Modeling Search Computing Applications

Alessandro Bozzon, Marco Brambilla, Alessandro Campi, Stefano Ceri, Francesco Corcoglioniti, Piero Fraternali, and Salvatore Vadacca

Politecnico di Milano, Dipartimento di Elettronica ed Informazione,
V. Ponzio 34/5, 20133 Milano, Italy
{bozzon,mbrambil,campi,ceri,corcoglioniti,
fraterna,vadacca}@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 models, supported by a framework, that make it possible for *expert users* to configure the data sources to be searched and the interfaces for query submission and result visualization, by using for such source and interface configurations mash-up tools which do not require programming. This paper presents Search Computing design process and developer roles, together with the conceptual models that represent the foundation of a model-driven approach to Search Computing.

Keywords: Search Computing, development process, search engine, models.

1 Introduction

Although Internet search is primarily performed by means of search engines, not all information needs are satisfied by individual Web pages. Web search queries become more and more complex: their formulation does not fit in few keywords, the result is richer than a simple list of Web pages, and general-purpose search engines perform poorly upon them. *Vertical search engines* (e.g., *Expedia* or *Lastminute* in the travel sector) provide a partial solution: they aggregate domain-specific information from a fixed set of sources; they do a better job than general-purpose search engines *in their domain*, but cannot be used or extended for other topics.

Complex *search processes* [1] addressing different domains remain to be supported. 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. A new class of applications, *search computing applications* [8], aims at responding to **multi-domain queries** like the one above, i.e., queries over multiple semantic fields of interest, by helping users to decompose queries and assemble complete results from partial answers; this class of applications fills the gap between generalized search systems, which are unable to find information spanning multiple topics, and domain-specific search systems, which cannot go beyond their domain limits. Examples of Search Computing queries are: “Where can I attend an interesting

F. Daniel and F.M. Facca (Eds.): ICWE 2010 Workshops, LNCS 6385, pp. 61–72, 2010.

scientific conference in my field and at the same time relax on a beautiful beach nearby?" or "Where is the theatre closest to my hotel, offering a high rank action movie and a near-by pizzeria?". Further motivating examples (and online demonstrations) can be found online at: **`http://demo.search-computing.com/`**.

Data source integration is a nontrivial task *per se*, as its solution requires solving schema matching problems, providing suitable coercion functions in the context of mismatching concrete types, and applying data cleansing and entity identification[15]. Ranked data source integration adds the problem of defining global rank functions over heterogeneous ranking attributes. Multi-domain queries are answered by selecting a subset of the available search services, by individually extracting their responses, and then by joining them thereby building combinations that collectively constitute the results of the query; combined and ranked results must be presented to the user for inspection and query refinement.

In this paper, we advocate a model-based stepwise refinement approach to the development of search computing application, which distinguishes the role of the developer, domain expert and end user, and adopts models to progressively abstract the complexity of query processing and data source wrapping.

The contribution of this paper is to discuss how to: (1) organize the development tasks in a coherent process, (2) identify the actors involved in such process, and (3) illustrate the models of the essential concepts that constitute a search computing application.

The paper is organized as follows: Section 2 gives an overview of the Search Computing framework; Section 3 presents the roles and process for SeCo application development; Section 4 discusses the main models managed by Search Computing framework; Section 5 discusses the related works and Section 6 concludes.

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?" This paper proposes an approach for building Search Computing applications by exploiting a configurable framework that delegates domain-neutral computation to a generic architecture which is configured with domain-dependent information and with the help of suitable models. As shown in Figure 1, several sub-frameworks constitute the Search Computing framework.

The *Service Mart Framework* provides the scaffolding for wrapping and registering data sources. Data sources can be heterogeneous (examples include Web site wrappers, Restful data sources, WSDL web services, data sources exposed by means of the Yahoo! Query Language [http://developer.yahoo.com/yql/], etc.).

A service mart is defined as an abstraction (e.g., Hotel) of one or more concrete Web services (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 (input and output data types) and connection information. A connection is defined as an input-output relationship between pairs of service marts that can be exploited for joining them.

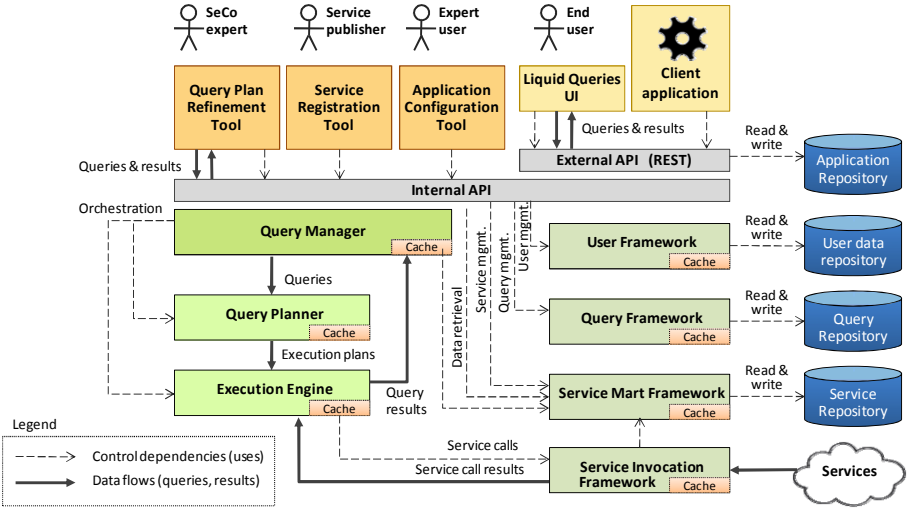

Fig. 1. Overview of the Search Computing framework

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, and modified.

The *Service Invocation Framework* masks the technical issues involved in the interaction with the registered service marts, e.g., the Web service protocol and data caching issues.

The *Query Processing Framework* provides the service for executing multi-domain queries. The *Query Manager* splits the query into sub-queries and binds them to the respective relevant data sources; the *Query Planner* produces an optimized plan with the sequence of steps for executing the query; finally, the *Execution Engine* executes the query plan, by submitting the service calls to designated services and then building the query results by combining the outputs produced by the called services.

Information persistence is delegated to dedicated object stores: the *service repository* (storing service mart descriptions), the *query repository* (storing queries saved by users), the *user data repository* (storing profiles information), and the *application repository* (storing application configurations). To obtain a specific application, the general-purpose architecture of Figure 1 is customized with the help of models and tools targeted to programmers, expert users, and end users.

3 Development Process and Roles

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

- **Service Publishers** are in charge of wrapping data sources, to make them compatible with the service mart standard interface, and register service mart definitions and their connections in the service repository.

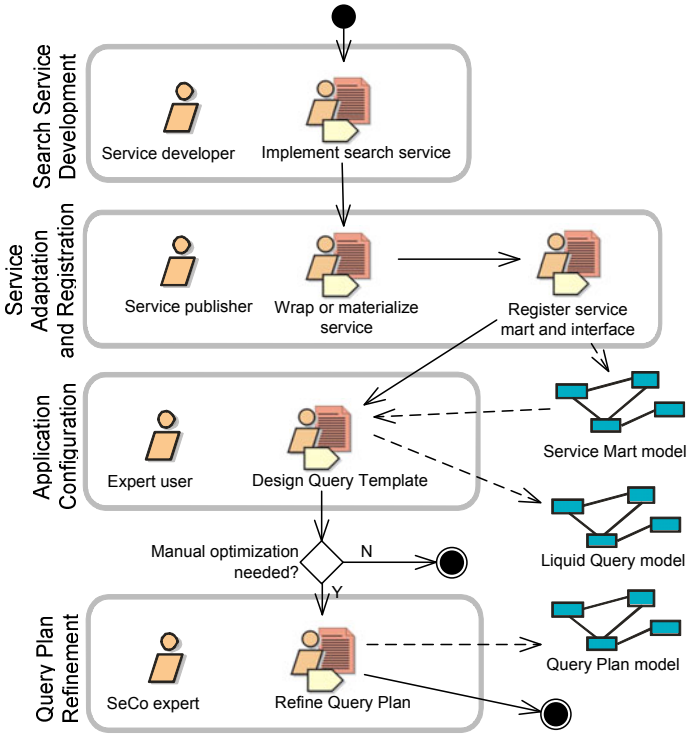

Fig. 2. Development process for SeCo applications (SPEM notation)

- **Expert Users** configure Search Computing applications, by selecting the services of interest, choosing the connection patterns to be used, and configuring the complexity of the user interface.
- **End Users** use Search Computing applications configured by expert users according to an exploratory information seeking approach [3].

These user's roles operate according to the SPEM process scheme shown in Fig. 2:

- **Search service development:** this phase consists in the development of Web services exposing on the Web the data requested by the application; it must be done in all those situations in which the data consumed by the Search Computing application are not available as Web services and must therefore be exposed, e.g., by wrapping a legacy system by means of a Web service interface; the activity is done by programmers, since it involves the production of the adapter code necessary to expose legacy data sources as Web services.
- **Service adaptation and registration:** this phase is necessary in the majority of cases, in which the data required by the application are already present on the Web, but the Web services that expose them do not adhere to the service mart interface and protocol. Therefore, several activities are needed for adapting the existing services to the Search Computing framework: creation of service wrappers, possible design of data caching policies, normalization of the data, and registration as service

marts in the service repository; these tasks are performed by service publishers, which need programming skills for the construction of data and protocol adaptation components. However, since the service mart APIs and protocol are fixed, the development of adapters can be supported by a template-based approach, in which standard adapter components are tailored to the specific Web service to wrap.

- **Application Configuration:** this phase consists in configuring the Service Invocation Framework and the User Interface and is conducted by an expert user. The former task requires selecting from the service repository several service marts and service implementations (for those service marts associated with multiple sources) and in declaring their configuration, including 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). Notice that multiple implementations could be available for the same service mart (e.g., Expedia and eDreams) and the same pair of services could be connected in different ways (e.g., theatres offering musical events could be connected to GIS data sources by using location names or geographic coordinates).

  The GUI configuration activity requires customizing the structure of a generic interface, by choosing: 1) optional selection predicates to restrict the objects retrieved by the query (e.g., a maximum price target for events or flights); 2) default ranking criteria for the results; 3) visual preferences on the display of the result set (e.g., sorting, grouping, and clustering attributes or the size of the result list); 4) a set of extra service marts usable by the end user to expand the current query (e.g., to expand a query on movies by means of a service that joins selected movies to their reviews and the theatres where they are programmed to nearby restaurants).
- **Query plan refinement:** this phase, in charge to the SeCo expert, consists in manually refining the optimized query plan produced by the SeCo platform starting from the query specification. In general, query plans are self-tuned, and therefore this phase is usually not needed. However, an expert can decide to manually override the optimal plans to comply with complex queries, or manually select alternative data sources, or improve scalability through parallelism, or provide customized choices not covered by the optimization.

The previous development steps lead to the final application accessed by the end user. The GUI, 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 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) [3].

The described development process takes into account the trend towards empowerment of the user, as witnessed in the field of Web mash-ups [10]. Indeed, only service development and service mart adaptation require programming expertise. All the other design activities are moved to service registration time and to application configuration

time, so that designers only need a conceptual understanding of services and queries, and do not need to perform low-level programming.

4 Search Computing Object Models

In this section we define the models describing the main objects involved in Search Computing applications. For illustration, throughout the paper we use a running example, in which a user plans a leisure trip, and wants to search for upcoming concerts (described in terms of music type, e.g., Jazz, Rock, Pop, etc.) close to a specified location (described in terms of kind of place, like beach, lake, mountain, seaside, and so on), considering also availability of good, close-by hotels. Additionally, the user can expand the query with information about available, nearby good quality restaurants for the candidate concert locations, the news associated to the event, photos that are taken close to the location, and possible options to combine further events scheduled in the same days and located in a close-by place.

4.1 Service Marts Model

Service mart**s** are abstractions that represent the properties (attributes and data types) of Web objects in a standardized way, according to the concepts of the Marts package shown in Fig. 3. Every service mart definition includes a name and a signature (a collection of exposed attributes). Attributes are strongly typed and can be atomic (i.e., defined by single-valued, basic types), composed (i.e., defined by a set of sub-attributes), or multi-valued (i.e., allowing multiple instances). The association between service marts is expressed by **connection patterns** (composition package of Fig. 3). Every pattern is described by a conceptual Name and a set of comparison predicates between pairs of attributes of the two services, which are interpreted as a conjunctive Boolean expression.

The Patterns package of Fig. 3 describes a lower level characterization, whereby each service mart is 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. In the context of logical databases, an assignment of I/O labels to the attributes of a predicate is called predicate adornment [7]. Moreover, an output attribute can be designed as ranked (R).

Finally, **service interfaces** (Interfaces package of Fig. 3) 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 sets of objects that satisfy a condition). Service interfaces are characterized by *chunking size* (number of result instances in the chunk), *caching properties***,** and *cost descriptors* (e.g., the *response time* and/or as the *monetary cost of invocation*). Examples of access patterns for the running case are:

```
Concert(location[I], radius[I], minDate[I], maxDate[I], genre[I] , name[O],
   date [O][R], lat[O], long[O], distance[O][R], Price[O][R], address [O])
Restaurant(category[I], minRating[I], location[I], radius[I], name[O],
   address[O], lat[O], long[O], Rating[O][R], distance[O][R], url [O])
```

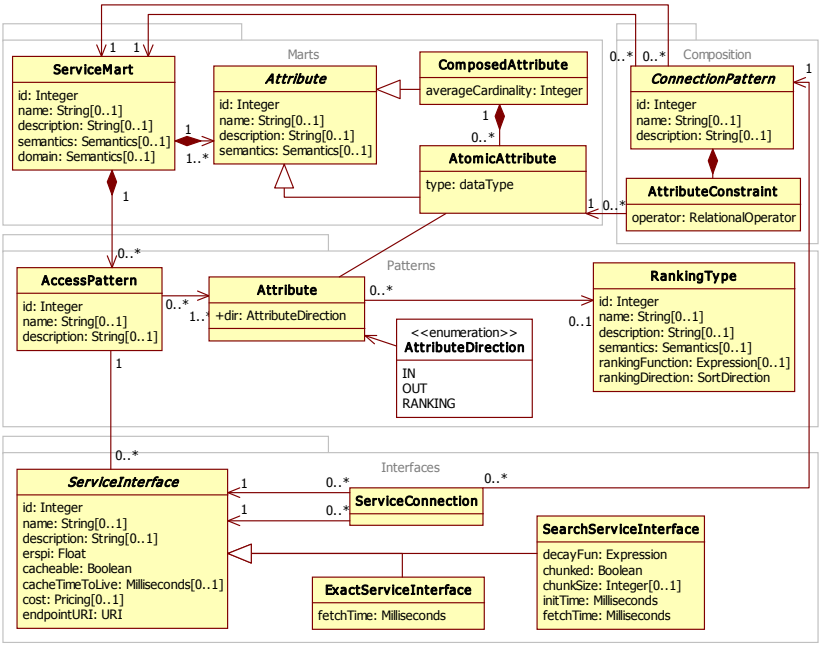

Fig. 3. Service mart model

4.2 Query Model

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. Fig. 4 shows that a query clause can refer to the service mart level (and thus requires automatic translation down to the concrete service interfaces) or at the Service Interface level. A clause may describe the service to invoke, conditional predicates over service attributes, join operations, and ranking of results.

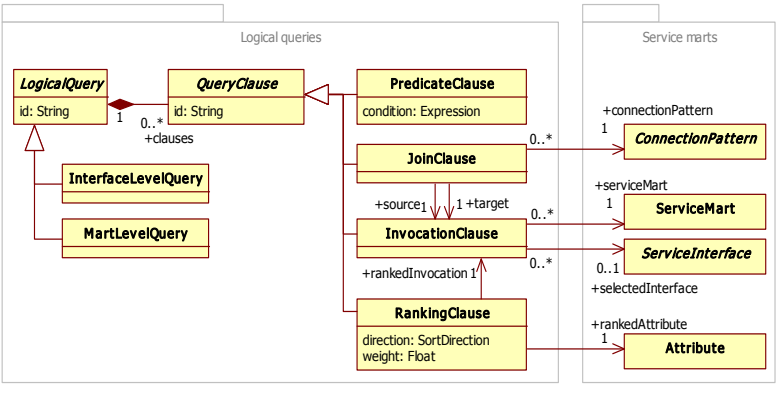

Fig. 4. Query model

An example of complex query is “Where can I find a jazz concert in San Francisco close to a nice vegetarian restaurant?” represented as:

```
Concert("San Francisco", "1.0 m", "10/30/2010", "11/30/2010", "Jazz", name[O],
   date [O][R], lat[O], long[O], distance[O][R], Price[O][R], address [O])
Restaurant("vegetarian", "3stars", {Concert.lat, Concert.long}, "1.0 m",
   name[O], address[O], lat[O], long[O], Rating[O][R], distance[O][R], url[O])
```

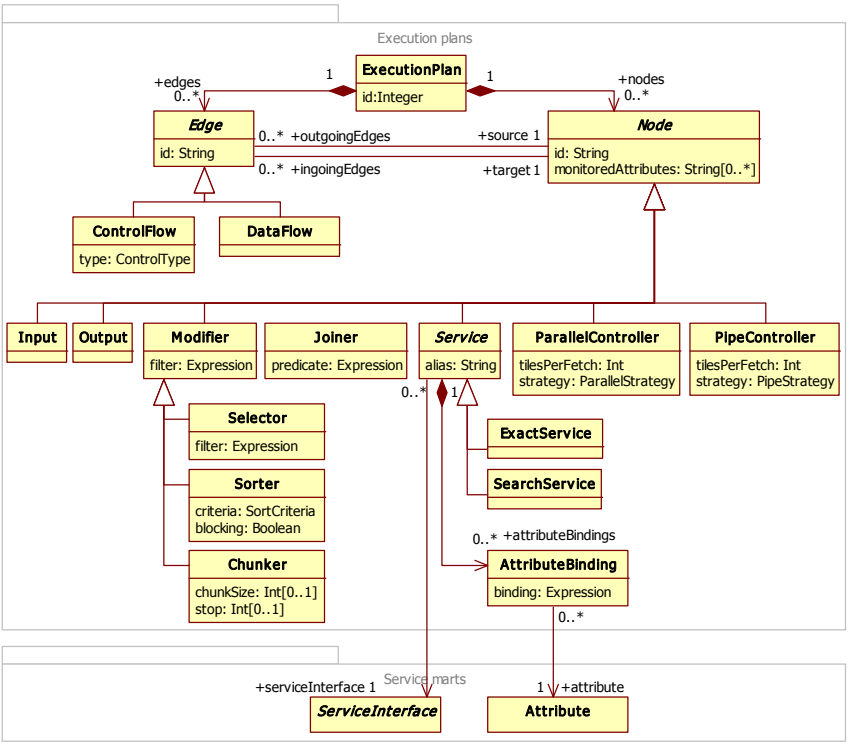

Fig. 5. Query plan model

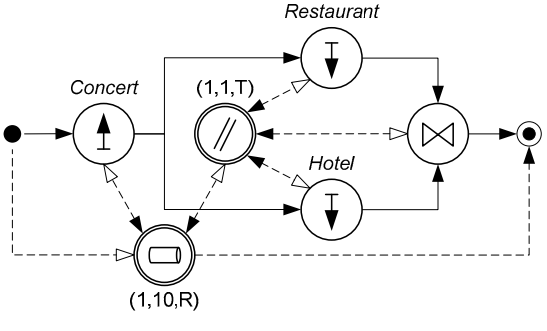

Fig. 6. Example of query plan according to the *SeCo Panta Rhei* DSL

4.3 Query Plan Model

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 search services return results to rank the combined query results. A query plan (see Fig. 5) is a graph composed by nodes (invocations of services or other operators) and edges (control flow and the data flow between nodes). Several types of nodes exist, including service invocators, joiners, controllers and modifiers (chunkers, sorters, selectors), according to the Panta Rhei *Domain Specific Language* [5].

As an example of query plan model, Fig. 6 shows a pipe join which first extracts a list of *Concerts* from a search service and then uses these results to retrieve close-by *Restaurants* and *Hotels*, whose invocations are performed in parallel according to a predefined join strategy. Finally, results are joined by suitable join units, which execute the join of search engine results [4]. Two strategy nodes—one for the outer pipe join and one for the inner parallel join—orchestrate the execution of the query by observing the output of data units (such as service invocators and joiners) and deciding the service to be invoked accordingly.

4.4 Interaction Model

Fig. 7 shows the model that describes the set of abstractions supporting exploratory search within the Search Computing user interface. Following the distinction of the development process into application configuration and usage, the model represents interaction at two levels. At the specification level, when configuring an application, a user's query is defined as a selection of an underlying *logical query*, which is in turn

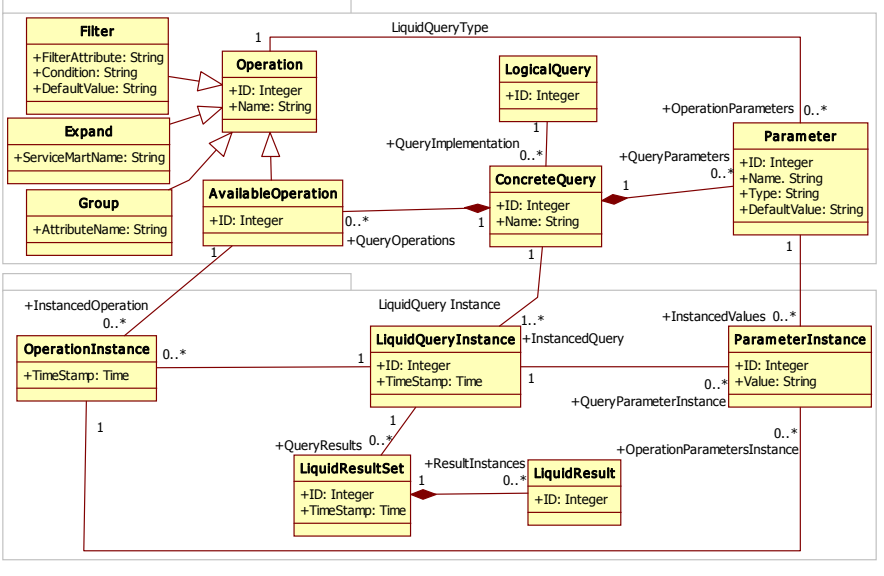

Fig. 7. The interaction model

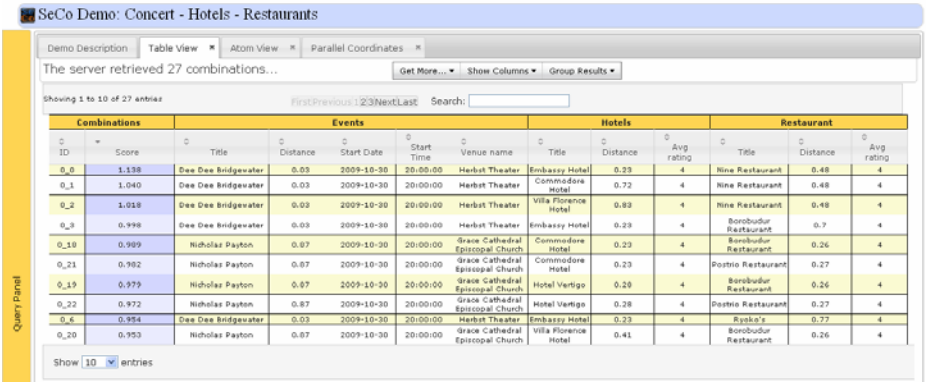

Fig. 8. Example of resultset retrieved by the system for the Concert-Restaurant-Hotel case (see the online demo at `http://demo.search-computing.org`)

mapped to a *concrete query*. Such concrete query is then associated with: 1) a set of input parameters; 2) a set of *AvailableOperation*s (e.g., Filter, Group, Expand, etc.). Operations might be parametric, thus requiring the end user to supply parameters.

Once specified, a logical query can be instantiated at runtime (as denoted by the *QueryInstantiation* class). When a query is instantiated, the query parameters assume a value, thus allowing the execution engine to process the associated logical query, which in turn produces a *ResultSet*, composed by a set of *Results*. Then, a user can decide to further explore the results by applying one of the *AvailableOperations*, thus altering the *ResultSet* according to the semantic of the selected operation. Some operations simply apply local transformations to the extracted results (e.g., re-sorting, grouping, and so on), while others need the intervention of the query execution engine, e.g., to ask for more results or to expand the result set with information on additional service marts; in the latter case, in particular, some additional pieces of query plans can be activated. Fig. 8 shows an example of result table for the running case. An online demo is available at: **`http://demo.search-computing.org`**.

5 Related Work

Model-driven engineering is the main research field SeCo takes inspiration from. All the SeCo artifacts are modeled as conceptual entities and are managed by model-driven transformations (a coarse approach to the transformations is described in [6]). The existing works in the field, which include general-purpose MDA techniques and web-specific languages and tools (e.g., WebML [9], OO-HMETHOD [11], UWE [13], HERA, and others), do not consider the search-specific aspects of application development and in particular completely ignore the need for an intermediate role between the user and the developer (that we call expert user) that has the duty of configuring search applications starting from basic services.

None of the **tools for object-oriented design**, including the ones focusing on databases (e.g., Oracle JDeveloper 10g [http://www.oracle.com/tools]), on Web applications design (e.g., Code Charge Studio [http://www.codecharge.com], WebRatio

[www.webratio.com]), and standard application design (e.g., Rational Rapid Developer or jABC, Java Application Building Center), explicitly support the design of search applications. From various tools, SeCo borrows the ideas of *visual composition* of the applications and *automatic deployment* of the running prototype.

Mashup approaches are even more inspiring in this sense, since they exactly comply with the SeCo expert users need of an easy online toolsuite to quickly configure and deploy applications. The most famous mashup environments are Yahoo Pipes and Microsoft Popfly (recently discontinued), together with several scientific investigations still ongoing (e.g., [12] proposes a spreadsheet based mashup development framework).

The **service engineering** field provides service description languages and protocols such as WSDL and SOAP, with limited support in mechanizing service recognition, combination, and negotiation. Proposals like WSFL, DAML-S, OWL-S, and WSMF attempt at formalizing the semantics of Web services using ontology technology. Data exchange formats (e.g., SOAP-XML, JSON, and others) and service orchestration languages (BPEL, BPMN, and so on) inspired our approach. Other ongoing efforts such as W3C's Web Services Architecture and OASIS standards aim at creating a framework for service modeling. With respect to the SOA terminology of orchestration and choreography, we adopt a centralized orchestration approach.

Finally, **search-based application development** has largely inspired SeCo too. The Symphony platform by Microsoft enables non-developers to build and deploy search-driven applications that combine their data and domain expertise with content from search engines and other Web Services [14]. Other approaches to search-based development (like Google Base API [http://code.google.com/apis/base/] and YQL, Yahoo Query Language [http://developer.yahoo.com/yql/]) target the skilled software developer. These solutions, combined with mashup approaches, can get close to the SeCo approach, although several critical aspects would still be different (e.g., join and ranking would be missing anyway).

With respect to our previous work, this paper advances in the direction of exploring the development process of search applications. A preliminary proposal for process and roles was introduced in [2], while model transformations are discussed in [6]. In this paper instead we refined the process, we redefined the user roles and provided a detailed the conceptual models (with a set of sample instances).

6 Conclusions

Search Computing is an emerging paradigm for supporting complex search. Search Computing design choices are supported by a cohesive framework which integrates several interacting models, thereby partitioning the design space and responsibilities to the different roles and involved expertise, in a non-trivial way; the objective is to replace programming with model driven development wherever possible, yielding to flexibility and efficiency.

Acknowledgements. This research is part of the Search Computing (Seco) project, funded by ERC, under the 2008 Call for "IDEAS Advanced Grants".

References

[1] 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)
[2] Bozzon, A., Brambilla, M., Ceri, S., Corcoglioniti, F., Gatti, N.: Building search computing applications. In: Ceri, S., Brambilla, M. (eds.) Search Computing. LNCS, vol. 5950, pp. 268–290. Springer, Heidelberg (2010)
[3] Bozzon, A., Brambilla, M., Ceri, S., Fraternali, P.: Liquid Query: Multi-Domain Exploratory Search on the Web. In: World Wide Web Conference (WWW 2010), Raleigh, USA, pp. 161–170. ACM, New York (April 2010)
[4] Braga, D., Campi, A., Ceri, S., Raffio, A.: Joining the results of heterogeneous search engines. Information Systems 33(7-8), 658–680 (2008)
[5] Braga, D., Ceri, S., Corcoglioniti, F., Grossniklaus, M.: Panta Rhei: A Query Execution Environment. In: Ceri, S., Brambilla, M. (eds.) Search Computing. LNCS, vol. 5950, pp. 225–243. Springer, Heidelberg (2010)
[6] Brambilla, M., Ceri, S., Tisi, M.: Search Computing - A Model-Driven Perspective. In: Tratt, L., Gogolla, M. (eds.) Theory and Practice of Model Transformations. LNCS, vol. 6142, pp. 1–15. Springer, Heidelberg (2010)
[7] Calì, A., Martinenghi, D.: Querying Data under Access Limitations. In: ICDE 2008, pp. 50–59 (2008)
[8] Ceri, S., Brambilla, M. (eds.): Search Computing. LNCS, vol. 5950. Springer, Heidelberg (March 2010)
[9] Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., Matera, M.: Designing Data-Intensive Web Applications. Morgan Kaufmann, USA (December 2002) ISBN 1-55860-843-5
[10] 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)
[11] Gómez, J., Cachero, C., Pastor, O.: Conceptual Modeling of Device-Independent Web Applications. IEEE MultiMedia 8(2), 26–39 (2001)
[12] Kongdenfha, W., Benatallah, B., Vayssière, J., Saint-Paul, R., Casati, F.: Rapid development of spreadsheet-based web mashups. In: WWW 2009, Madrid, pp. 851–860. ACM, New York (April 2009)
[13] Knapp, A., Koch, N., Moser, F., Zhang, G.: ArgoUWE: A CASE Tool for Web Applications. In: EMSISE Workshop (2003)
[14] Shafer, J.C., Agrawal, R., Lauw, H.W.: Symphony: Enabling Search-Driven Applications. In: USETIM (Using Search Engine Technology for Information Management) Workshop, VLDB Lyon (2009)
[15] Tejada, S., Knoblock, C.A., Minton, S.: Learning object identification rules for information integration. Information Systems 26(8), 607–633 (2001)