms need to be embedded into software modules devised to solve the diverse sub-problems, including query specification, query planning and optimization, query execution, service invocation, user interface rendering, and so on. On the other side, these software modules must be integrated within a software environment supporting collaborative executions. This paper describes first our vision on how search systems should evolve towards modular software components and interfaces; then, as a guarantee of the feasibility of such vision, we describe the "anatomy" of the Search Computing framework that we are currently developing, discussing the main architectural challenges and solutions.

2 Vision

Supporting search over data sources requires a new class of data services, denoted as **Search Services**, and of data integration systems, denoted as **Search Computing Systems**.

2.1 Data Provisioning

An increasing number of data sets is becoming available on the Web as (semi) structured data instead of user-consumable pages. Linked Data plays a central role in this, thanks to initiatives such as W3C Linked Open Data (LOD) community project,

which are fostering LD best practice adoption [2]. An important aspect of LD is their use of universal references for data linking; this aspect raises the hopes of solving the data-matching problem, which has so far limited the practical applicability of data integration methods.

LD and other open or proprietary data are made available through Web APIs (e.g., see Google Places API) and/or search-specific languages (e.g., see the Yahoo Query Language (YQL) framework [19]). Methods, models and tools are being devised for efficiently designing and deploying applications upon such new data services and data access APIs. However, data access and linking so far has not been concerned with data search. The quality of these data sources can be fully exploited with the growth of new search applications, which federate and compose data sources.

Data services must expose ranking criteria and must be optimized for ranked retrieval; top-k queries are well supported by relational sources, while extensions in this direction are being pursued by the Semantic Web community, e.g. through language extensions covering tuple orderings in time and rank (e.g. Continuous-Sparql and RankSparql).

2.2 Service Composition

The efficient support of such data services requires mastering both data and control dependencies, and strategy optimization must consider rank aggregation, optimal result paging, and so on [7]. When data sources must be joined, the join operation must take into account ranking; join can either be based on exact methods, according to the rank-join theory, or on approximate methods, that favor the speed of result production [15]. Data integration strategies should aim at obtaining a suitable number of results (e.g., neither too few, nor too many). Normally, a computation should not be set up so as to exhaust a searchable data source, as the user is rarely interested to inspect all of them. The ultimate controller is the user, who sees service results or their compositions and can halt their production.

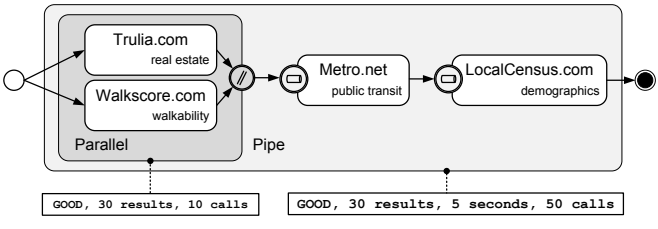

Fig. 1. Search Service integration example, high level view

To be concrete, think about developing a real estate search system in a given region, by using both global and local resources. The high-level process orchestration offered by such system may consist in the composition of data sources providing real estate offers (e.g., Trulia.com or Zillow.com) with local services (e.g., Metro.com for public transport) and indexes (e.g., WalkScore.com or LocalCensus.com), as shown in Figure 1. In the example, the Trulia and WalkScore services are invoked in parallel to extract housing offers and the walkability index (i.e., a value describing if the

neighborhood is walking-friendly, in terms of services, shops, and restaurants at walking distance) of the various districts. This parallel execution is set to extract at most 30 good results by using at most 10 service calls. Then, results are piped to the Metro and LocalCensus services, which extract demographics and crime rate information for each solution. Three global thresholds are set on the number of results, on the time limit and on the number of service calls (30 results, 5 seconds, 50 calls respectively). Thresholds correspond to execution constraints: execution halts if any threshold is met. Notice that this is not a simple mashup of service, as it includes, e.g., complex optimization strategies and result composition logics [6].

From a system-oriented point of view, search-oriented engines for data integration should be deployed in "cloud computing" environments, to enable flexibility and scalability for application developers. In this way, the complexity of engine design and optimization would be totally outsourced and become transparent to the developer. In essence, the developer should not only resort to external data sources, but also to external data integration systems.

2.3 User interaction

From the end user standpoint, the way in which the informative need can be described may be crucial for determining the acceptance of a search application. Hence, different query formulations and user interactions should be allowed. In particular, one can think of three types of query: one-shot form-based queries, natural language queries, and exploratory queries.

One-shot queries based on forms correspond to the behavior of several domain-specific search engines in which the objective of the search is predefined and the user submits a fixed set of parameters for finding the information. Typical examples are flight search or hotel search applications. In case of multi-domain queries one can expect a more complex form, comprising more fields and parameters. In this case the set of focal entities are typically known a priori and the input parameters are exactly matching the exposed fields in the form.

Natural language queries offer instead the maximum freedom to the user, who can express his information needs in a free text form. The submitted sentence might contain query parameters, references to the objects of interest, and possibly information on the expected results (i.e., the expected output of the search). NL queries represent an example of incremental query methods, where incrementality is achieved with a process by which, given his/her information need, the user is led to full specification of the most suitable search services to satisfy it. This scenario is much harder to deal with, as not only the text might not specify the complete set of expected input parameters for producing a response, but it also requires a separation of the original query into a number of sub-queries based on probabilistic mappings of concepts and properties.

Finally, **exploratory queries** are queries where, given the current context of interaction, the user is able to follow links to connected concepts, thus adding a new query fragments or rolling back to a previous result combination. Exploratory queries are by nature incremental. To give users some flexibility in exploring the result, we have proposed the Liquid Query paradigm [3], which allows users to interact with the search computing result by asking the system to produce "more result combinations",

or "more results from a specific service", or "performing an expansion of the result" by adding a sub-query which was already planned while configuring the query.

By means of such paradigm, the user is supported in expressing fully exploratory queries, starting from an initial status with no predefined query, and enabling a progressive, step-by-step construction of the query itself. The new paradigm consists of exploring a network of connected resources, where each resource corresponds to a clearly identified real-world concept (an "hotel", a "flight", a "hospital", a "doctor"), and the connections have predefined semantics ("hotels" are close to "restaurants", "doctors" care "diseases" and are located at "hospitals"). Such network, called the "Semantic Resource Framework", is built as a conceptual description of the search services exploited by the multi-domain framework. The proposed exploration paradigm exploits query expansion and result tracking, giving the user the possibility to dynamically selecting and deselecting the object instances of interest, and move "forward" (adding one node to the query) or "backward" (deleting one node) in the resource graph. Result presentation paradigms support the exploration by visualizing instances of different objects in separate lists, at the same time displaying the combinations they belong to and their global rank. This view allows users to focus at each step on the new results, and therefore is most suitable for a progressive exploration.

2.4 Application Development

A complex search application can be generated from such a high-level design after a preliminary data source registration. During such process, sources become known in terms of: the concepts that they describe (conceptual view), their "access pattern", i.e. the input-output parameters and supported ranking (logical view), and the actual supported interaction protocol, with a variety of quality parameters (physical view). The registration process should be as simple and encompassing as possible, and include a mix of top-down acts (when starting from concepts) and bottom-up acts (when starting from data source schemas or service calls).

We envision semi-automatic tagging of sources during registration and use, and we suggest that tags be extracted from general ontologies (such as Yago) so as to build an abstract view of the sources that can be helpful in routing queries to them [18]. The search system Kosmix [17] already uses such an approach for describing data sources that is sufficient for selecting relevant sources and driving simple queries to them, but it currently does not support the combination of data sources through operations. Combining data sources within queries requires going beyond the registration or discovery of data sources as entities, and supporting the registration of the relationships semantically relating such entities, at the conceptual, logical, and physical level. Relationship discovery is the most difficult step and the key to ease service registration and application development; early work on linked data and on knowledge extraction (e.g. from social networks) proves that relationship discovery is becoming feasible.

Application developers could act as brokers of new search applications built by assembling data sources, exposed as search services. We envision a growing market of specialized, localized, and sophisticated search applications, addressing the long tail of search needs (e.g., the "gourmet suggestions" about slow-food offers in given geographic regions). In this vision, large communities of service providers and brokers (e.g., like ProgrammableWeb.com and Mashape.com for mashups) could be

empowered by support design environment and tools for executing search service compositions and orchestrations. Thanks to the lowering of programming barriers one could expect an ultimate user's empowerment, whereby end users could compose data sources at will from predefined data source registries and collections; e.g., the orchestration of Figure 1 could be built by starting from simple, menu-driven interfaces where the user is just asked to select the concepts and the orchestration is then inferred. We also envision that, with suitable ontological support and possibly within a narrow domain, queries could be generated from keywords, as with conventional search engines.

3 Reference Architecture

The Search Computing project covers many research directions, which are all required in order to provide an overall solution to complex search. The core of the project is the technology for search service integration, which requires both theoretical investigation and engineering of efficient technological solutions. The core theory concerns the development of result integration methods that not only denote ``top-k optimality'', but also the need of dealing with proximity, approximation, and uncertainty. A number of further research dimensions complement such core. Service integration requires solving schema integration problems, as well as ontological description and labeling of resources to facilitate queries. Efficient execution requires optimization, caching, and a server configuration supporting scalability through distribution and parallelism. Support of user interaction requires powerful client-side computations, with rich interfaces for assisting users in expressing their needs and exploring the information. Design tools for building Search Computing applications employ mashup-based solutions and specialized visualization widgets.

Therefore, Search Computing systems deal with many typical Web engineering problems. Figure 2 shows the architecture of the Search Computing system. The software modules in Figure 2 are vertically subdivided into processing modules, repositories, and design tools, and vertically organized as a two-tier, three-layer infrastructure, with the client tier dedicated to user interaction and the server tier further divided into a control layer and execution layer; the client-server separation occurs between processing layers and repositories, and communications are performed using Web-enabled channels. Tools address the various phases of interaction.

3.1 Processing Modules

We describe processing modules bottom-up, starting with the *Execution Layer*. The lower module, the *Service Invocation Framework,* is in charge of invoking services that query the data sources. Such services typically have few input parameters (which are used to complete parametric queries) and produce results constituted by a "chunk" of tuples, possibly ranked, each equipped with a tuple-id; thus, a service call maps given input parameters to a given chunk of tuples. The framework includes built-in wrappers for the invocation of several Web based infrastructures (e.g., YQL, GBASE), query end-point (e.g. SPARQL) and resources (e.g., WSDL- and REST-based Web services). It also supports the invocation of legacy and local data sources.

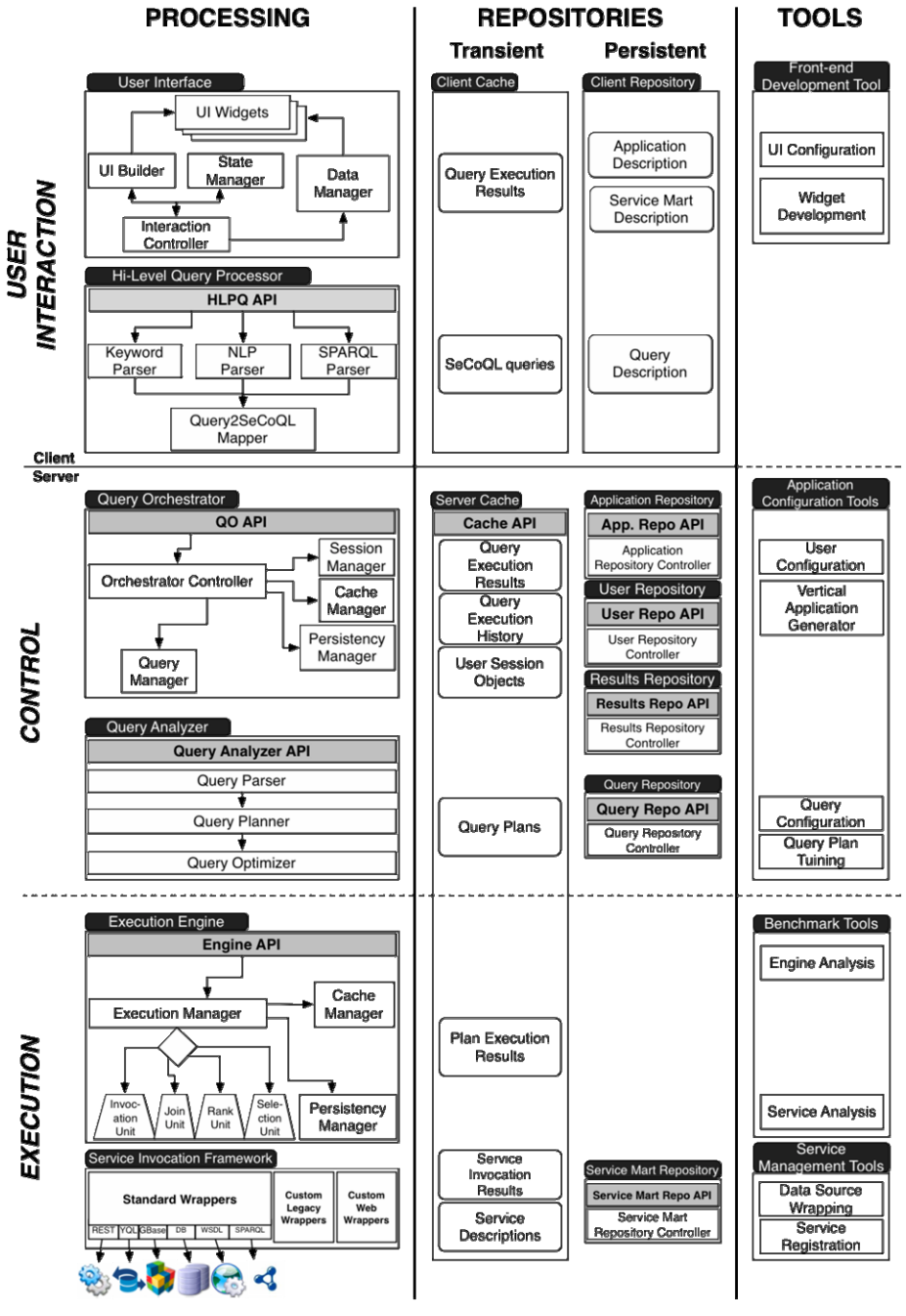

Fig. 2. The architecture of the Search Computing platform

The *Execution Engine* is a data- and control-driven query engine specifically designed to handle multi-domain queries [9]. It takes in input a reference to a query plan and executes it, by driving the invocation of the needed services trough the *Service Invocation Framework*. The *Execution Engine* includes a *Cache* and a *Persistency Manager* module, devoted to in-session and cross-session level caching of results.

The *Control Layer*, as the name suggests, is the controller of the architecture; it is designed to handle several system interactions (such as user session management and query planning) and it embodies the *Query Analyzer* and the *Query Orchestrator*. The *Query Analyzer* is a component devoted to the parsing, planning, and optimization of queries (possibly expressed in a declarative language) into query plans to be executed by the *Execution Engine*. The *Query Orchestrator* is the main access point to the platform, as it acts as a proxy toward all the internal components, offering a set of APIs. The *Query Orchestrator* also handles the management of user sessions and authentication, and communicates with the *Execution Engine* to perform queries; each query is univocally identified, as well as the result set produced by its execution.

The *User Interaction Layer* is the front-end of the SeCo system, and it embodies the *User Interface* and the *High-Level Query Processor*. The latter is a component designed to allow users to express unstructured queries (e.g., a set of keywords, a natural language sentence, or a structured English sentence) or declarative queries expressed over an ontology (or, more in general, a schema), which are translated into processable multi-domain queries. The *User Interface*, instead, is a client-side Rich Internet Application dynamically configured according to a designed SeCo application. It accepts in input user commands and produces queries and query expansion commands for the *Query Orchestrator*.

The rationale of this architecture is a clear separation of modules by functions. Thus, the lower components separate the service interaction logics from the service composition logics, the separation between the orchestrator and the engine guarantees that the former is focused on session management while the latter is focused on efficient execution of one-shot queries, thereby simplifying the engine – which is the most critical component. The design of the *Query Planner* from as a component external to the query engine allows its invocation from the *Query Orchestrator*, thus offering planning services to several other components Finally, the separation of the client-side query processor from the user interface guarantees that all the specific processing depending on the type of input is performed in the former, while all the device-specific aspects of presentation are considered in the latter.

3.2 Repositories

The *Repository* contains the set of components and data storages used by the system to persist the artifacts required for its functioning. On the server side, the *Service Mart Repository* contains the description of the search services consumed by the system, represented at different level of abstraction; it is accessed by the *Query Orchestrator* to provide users with information for navigating the service resource graph while it provides to the *Execution Engine* the access to services. The *Query Repository* and *Results Repository* are used in order to persist query definitions and query results which are heavily used, while the *User Repository* and *Application Repository* store, respectively, a description of users accessing an application and of the configuration

files (query configuration, result configuration, etc.) required by the user interface for an application.

On the client side, three persistent repositories store have been designed, respectively: the *Query Descriptions*, which are used by the High-level Query Processor as reference definitions of the query models; the *Service Mart Descriptions* and the *Application Descriptions*, managed by the user interface on the user's browser to persistently cache application's configuration files and service descriptions, thus reducing the time required for the application to be downloaded and started.

3.3 Caching

The efficient handling of query executions is guaranteed by the combined design of the processing module and of a shared and distributed caching system, which maximizes artifacts reuse while minimizing redundant computation and network data transfer: for instance, caching the results of service invocations allows for faster query answers (as the execution time of a query is typically dominated by the response latency of the search service); repeated executions of the same query or subquery allows for reduced computations, thus enhancing the scalability of the system. Client-side caching of query results minimize the need for client-server round-trip, thus improving the user experience.

Figure 3 illustrates an extract of the sequence diagram of the interactions occurring within the processor layer and its cache system: chunks extracted by each service calls are cached; query plans are cached with the data which are produced by its execution; query execution histories are cached together with user inputs and results at each step. Caches indexes support a fast lookup, e.g. query results are accessed by query plan, user input and chunk number; cache items remain valid throughout a user session (to guarantee consistency in the retrieved data, while improving the reactivity of the system) or across several sessions (to maximize reuse, when possible), unless explicitly invalidated. Query execution occurs by systematically checking whether a resource exists in the cache, else it is produced: when a query created by the user interface, the *Query Orchestrator*, which manage user session histories, looks up in the cache for existing plans associated with such query; if not hit is found, the *Query Orchestrator* interacts with the *Query Planner* to create a new query plan, and looks again in the cache for existing results associated with the plan invocation. If no results have been cached, the control is redirected to the *Execution Engine*, which, in turn, progressively invokes the services included in the query plan to retrieve (possibly cached) data and compose results. A service invocation in a query plan can be directed to another query plan; therefore, the execution engine is able to recursively execute plans, re-using the results of previous executions when possible. The *Query Orchestrator* also handles user session objects, to manage the navigation *history* of the user during exploratory search tasks: the state of navigation is made of a set of user interactions on the system, together with the produced results.

3.4 Tools

To support and configure the set of objects and components involved in the SeCo architecture, a tool suite has been devised that comprises a variety of instruments. The

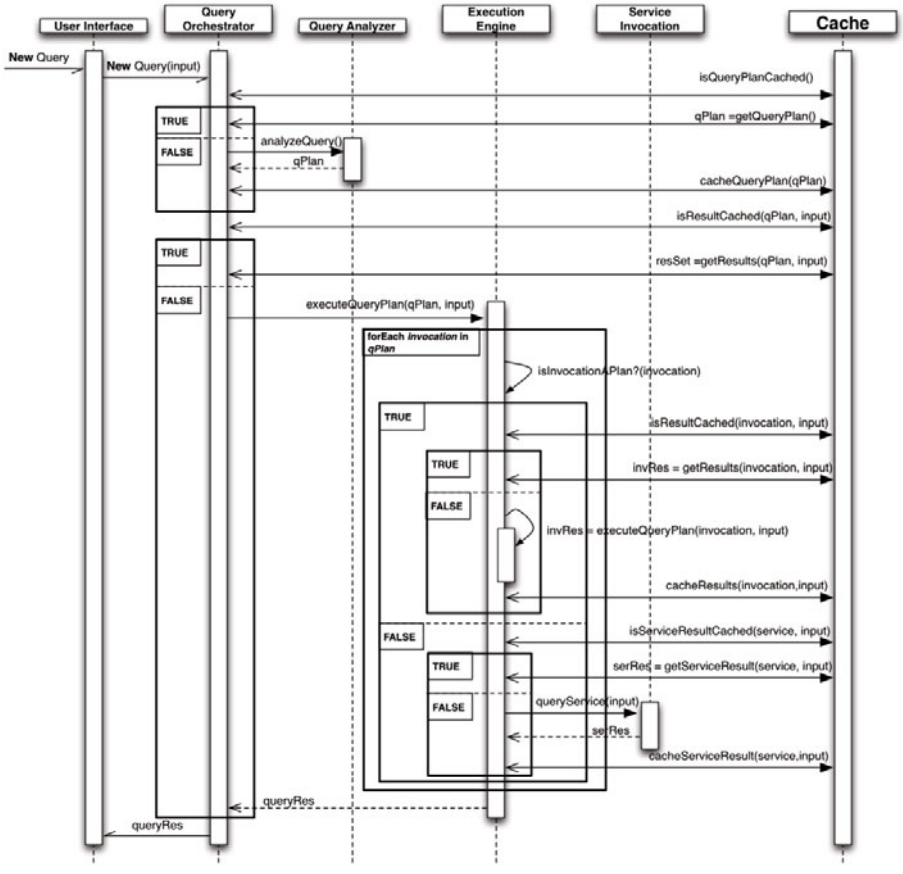

Fig. 3. Sequence diagram showing processing and cache modules

tool suite has been structured as an online development platform in which developers can login and, according to their role, access the right set of tools for building SeCo applications. The availability of the tools as online applications aims at increasing SeCo application design productivity, by reducing the time to deployment and avoiding the burden of downloading and installing software.

At the service management level, tools are crucial for service registration and wrapping. The *Service wrapping tool* allows normalization of service interfaces by supporting the definition of service wrappers, i.e., software components that make heterogeneous services fit into the search computing search services conventions. The *registration tool* [10] consists of a set of facilities for the mapping of concrete services to their conceptual descriptions in terms of service marts, access patterns, and connection patterns. We are currently extending the service registration process so that terms used in naming concepts are extracted from Yago [17], a general-purpose ontology. The tools consist of mapping-based interfaces that allow picking elements from the service input/outputs (and domain descriptions) and populating the

conceptual service mart models. Furthermore, a *service analysis tool* supports monitoring and evaluation of the service performance. At the engine level, the *execution analysis* tool is a benchmarking dashboard that allows monitoring and fine-tuning of the engine behavior. The execution of queries can be visually monitored, recorded, and compared with other executions (e.g., for evaluating the best execution strategies of the same query, or for assessing the bottlenecks of an execution) [5].

The *application configuration tools* allow designers to define applications based on the composition of search services: the *query configuration tool* supports the visual definition of queries as subsets of the service description model with predefined condition templates. The tool supports the designer in exploring the service repository, through visual navigation, selecting the services of interest for the application and the respective connection patterns, and defining the conditions upon them. The *query plan tuning tool* is visual modeling environment that allows SeCo experts to refine the edit query plans specified according to the Panta Rhei notation. The *user configuration tool* allows one to define user profile properties and the *vertical application generator* produces application models that are stored in the application repository.

The client side issues are addressed by two tools: the *UI configuration tool* aims at supporting the design of the interface of the query submission form and of the result set, together with the default settings for the application and the allowed Liquid Query operations [3]. The *widget development environment* consists of a framework in which the developer can encode his own new visual components to be inserted into the pages (e.g., new result visualizers or new query submission widgets).

4 Conclusions

This paper presented our vision for a novel class of search systems, advocating that a new generation of search infrastuctures with a modular software organization is required for addressing the needs of a fragmented market of new search applications. We also showed how this vision is partially instantiated by the prototype architecture currently under development within the Search Computing project. Demos providing some evidence of the feasibility of this approach are presented at WWW [4], ACM-Sigmod [5] and ICWE [1].

Acknowledgements. This research is part of the Search Computing (SeCo) project, funded by ERC, under the 2008 Call for "IDEAS Advanced Grants" (http://www.search-computing.org). We wish to thank all the contributors to the project.

References

[1] Barbieri, D., Bozzon, A., Brambilla, M., Ceri, S., Pasini, C., Tettamanti, L., Vadacca, S., Volonterio, R., Zagorac, S.: Exploratory Multi-domain Search on Web Data Sources with Liquid Queries. In: ICWE 2011 Conference, Demo session, Paphos, Cyprus (June 2011)

[2] Bizer, C., Heath, T., Idehen, K., Berners-Lee, T.: Linked Data on the Web. In: Proceedings WWW 2008, Beijing, China (2008)

[3] Bozzon, A., Brambilla, M., Ceri, S., Fraternali, P.: Liquid Query: Multi-Domain Exploratory Search on the Web. In: WWW 2010, Raleigh, NC, pp. 161–170. ACM, New York (2010)
[4] Bozzon, A., Brambilla, M., Ceri, S., Fraternali, P., Vadacca, S.: Exploratory search in multi-domain information spaces with Liquid Query. In: WWW 2011 Conference, Demo session, (March 31, 2011)
[5] Bozzon, A., Brambilla, M., Ceri, S., Fraternali, P., Vadacca, S.: Exploratory search in multi-domain information spaces with Liquid Query. In: Bozzon, A., Brambilla, M., Ceri, S., Corcoglioniti, F., Fraternali, P., Vadacca, S. (eds.) Search Computing: Multi-domain Search on Ranked Data, ACM-Sigmod 2011 Conference, Demo session (June 2011)
[6] Braga, D., Campi, A., Ceri, S., Raffio, A.: Joining the results of heterogeneous search engines. Inf. Syst. 33(7-8), 658–680 (2008)
[7] Braga, D., Ceri, S., Daniel, F., Martinenghi, D.: Optimization of Multi-Domain Queries on the Web. In: VLDB 2008, Auckland, NZ (2008)
[8] Braga, D., Ceri, S., Daniel, F., Martinenghi, D.: Mashing Up Search Services. IEEE Internet Computing 12(5), 16–23 (2008)
[9] Braga, D., Grossniklaus, M., Corcoglioniti, F., Vadacca, S.: Efficient Computation of Search Computing Queries. In: Ceri, S., Brambilla, M. (eds.) Search Computing II. LNCS, vol. 6585, pp. 141–155. Springer, Heidelberg (2011)
[10] Brambilla, M., Tettamanti, L.: Tools Supporting Search Computing Application Development. In: Ceri, S., Brambilla, M. (eds.) Search Computing II. LNCS, vol. 6585, pp. 169–181. Springer, Heidelberg (2011)
[11] Ceri, S., Brambilla, M. (eds.): Search Computing II. LNCS, vol. 6585. Springer, Heidelberg (March 2011)
[12] Ceri, S., Brambilla, M. (eds.): Search Computing. LNCS Book, vol. 5950. Springer, Heidelberg (March 2010) ISBN 978-3-642-12309-2
[13] Danescu-Niculescu-Mizil, C., Broder, A.Z., Gabrilovich, E., Josifovski, V., Pang, B.: Competing for users' attention: on the interplay between organic and sponsored search results. In: WWW 2010, Raleigh, NC, pp. 291–300. ACM, USA (2010)
[14] Google: Fusion Tables (2009), http://tables.googlelabs.com/
[15] Ilyas, I., Beskales, G., Soliman, M.: A survey of top-k query processing techniques in relational database systems. ACM Comput. Surv. 40(4) (2008)
[16] Parameswaran, A., Das Sarma, A., Polyzotis, N., Widom, J., GarciaMolina, H.: Human-Assisted Graph Search: It's Okay to Ask Questions. In: PVLDB, vol. 4(5), pp. 267–278 (February 2011)
[17] Rajaraman, A.: Kosmix: High Performance Topic Exploration using the Deep Web. In: VLDB 2009, Lyon, France (2009)
[18] Suchanek, F., Bozzon, A., Della Valle, E., Campi, A.: Towards an Ontological Representation of Services in Search Computing. In: Ceri, S., Brambilla, M. (eds.) Search Computing II. LNCS, vol. 6585, pp. 101–112. Springer, Heidelberg (2011)
[19] YQL (2009), http://developer.yahoo.com/yql/