Engineering Search Computing Applications: Vision and Challenges

Marco Brambilla, Stefano Ceri
Dipartimento di Elettronica e Informazione, Politecnico di Milano
Piazza L. Da Vinci, 32
20133 Milano, Italy
+39 02 2399 7621
marco.brambilla@polimi.it, stefano.ceri@polimi.it

ABSTRACT
Search computing is a novel discipline whose goal is to answer complex, multi-domain queries. Such queries typically require combining in their results domain knowledge extracted from multiple Web resources; therefore, conventional crawling and indexing techniques, which look at individual Web pages, are not adequate for them. In this paper, we sketch the main characteristics of search computing and we highlight how various classical computer science disciplines - including software engineering, Web engineering, service-oriented architectures, data management, and human-computing interaction - are challenged by the search computing approach.

Categories and Subject Descriptors
D.2.2 [**Software Engineering**]: Design Tools and Techniques.

D.2.9 **[Software Engineering]**: Management – *Lifecycle*.

D.2.11 **[Software Engineering]**: Software Architectures – *Domain-specific Architectures*.

H.3.3 **[Information Storage and Retrieval]**: Information Search and Retrieval.

General Terms
Algorithms, Design, Human Factors.

Keywords
Search computing, Software Engineering, SOA, Web Services, Search Services.

1. INTRODUCTION
The current evolution of the Web is characterized by an increasing availability of online services and novel search facilities (e.g., services for searching scientific literature, photos, videos, vacation offers, travels, slow-food restaurants, online shops, and so on). Being specific to a restricted domain, the quality of their answers goes much beyond what can be achieved via conventional, general purpose search engines. The overall amount of data that can contribute to such queries is continuously growing, mainly within the so-called deep Web, i.e., in a form not immediately indexed by search engines.

ESEC/FSE'09, August 24–28, 2009, Amsterdam, The Netherlands.
ACM 978-1-60558-001-2/09/08.

While these services can be individually accessed, they often provide value to users in the context of complex interactions, where users try to solve specific tasks, such as planning their vacations, or searching for medical treatments offered by local doctors and hospitals, or checking that local stores have in stock goods which are described by their manufacturers. Typically a user is not only concerned with queries about a single domain; while current technological limitations confine a user to such interaction, in reality users' need for information typically spans over multiple domains, which must be semantically connected. In light of these considerations, multi-domain queries, i.e., queries that can be answered by combining knowledge from two or more domains, no longer represent a mere academic exercise; rather, they demonstrate how intricate real life queries may be.

Answering multi-domain queries requires the combination of knowledge from various domains. These queries are hardly managed by general-purpose search engines, because they cannot be found on a single page, where a page is the classical unit of crawling and indexing. Moreover, domain-specific systems exhibit more sophisticated knowledge than general-purpose search systems about their own field of expertise; such expertise (about cultural events, medical specializations, popular rock songs, and so on) is contributed through social processes (e.g., rating, tagging, commenting) or through a long and careful process of knowledge construction by experts. At the current state of the art, multi-domain queries over such engines can be answered only by patient and expert users, whose strategy is to interact with specialized engines one at a time, and feed the result of one search in input to another one, reconstructing answers in their mind.

With the advent of service computing and the growing interest for the Web as the predominant interface for any human activity, we expect such knowledge to become more and more exposed in the form of search services. But the mere composition of such services by sequential invocation will not solve multi-domain queries, as their interplay usually requires a lot of expertise, especially in handing and composing the search results.

In this paper, we present a conceptual framework for addressing the composition of search services for solving multi-domain queries, and we discuss the novel challenges that such systems are posing to the software engineering discipline. We will investigate the meaning and intrinsic properties of search services and the ways such services can be orchestrated. This setting leaves within each search system the responsibility of maintaining and improving its domain knowledge, and defines new search computing systems that provide the glue between search service competences.

We are aware that the general formulation of the search computing problem, going from registration of arbitrary services and acquisition of arbitrary queries to the production of sensible results, is very complex; many simplifying assumptions can be used to reduce the problem complexity, ranging from a pre-selection of the domains of interest and of the search engines, to a progressive reduction of the expressive power of the query.

To better appreciate the approach, we consider a running example, consisting of the domain, service, and query analysis steps required to answer the query: *Where can I attend a DB scientific conference in a city within a hot region served by luxury hotels and teachable with cheap flights?*

The paper is organized as follows: Section 2 presents a bird eye's view of search computing through a brief description of its registration and control flows; Section 3 deals with search services and their orchestration; Section 4 introduces the software development environments that we plan to build for service registration and composition, while Section 5 describes the search environment framework that will host the search computing applications, including its deployment upon scalable architecture and the availability of flexible and generic user interfaces. Section 6 sketches how the development process of search computing differs from other software systems and illustrates the roles of developers, designers, and users in the various phases of the process, and finally Section 7 presents our vision about how search computing could lead to the development of communities of content developers and application providers.

2. ARCHITECTURE OF SEARCH COMPUTING SYSTEMS

Figure 1 shows the overall architecture of a search computing system, together with the main execution flows. We identify two main activity flows: the ***registration flow*** - that deals with the declaration and description of domains, and the registration of search services and their association to domains- and the ***query execution flow*** - that deals with the actual processing of the queries. Registration is performed by administrators of the platform, and the framework helps them in selecting the domains and services, annotating them, and creating mappings between them. Queries are performed by final users.

The objects managed by the activity flows include: domains and their properties, search services, high level multi-domain user queries, low-level queries (adorned conjunctive datalog queries), query plans (coarse-granularity descriptions of query execution strategies); and query execution schedules (well-defined schedules of fine-granularity operations).

In the registration flow, we address the following problems: semantic description, storage, management, and access to search services; clustering of services into domains; and definition of admissible join conditions between services.

In the query execution flow we address the following problems: definition of proper interfaces for submission of multi-domain user queries; splitting of the query into subqueries; mapping of subqueries to domains and to associated search services, for defining low-level queries; generation of query plans and their evaluation against several cost metrics so as to choose the most promising one for execution; generation and processing of query execution plans; and transformation and rendering of the results for user consumption.

The role of domains is relevant in the context of query decomposition, i.e., the process of decomposing user query into subqueries, each specific to a given domain. Such problem is relevant in order to bring search computing up to dealing with "universal queries". However, in this paper we take a focused view, where queries are addressed to specific services via a software engineering approach which deals with the delivery and composition of services by developers. In such context, the role of domains is less crucial.

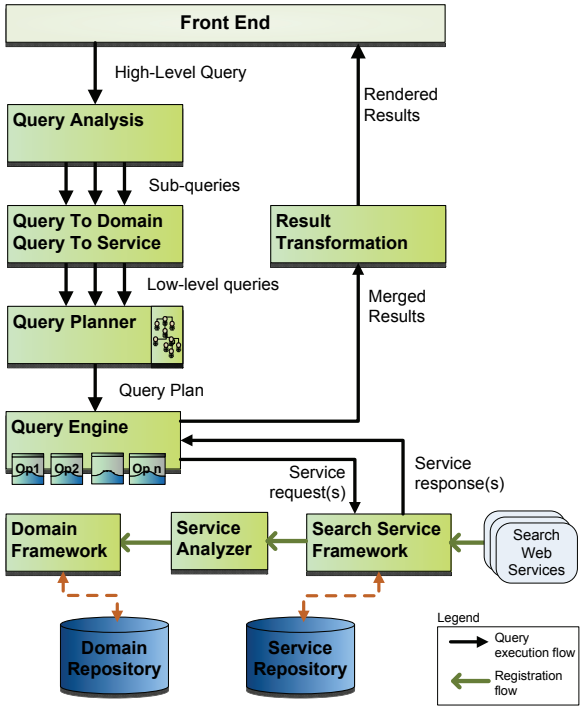

Figure 1. Overall architecture and execution flows.

3. SERVICES FOR SEARCH COMPUTING

3.1 Search Services

The spreading of search computing relies on the availability of components capable of effectively performing search tasks. Such systems range from generalized search engines, such as Google and Yahoo, to very specific and sophisticated search systems, e.g., capable of locating slow-food restaurants within given geographic districts, or to characterize the last-minute world-wide offers of specific hotel chains. Such components are software services which receive queries as input, and respond objects with associated rankings, where the ranking reflects a combination of: item-based rank computation algorithms, social recommendations, context-based adaptation to the user, business incentives, and so on. In search computing, we assume that such services will be characterized by the use of uniform interfaces, which follow the service-oriented paradigm, such that:

- Interaction is based upon request-responses, where:
  - Requests = queries
  - Responses = ranked lists of items

- Requests are associated with input parameters, which may represent search keywords or attribute-value pairs.
- Output items contain attribute-value pairs, links to resources, and possibly unstructured content.
- Rankings are normally opaque, although it is fair to assume that items are presented in decreasing order of the ranking function.
- Lists are too long and should be partially retrieved. Therefore lists are presented in chunks, and each chunk normally contains a fixed number of elements.

Thus, search services give to their users a greater amount of flexibility in interaction, as the same query can be answered by a variable number of requests; every new request retrieves the successor chunk of results given what had been retrieved so far. In general, however, user interaction requires only the first chunks of results for a given query, because users are only interested in the top results.

Such services will live in the context of other generic services, characterized by classical request-response interfaces, and offering results which are not ranked, but instead all equal in relevance (as a consequence, no result can be omitted in order to fully answer to a query). If we consider a service as part of a data-flow, such services may be associated to a cardinality, seen as the average ratio between service outputs and inputs, and as such some of the services are "selective" (because the flow halts after the service result is produced), other are "proliferative" (because one input is associated with many outputs).

The originality of the model resides in introducing a simple and yet effective classification of services: *exact services* have a "relational" behavior and return either a single answer or a set of unranked answers; *search services* return a list of answers in ranking order, according to some measure of relevance. The interplay of "exact" services with "search" services is one of the relevant aspects of the search computing research [2].

3.2 Orchestration of Search Services

A search computing engine is mainly an orchestrator of search services, possibly interleaved with other generic services. The orchestration is started at query presentation, but requires a lot of preparatory work, distributed among three distinct phases: service registration, query registration, and query execution.

At registration, services are described through a complex and detailed interaction with designers, described in the next Section, so as to capture their semantics (what the service does) and their composition potential (how services relate to each other). The typical operations to be performed upon service results, after their invocation, are algebraic operations such as joins, unions, and differences. These enable to build "compositions" of service results, which represent complete answers to a query, such that every composition is associated with a global ranking value. The goal of orchestration is to generate strategies that present compositions to users in descending order of the aggregate function, however such requirement is not strict, as the orchestration attempts as well at generating the first compositions very fast. The generation of an orchestration is very similar to the generation of "query execution plans" by database management systems, although data are retrieved from services rather than from internal storage. Therefore, most of the work is performed when a query is registered, assuming that a given query registration process can be reused for multiple query executions.

An example of service orchestration is shown in Fig. 2. The query "*where can I attend a DB scientific conference in a city within a hot region served by luxury hotels and teachable with cheap flights?"* requires combining 4 services, about conferences, weather, cities, and travels, such that the first two are exact services and the last two are search services. The conference service is proliferative, returning all DB conferences (such as: Sigmod, VLDB, ICDE, EDBT,...) without ranking them, the second one is selective, as it excludes the cities with cold weather, and the third and fourth are search services, extracting cheap flights (in descending order of price) and luxury hotels (in descending order of stars), while a global aggregate function composes the number of stars and travel costs so as to assign a global ranking to the compositions. The query plan uses a particular operation called "merge scan join", defined in [1].

The annotations of the query plan indicate that, in order to produce 15 compositions in the end, the plan calls for invoking three times the flight service and four times the hotel, while initially 20 conferences are selected but then only one is filtered. This query plan is used at execution time to orchestrate service calls and join operations; it consists of invoking the conference service, then the weather service, and finally the flight and hotel services according to a three-to-four fixed relationships, with the expectation of generating "enough" results (i.e., 15) after one such collection of invocations. The query could be installed and be made parametric, e.g. with regard to the topic of the conferences, the kind of weather, the time of the event, the maximum number of flight connections or of flight time, and so on.

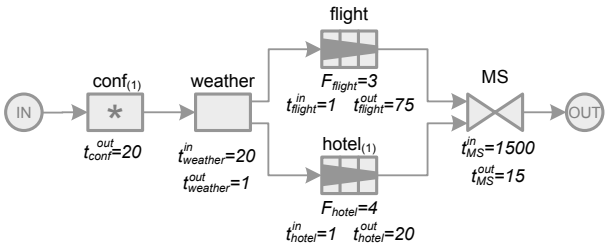

Figure 2. A fully specified search service orchestration

4. DEVELOPMENT ENVIRONMENT

The peculiar features of search computing systems must reflect into proper support by C.A.S.E. (Computer Aided Software Engineering) tools. Given the needs for flexibility and adaptability of search applications, several development activities move from the traditional design and development phases to a phase subsequent to the search framework deployment, when developers can "configure" the framework for the actual needs. Thanks to this paradigm shift, most of the development cost involved into new search application development moves from design time to configuration time. Indeed, the main multi-domain, configurable platform is developed once and for all, and then is properly configured according to the needs.

We distinguish between tools to be used at registration time (for registering and describing services), and tools to be used at integration time (for defining the integration strategies between the services).

4.1 Registration-time tools

The development environment that supports developers at registration time will cover the following aspects:

- *High-level representation of services*: search services are described through a high-level model that comprises: service *name* and *parameters*, i.e., name-type-value(s)-direction(I/O) tuples. Such representation abstracts from several details of the service, thus providing information hiding and the correct abstraction level for describing a search service, including all its peculiar properties. This representation is suitable for a large set of implementation options, that span from single web services, to compositions of services whose combined result is exposed through a single interface, to one or more materialized tables. Such complex implementations can be hidden in the code developed for defining the component, and are masked by the abovementioned abstract interface.
- *Web service wrapping*: search services on the Web are available in a plethora of formats, including SOAP Web services, REST services, and user oriented search applications. Each of them exposes a specific interface, which needs to be adapted to the search framework needs. Therefore, the first step in the integration is the wrapping of the services so as to have a common interface towards all of them. For some formats, appropriate tools can be used. For instance, user oriented search applications could be wrapped using tools like Lixto [10], which in turn could be refined for adapting to the specific needs of search interfaces.
- *Web service registration*: once the service is properly wrapped, it must be referenced within the service repository. This requires to store the endpoint of the service (or the wrapper), together with the description provided by the service provider (e.g., WSDL specifications) and all the information for accessing it;
- *Access paths definition*: for each service, the access paths that are allowed for its use must be defined. This basically consists in specifying the correct sets of inputs and outputs that are allowed for the service.
- *Web Service description*: furthermore, services need to be described through some lightweight semantics mechanisms, for allowing unique semantics to be associated to service operations, parameters, and results. This can be achieved by annotating the service API with tags. For instance, tags could be *synsets*[1] taken from repositories like Wordnet [9], and domains defined in the design environment as clusters of services that refer to the same field.
- *Join path definition*: finally, the designer is in charge of the definition of admissible join paths between services. The goal of this task is to identify, for every pair of services that can be invoked for answering a query, the join attributes that will be used for composing their results. This can be performed by the designer through a proper visual mapping interface between services, but can also be supported by a set of recommendation algorithms that suggest the best candidates for matching based on the service clustering, and their operation interfaces (name and parameters annotations and types). For each pair of classes belonging to different domains, the tool can identify parameters having the same type and annotations, which are candidates for being qualified as join attributes. Then, the process of pairing services is progressively performed, with the help of developers, who can tell if the join paths identified by the system can indeed be used for connecting domains, and, if so, how elements of join paths should be paired and join conditions be fully qualified.
- *Default presentation of service data*: given that a service is provided with a set of inputs and outputs, the designer can optionally specify a set of default presentation rules that will be used at query time for building the user interface, for allowing both query submission and result browsing by the user. The tool should provide predefined rules based on parameter types and annotations.

[1] A Synset, or Synonym ring, is a group of data elements considered semantically equivalent for the purposes of information retrieval.

4.2 Integration-time tools

Once search services are registered in the repository according to the guidelines provided above, the development of a search computing application becomes a matter of combining the proper search services and defining the user interfaces for accessing them. More precisely, the actions to be performed are:

- *Choice of the domains of interest*: the analyst is in charge of understanding the search problem and of identifying the domains that are relevant for the search application;
- *Selection of the services*: the developer selects the kinds of services that are useful, based on the selected domains and requirements from the analyst. This task can be supported by tools that allow to prune the list of available services (which in principle could be huge, for some domains), based on their interface and annotations;
- *Selection of the best join paths:* the designer selects the most suitable join paths among the ones that connect the services;
- *Design of the user interfaces:* the designer builds the user interfaces that allow the final user to submit the queries and to browse the results. This task is simplified thanks to the available default presentation rules specified at registration time, that allow to automatically get a coarse interface once the services and the join paths are selected.

5. SEARCH ENVIRONMENT

Although the main design choices for the search application are taken at registration and composition time, the framework must be flexible enough to provide the possibility of refining or optimizing the behavior of the application even at query time.

Three challenges must be faced to obtain this result: a flexible infrastructure for planning and executing the queries; the configurable deployment of the framework to grant easy access to search computing and corresponding scalability properties; and a user interface that satisfies the need for flexibility in the query and result specification.

5.1 Infrastructure

The core search computing infrastructure must deal with two main aspects: *query execution* and *query plan optimization*. The latter is optional, in the sense that a trivial plan can be defined starting from the specification of the designer at composition time.

5.1.1 Query execution

The processing of query execution schedules is in charge of a query execution engine. The executable schedules include fine-granularity operations, like service invocations, and control structures, that define parallelism, sequence, branching, and so on.

The execution schedule is a lower level representation of the visual language that describes query plans. At this level, the plan could include an explicit allocation of cache memory, as well as the exploration strategies for the join executions.

Apart from enacting the execution and orchestrating the prescribed service invocations, it is the query engine responsibility to cope with any unexpected behavior, and apply correction policies, including the cases of: anticipated stopping policies if the query is likely to generate more results than needed; heuristics to restart the computation when the query returns fewer results than expected; dynamic change of the join strategy in the presence of trends in the scoring functions that clearly contradict the expected ones. In order to leverage parallel execution as much as possible, all invocations must be performed by different threads (normally one per node in the query plan) and results are pushed forward in a continuous way, as soon as they are available, according to a producer-consumer paradigm. Nodes that accept input from more than one node may be blocked waiting for delayed data, but his doesn't prevent other branches from proceeding with the computation.

5.1.2 Query plan optimization

A *query plan* is an orchestration of service invocations that complies with their access modes and exploits the ranking order in which search services return individual results to rank the global query results. To optimize the execution of queries the search system must address the problem of generating query plans and evaluating them against a cost metric so as to choose the most promising one for execution. A preliminary version of query planner was presented in [2].

The optimization accepts as input low-level queries, i.e., conjunctive queries that list the specific services to be invoked. Query plans schedule the invocations of Web services and the composition of their inputs and outputs. A plan is defined as the orchestration of service invocations, possibly in parallel, which takes into account the most significant features of the service, including its ability to chunk the results (i.e., to return a given number of answers with a single request-response). Within plans, the main operations are joins between Web service results, whose execution can take place according to several join strategies, already investigated in [1].

The optimization strategies progressively refine choices and produce an access plan by performing the following steps:

1. Choice of the specific access patterns for each of the services involved in the query;
2. Definition of the order of invocation of the different services, some of which may be invoked in parallel;
3. Definition of an execution strategy for each join operation between services;
4. Definition of an execution cost for each plan, based on the cost, time of execution, and number of calls to services.

The Query Planner searches for an optimal query plan by considering all feasible choices in the above context, yet reducing its search space by a branch-and-bound exploration that associates expected costs with every choice. A suitable cost metrics is the total execution time, but others are possible.

The outcome of the query planner is the selection of the access plan that minimizes the cost of interaction with the services, while producing a given expected number of results in output; results are lists of entries, ranked by the combination of low cost and high number of beach stars (which are clearly independent criteria).

While query plan designate the orchestration of several services and the methods used for their integration, more sophisticated optimization methods can be devised for their join, including methods which guarantee the optimality of top-k result extraction. The result is a detailed *query schedule*. For instance, a more refined model could consider objects that can be accessed according to various methods, broadly classified as *sorted*, producing a very long ranked list of objects, or *attribute-based*, producing a narrower set of objects, normally not ranked, which satisfy a selection over the attributes. The query planner formulates the problem of optimal extraction of top-k combinations, whereby the optimization is performed with respect to the access costs involved with the different services and the available access methods. For the specific case of the binary join between two Web services (e.g. finding the top ranked hotel-restaurant combinations, i.e. with highest combined score, in the same city district), we devise an iterative execution strategy that, at each step, determines the way of accessing services, such that the probability of obtaining the combinations with the highest combined scores is maximized, while the overall cost of accessing the services is minimized.

5.2 Deployment

The "operational semantics" defined by the producer-consumer approach perfectly matches the requirements of deployment on highly parallel computing infrastructures. While a monolithic deployment configuration is always possible, search computing is flexible enough to be deployed on cloud computing environments. This provides the option of delivering the framework as a standalone application to be installed at the customer premises, or to provide it according to the SaaS (Software as a Service) paradigm, exploiting commercially available services.

This approach provides the additional advantage of allowing automatic load balancing in case of peaks of requests. Indeed, once the search computing system is configured and in place, massive query submissions could compromise its performances, as for any other open web system. In case of search computing, three main potential bottlenecks can be identified:

- *Search services:* search services generally have a limited throughput and may be described by a set of non functional properties that describe their behavior with respect to changes to the workload. An overwhelming number of requests may overload the service, finally producing a denial or service error;

Table 1. Summary of activities, involved roles, and expertise needed for search computing.

	Service development and framework adaptation		Service registration				Application development			Application execution		
Activity	Search service development	Framework adaptation	Wrapping and registration	Access path and join path definition	Service annotation	Default presentation definition (UI default)	Choice of domains	Selection of services and joins	Design of the user interface	Query submission	Query refinement	Result browsing and refinement
Type of activity	Software development	Software development	Lightweight software develop.	Visual mappings and selections	Visual mappings and selections	Graphics and stylesheet definition	Analysis and requi-rement specificat.	Visual mappings and selections	Graphics and stylesheets	Web interface interaction	Web interface interaction	Web interface interaction
Role	Developer	Developer	Software Designer	Software Designer	Software Designer	Graphic Designer	Software Designer	Software Designer	Graphic Designer	Final user	Final user	Final user
Needed Expertise	Programming knowledge (Java, .NET, WSDL, ...)	Programming knowledge (Java, .NET, WSDL, ...)	Web Services Interfaces	Search Services basics	Search Services basics	Graphics	Domain and requi-rements	Search Services basics	Graphics	No specific skills	No specific skills	No specific skills

- *Choreography optimizer* (aka, query planner): in case of open multi-domain search infrastructure, the optimization might be needed at query time too, which could lead to an additional bottleneck instead of providing performance improvements;
- *Choreography execution engine*: the execution engine could become a limited resource too, thus slowing down the performance of all the queries.

Although several specific strategies could be applied to these different components to improve their performances (that should be applied by the producers of the components at the various levels of the architecture), at this stage we mainly envision a unified view to the scalability problem. Basically, we identify each component of the framework as a valuable resource, which can be shared among various processes, and therefore may need some load balancing strategy to overcome the risk of low performances. Therefore, we rely on the *cloud computing* vision [10]. Ideally, all the resources and subsystems of the architecture can be virtualized and assigned to a cloud computing platform, which allows for automatic replication of the resource and load balancing upon traffic overload. Several cloud computing environments now exist and can be exploited as state-of-the-practice systems for any purpose. One of the most know examples is Amazon EC2[1], a web service that provides resizable compute capacity in a cloud of machines. It is designed to make web-scale computing easier for developers, who can setup and configure a cloud structure for a single-server Web application with limited efforts. Thanks to these features, cloud computing appears to be in line with the search computing approach.

5.3 Flexibility of the Interface

The design of a user interface for search computing systems must deal with:

- building a interface for the user to express multi-domain queries in a facilitated way, by also providing hints about his expected semantics (e.g., personal service preferences, a priori disambiguation of terms, etc.);
- building an interface for presenting results, incorporating an explanation facility, whereby the user can drill down the result set and understand where each piece of information comes from;
- enabling query refinement, whereby the user can peruse the results of past queries to better reformulate his information need (e.g., using a faceted query modality over the result set to narrow down the scope of query processing to selected services/domains, adding terms to the query to make it more precise, and so on);
- enabling result refinement, by allowing to change the shape and the extent of the results, (e.g., by dropping extracted details, asking for further information, and so on).

These aspects blend together in a vision towards fluid treatment of queries, whose borders become flexible and allow continuous query and result refinement. Support of such flexibility can be performed within the environment instead of being delegated to developers, as the search computing framework can automatically provide this behavior, by exploiting the information gathered on services at registration and composition time. Interfaces and interaction features can be automatically calculated, given a set of properties of services and their orchestrations.

6. DEVELOPMENT PROCESS

Once the system and the development tools are in place, the actual development process for search computing applications must be put in place. Although traditional development cycles (e.g., waterfall, Bohem's spiral, or fast prototyping) could be applied to search computing, the peculiar features of search systems must be taken into account when defining the development methods. Among the various aspects to be considered, the most crucial ones are:

- The *general vs. vertical focus* of the application: the first decision when designing a search system is whether to implement a general purpose search or a domain-specific vertical search. In the former case, a generic user interface must be designed and knowledge and services from several domains must be gathered and query plans must be completely flexible. In the latter case, a specific canned interface must be devised, allowing the user to submit only the required parameters; selections of services and plans is

more limited. This must be considered both at the requirement collection and design phases.

- The need for *components provided by third parties* (in particular: search services, as discussed so far; description of domains; tagging systems): this implies that a novel phase must be introduced in the process, for scouting and investigating the existing ecosystem of the domains of interest.
- The need for *configurability* of the interfaces and of the overall applications: the continuous evolution of several pieces of the architecture (services, tags and descriptions, interfaces, results) makes several steps of the development more conveniently located at runtime instead of design time. For instance, decision over possible "join paths" between search services can be postponed at query registration and execution time, while the definition of connections between classes/entities was a crucial design step in for traditional systems (e.g., think to database schema design).

Overall, these features push forward a trend towards empowerment of the user, who can get more and more flexibility and facilities in the query process. This is matched with a change in the perspective for development too, as summarized in Table 1. Only the basic tasks that deal with service development now require programming expertise. Several design activities are now moved to the service registration and composition phases, where the profiles of the designers only require conceptual understanding of services and queries, and do not ask for low-level programming, since only graphical model-based tools are exposed to the designer.

7. SEARCH COMPUTING VISION

The vision behind search computing is to develop enabling technologies for two new communities of users:

a. *Content providers*, who want to organize their content (now in the format of data collections, databases, web pages) in order to become available for search access by third parties. They will be assisted by the availability of a developer environment facilitating at most their task, and will be provided with the possibility to register their data within a community. In this way, the "long tail" of content providers will see a concrete possibility to deliver searchable information.

b. *Application developers*, who want to offer new services built by composing domain-specific content in order to go "beyond" general-purpose search engines such as Google and the other main players. They will be assisted as well by the availability of a developer environment facilitating at most their task, and will in addition find a deployment environment, either obtained by installing run-time components upon their servers, or - most interestingly - by finding servers already deployed within cloud computing architectures where they could run their applications.

In the simplest scenario, the same person or organization may play the role of service and application developer, and provide to generic users the access to a specific content. In the most interesting and challenging scenario, application developers would act as the "brokers" of new search applications, built by assembling arbitrary services, some of which could be generic, world-wide, and powerful (e.g., general purpose search engines or other generic utilities, such as geo-localization services), while other could be specialized, localized, and sophisticated (e.g., the "gourmet suggestions" about slow-food offers in given geographic regions).

In this vision, a new market of service providers and brokers could be established, with appropriate regulations concerning right to contents and the sharing of profits based upon accountability of the click-through generated traffic or of actual committed transactions. This market is compatible with the current business models adopted by the major search engine companies (e.g., Google or Yahoo), but it may enable much larger communities of content providers and application brokers.

8. CONCLUSIONS

This paper presented a set of problems that need to be solved when addressing multi-domain queries, highlighting the challenges posed to the software engineering field. The main trends that we envision for this kind of applications are: the availability of search services, with specific interfaces that provide more accurate web information retrieval capabilities; a slow but continuous move from one time design to runtime configuration of applications. Thanks to this paradigm shift, most of the development cost involved into new search application development moves from design time to configuration time. This might exploit results proposed by mashup tools, cloud computing frameworks, and user-oriented development.

9. ACKNOWLEDGEMENTS

This research is funded by the "Search Computing" (SeCo) project, funded by the European Research Council (ERC), under the 2008 Call for "IDEAS Advanced Grants", dedicated to frontier research. SeCo started on November 1st, 2008 and will last 5 years, until October 31, 2013.

10. REFERENCES

[1] Amazon. Elastic Compute Cloud (EC2). http://aws.amazon.com/ec2/

[2] D. Braga, A. Campi, S. Ceri, A. Raffio. Joining the results of heterogeneous search engines. Inf. Syst. 33(7-8): 658-680, 2008.

[3] D. Braga, S. Ceri, F. Daniel, D. Martinenghi. Optimization of Muti-domain queries on the Web. VLDB'08, pp. 562-573, 2008.

[4] D. Braga, S. Ceri, F. Daniel, D. Martinenghi. Mashing Up Search Services. IEEE Internet Computing 12(5): 16-23, 2008.

[5] I. Elgedawy, Z. Tari, and M. Winiko. Exact functional context matching for web services. In ICSOC, 2004.

[6] R. Fagin. Combining fuzzy information from multiple systems. J. Comput. Syst. Sci., 58(1):83-99, 1999.

[7] R. Fagin, R. Kumar, M. Mahdian, D. Sivakumar, and E. Vee. Comparing partial rankings. SIAM J. Discrete Math., 20(3):628-648, 2006.

[8] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. J. Comput. Syst. Sci., 66(4):614-656, 2003.

[9] C. Fellbaum, ed. WordNet: An Electronic Lexical Database (Language, Speech, and Communication). MIT Press, May 1998.

[10] G. Gottlob, C. Koch, R. Baumgartner, M. Herzog, S. Flesca. The Lixto data extraction project: back and forth between theory and practice. ACM PODS 2004, Paris.

[11] B. Hayes. Cloud computing. Communications of the ACM 51(7): 9-11 (2008).

[12] I. F. Ilyas, W. G. Aref, and A. K. Elmagarmid. Supporting top-k join queries in relational databases. VLDB J., 13(3):207-221, 2004.

[13] I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top-query processing techniques in relational database systems. ACM Comput. Surv., 40(4), 2008.

[14] D. Kossmann, F. Ramsak, S. Rost. Shooting stars in the sky: an online algorithm for skyline queries. In VLDB'02, pp. 275-286.

[15] N. Mamoulis, M. L.Yiu, K. H. Cheng, and D. W. Cheung. Efficient top-k aggregation of ranked inputs. ACM TODS, 32(3), 2007.

[16] C. D. Manning. Probabilistic Syntax. In Rens Bod, Jennifer Hay, and Stefanie Jannedy (eds), Probabilistic Linguistics, pp. 289-341. Cambridge, MA: MIT Press, 2003.

[17] MetaSearch. http://www.lib.berkeley.edu/TeachingLib/Guides/Internet/MetaSearch.html.

[18] D. Papadias, Y. Tao, G-Fu, and B. Seeger. Progressive skyline computation in database systems. ACM TODS, 30(1):41-82, 2005.

[19] M. Papazouglu and K. Pohl eds, Wp 2009-2010 expert group: Longer term research challenges in software & services. 2008.

[20] A. A. Patil, S. A. Oundhakar, A. P. Sheth, and K. Verma. Meteor-s web service annotation framework. In WWW 2004, pp. 553-562.

[21] S. Ran. A model for web services discovery with QOS. SIGecom Exch., 4(1):1-10, 2003.

[22] Stanford Natural Language Processing Group. Statistical parser. http://nlp.stanford.edu/software/lex-parser.shtml

[23] M. Stollberg, U. Keller, H. Lausen, and S. Heymans. Two-phase web service discovery based on rich functional descriptions. In ESWC '07: pp. 99-113. Springer-Verlag, 2007.