Part III

Issues in Search Computing
Introduction to Part III:
Search Computing in a Nutshell

Prior to delving into chapters discussing search computing in greater detail, we give a bird’s eye view of its various phases and components, by providing an architectural view of the search computing prototyping environment.

Search computing systems support their users in making multi-domain queries; for instance, “Where can I attend a DB scientific conference close to a beautiful beach reachable with cheap flights?” A system decomposes the query into sub-queries (in this case: “Where can I attend a DB scientific conference?”; “which place is close to a beautiful beach?”; “which place is reachable from my home location with cheap flights?”) and maps each sub-query to a domain-expert server (in this case, calls to servers named “Conference”, “Tourism”, “Low-Cost-Flights”); it then analyzes the query and translates it into an internal format, which is then optimized, thereby yielding to an optimal plan for query execution; plan execution is supported by an execution engine, which submits service calls to services through a service invocation framework, builds the query results by combining the outputs produced by service calls, computes the global rankings of query results, and outputs query results in an order that reflects, although with some approximation, their global ranking.

These transformation steps are shown in the bottom-left side of Fig. 1; they are performed by the query mapper, query analyzer, query planner, and execution engine, under the responsibility of a query orchestrator that starts query execution and collects query results. The figure shows that each of the four modules directly accepts user-provided input through suitable interfaces; in this way, prototype implementation in search computing can take place bottom-up, by starting with the execution engine, which can execute a given plan, then adding the query planner, which produces the optimal plan for a given internal query, then adding the query analyzer, which reads an abstract query, checks that the query is legal, and produces an internal query; and finally adding a query mapper, capable of decomposing a multi-domain query into several domain-specific queries. In this book we do not address query mapping, while we address the other steps. The search computing prototyping architecture is currently well-defined in terms of interactions and of functionalities; prototypes will be delivered throughout the course of the SeCo Project.

Services are made available to search computing through a standard format, called service mart; by this term we mean an abstraction that masks the different implementation styles of services and is tailored to the specific need of exposing search services – i.e., services whose primary purpose is to produce ranked lists of results. Moreover, service marts offer a classification of service properties (that represent either the call or the result of a service invocation; output results may represent the ranking values) and a definition of composition patterns allowing the combination of service marts.
Search computing users mainly belong to two categories. **End users** can only launch predefined applications and submit input to them through forms; **expert users** may also compose queries in the context of repositories of service marts and of their composition patterns (we say in such cases that users can build liquid queries, where their liquid nature comes from the fact that queries extend upon service marts more or less as stains over surfaces). In both cases, however, we expect users to have some experience in data analysis (similar, e.g., to the basic skills required by spreadsheets) and we expect them to use such skills in manipulating results, which are shown in tabular format, and can be dynamically augmented online – we call them liquid results to highlight the dynamic and plastic nature of such results, which can be manipulated by means of user controls.

Tools are intended to support three kinds of experts:

- **Service designers** register data sources in the system through the Service Mart Framework, by either interacting with existing Web services, or by exposing existing data sources, or by wrapping existing Web pages. They play the role of “data providers”.
- **Application developers** preselect some of the services and configure them so as to turn them into applications; specifically, they build user interfaces which either expose service marts and their connections to expert users or simple forms accepting typed input to end users. They play the role of “data brokers”.
- **SeCo developers** install, open and configure the SeCo modules on suitable hardware resources and may perform fine tuning (or creation from scratch) of query and execution plans.
The upper part of Figure 1 shows that the tools provided to the designer and developer communities plug to an internal API, while end-user applications and interfaces in turn are accessible via an external API and therefore callable from any client environment. Finally, the right part of Figure 1 shows three kinds of repositories, called service repositories (i.e., registries of search services registered in the framework), query repositories (i.e., queries that have been saved by the user for subsequent restore operations), and user data repositories (i.e., profiles and administrative information). An additional application repository loads applications and stores the user’s interactions, so as to be able to remember and re-apply such interactions to new queries or to new results.

The various architectural elements forming the search computing prototype architecture defined above are described in different, autonomous chapters.

Chapter 9 deals with service marts, a novel concept for enabling the engineering and deployment of search services, i.e., of services whose main feature is the ability to respond to ranked results organized by chunks (so as to enable a fine-grain control by the execution engine). Such results are produced by interacting with concrete data sources, which are made available through service interfaces, wrappers, or direct access to extensional data collections (databases, excel files, and so on). Thus, service marts are conceptual abstractions providing information hiding, mapped to service interfaces which directly interact with concrete data sources.

Chapter 10 describes our framework for query execution; specifically it addresses the description of a query language for search computing, then the mapping of queries to service interfaces, then the composition methods that have been defined so far in search computing under the classical format of join methods, suitably extended to the search and web context. This chapter discusses query formulation and optimization up to the choice of join methods.

Chapter 11 deals with ranking aggregation in its most general formalization, and shows how ranking aggregation methods can be adapted to search computing in generating a join method which is capable of guaranteeing that the top-k results are selected.

Chapter 12 describes a flexible architecture for search computing (named Panta Rhei) which includes suitable abstractions for data production, consumption, and caching, with both data-driven and event-driven synchronization. Operations and flows of the Panta Rhei model are described at a high level, but they are designed for supporting the scalable execution of search computing queries in a variety of deployment architectures.

Chapter 13 shows a paradigm for making search computing queries, called liquid queries, that can be articulated upon a flexible architecture, where a liquid query is capable of run-time modification by the addition or dropping of sub-queries and by drilling down and rolling up information, much in the same way as with a data cube expressing the results in data analysis environments.

Chapter 14 shows how to build and deploy applications by means of a software engineering environment involving both “data providers”, who will register service marts, and “data brokers”, who will assemble applications. SeCo servers can be deployed upon a variety of computing architectures, hinting to future prototypes running upon highly scalable architectures and/or cloud computing systems. The
chapter also discusses the business models that may favor the spreading of both data providers and data brokers.

Chapter 15 discusses ranking opportunities in the context of life sciences, which are characterized by a wide use of ranked information, thereby anticipating some of the specific issues featured by an appealing search computing application.