Data-driven optimization of search service composition for answering multi-domain queries

Davide Barbieri, Alessandro Bozzon, Daniele Braga, Marco Brambilla, Alessandro Campi, Stefano Ceri, Emanuele Della Valle, Piero Fraternali, Michael Grossniklaus, Davide Martinenghi, Stefania Ronchi, Marco Tagliasacchi
Dipartimento di Elettronica e Informazione, Politecnico di Milano
P.zza L. Da Vinci, 32. I-20133 Milano, Italy
[firstname.lastname]@polimi.it

ABSTRACT
Answering multi-domain queries requires the combination of knowledge from various domains. Multi-domain queries are inadequately answered by general-purpose search engines, because domain-specific systems typically exhibit sophisticated knowledge about their own fields of expertise. Moreover, multi-domain queries typically require combining domain knowledge in the result, in a way that is not conveyed by single web pages, therefore conventional crawling and indexing techniques are not adequate. In this paper we present a conceptual framework for addressing the composition of search services for solving multi-domain queries. The approach consists in building an infrastructure for search service composition that leaves within each search system the responsibility of maintaining and improving its domain knowledge, and whose main challenge is to provide the glue between them; such glue is expressed in the format of joins upon search service results, and for this feature we regard our approach as "data-driven". In this paper, we present an overall architecture, and also present the work that has been done so far in the development of some of the modules.

1. INTRODUCTION
The current evolution of the Web is characterized by an increasing availability of online services (e.g. book search services provided by online stores or libraries) and novel search facilities (e.g. flight search Web sites, provided by most commercial airlines or travel offers integrators). Being specific to a restricted domain, they offer a quality of their answers that 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 index-able by search engines.

In light of these considerations, multi-domain queries, i.e., queries that can be answered by combining knowledge from two or more domains (i.e., commercial sectors, cultural fields, and so on), no longer represent a mere academic exercise; rather, they witness how intricate real life queries may be, and what a user would like to find available in order to fulfill real needs. However, we are still lacking effective query systems on the Web allowing users even to ask similar queries.

Answering multi-domain queries requires the combination of knowledge from various domains. These queries are hardly managed by general-purpose search engines, because domain-specific systems exhibit more sophisticated knowledge about their own field of expertise. Examples include expertise about cultural events, medical specializations, popular rock songs, and so on; knowledge can be contributed through social processes or through a long and careful process of knowledge construction by experts.

With the advent of service computing and the growing interest of 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 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 need to be semantically connected. At the current state of the art, the above needs can be answered only by patient and expert users, whose strategy is to interact with specialized services one at a time and then feed the result of one search as input to another, reconstructing answers in their mind.

In the search computing project, we present a conceptual framework for addressing the composition of search services for solving multi-domain queries. Our approach consists in building an infrastructure for search service composition that leaves within each search system the responsibility of maintaining and improving its domain knowledge, and whose main challenge is to provide the glue between search service competences; such glue is expressed in the format of joins upon search service results, and for this feature we regard our approach as "data-driven". Search computing is a five-year project sponsored by the ERC, and this paper presents a preliminary vision on the constellation of problems to be solved and of software components to be developed in such context. In this paper, we give a high-level description of the approach in terms of an overall architecture, and also present work that has been done so far in the development of some of the models and software modules required by the architecture.
We are aware that the general formulation of the search computing problem, going from the registration of arbitrary services and the 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. Ultimately, vertical solutions combining a limited number of domains could be encoded with a predetermination, within suitable query forms, of distinct keywords for each domain of interest. However, at least in this early phase of the search computing project, we like to explore the problem in its full complexity, and approach it with an experimental attitude, knowing that the most difficult tasks can be addressed in a simplified way, and yet attempting general solutions.

To overcome the difficulties of addressing a full English vocabulary and natural language understanding we resort to the use of consolidated resources, such as WordNet [17] and the Stanford Natural Language Parser. We attempt a lightweight semantic approach, which does not exploit full expressive power of domain-specific ontologies and reasoning techniques, but instead leverages on WordNet and its functionalities for the annotation of queries, services, and domains. Given that semantics provided in this way is too limited, we add a significant amount of user feedback and manual refinement, both in the definition of services, domains, and queries. Indeed, our framework (once developed in all its components) will be a very interesting testbed for tuning the level of user feedback and refinement, and at the same time measuring the amount of deviation from expected results that the system will give in the lack of such tuning.

2. OVERALL ARCHITECTURE AND EXECUTION FLOWS

Within the multi-domain query answering problem we identify two main activity flows: the registration flow - that deals with the creation of new domains, domain descriptions, and search services within the framework - and the query execution flow - that deals with the actual enactment of the queries. Figure 1 shows the overall architecture of the system, together with the two main execution flows.

The conceptual models that are needed in order to describe the various ingredients of the objects we propose are: (1) domains and their definitions (both in terms of taxonomies and of concepts which describe domains); (2) search services (request/response interfaces with annotations for in/out parameters and response description, including also functional and non functional properties); (3) high level multi-domain user queries (simplified natural language queries, composed by subqueries); (4) low-level queries (adorned conjunctive datalog queries); (5) query plans (description of query execution strategies, using coarse-granularity operations which comply with access limitations and define ranking-aware strategies for building results); and (6) query execution schedules (well-defined schedules of fine-granularity operations, including service invocations, which highlight execution controls and the possibility of parallelism).

Along the registration flow we address the following problems: (a) semantic description, storage, management, and access to domains and their descriptions; (b) semantic description, storage, management, and access to search services; (c) clustering of services based on similarity; (d) mapping of services to domains; and (e) definition of join conditions between services.

Along the query execution flow we address the following problems: (f) definition of proper interfaces for submission of multi-domain user queries; (g) splitting the query into subqueries; (h) mapping of subqueries to domains; (i) mapping of subqueries on given domains to associated search services, at the purpose of defining low-level queries; (j) generating query plans and evaluating them against several cost metrics so as to choose the most promising one for execution; (k) generating and processing query execution schedules; and (l) transformation and rendering of the results for user consumption.

In the following two sections, we describe a general architecture for addressing the multi-domain query problem, define its decomposition into sub-problems, assign each sub-problem to a component, and sketch a technical solution for developing each of them. Registration is performed by developers, and in such case the framework helps them in selecting the domains and services, annotating them, and developing mappings between them. Queries are performed by users, whose feedback helps in resolving ambiguities and confirming interpretations. We address the registration flow in Section 3 and the query flow in Section 4. To better appreciate the approach, we consider a running example, consisting in the domain, service, and query analysis steps required to answer the query: “Where can I attend a DB scientific conference close to a beautiful beach reachable with cheap flights?”

![Figure 1. Overall architecture and execution flows.](image-url)
services. These aspects are addressed by the following components.

3.1 Domain framework

The domain framework deals with domains and their definitions (1) and addresses the problems of semantic annotation, storage, management, and access to domains and their descriptions (a).

The whole infrastructure of the multi-domain search engine is based on the concept of domain. Intuitively, we consider a domain as a self-standing field of interest for the user, such as music, sport, arts, tourism, computer science, and so on. Every domain is associated to a distinctive label and associated with a bag of Wordnet synsets\(^1\); each synset can be associated with more domains, and such association is further characterized by a probability distribution. This allows to characterize a domain in terms of most frequently used terms for describing concepts in that domain, and vice versa to identify for each synset the list of domains it refers to. Moreover, domain definition will take advantage of Wordnet Domains [18], a definition of about 200 domains which have been produced in order to partition the Wordnet vocabulary into parts and to associate each part with a specific domain of interest. One of the most interesting and urgent task in search computing is to investigate if Wordnet Domains can facilitate the task of partitioning queries and associating them to specific search engines and data sources.

The domain repository is a data structure that is able to store domains as described above. In our solution, we assume that domains are organized as a taxonomy, representing a tree of domain-subdomain relationships. Information about the domains is made available to the other components through an API that exposes interfaces for querying and updating the domain structure (i.e., creation, deletion, and update of domain information, including associated synsets and services).

In the proposed example, domains concerning Scientific Conferences, Beaches, and Flights are required. Beach is included within a more general domain of Geographic Resource, and both Geographic Resource and Flight are found as sub-domains of Travel.

3.2 Search service framework

The search service framework defines a conceptual model of search service (2) and addresses the semantic annotation, storage, management, and access to search services (b). The core function performed by the framework is to enable the annotation of the request/response interface of the services. Such annotation uses the Wordnet vocabulary and adds labels to each service, its operations, and the input - output parameters of each operation. In our framework, we are concerned only with those operations belonging to a Web service which perform data retrieval, and we are particularly interested in those operations which return itemized and ranked information.

Our repository exposes services in terms of operations; each operation is described by means of a set of functional or not functional attributes. The qualifying attributes are the id, the name, the descriptor (Wordnet annotation), the serviceName (the name of the service exposing the operation), the input and output descriptors and types, the average response time and cost of interacting with the operation. In addition, several parameters describe the qualifying aspects of search services: the ranking description (a parameter indicating if the result is ranked), the caching description (an indication whether the service results can be cached, and if an indication of the validity time of cached data), the decay description (an indication of the decay trend of service results), the chunking description (an indication whether service results are returned as a single reply or instead are returned by chunks, where every chunk is a given number of items, that can be requested by means of an iteration-based interface).

While the above parameters are essential to our framework, more information characterizes a classical service repository. According to [15], the most important QoS parameters are scalability, capacity (describing the limit of concurrent requests for guaranteed performance), performance (a measure of the speed in completing a service request), response time, latency, throughput, reliability (expressed in terms of mean time between failure and of mean time to failure), robustness, accuracy (defines the error rate produced by the service), and completeness.

The registration process is semi-automatic: a user interacts with the service registration framework while registering a particular service. The system analyzes the names of the service, of the operations and of the parameters, and tries to assign each of them to a domain, in particular try to assign one possible term and one possible synset in WordNet. This process is semi-automatic since it requires different interactions with user (for example for each term the system shows to the user different possible synsets and ask the user to choose the most appropriate). Once this phase is ended, the system automatically starts to collect information implicitly hidden [16]. For example, the availability and response times of a particular operation can be collected simply measuring them during the interaction with the service.

It is important to note that, while the information about a given domain could be available on the Web, it could be unavailable in the form of a service. Thus, an extension of this step covers the aspect of wrapping existing data sources so that the wrapper eventually exhibits some of the mentioned properties of a service – most important, exhibit a request-response interface and is capable of some form of aggregation, chunking, and caching of results. We aim at providing developers with tools that will make such wrapping rather easy in the classical case of web sites which are accessible through entry forms, thereby assuming that the form can provide the same information as a request of service, and that the results, once extracted from one or more result pages, can also provide structured information in the form of a set or list of entries, each described by a given number of parameters.

In the proposed example, registration of services relative to conferences may lead to availability of several services, e.g. with different choices of input and output parameters, since conferences can be searched by topic, name, place, start date, or some of these inputs in combination. One such service, relevant to our running example, could be:

\[\text{ConfSearch(topic, nameX, placeX, dateY)}\]

where topic is an input parameter, name, place and dates are output parameters, and the service and parameters are further annotated through wordnet descriptors. Results are collections of triples of names, places, and dates extracted once the input is assigned to a given topic, that can be either ranked or not ranked;
ranking may take into account the relative importance given by a community of users to the conference itself; the service may be characterized in terms of availability and of accuracy.

### 3.3 Service analyzer

The service analyzer will primarily address the following problems: the clustering of the available services, based on their similarity (c); the mapping of services to domains (d); and the definition of join connections between services (e). This part of the framework requires the other parts to be prototyped, hence is in a very early stage.

Items (c) and (d) above require the use of clustering algorithms, such as Lingo [12], in order to gather together similar web services and to map them to domains on the basis of their content description.

The aim of the clustering process is the grouping of all the available web services and their probabilistic association with a set of domains. The clustering process will take advantage of the availability of synset annotations for services and their input/output parameters, and will also exploit the presence of a light weight semantic associated with registered web service in order to associate them to one of the Wordnet Domains. While the clustering process will be periodically performed, we also envision an incremental process of adding a service to an existing cluster. Developers will be asked to verify the correctness of the choices made by the system. In addition, they will be offered the possibility to expand the domain ontology of WordNet Domains with other terms and relationships.

While clustering the services and associating them to domains, we will also build information about the “belonging degree” of each service to the corresponding cluster, evaluated as the cosine distance between the vectorial representation of the service in a term vectorial space, and the centroid of the cluster. This information can be seen as an expression of how much a specific service is correlated to the domain/s identified from its cluster/s of belonging. We will then see if this information can be pragmatically used in the selection of search services.

The second major task of the service analyzer is item (e) of the registration workflow, i.e. the definition of admissible join paths between services. The goal of the 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. A possible solution to this problem requires first the definition of a classification of the services within each cluster, computed on the basis of their operations interfaces (name and parameters annotations and types). In this way, for each pair of classes belonging to different domains, we 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 are offered to indicate if the join paths as 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.

In the running example, the goal of this step is to identify that services such as ConfSearch (describing conferences) and Flights (defining available flights to reach the conference) can be connected with matching place, starting and finishing dates of the conference and of a roundtrip flight, whereas the conference place is used as destination of a roundtrip from a user designated location, the starting date is used to designate the date for the first travel, and the termination date is used as return date. Moreover, while building such connection, the flight service is identified as one associated with ranking, and specifically its total cost.

### 4. Query execution flow

#### 4.1 Query analysis

In this section we discuss the conceptual model of high level multi-domain user queries (3) and we address the problem of splitting the high-level query into subqueries (g). A high level query is the specification of an information need of a user at a high level of abstraction. In our work we assume high level queries to be quasi-natural language descriptions of the user request which may require to extract information from multiple domains. The only restriction we impose on the queries is that they must consist of a set of noun phrases, i.e., phrases whose head is a noun or a pronoun, optionally accompanied by a set of modifiers (e.g., adjectives).

The query analysis component decomposes the high-level queries into sub-queries, each representing one search objective in a specific domain. As an example, a query like “Where can I attend a DB scientific conference close to a beautiful beach reachable with cheap flights?” would be split into Q1= “DB scientific conference?”, Q2=“Place close to a beautiful beach?”, and Q3=“Place reachable with a cheap flight?”.

For processing the natural language query, we exploit an open source tool developed by the Stanford Natural Language Processing Group. The tool implements a probabilistic lexical parser of English natural language sentences. The outcome of the parser is a tree representation of the sentences that is suitable for the problem of splitting the queries into subqueries to be assigned to different domains.

The most promising approach seems to consist in applying a first splitting of the sentence, and then assessing whether the generated subqueries map consistently to separate domains, by invoking the Query-Domain mapper. If the mapping is not coherent (e.g., several very different domains refer to the same subquery), we conjecture that the splitting may not be solid enough, and therefore we: (i) ask for feedback from the user; or (ii) we try a different splitting based on cohesion of words with respect to domains. The final result of the splitting in (high-level) subqueries is therefore just a first step towards the mapping of subqueries to domains.

#### 4.2 Query to domain and service mapping

This component addresses the problems of mapping of subqueries to domains (h) and mapping of subqueries to associated search services, at the purpose of defining low-level queries (i). The operation of mapping a query to a domain can be successful only if: (i) each subquery comprises only requests to one domain; and (ii) the words used in the subquery are unambiguous, thus allowing a crisp identification of their semantics.

Several techniques can be applied to optimize the recognition of query-subquery structures which comply with the separation into distinct domains of concern so as to achieve the objective (i); these include:

---

• iterative invocation of the NLP tool based on defined lexical interpretation obtained from feedback from user, or feedback from other components;
• exploitation of annotations of search services or domains for assessing the correctness of the query splitting;
• syntax/logic analysis results on the sentence.

The second precondition can be satisfied by replacing all the words in the query with the correct synsets. The latter task is in general hard. We apply some heuristics for reaching the goal:

• synset domain coherency: as a first step, we try to infer the correct meaning of the word by: (1) extracting all the synsets of the words used in the queries; (2) calculating the groups of synsets that better map to one domain (or to nearest domains); (3) selecting one synset per word, according to the grouping defined before.
• user feedback: in case no final decisions can be taken in the selection of a group of synsets for the words, the user feedback is requested.

To clarify the approach, consider the following query: "rainy US states". The problem of identifying the domain of this query can be reduced to:

• identifying the synsets of the words: rainy has just one synset (ADJ: (1) showery, rainy); US has four synsets (NOUN: (2) uracil, (3) uranium, (4) U, (5) United States); state has several synsets, we report only a few here (NOUN: (6) state, province; (7) the way something is; (8) group of people comprising the government of a sovereign state; (9) state of matter).
• grouping the synsets by domain: for the example, we assume that: synset (1) is associated to the domain `weather`; synsets (2), (3), (9) are associated to `chemistry`; synsets (4), (7) are associated to `grammar`; synsets (5),(6),(8) are associated to `geography`.
• selecting one synsets for each word: considering the closeness of the domains, we can infer that probably the significant synsets involved here are: (1) (5) (6/8) (related to the geographical domain, which is the correct interpretation of the sentence), or (1) (2/3) (9) (related to the chemical domain, which is not correct and indeed has a lower probability because of the high distance of (1) from (2/3/9)).
• if the probability difference between the two options is not so high, the user feedback could be requested in terms of a question like: `are you asking information about geography or chemistry?".

Once the domain is identified, analogous techniques can be applied for mapping the query to the services. The ultimate goal of this task is to match each of the identified subqueries to one or more services, so that the following processing of the query can take place according to a well-defined query execution strategy. Of course, this step is extremely difficult, and therefore users' feedback may be required at various levels, in particular regarding the query rewrite, the domain selection, and the mapping to specific services with well identified join paths connecting them.

By taking again as input the running query: "Where can I attend a DB scientific conference close to a beautiful beach reachable with cheap flights?" which was already broken down into Q1="Where can I attend a DB scientific conference?", Q2="Place close to a beautiful beach?", and Q3="Place reachable with a cheap flight?", the ultimate result of such processing is the selection of services:

```
S1=ConfSearch("DB",Title,Place,StartDate,EndDate);
S2=PhotoSearch("Beach",Stars, PhotoID,Place);
S3=RoundTripFlight(From, To, FromDate, ToDate, TotalTime, TotalCost, RoundTripID)
```

With joins pairing S1 to S2 by Place and S1 to S2 by Place=To, StartDate=FromDate, and EndDate=ToDate. Moreover, the word "cheap" indicates interest in a ranking of flights, and the word "beautiful" indicates an interest in an evaluation of the beach's beauty, that can be obtained by an indication of the "number of stars" that the beach has been given, and by linking the result to a photo of the beach so as to enable an individual evaluation. While we understand that doing all such steps dynamically, at query presentation time, and without help from the user goes beyond the current state-of-art, we envision progressive steps to achieve this result at least in part through a well-designed user interaction. We therefore anticipate a huge amount of experimentation in this component, leading us to draw conclusions concerning the practicability of the approach; of course, the complexity of this step can be arbitrarily reduced by constraining the choice of queries, domains, search systems, and their pairing through join connections.

We also designed and partially implemented a visual language, in the form of mash-up, for query plan specification that allows designers to submit low-level queries [], thereby substituting in full steps 4.1 and 4.2.

### 4.3 Query planner

A low-level query is a conjunctive query over services. A query plan is a well-defined scheduling of service invocations, possibly parallelized, that complies with their access modes and exploits the ranking order in which searches return results to rank the query results. The Query Planner addresses the problem (j) 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 [], while we are currently engineering an extended version.

The Query Planner accepts as input low-level queries, i.e., conjunctive queries that list the specific services to be invoked, already chosen at the Query-To-Domain Mapper. It is assumed that each such service is associated with a description in the service repository.

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.

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 page 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 [].

The Query Planner progressively refines choices and produces an access plan by performing the following steps:
1. Given that services may be accessed according to different patterns, the Query Planner chooses specific access patterns for each of the services involved in the query, provided that they are compatible with the query.
2. Once the access patterns are fixed, there may still be some indeterminacy on the order of invocation of the different services, some of which may be invoked in parallel. The Query Planner fixes such order.
3. The main operation for combining search services in our conjunctive setting is the join. The Query Planner selects an execution strategy for each join.
4. Optimality of execution primarily depends upon the cost and time of execution of request/responses to services. The Query Planner determines the expected number of requests associated with each service request in order to obtain the desired number of results, so as to associate to each plan an execution cost.

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.

An example of complex query that can be considered by the optimizer is:

\[
\text{BestMatch(ConfTitle, Place, Stars, DateStart, DateEnd, TotalCost, TripId, PhotoID); -}
\]

\[
\text{ConfSearch("DB", ConfTitle ,Place, DateStart, DateEnd),}
\]

\[
\text{PhotoSearch("Beach", Stars, PhotoID, Place),}
\]

\[
\text{RoundTripFlight(City, Place, DateStart, DateEnd, TotalTime, TotalCost, TripID),}
\]

\[
\text{City=InputCity, DateStart=InputPeriodStart, DateEnd<InputPeriodEnd.}
\]

This query is presented with user input indicating the city and period of interest. One can think of addressing progressively more contexts in which such formalization can be used, staring from a situation where the query is installed and made available to users within a vertical application (e.g., offered to the university’s travel agent) leaving the initial city of the user and the period description as the only variable parts of the query, up to a context where the query is dynamically presented, understood, mapped, and executed. Of course, in a fully dynamic setting, there would be no difference between the “Input” values (user’s city of residence and suggested period) and the other constants in the query (such as “beach” and “db”), therefore the complete query to be considered in such case is “Where can I attend a DB scientific conference close to a beautiful beach reachable with cheap flights, staring from Milano, in the next 4 months?”; “Milano” and suitable dates would then be included in the query.

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). An example of access plan, taken from [1] for a slightly different version of this same query, is given in Figure 2. In the model:

- All service invocations are marked with an expected number of items provided in input (\(t^{\text{in}}\)) and in output (\(t^{\text{out}}\));
- Exact services (conf and weather) are represented as simple boxes, chunked services (flight and hotel) are segmented by vertical lines. The latter are also search services, marked with a grey triangle recalling the decay of the ranking of their results, the former are exact services (conf is marked with a * because it is proliferative service, i.e. in average it returns several items in response to every input, whereas weather is a selective service which selects some of the input items; all search services are proliferative);
- The number F of subsequent invocations to be performed on each chunked service is specified;
- Join nodes are marked with a join strategy (MS stands for Merge-Scan, see [1]) and are also marked with estimations of the size of the input and their output (derived by applying the expected reduction due to the selectivity of the join predicate).

The aforementioned annotations result from the static optimization criteria applied by the query planner, and if the runtime behavior strictly matches the expected one the query will return exactly the top K results (in the example, K=15).

![Figure 2. A fully specified query plan](image-url)

While query plan designate the orchestration of several services and the methods used for their integration, we are studying more sophisticated methods for their join, including methods which guarantee the optimality of top-k result extraction. More specifically, for point 3 we recently described a join strategy suitable for the case of Web services that output data objects ranked by score, which adapts to the join of ranked services the FA method designed by Fagin [1]. The ranked lists may contain a high number of objects, typically presented in pages, and accessing such pages is costly. Moreover, objects 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. Such optimization strategy can be
practically deployed in a search computing setting, since it requires a minimal set of parameters that characterizes the joined services, which can be obtained at the time of service registration and possibly refined during the execution of queries involving those services. These parameters include, for example, estimates of sorted and attributed-based access costs, cardinality of objects returned by the services (e.g., total number or hotels/restaurants), average number of distinct join attributes (e.g., average number of hotels/restaurant per district). We are currently working towards the extension of the aforementioned optimization strategy to the case of joins involving more than two services.

4.4 Query engine

The query engine deal with the generation and processing of query execution schedules (k); these include fine-granularity operations, including service invocations, and thus facilitate execution controls, also in the presence of parallelism. The input of this step is a query plan generated by the planner, like the one in Figure 2.

The execution schedule is a lower level representation of the visual language that we use to specify the behavior of joins in terms of number of iterations and interleaving of fetches from the different input items. A very simple example is in Figure 3, where a Nested Loop option for the join between the results of flight and hotel [] is expanded into a specification in which:

- Iterators are represented by “circle” units with the specification of the iteration range (in this case, simply chunks 1-to-4 and 1-to-3 for);
- Fetching of pages of results is represented by units with the upgoing arrow as icon;
- Dashed arrows represent the control flow in the execution, whereas regular lines represent the data flow.
- The join node represents a simple operation that applies the join predicate to the data items it gets in input in the order in which they are provided.

![Figure 3. Unit-based specification of the Nested Loop join](image)

At this level, the plan could include an explicit allocation of cache memory to store partial results of sub-queries and portions of pre-computed joins between the results of frequently invoked services with the most frequent inputs, as well as the specification of exploration strategies for the join search space that are not simply expressed by combination of simple iterations, but follow sophisticated methods like the FA method extension, discussed in the previous section. We are currently working on the selection of a limited number of nodes representing reusable operations and on efficient schemes for parameter passing between nodes, so as to give to the language, at the same time, high expressive power and good ability of being used for join strategy generation and testing.

The results generated by the service nodes and the combinations returned by join nodes are collected in their “raw” format of tuples of values, and passed to the Result Transformation module, to be processed in order to be presented to the user.

Apart from enacting the execution and orchestrating the prescribed service invocations, it is the query engine responsibility, instead, to cope with any unexpected behavior, and apply correction policies. We are currently investigating:

- Anticipated stopping policies if the query shows to be likely to generate more than K results. Whenever a service that is “initial” in the query graph provides more results than expected, this heuristically allows for limiting the search space of the subsequent ones;
- Heuristically effective strategies to restart the computation of “completed” nodes 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.

All these issues also correspond to situations in which the planning was not accurate enough, and feedback to the planner has to be provided. The optimal balance between heuristic deviations from the optimized plan and continuous feedback (which may even be pushed to halt the execution and request a new overall optimization with new parameters) is, in turn, a challenging research problem.

In order to leverage parallel execution as much as possible, invocations are all 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. Nodes that accept input from more than one node (all join nodes, but possibly service nodes as well) may be blocked waiting for delayed data, but their doesn’t prevent other branches from proceeding with the computation. It is worth noting that this “operational semantics” ideally leads itself to be deployed on highly parallel computing infrastructures.

We will work on the deployment of execution environments which will initially support simple schedule executions, and will then be augmented to deal with exception handling and dynamicity, and finally support optimal caching, pipelining, and parallelism.

4.5 Human-computer interface

This component will address the following problems: (f) definition of proper interfaces for submission of multi-domain user queries; (k) transformation and rendering of the results for user consumption; it will deal with:

- building a interface for the user to express multy domain queries in a facilitated way, by also providing hints about his expected semantics (e.g., personal service preferences, a priori disambiguation of term, 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 faceted query 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, etc.)

5. RELATED WORK

A great deal of interest is being devoted to extending service-orientation capabilities of software systems, as testified to by current research trends [2]. Search computing is the discipline meant to mark the transition towards better behaved and more reliable systems thanks to a better use of search and composition. Foreseeable extensions include, e.g., achieving better guarantees that can be given to users in the context of dynamically assembled systems, where unsatisfactory or failing services can be changed and when user's requests may vary at each time. We now propose an overview of the most closely related fields.

Meta-search consists in a shallow-level integration of search engines working in restricted domains (e.g. searching the best fare for books or flights). Meta-search engines are Web applications that aim at integrating the results of several search engines that are queried with the same search string. The user typically submits a search request in the meta-search submission form, and the meta-search system forwards the search simultaneously to several individual search engines, whose responses are then shown together in a single result page. A comprehensive review of meta-search (discussing its qualities and limitations) can be found in [1]. Albeit related to search computing, meta-search profoundly differs from it. The set of search engines used by a meta-search system is fixed and predefined, while search computing will foster context-based dynamic selection of search engines as well as source ranking; each source is queried with the specific part of the query that is pertinent to its domain. In meta-search, results are merged with no composition, possibly after sorting based on single-domain common information (e.g., price, departure time, etc.). Search computing comprises a rich compositional framework that allows several strategies to deal with the results of multiple search engines included in a query, and supports several operations other than the simple merge. Moreover, user interaction is articulated through protocols rather than the simple result presentation.

Web service composition provides the basis to be used in search computing to translate a user query into calls to several existing services. At the current state-of-art, composition is mostly the result of human selection, but we can envision situations and systems which will be able to perform composition either partially or fully automatically. Research on automation of Web service compositions exploit different notions, such as functional substitutability[3], service annotation wrt. relevant ontologies[14], and others.

Top-k query answering is a topic that has been addressed by a large body of recent research (see [9] for a survey). The topic is very relevant, as it shows how ranking has been managed in the simpler context of database management. Top-k queries produce results that are ordered on some computed score. Typically, these queries involve joins, where users are usually interested only in the top-k join results. Top-k queries are dominant in many emerging applications, e.g., multimedia retrieval by content, Web databases, data mining, middleware, and most information retrieval applications. The foundations of top-k queries are rooted in the simpler problem of rank aggregation, i.e., the problem of combining several ranked lists of objects in a robust way to produce a single consensus ranking of the objects [5]. Rank aggregation has resulted in a number of algorithms, such as the so-called Fagin's algorithm [4], and the threshold [6], which have been adapted to various extents to the context of top-k queries. Under suitable assumptions, some of these extensions can be proved to be instance optimal, in the sense that the cost incurred by their execution is the smallest possible in every database (modulo a fixed constant). Among these, we mention [8][11].

A known approach to answering queries that pursue optimality with respect to more than one criterion is that of skyline queries. The skyline of a set of d-dimensional points is the locus of the points that are not dominated by any other point on all dimensions. A point dominates another point if it is as good or better in all dimensions and better in at least one dimension. Skyline computation has recently received considerable attention in the database community, especially for progressive (or online) algorithms that can quickly return the first skyline points without having to read the entire data file. Among the most efficient algorithms, we cite NN (Nearest Neighbors) [10] and the IO-optimal BBS (Branch-and Bound Skyline) [13].

6. CONCLUSIONS

This paper presented a set of problems that need to be addressed when addressing multi-domain queries, an architectural view of the problems, and a sketch of the solution techniques adopted for each of them.

7. 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", a program dedicated to the support of frontier research. Seco started on November 1st, 2008 and will last 5 years, until October 31, 2013.

8. REFERENCES


End of Columns (be careful when removing)