Zuerst ersch. in : IEEE Internet Computing ; 14 (2010), 6. - S. 14-22
http://doi.ieeecomputersociety.org/10.1109/MIC.2010.106

Search Computing

Managing Complex Search Queries

Search computing focuses on building answers to complex search queries (for example, “Where can I attend an interesting conference in my field near a sunny beach?”) by interacting with a constellation of cooperating search services, using result ranking and joining as the dominant factors for service composition. The service computing paradigm has so far been neutral to the specific features of search applications and services. To address this weakness, search computing advocates a new approach in which search, join, and ranking are the central aspects for service composition.

Stefano Ceri, Adnan Abid, Mamoun Abu Helou, Davide Barbieri, 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, and Salvatore Vadacca
Politecnico di Milano

In the past decade, search has become the most popular way to access information over the Internet. But as users have grown more sophisticated, their queries have become increasingly challenging for search engines. For example, users typically approach search as an interactive process rather than a single query. Single queries are also more complex in terms of the amount and extension of information requested. Often, the information to be retrieved is hidden in the deep Web, which contains information perhaps more valuable than what can be crawled on the surface Web. Furthermore, results returned by different search engines are often ranked according to diverse criteria, thereby raising the challenging task of combining them into globally good answers to the original complex query.

In addition to general-purpose search engines, different classes of search systems have emerged to cover this information need.[1] Metasearch engines, for example, query several engines and build a unified result set, whereas vertical search engines aggregate domain-specific information from a fixed set of relevant sources and let users pose more sophisticated queries (for example, finding proper combinations of flights, hotels, and car rental offers). Exploring such a wide spectrum of candidate combinations while controlling result size and user satisfaction requires methods, such as top-*k*

Related Work in Software Development

Our work stands at the intersection of four main research trends in software development: service integration, model-driven Web engineering, mashups, and search-driven application development.

The service integration field has produced many contributions: service definition (Web Service Definition Language and ontologies such as Web Service Modeling Ontology and OWL-S), service registration (UDDI and derived approaches), service orchestration (Web Services Business Process Execution Language [WS-BPEL], Business Process Modeling Notation [BPMN], and interchange formats such as the XML Process Definition Language). Associated tools include BPEL-based service orchestration tools and more general BPM tools. The Search Computing (SeCo) project takes inspiration from the service-oriented architecture vision, defining service-based invocation, collaboration among services to achieve a common result, and orchestration of query plans.

From model-driven engineering approaches (such as WebML[1]), SeCo borrows visual design and composition of applications. Modeling is ubiquitous in the search computing approach, where it's applied in all development phases.

Mashup approaches (such as Yahoo Pipes) are even more inspiring, because they comply with the expert user need of an easy online tool to quickly configure and deploy applications. Our design tools rely on mashup-like approaches.

Finally, search-based application development aims to cover the increased sophistication and diversification of search application requirements. Microsoft's Symphony platform, for example, lets nondevelopers build and deploy search-driven applications that combine their data and domain expertise with content from search engines and other Web services.[2] Other approaches — such as Google Base API (http://code.google.com/apis/base) and Yahoo Query Language (http://developer.yahoo.com/yql) — target skilled software developers and rely on APIs or query languages. Google Squared (www.google.com/squared) and Fusion Tables (http://tables.googlelabs.com) produce tabular results of searches over Web and proprietary information, respectively. Kosmix (www.kosmix.com) is a general-purpose topic discovery engine that responds to keyword search using a topic page.

All of these proposals miss several significant features that SeCo provides, including join of results, cost awareness, and definition and optimization of query plans.

References

1. S. Ceri et al., *Designing Data-Intensive Web Applications*, Morgan Kaufmann, 2002.
2. J.C. Shafer, R. Agrawal, and H.W. Lauw, "Symphony: Enabling Search-Driven Applications," *Proc. VLDB Workshop Using Search Engine Technology for Information Management* (USETIM), 2009; http://vldb2009.org/files/Proceedings USETIM/paper-4.pdf.

techniques,[2] that effectively prune the search space and focus on the best candidates.

However, none of these systems addresses the case in which a user performs a complex search process over different domains, possibly covered by distinct vertical search engines.[3] Users perform this type of search by separately looking for each piece of information and then mentally collating partial results to get a combination of objects that satisfies their needs. Such a procedure is cumbersome and error-prone.

The Search Computing (SeCo) project seeks to define a new class of applications that respond to multidomain queries[4] – that is, queries over multiple semantic fields of interest. These *search computing applications* aim to help (or replace) users in decomposing queries and manually assembling complete results from partial answers, each of which might also have its own ranking. A preliminary description of the SeCo project and the set of challenges it poses to various IT fields are available elsewhere.[5] The "Related Work in Software Development" sidebar discusses search computing's relationship to other trends in software development.

Core Concepts

Throughout this article, we use a running example in which a user is planning a leisure trip and wants to search for upcoming concerts (described in terms of music type, such as jazz, rock, and pop) near a specified location (described in terms of place type – beach, lake, mountain, seaside, and so on) with good hotels nearby. The user can expand the query with information about nearby high-quality restaurants for the candidate concert locations, news associated with the event, photos taken near the location, and possible options to combine further events scheduled in the same days and located near the first event. This is a complex search prototype, in which the user doesn't express all needs at once, but they become progressively involved as the user interacts. The search result is a complex combination of several individual choices.

Search computing involves four main concepts: resource framework, queries, query plans, and liquid interaction.

Resource Framework

Search computing is based on a resource frame-

Concert(“San Francisco”, “1.0 m”, “10/30/2010”, “11/30/2010”, “Jazz”, name[O], date [R], lat[O], long[O], distance[R], price[R], address[O])

Restaurant(“vegetarian”, “3stars”, {Concert.lat, Concert.long}, “1.0 m”, name[O], address[O], lat[O], long[O], rating[R], distance[R], url[O])

Hotel(“hotel”, “3stars”, {Concert.lat, Concert.long}, “1.0 m”, name[O], address[O], phone[O], lat[O], long[O], rating[R], distance[R], url[O])

Figure 1. Formal representation of the query, “Where can I find a jazz concert in San Francisco close to a nice vegetarian restaurant and a good hotel?” Attributes are denoted as I = input, O = output, and R = ranked, depending on their role in the service call.

work describing the information sources at different levels of abstraction. Service marts are abstractions of several Web services dealing with the same conceptual objects available on the Web (such as flights, hotels, restaurants, jobs, and houses for sale). They’re modeled at three levels of abstraction — service marts, access patterns, and service interfaces — leading from the conceptual representation of Web objects to the implementation of search services.

At the conceptual level, object properties are represented as a collection of strongly typed attributes that can be atomic, composed, or multivalued. Connection patterns express associations between service marts. These patterns consist of comparison predicates between pairs of attributes, which are interpreted as conjunctive Boolean expressions. They support joins between the objects returned by search services. We extend equality predicates to support partial matching, type coercion, and proximity-based connections (giving special support to time, space, and money distances). The identification and design of service marts is both a top-down and bottom-up process because service marts reflect a real-world object’s conceptual description as well as the availability of physical services providing information on that object.

At the logical level, 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 input (I), output (O), or ranked (R), depending on the role it plays in the service call.

At the physical level, service interfaces are characterized by chunking properties (a service is chunked when it returns objects one subset at a time, with a specified chunk size, to allow the progressive retrieval of all objects in the result set), caching (if the query’s results can be cached, the expiration time associated with cache decay), and cost (expressed as the response time and/or the monetary cost of invocation).

Our approach doesn’t explicitly address data integration, data cleansing, entity resolution, duplicate detection, and other issues that are typical of the general data-integration problem.

Service marts used in the running example include `concert (date, hour, genre, title, lat, long, price, address, singer, duration)` and `restaurant (category, rating, name, address, lat, long, rating, url)`. Connection patterns include `CloseByRest: near(Concert.address, Restaurant.address)`. Access patterns include the following:

- `Concert(location[I], radius[I], minDate[I], maxDate[I], genre[I], name[O], date[R], lat[O], long[O], distance[R], price[R], address[O])`
- `Restaurant(category[I], minRating[I], location[I], radius[I], name[O], address[O], lat[O], long[O], rating[R], distance[R], url[O])`
- `Hotel(category[I], minRating[I], location[I], radius[I], name[O], address[O], phone[O], lat[O], long[O], rating[R], distance[R], url[O])`

At the service mart level, some attributes are missing with respect to the access pattern level (for example, the location the user looks for, the distance radius, and the date range) because they aren’t relevant at the conceptual level for describing the abstract objects `concert` and `restaurant`.

Query

A search computing query is a conjunctive query over services. It includes both the logical query clauses over the relevant data sources and the result ranking criterion. Although queries might initially be expressed over the conceptual description of service marts, eventually all queries are translated into adorned queries over service interfaces. An example complex query is, “Where can I find a jazz concert in San Francisco close to a nice vegetarian restaurant and a good hotel?” Although in principle this approach could consider transformations from the natural language query specification to lower-level descriptions, we don't address this problem here.

We start from the formal representation of this query in the shape shown in Figure 1.

Query Plan

A query plan is a well-defined scheduling of service invocations, possibly parallelized, that complies with the service interface and exploits the ranking in which search services return results to rank the combined query results. To specify these plans, we defined Panta Rhei.[6] We decided to define a new language rather than use or adapt an existing formalism because of the specific requirements of evaluating complex queries over search services. For example, we achieve runtime adaptivity by separating the data flow and the control flow.

Panta Rhei represents a plan as a graph composed of operator nodes and edges, which describe the control and data flow between nodes. Node types include service invocators, joiners, controllers, and modifiers (chunkers, sorters, and selectors). To support top-*k* answers (or approximations thereof), Panta Rhei provides strategy nodes that control the evaluation of joins by scheduling the corresponding service invocations.

Figure 2 shows an example query plan model. In the figure, a pipe join extracts a list of concerts from a search service and uses these results to retrieve nearby restaurants and hotels. The plan invokes the restaurant and hotel services in parallel according to a predefined join strategy. Finally, a join unit executes the join of the two result sets.[7] Two strategy nodes — one for the outer pipe join and one for the inner parallel join — orchestrate the query's execution by observing the output of data units (such as service invocators and joiners) and deciding the service to be invoked accordingly.

Liquid Interactions

A liquid query consists of a set of abstractions supporting exploratory search.[8] The liquid query interface visibly highlights each search service's contribution to the composite result set and helps users fine-tune their requests to the underlying search services by

- expanding the query with an extra search service,
- adding or dropping an object's attributes,

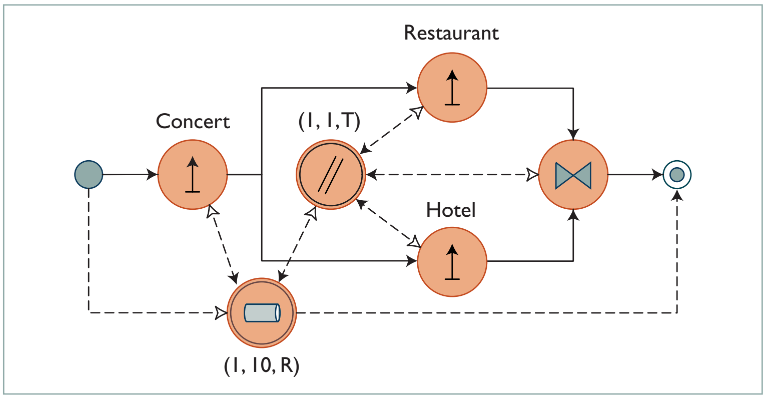


Figure 2. Example of a Panta Rhei query plan. The concert service is invoked first. Next, restaurant and hotel are invoked in parallel based on the inputs received by the concert service. The pipe and parallel control units (marked with a double circle) control the execution flow, the invocation units (represented by circled arrows) call the services, and the join node (circled join symbol) collects and composes the results.

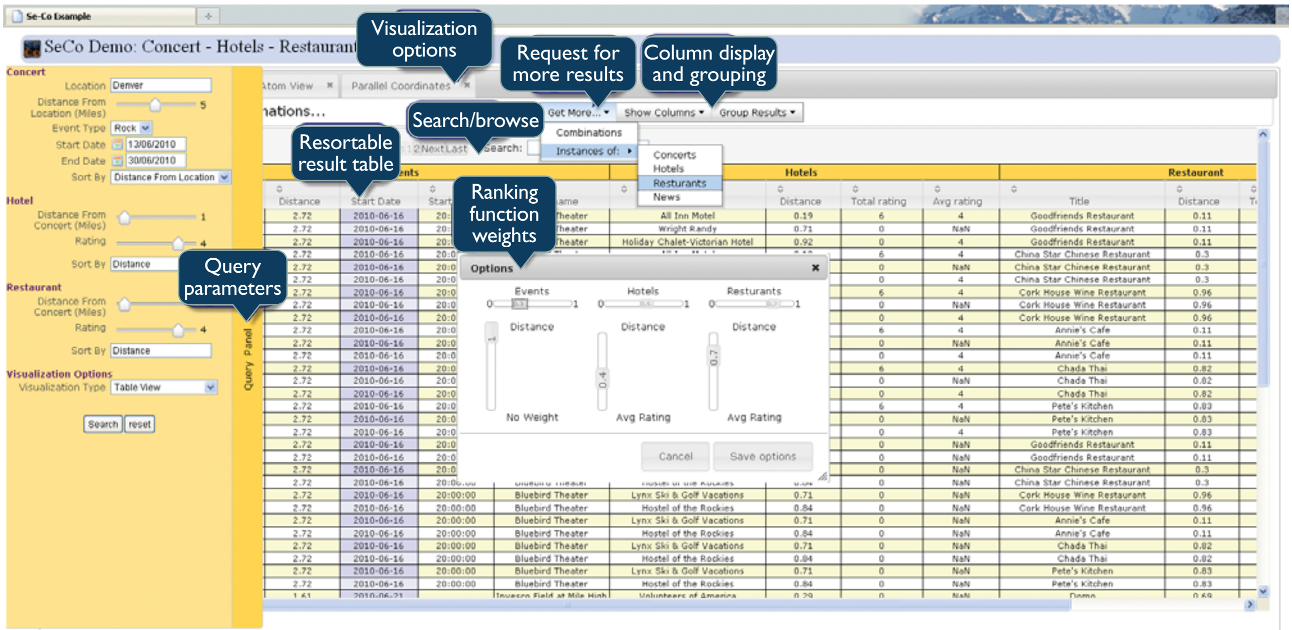
- asking for more results from a specific service,
- aggregating and reordering results;
- adding (drill-down) or removing (roll-up) details, and
- choosing the best data visualization format at the level of individual object, combination, or results set.

In playing with the query and its result set, users alter the result set container's schema. Results then dynamically flow inside the new schema, adapting to it as a liquid to its container. All the provided manipulation operations apply to a tabular data representation similar to that of Google Squared, but with increased functionality derived from the knowledge that each item in the result set is actually a combination of objects coming from different search services. Figure 3 shows an example result table and a set of exploration options. A live demo is available at http://demo.search-computing.org.

Framework

As Figure 4 illustrates, the search computing framework consists of several subframeworks.

The *resource* framework provides the scaffolding for wrapping and registering data sources, which can be heterogeneous (for example, a website wrapper, a RESTful data source, a Web Services Description Language Web service, or a Yahoo Query Language data source). Registering them requires standardizing their interfaces to comply with a service invocation protocol.



SeCo Demo: Concert - Hotels - Restaurant

Concert
Location: Denver
Distance From Location (Miles): 5
Event Type: Rock
Start Date: 13/06/2010
End Date: 30/06/2010
Sort By: Distance From Location

Hotel
Distance From Concert (Miles): 1
Rating: 4
Sort By: Distance

Restaurant
Distance From Concert (Miles)
Rating: 4
Sort By: Distance

Visualization Options
Visualization Type: Table View

Search | reset

Options

Events	Hotels	Resturants
0 — 1	0 — 1	0 — 1
Distance	Distance	Distance
	0.4	0.7
No Weight	Avg Rating	Avg Rating

Cancel | Save options

...ents				Hotels				Restaurant	
Distance	Start Date	Start	...ame		Distance	Total rating	Avg rating	Title	Distance
2.72	2010-06-16	20:	Theater	All Inn Motel	0.19	6	4	Goodfriends Restaurant	0.11
2.72	2010-06-16	20:	Theater	Wright Randy	0.71	0	NaN	Goodfriends Restaurant	0.11
2.72	2010-06-16	20:	Theater	Holiday Chalet-Victorian Hotel	0.92	0	4	Goodfriends Restaurant	0.11
2.72	2010-06-16	20:0				6	4	China Star Chinese Restaurant	0.3
2.72	2010-06-16	20:0				0	NaN	China Star Chinese Restaurant	0.3
2.72	2010-06-16	20:0				0	4	China Star Chinese Restaurant	0.3
2.72	2010-06-16	20:0				6	4	Cork House Wine Restaurant	0.96
2.72	2010-06-16	20:0				0	NaN	Cork House Wine Restaurant	0.96
2.72	2010-06-16	20:0				0	4	Cork House Wine Restaurant	0.96
2.72	2010-06-16	20:0				6	4	Annie's Cafe	0.11
2.72	2010-06-16	20:0				0	NaN	Annie's Cafe	0.11
2.72	2010-06-16	20:0				0	4	Annie's Cafe	0.11
2.72	2010-06-16	20:0				6	4	Chada Thai	0.82
2.72	2010-06-16	20:0				0	NaN	Chada Thai	0.82
2.72	2010-06-16	20:0				0	4	Chada Thai	0.82
2.72	2010-06-16	20:0				6	4	Pete's Kitchen	0.83
2.72	2010-06-16	20:0				0	NaN	Pete's Kitchen	0.83
2.72	2010-06-16	20:0				0	4	Pete's Kitchen	0.83
2.72	2010-06-16	20:0				0	NaN	Goodfriends Restaurant	0.11
2.72	2010-06-16	20:0				0	NaN	Goodfriends Restaurant	0.11
2.72	2010-06-16	20:0				0	NaN	China Star Chinese Restaurant	0.3
2.72	2010-06-16	20:00:00	Bluebird Theater	Hostel of the Rockies	0.84	0	NaN	China Star Chinese Restaurant	0.3
2.72	2010-06-16	20:00:00	Bluebird Theater	Lynx Ski & Golf Vacations	0.71	0	NaN	Cork House Wine Restaurant	0.96
2.72	2010-06-16	20:00:00	Bluebird Theater	Hostel of the Rockies	0.84	0	NaN	Cork House Wine Restaurant	0.96
2.72	2010-06-16	20:00:00	Bluebird Theater	Lynx Ski & Golf Vacations	0.71	0	NaN	Annie's Cafe	0.11
2.72	2010-06-16	20:00:00	Bluebird Theater	Hostel of the Rockies	0.84	0	NaN	Annie's Cafe	0.11
2.72	2010-06-16	20:00:00	Bluebird Theater	Lynx Ski & Golf Vacations	0.71	0	NaN	Chada Thai	0.82
2.72	2010-06-16	20:00:00	Bluebird Theater	Hostel of the Rockies	0.84	0	NaN	Chada Thai	0.82
2.72	2010-06-16	20:00:00	Bluebird Theater	Lynx Ski & Golf Vacations	0.71	0	NaN	Pete's Kitchen	0.83
2.72	2010-06-16	20:00:00	Bluebird Theater	Hostel of the Rockies	0.84	0	NaN	Pete's Kitchen	0.83
1.61	2010-06-21		Invesco Field at Mile High	Volunteers of America	0.29	0	NaN	Domo	0.69

Figure 3. Liquid query multidomain result set table and some user interaction options. Options include visualization perspectives, rank tuning, local sorting, grouping, filtering, and browsing.

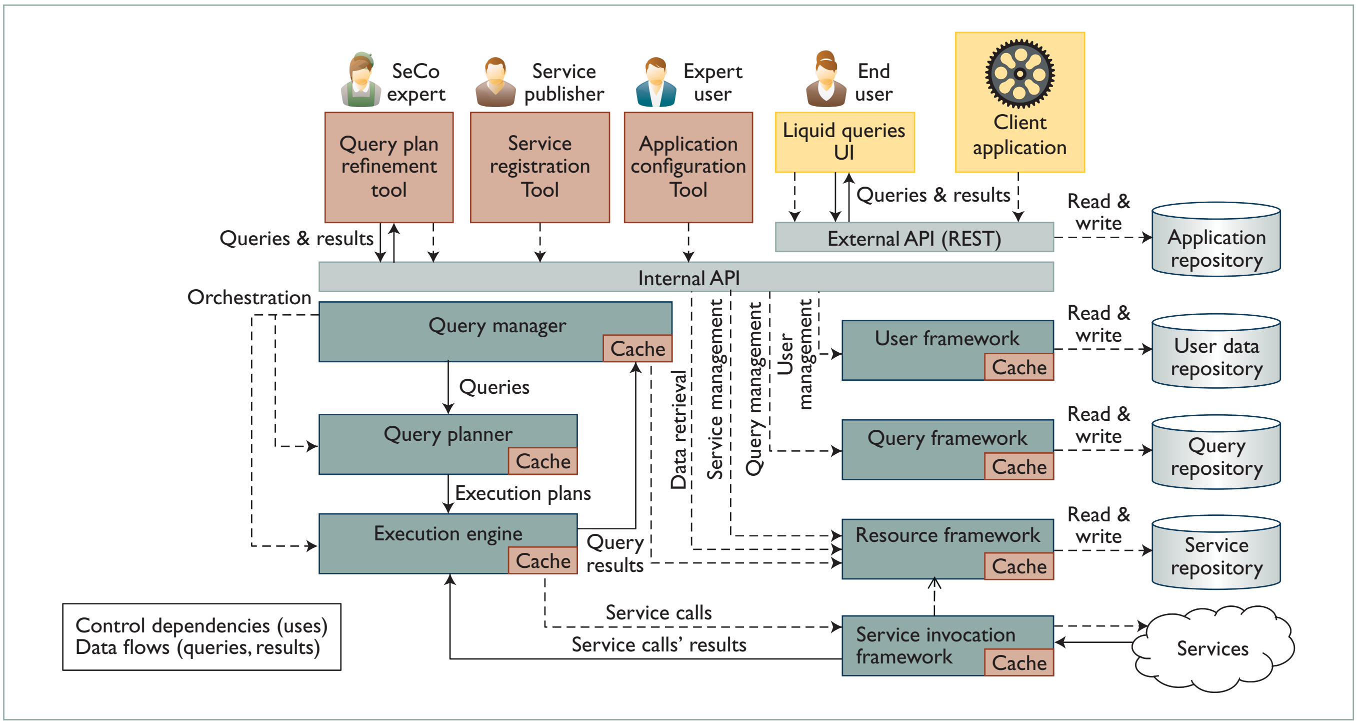


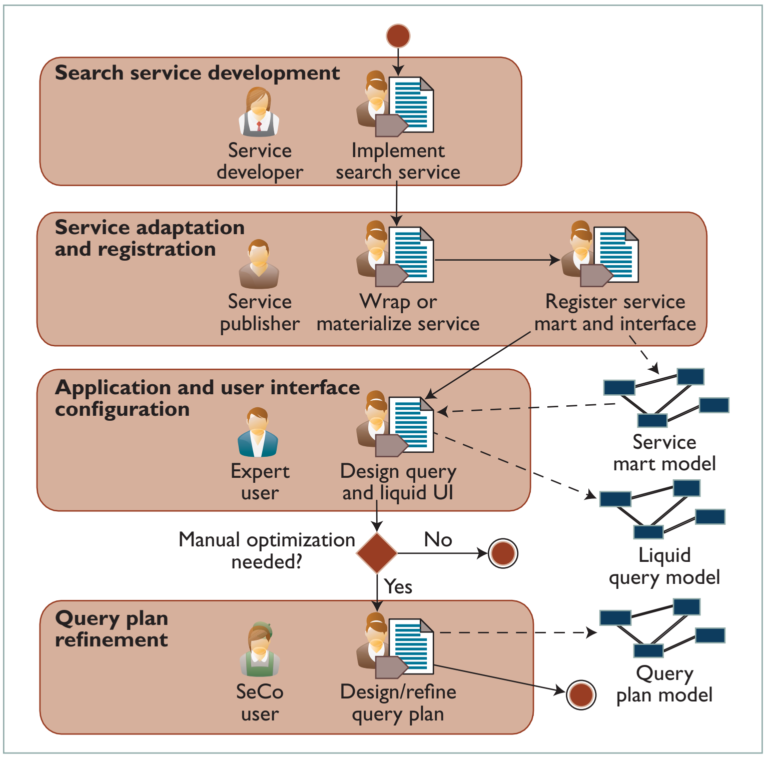
Figure 4. Overview of the search computing framework. A set of repositories and related management frameworks store the information of interest (search services, queries, users, and configured applications); the query manager orchestrates the system behavior, which includes a query planner and a query execution engine; the tools layer supports the user in the configuration of the system.

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.

Finally, the *service invocation framework* masks the technical issues involved in the interaction with the registered service mart — for example, the Web service protocol and data-caching issues.

The framework's core aims to execute multidomain queries. The query manager splits the query into subqueries (for example, "Where can I listen to a good jazz concert?" "Which three-star hotel nearby has the best rates?" and "Where can I find a good vegetarian restaurant close to the hotel?") and binds them to the respective relevant data sources registered in the service mart repository. 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 submits the service calls to designated services through the service invocation framework, builds the query results by combining the outputs produced by service calls, computes the global ranking of query results, and produces the query result outputs in an order that reflects their global relevance.



Search service development
Service developer
Implement search service
Service adaptation and registration
Service publisher
Wrap or materialize service
Register service mart and interface
Application and user interface configuration
Expert user
Design query and liquid UI
Service mart model
Liquid query model
Manual optimization needed?
No
Yes
Query plan refinement
SeCo user
Design/refine query plan
Query plan model

Figure 5. Development process for search computing applications: search services are developed by third parties and registered in the search computing repository. Then, expert users can define the queries, user interfaces, and refined execution plans to be performed.

Application Development

Although our framework is generic, we can build specific search computing applications by progressively registering services, constructing queries, and enabling them through suitable user interfaces. To obtain a specific search computing application, we customize the general-purpose architecture in Figure 4 using tools targeted to different user types:

- *Service developers* implement their own services, independently from their usage within search computing.
- *Service publishers* implement service mediators, wrappers, or data-materialization components, and register service mart definitions.
- *Expert users* configure search computing applications by defining the queries of interest and the liquid user interface.
- *End users* consume search computing applications configured by expert users. They interact by submitting queries, inspecting results, and refining/evolving their information needs according to the liquid query approach.

These users interact in developing search computing applications.

Development Process

As the process scheme in Figure 5 illustrates, development consists of five main stages.

Search service development consists of the development of Web services exposing on the Web the data requested by the application. Programmers perform this activity unassisted by the envisioned tools.

Service adaptation and registration is necessary whenever the Web services that expose information don't adhere to the service mart interface and protocol. Adapting existing services to the framework includes creating service wrappers, possibly designing data-caching policies, normalizing the data, and registering them as service marts. Service publishers perform these tasks. These users need programming skills to construct data and protocol adaptation components. Publishers can use a template-based approach to tailor standard adapter components to the specific Web service to wrap.

Application configuration consists of configuring the service invocation framework and the user interface. Expert users perform this task, which requires selecting several service marts and service implementations (for those service marts associated with multiple sources) from the service repository and declaring their configuration. The configuration includes the input and output parameters (in case a service can be called in different ways) and the connection patterns (used for joining pairs of services). Note that multiple implementations could be available for the same service mart (for example, Expedia and eDreams), and the same pair of services might be connected in different ways.

Query interface configuration requires customizing the liquid query generic interface's structure. In this task, the expert user can choose

- optional selection predicates to restrict the objects retrieved by the query (for example, a maximum price target for events or flights);
- default ranking criteria;
- visual preferences for the result set display (for example, sorting, grouping, and clustering attributes or the size of the result list); and
- a set of extra service marts that end users can use to expand the current query (for example, to expand a query on movies through a service that joins selected movies to their reviews and the theaters where they're appearing to nearby restaurants).

Query plan refinement, performed by the search computing expert, consists of manually refining the optimized query plan produced by the search computing platform, starting from the query specification. In general, query plans are self-tuned, so this phase is usually not needed.

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: search is based on the continuous evolution, manipulation, and extension of queries and results. The query life cycle consists of iterations of the steps of query submission, during which the end user submits an initial liquid query. Query execution produces a result set that is displayed in the user interface. Finally, during result browsing, end users can insert and manipulate the result using appropriate interaction primitives, which update either the result set (for example, by reranking or clustering the results) or the query (for example, by expanding it with additional service marts or requesting additional results).

This development approach accounts for the trend toward user empowerment, as witnessed in the field of Web mashups.[9] Indeed, only service development and service mart adaptation require programming expertise. Search computing moves the other design activities to service registration and application configuration time, so designers don't need to perform low-level programming.

Development Tools

Search computing's success depends on, among other factors, its attractiveness to early adopters, who will need to invest effort in building applications. We've therefore developed a search computing tool suite that minimizes such effort, so as to lower the entrance barrier as much as possible. To support the complex search computing life cycle, our tool suite comprises various instruments, structured as an online development platform in which developers can log in and, according to their role, access the right set of tools for building search computing applications. To support our design choices, a cohesive framework integrates several interacting models, thereby partitioning the design space and responsibilities to the different roles and required expertise in a nontrivial way.

In addition, the mashup approach replaces programming wherever possible, allowing flexible and efficient application development. Different tools support service registration, query definition, liquid interface configuration, and query plan refinement. Developers can work at a single model level (for example, only at the service mart level) or can define a workspace that gives a coordinated view of all the models of a complex project (for example, the service mart, query plan, and liquid interface models for a given application).

Search computing is a multidisciplinary effort that requires adding to sound software design principles contributions from other disciplines such as knowledge representation, human-computer interfaces, user-centered interaction design, business models, economics, and legal sciences.

In addition to the theoretical, technical, and architectural issues already addressed at the current stage of the project, ongoing and future research focuses on additional aspects such as result diversification, complex result visualization, integration with semantic knowledge sources, interaction design, and business models associated with multidomain search.

Acknowledgments

The European Research Council (ERC) funds the SeCo project as part of its call for IDEAS Advanced Grants, a program dedicated to supporting investigation-driven frontier research.

References

1. R. Kumar and A. Tomkins, "A Characterization of Online Search Behaviour," *Data Eng. Bull.*, vol. 32, no. 2, June 2009, pp. 3–11.
2. D. Braga et al., "Optimization of Multi-Domain Queries on the Web," *Proc. Very Large Databases* (VLDB), ACM Press, 2008, pp. 562–573.
3. R. Baeza-Yates and P. Raghavan, "Next Generation Web Search," *Search Computing Challenges and Directions*, S. Ceri and M. Brambilla, eds., LNCS 5950, Springer, 2010, pp. 11–23.
4. S. Ceri and M. Brambilla, eds., *Search Computing Challenges and Directions*, LNCS 5950, Springer, 2010.
5. M. Brambilla and S. Ceri, "Engineering Search Computing Applications: Vision and Challenges," *Proc. 7th Joint Meeting of the European Software Eng. Conf. and the ACM SIGSOFT Symp. Foundations of Software Eng.* (ESEC/FSE 09), ACM Press, 2009, pp. 365–372.
6. D. Braga, "Panta Rhei: A Query Execution Environment," *Search Computing: Challenges and Directions*, LNCS 5950, Springer, 2010, pp. 225–243.
7. D. Braga et al., "Joining the Results of Heterogeneous Search Engines," *Information Systems*, vol. 33, nos. 7–8, 2008, pp. 658–680.
8. A. Bozzon et al., "Liquid Query: Multi-Domain Exploratory Search on the Web," *Proc. 19th Int'l World Wide Web Conf.* (WWW 2010), ACM Press, 2010, pp. 161–170.
9. F. Daniel et al., "Understanding UI Integration: A Survey of Problems, Technologies, and Opportunities," *IEEE Internet Computing*, vol. 11, no. 3, 2007, pp. 59–66.

Stefano Ceri is a professor of database systems in the Department of Electronics and Information at the Politecnico di Milano. His research interests include extending database technology to incorporate data distribution, deductive and active rules, object orientation, XML query languages, and design methods for data-intensive websites, stream reasoning, and search engines. Ceri is the principal investigator of the SeCo project. Contact him at Stefano.Ceri@polimi.it.

Adnan Abid is a PhD student in information engineering at the Politecnico di Milano. His research interests include information retrieval, databases, and ranked query processing. Abid has a master's degree in information technology from the National University of Science and Technology, Pakistan. Contact him at Adnan.Abid@polimi.it.

Mamoun Abu Helou is a PhD student in information engineering at the Politecnico di Milano. His research interests include information retrieval and keyword clustering for service matching. Abu Helou has an MSc in engineering of computing systems from the Politecnico di Milano. Contact him at Mamoun.Abu@mail.polimi.it.

Davide Barbieri is a PhD student in the Department of Electronics and Information at the Politecnico di Milano. His research interests include data stream processing, query processing, and the Semantic Web. Barbieri has a master's degree in computer science from the Politecnico di Milano. Contact him at dbarbieri@elet.polimi.it.

Alessandro Bozzon is a postdoctoral researcher in the Department of Electronics and Information at the Politecnico di Milano. His research interests include user interaction paradigms, data visualization, and result diversification techniques in multidomain search over heterogeneous data collections. Bozzon has a PhD in information engineering from the Politecnico di Milano. Contact him at Alessandro.Bozzon@polimi.it.

Daniele Braga is an assistant professor in the Department of Electronics and Information at the Politecnico di Milano. His research interests include the manipulation of semistructured data, service integration, stream reasoning, and data mapping. Braga has a PhD in information engineering from the Politecnico di Milano. Contact him at Daniele.Braga@polimi.it.

Marco Brambilla is an assistant professor in the Department of Electronics and Information at the Politecnico di Milano. His research interests include conceptual models, tools, and methods for Web applications, services, and search; user interaction; semantic Web; and business processes. Brambilla has a PhD in information

engineering from the Politecnico di Milano. Contact him at Marco.Brambilla@polimi.it.

Alessandro Campi is an assistant professor in the Department of Electronics and Information at the Politenico di Milano. His research interests include extending XQuery, automatic construction and verification of data-intensive websites, and methodologies and tools for e-learning. Campi has a PhD in information engineering. Contact him at Alessandro.Campi@polimi.it.

Francesco Corcoglioniti is a research assistant in the Department of Electronics and Information at Politecnico di Milano. His research interests include query execution within the search computing framework. Corcoglioniti has a master's degree in computer systems engineering from the Politecnico di Milano. Contact him at Francesco.Corcoglioniti@polimi.it.

Emanuele Della Valle is an assistant professor in the Department of Electronics and Information at the Politecnico di Milano. His research interests include Semantic Web and linked data and reasoning on streams. Della Valle has a master's degree in computer systems engineering from the Politecnico di Milano. Contact him at Emanuele.DellaValle@polimi.it.

Davide Eynard is a postdoctoral researcher in the Department of Electronics and Information at the Politecnico di Milano and Università della Svizzera Italiana, Lugano. His research interests include the semantic extension of participative systems such as wikis, folksonomies, and annotation systems. Eynard has a PhD in computer science from the Politecnico di Milano. Contact him at Davide.Eynard@polimi.it.

Piero Fraternali is a full professor in the Department of Electronics and Information at the Politecnico di Milano. His research interests include software engineering, advanced search systems, and methodologies and tools for Web application development. Fraternali has a PhD information engineering from the Politecnico di Milano. Contact him at Piero.Fraternali@polimi.it.

Michael Grossniklaus is a senior research associate in the Department of Electronics and Information at the Politecnico di Milano. His research interests include object-oriented database systems, context-aware applications, and multidomain search. Grossniklaus has a PhD in computer science from ETH Zurich. Contact him at Michael.Grossniklaus@polimi.it.

Davide Martinenghi is an assistant professor in the Department of Electronics and Information at the Politecnico di Milano. His research interests include data integrity maintenance, Web data access, top-*k* query optimization, and, in a broad sense, applications of logic to data management. Martinenghi has a PhD in computer science from Roskilde University, Denmark. Contact him at Davide.Martinenghi@polimi.it.

Stefania Ronchi is a PhD student in the Department of Electronics and Information at the Politecnico di Milano. Her research interests include services modeling, service description, service composition, information retrieval, and clustering techniques. Ronchi has a master's degree in computer science from the University of Bergamo. Contact her at Stefania.Ronchi@polimi.it.

Marco Tagliasacchi is an assistant professor in the Department of Electronics and Information at the Politecnico di Milano. His research interests include multimedia signal processing and information retrieval. Tagliasacchi has a PhD in information engineering from the Politecnico di Milano. Contact him at Marco.Tagliasacchi@polimi.it.

Salvatore Vadacca is a research assistant in the Department of Electronics and Information at the Politecnico di Milano. His research interests include distributed computing (grid and cloud computing) and its application to data management. Vadacca has a cum laude degree in computer engineering from the University of Lecce. Contact him at Salvatore.Vadacca@polimi.it.